université
PARIS-SACLAY

POGL : ennoncés TDs TPs, projet, corrigés 2025-2026 F. Gruau.

Ce poly contient tous les énoncés TD et TP mais aussi leurs corrigés, écrit en petit tout à la fin. J'ai adapté des ennoncés originaux élaborés par T. Balabonsky. J'ai décomposé les grosses questions qui s'y trouvaient, en beaucoup de petites questions, afin de favoriser une progression régulière et synchronisée du groupe. Chaque TP aborde une technique objet, illustrée sur une structure de donnée fondamentale ; vous aprenez ainsi deux choses en même temps. Le receuil de tp et td c'est le graal, votre bible. Vous les avez avec vous pour les consulter en tp et td, mais aussi aux examens, car plus de la moitié de l'examen, (partiel aussi) consiste en des questions additionnelles à ces tps, et tds. Pour être bien préparé, il vous faut donc les terminer chez vous si vous n'avez pas eu le temp de les finir en cours, pour en maitriser le contenu.

Mes défauts Je vous prie de bien vouloir m'excuser pour mon othographe parfois appoximatif, lequel peut rendre plus difficile la lecture ; j'enseigne l'informatique pas le francais, j'essaie de faire de mon mieux, mais je suis un peu névrosé de ce coté la. Je fais les diagrammes à la main, car j'ai des troubles musculo-squeletal épaule/bras/mains qui réduisent mon potentiel d'intéraction sur l'ordi. Parfois c'est pas très lisible, n'hésitez pas à vous plaindre. Je parle vite, n'hésitez pas à me faire répeter si c'était pas clair. Je le ferais avec plaisir et mon respect pour vous augmentera, car je considère les éleves qui participent.

The archive java Tout vos sources pour les TPs sont situés dans une seule et même archive : "eTDTPlatex.tar". Elle se trouve sur ecampus, mais aussi sur les ordis du 336 dans /public/pogl, donc si vous êtes en salle TP, vous n'avez donc pas besoin de rien télécharger. Cette archive contient un module par TP, et chaque module peut s'executer indépendemment des autres. Ensuite dans chaque module de TP, un dossier et deux sous-dossiers contenant :

— Un squelette java à compléter, qui s'éxecute.
— Une correction qui est ce même squelette complété.

L'enoncé du TP1 commence en vous expliquant comment installer. C'est à vous, de réaliser progressivement la trajectoire du squelette "sec" vers le squelette "charnu", au fur et à mesure des questions du TP. Squelettes et squelettes complétés réutilisent parfois les même noms de classe, et cela peut créer des conflits. Pour passer de l'un à l'autre, il se peut que vous deviez exclure le squelette du path ou java cherche ses fichiers, et inclure le squelette complété, ou vice-versa. Pour ce faire, sous InteliJ, vous sélectionnez le dossier du squelette, et faites "mark directory as", puis ensuite au bien "excluded", ou bien "cancel exclusion" suivi de "mark directory as" "sources root".

Executer vos prog toutes les 2mm. Très important : après chaque modif de qq minutes, executer votre code, afin d'en vérifier la correction. Si il y a bien un comportement dénué de sens, mais pourtant trés fréquent, c'est de pisser de la ligne de code sans rien tester en pensant que si ca compile c'est bon. Ben non, c'est souvent pas bon.

Mode d'emploi corrigés. Les corrigés vous offrent la possibilité de travailler en autonomie si cela s'avére nécessaire (éloignement, pb de santé). Les corrigés TP vous montrent au fur et à mesure des bouts du squelette complétés correspondant aux questions, comme cela, le spectacle de la solution totale n'est pas spoilé. On vous donne aussi tout le squelette complété dans l'archive, car c'est quand même sympa de pouvoir consulter/tester/modifier la solution compléte pour bien comprendre avant les exams. Ne regardez surtout pas les corrigés avant d'avoir essayé de faire par vous-même : en effet, on a besoin de sécher sur une question pour bien enregistrer la correction, car sécher produit une forte émotion (cette fameuse émotion : je suis un.e gros.se nul.le, j'y arriverai jamais) et la mémoire imprime lorsqu'il y a émotion. Par contre, si vous bloquer toujours au bout de 5mm sur une question qui demande du code, oui, n'hésitez pas à consulter le corrigé, beaucoup d'entre vous s'y refusent par pudeur. Le corrigé, c'est moi qui suis toujours derriére vous. Mais ne recopiez pas ce code : 1-comprenez le, 2- cachez le 3- reproduisez le. Ainsi vos neurones vont imprégner.

Pour les mordus. Dans certains TPs, il existe des parties considérées comme non-essentielles. Elles sont en général très sympa, mais si vous n'êtes pas fan, vous n'êtes pas obligé de faire. L'objectif est de ne pas surcharger les pas fan. Elles sont marquées par l'indication : "pour les mordus".

Pb de quota Vous arrivez pour faire votre TP tranquillement, et non, rien ne marche ; Il est plus que probable que vous avez dépasser votre quota de mémoire. Lisez bien le message. Il vous faudra dégager vos gros fichiers. Egalement il se peut que vous ayez un trop grand nombre de petit fichiers. Un crash de chrome peut générer des gros fichiers qui vous mettent dans le rouge sans que vous compreniez pourquoi. Dans le tp1 je donne plusieurs pistes pour nettoyez les fichiers.

La solution royale Elle consiste à venir en TP avec votre propre ordi sur lequel vous aurez installez intellij et java. Comme cela on fait face aux salle d'ordi parfois en manque d'ordis.

Le projet Dans vos feedback, on me dis souvent que le projet est ce que vous préferez. L'énnoncé est déja inclus dans ce receuil. Les cours et td relatif au projet on lieu le plus tôt possible, la deuxiéme semaine après le retour des vacances d'hiver, afin de maximiser le temps que vous pourrez y

consacrer. Vous démarrerai immédiatement, c'est la garantie du succès. Je répéte : vous démarrerai immédiatement.

1 TP1, L'énigme des N reines : révisions Java, interface graphique

1.1 Step by step installation.

C'est un peu délicat, lisez bien, allez-y doucement, dou-ou-oucement. Après vous serai bon pour tout le semestre. D'expérience, ceux qui vont trop vite galérent plusieurs semaines ensuite ! Je refais cette manip chaque année avant vous, en condition réelle au 336 (mais à distance), pour m'assurer que tout vas glisser.

Créer un dossier "pogl", c'est important de le faire au bon endroit, le bon endroit c'est chez vous, mais chez vous c'est pas sur le bureau qui présente parfois des rélles embrouilles spécifiques. Ouvrez une fenetre de commande, faites "cd" pour allez chez vous. C'est capital, si vous ne faites pas "cd" pour allez chez vous vous risquer de vous retrouvez dans le sable, dans un répértoire bizarre à cause d'un système compliqué d'anonymisation.

Faites ensuite "cd Document" pour allez dans le repertoire Document, puis mkdir pogl. Prenez une respiration. Copier l'archive dans votre dossier pogl, puis extrayer le contenu toujours dans votre dossier pogl.

Tous les fichiers se trouvent à présent dans le sous dossier "eTDTPjava" de "pogl", il y a un fichier "eTDTP.iml" qui contient des metaDonnées importantes de configuratoion ; Lancer l'application Intellij, en tapant "idea.sh". Attention, attention, vous faite ce lancement en étant dans ce repértoire pogl. Cliquer "open project" et ouvrez le dossier eTDTPjava. Vous verrez tout vos TPs, et cela sera correctement configuré pour tous vos TPs. Attention, ne faites pas l'erreur d'ouvrir juste le TP1-NReines. En effet, l'archive est concue pour que vous ouvriez tout les TPs en même temps, en ouvrant le dossier eTDTPjava qui les contient tous, et non pas chaque TP individuel. Lorsque vous reviendrez, intellij ouvrira toujours le même projet qui contient tout les TPs, et vous serez en route pour la lune ;

En cas de problème allez voir la section problèmes. Il vous faut construire le module avec un click droit sur Nreines.java dans le dossier squelette du dossier TP1, vous devez pouvoir faire buildModule eTDTP, ca compile tout ;

Votre classe principale NReines dans le package nReinessquelette dans le dossier NReines, doit pouvoir s'exécuter. Executer la, il y a plusieurs facon de faire : par exemple en faisant un click droit sur le Nreines.java, puis click sur la fleche verte devant "Nreines.main()". Pour l'instant cela ouvre juste une fenêtre avec deux boutons. Vous consulterez l'énoncé qui suit, pour progressivement compléter le squelette. Vous pouvez si vous le souhaitez également exécuter le démineur qui est dans le dossier avant le TP1 et que nous avons commenté en cours.

1.1.1 Résolution Problèmes ?

Le premier TP, il y a toujours des étudiants qui se retrouvent avec qq problèmes, je vous donne ici toutes les clefs que je connais.

Utilisation des suggestions de Intellij Bien souvent intellij vous suggére directement ce qu'il faut faire pour corriger un bug d'installation, vous n'avez ensuite que a cliquer. Par exemple, il se peut que l'idea n'arrive pas à compiler le TP 7 "formule1", parceque il ne trouve pas junit4. Ce TP porte sur les tests. Lorsque l'idea veut compiler les tests, il dit : "java : package org.junit does not exist" Une solution simple est d'ouvrir par exemple vect.test, et vous déplacer le curseur sur le "junit" qui apparait en rouge dans la ligne au tout début "import org.junit.*;", l'idea vous suggére alors de cliquer sur add Junit 4 to class Path, vous cliquer la dessus, puis OK sur le bandeau suivant qui apparait, et ca résoud le problème.

Spécifier le jdk et la version de java Autre problème possible : si depuis l'idea vous faites la manip trés fortement déconseillée de ouvrir seulement le dossier TP1 au lieu de eTDTPjava qui contient tout les TPs, vous n'aurez pas le bon fichier ".iml" de configuration, il vous faudra spécifier un jdk (version 11) et une version de java (version 8),pour cela allez dans fichiers, puis "project structure" puis dans l'onglet project choisissez le 11 et dans l'onglet module, saisissez 8, on aura besoin du 11 plus tard donc vous pourriez commencez par essayer le 11 et si ca marche pas mettez 8, (j'ai testé pour 8 pas pour 11, parceque google me disait de mettre le 8).

Exclure ou ajouter des ressources dans le path. Si vous faites les choses mal, il vous faudra aussi peut etre "exclure du path" des ressources qui seraient définies deux fois. Cliquer sur le petit point rouge qui signale la présence d'erreurs, ca vous donnera la liste des fichiers à exclure du chemin considéré par Intellij pour regrouper les classes considérées. Pour exclure du path, il faut faire un clic droit sur le dossier et sélectionner "markDirectory as" (qui se trouve presque tout en bas du long menu déroulement) puis "Excluded". La manip inverse se fait en selectionnant "source Root" au lieu de "Excluded". Ce sont des manips que on peut être amenés à faire de temps en temps, c'est le prix a payer pour avoir tout vos TPs pré-installés en même temps.

Vous n'arrivez pas à trouver eTDTPjava depuis intellij Pour des raisons de configuraton du 336 cela peut arriver. Sachez que vous pouvez ouvrir le projet eTDTPjava en faisant glisser le dossier eTDTPjava sur Intelij. La plus part qui avait ce problème on réussi a ouvrir comme cela, (pas tous) cependant je ne suis pas sur que cela part sur une bonne base. Mon expéreince est qu'il est mieux de repartir a zero et ...de suivre religieusement les instructions du début (je depanne les étudiants de cette facon toute bête)

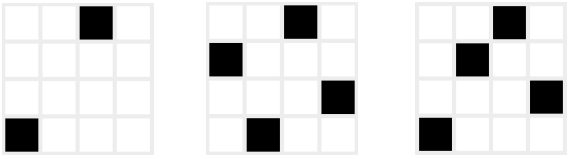
Problème de quota Parfois, votre login est bloqué au 336 parceque vous avez dépasser votre quota, par exemple

un crash de chrome peut générer des gros fichiers qui vous mettent dans le rouge sans que vous compreniez pourquoi. Mounir, ingénieur au SIF nous livre ici toutes les ficelles pour nettoyer votre compte des gros fichiers qui l'encombrent. Merci Mounir.

- la commande quota affiche la taille en octet que vous occuper sur le disque et ce que vous devriez occuper ; si vous avez un pb de quota, vous allez tout de suite le voir. pour Mounir : Occupation 7.6 Go, Limite soft 20 Go, et limite hard 29 Go
- Si la commande quota ne fournit pas une info très claire : Pour connaitre l'espace occupé dans le home se placer dans son home puis taper `du . -sh`
- Il y a un secteur qui favorise les gros fichiers, Lorsque il y a beaucoup de petit fichiers dans ce secteur, ils sont comptabilisés comme étant de gros fichiers d'où dépassement de quota. la commande `find Documents -type f | wc -l` vous dira combien de fichiers vous avez. Si il y en a des centaines, vous devrez supprimmer les fichiers inutiles.
- La commande suivante liste les sous dossier avec leur disk usage, si y a un dossier qui contient des gros truc on va dedans avec cd directory, et on recommence `du -shc ./* | sort -h`
- Attention aux fichiers cachés qui ne sont pas pris en compte avec la commande précédente notamment `.local` (`.local/share/Trash/` qui représente le contenu de la poubelle) et `.cache` (`.cache/mozilla/firefox` pour le cache firefox `du -shc .cache/* | sort -h` `du -shc .local/* | sort -h`
- Aprés qq descentes on trouve finalement un dossier avec plein de gros fichiers, alors on les liste avec leur taille et on les vire avec rm fichier la commande suivante cherche dans le reperoire courant tous les fichier dont la taille est supérieure à 200Mo `find . -type f -size` 200M -exec ls -lh
  Pour les supprimer ensuite (attention c'est définitif!) `find . -type f -size` 200M -exec fm -f
- on peut aussi directement lister récursivement la liste de tout les fichier avec leur taille, mais il y en a beaucoup c'est long a lire `ls -laShR`
- on peut chercher spécifiquement la liste tout les fichier suffixe dmp, généré lors de crash, qu'il faut de toute facon virer un jour ou l'autre `find . -type f -name "*.dmp" -exec ls -lh {} \;`
  Pour les supprimer ensuite (attention c'est définitif!) `find . -type f -name "*.dmp" -exec fm -f {} \;`

1.2 Le problème des N reines.

Il s'énonce ainsi : peut-on placer N reines sur un échiquier de taille N, de sorte qu'aucune reines ne soit menacées par une autre ? Autrement dit, nous disposons d'un tableau à deux dimensions carré de côté N, et nous voulons placer N points dans ce tableau de sorte qu'il n'y ait jamais deux points sur une même ligne, une même colonne, ou une même diagonale.



Conf.valide Solution Conf. invalide

Dans ce TP, nous allons créer un plateau de jeu graphique qui proposera à un joueur de résoudre cette énigme. Le joueur pourra effectuer les deux actions suivantes sur le plateau :

- Cliquer sur une case libre pour y placer une reine.
- Cliquer sur une case occupée pour retirer la reine qui s'y trouve.

On ajoutera également au jeu deux boutons :

- Un bouton de validation, qui indique si la configuration actuelle du plateau est acceptable (c'est-à-dire qu'aucune des reines présentes n'en menace une autre, indépendamment du nombre de reines présentes).
- Un bouton de demande d'indice. Si la configuration actuelle peut être complétée en une solution, alors l'interface propose au joueur une case sur laquelle jouer. Sinon elle indique que le joueur est bloqué.

1.3 Un peu de swing

Dans tout ce TP, on utilisera la mini-bibliothèque `IG` vue en cours, avec les classes ZoneClicquable, Grille, Fenetre et Texte. Cette ig permet d'utiliser la bibliothéque graphique Swing sans trop souffrir, On complètera le fichier `NReines.java`, du package `nReineSquelette.`

Question 1 Mise en route. dans le module TP1. Pour exécuter le squelette NReines.java, la premiére fois, vous faites un click droit dessus puis RUN. Les fois d'aprés, ca va s'ajouter dans vos configuration de run (flèche verte) en haut vers la droite, et vous n'aurez qu'a cliquer sur cette fléche verte. Pour faire apparaitre Nreines.java dans votre éditeur, vous double-cliquez énergiquement dessus, ou vous le glisser déposer.

Question 2 A présent, le programme se lance, mais il se contente d'afficher juste les boutons indices et validation. Que manque t'il dans le constructeur du plateau pour voir s'afficher l'échiquier ?

Question 3 Compléter le constructeur du plateau pour voir s'afficher l'échiquier. Ce constructeur a une taille `taille` que l'on pourrait passer en paramètre lors du démarrage de l'application.

Question 4 Par quel miracle les cases viennent t'elle se mettrent bien sagement dans un tableau de 8x8 alors que les bouton valider et indice se trouve à droite de cette grille 8x8 ? Question 5 A présent, on souhaite savoir pour chaque case s'il se trouve une dame. Combien y a t'il d'état pour une cases ? comment représenter cet état ? Comment accéder a cet état ? Programmez tout ca. Question 6 Un simple click gauche sur sur une case du tableau place une dame s'il n'y en avait pas, ou l'enléve si il y en avait une. Cela fait passer la case du du blanc au noir ou inversement. Compléter la méthode `clicGauche` de la classe `Case`, pour obtenir ce comportement. Pour colorer une case en noir, on pourra utiliser l'appel de méthod `setBackground(Color.BLACK);` . Bien sur il faut

tester en cliquant sur les cases, que elles se colorent bien en noir puis à nouveau en blanc si on re-clique.

1.4 Validation d'une configuration

Mots-clés : tableaux à deux dimensions, `if`, `for`, `for each`.

L'objectif de cette partie est de programmer le comportement du bouton de validation. Pour chaque méthode écrite dans cette partie il vous est demandé de créer des tests, que vous inclurez dans la méthode `main`. N'hésitez pas à créer des tableaux à la main, et à faire appel à des méthodes comme `System.out.println( ... )` pour suivre les résultats de vos tests.

Question 7 Pour vérifier que les reines sont bien placée, certaines méthodes de cette partie devront parcourir les cases de l'échiquier. Dans quelle classe faudra-t-il les définir ?

Question 8 Que font les méthodes compteDiagonales, compteAntidiagonale, verifieDiagonale, verifieAntidiagonale ?

Question 9 Ecrire une méthode qui renvoie vrai si toutes les diagonales et toutes les antidiagonales sont correctes.

Question 10 A présent, completer le bouton validation, afin que celui ci devienne rouge, si une diagonale, ou une antidiagonale contient deux reines, et reste vert sinon. Tester ensuite que cela fonctionne aussi bien pour le vert que pour le rouge.

Question 11 Écrire une méthode de signature
`private int compteLigne(Case[] l)` qui compte le nombre de reines présentes dans la ligne `l`, et en déduire une méthode de signature
`private boolean verifieLignes()` qui renvoie `true` si et seulement si aucune ligne de notre plateau contient au plus une reine. Enfin, procéder de même pour les colonnes

Question 12 Compléter la méthode `public boolean verifieConfiguration()` pour vérifier aussi les lignes et les colonnes, et Tester ensuite que cela fonctionne aussi bien pour le vert que pour le rouge.

1.5 Pour les "mordus" : Exploration des solutions et proposition d'un indice

L'article `http://sunnyday.mit.edu/16.355/wirth-refinement.html` replace le problème des Nreines dans le contexte de la technique de génie logiciel " conception par rafinement successif". Ce problème illustre bien le principe général de backtracking, pour explorer une arborescence de solutions possibles. Les étudiants qui arrivent la dessus avec de l'avance, trouveront probablement un certain plaisir à se mettre quelque chose de sérieux sous la dent. Tout étudiants qui réussi a soulever excalibur aura droit a sa photo au mur en noir et blanc avec un laius explicatif. L'enseignant qui n'aura plus que 20mm pourra simplement expliquer le corrigé.

L'objectif de cette partie est de programmer le comportement du bouton de demande d'indice. Comme à la partie précédente, chaque méthode écrite devra être méthodiquement testée.

On va pour cela utiliser une méthode verifieResolubilité qui a pour objectif de tester toutes les manières possibles de compléter le plateau jusqu'à trouver une solution. Les trois principaux ingrédients sont, chaque étape, :

- Verifier la validité de la configuration actuelle
- choisir une case, y placer une reine, et rappeler la méthode *récursivement* avec cette nouvelle configuration de départ.
- Si on ne trouve pas de solution, revenir au dernier choix et essayer une autre case (*backtrack*).

Question 13 Comment choisir la prochaine case a tester ?

Question 14 Ecrire une fonction qui renvoie l'indice de la première ligne non occupée, et tester la, en l'appelant systématiquement lors d'un click sur une cases.

Question 15 Quand est ce que vérifieRésolubilité renvoie Vrai ? et quand renvoie t'il faux ?

Question 17 Lors de l'appel récursif à vérifieRésolubilité, va t'on générer une copie entiére du plateau, en le passant par valeur ?

Question 18 : Si on ne passe pas le plateau par valeur, qu'est ce que cela implique ?

Question 19 : Ecrire la méthode vérifieRésolubilité, et tester la, sur un petit échiquier 4x4 pour lequel il n'y a pas de solution.

Question 20 compléter la méthode `void clicGauche()` de la classe `Indice`. On veut que le bouton s'affiche en vert si la configuration peut être complétée en une solution, et en rouge sinon. Tester sur un plateau 4x4 puis 6x6.

Question 21 A présent, ou souhaite aussi, lorsque une solution est possible, récupérer les coordonnées de la prochaine case a remplir, dans des variables indiceLigne, indiceCol. Lequel des appels a verifieResolubilité va calculer ces indices

Question 22 Un solution propre consisterai a ce que vérifieRésolubilité renvoie ces deux indices, en plus du boolean. Mais ca serait lourd a mettre en place. Comment faire plus simple ?

Question 23 Completer la méthode verifieRésolubilité afin qu'elle calcule ces indices.

Question 24 Compléter le code du bouton indice pour que dans le cas où une solution est trouvée, le jeu affiche en bleu une case dans laquelle il serait judicieux de placer une reine.

2 TP2 itérateurs : Réaliser une interface générique de deux façons différente

Dans ce TP, nous allons réaliser deux classes basiques de conteneurs inspirés de l'interface List de la bibliothèque standard. Puis nous allons itérer dessus via des "itérateurs" ! Dans le jargon de Java une liste désigne une collection ordonnée, ou *séquence*, d'éléments. La caractéristique principale d'une telle collection est qu'on peut ajouter des éléments, puis ensuite, accéder à un élément en fonction de sa position, donnée par un indice entier.

Question 1 : Executer le programme que fait il ?

L'interface List contient enormément de méthodes, allez le consultes sur internet pour vous faire une idée. Nous allons

nous contenter d'implémenter l'interface suivante, "List2" qui en est un sous-ensemble :

```
interface List2<T> {
    /** Renvoie l'élément d'indice [i]. */
    T get(int i);
    /** Ajoute l'élément [elt] à
    la fin de la liste. */
    void add(T elt);
    /**  Renvoi le nombre
    *  d'élements de la liste.*/
  int size();
}
```

2.1 Implémentation par un tableau

Des listes peuvent être représentées en utilisant les tableaux primitifs de Java. On obtient alors des listes dont la capacité est limitée par la taille du tableau utilisé. On vous donne un squelette de la classe `FixedCapacityList` avec les attributs et le constructeur, ainsi que la méthode size. Il s'agit de la complèter

Question 2 : Quel type de List2 implémente cette classe ?

Question 3 : Mais on veut stoquer des chaines de caractéres, cela peut il se ranger dans un conteneur d'objets ?

Question 4 : Que doit faire la methode get(i) de fixedCapacityList ?

Question 5 : Ecrire la méthode get de fixedCapacityList

Question 6 : Que doit faire la methode add(s) de fixedCapacityList ?

Question 7 : Ecrire la méthode add(o) de fixedCapacityList

Question-Test Votre classe FixedCapacity est prête, bravo ! Que proposez vous pour la tester ? Programmer le et . . .executer !

2.2 Implémentation par une liste chaînée

Pour implémenter l'interface `List2`, une alternative aux tableaux utilisés dans `FixedCapacityList` consiste à définir une structure de liste chaînée : pour une liste dont les éléments sont de type `T`, on utilise plusieurs blocs reliés entre eux et contenant chacun un élément.

Concrètement, on vous donne une classe `Block<T>` ayant :

- un attribut `contents` de type `T` pour l'élément contenu,
- un attribut `nextBlock` de type `Block<T>` désignant le bloc suivant, ou valant `null` s'il n'y a pas de bloc suivant.

La classe `MyLinkedList<T>` contient deux attributs `firstBlock` et `lastBlock`, tous deux de type `Block<T>`, désignant respectivement les premier et dernier blocs de la liste. Attention, il est important déclarer FirstBlock et LastBlock comme des `Block<T>`, sinon la variable de type T de ces block ne désignera pas le même type que le T de `MyLinkedList<T>` . Lors de la création d'une liste, les champs `firstBlock` et `lastBlock` sont initialisés à `null`

Question 8 Lorsqu'on veut ajouter un élement, la méthode `add` doit à présent ajouter un bloc après le dernier bloc (et mettre à jour le champ `nextBlock` du dernier bloc).quels sont les deux cas à considèrer ?

Question 9 Que faut il faire dans tous les cas

Question 10 Que faut il faire de spécifique dans chaque cas ?

Question 10 bis Les listes ne sont pas facile à bien comprendre, car il y a des pointeurs partout ; c'est seulement en ayant en tête une idée claire du diagramme d'objets que l'on ne vas pas se tromper. Ce diagramme contient un rectangle pour chaque objets, et dans le rectangle vous commencez par écrire le nom de la classe que l'objet instancie, puis les attributs de l'objet. Lorsque cet attribut est un pointeur vers un autre objet, vous tracez la flèche vers l'objet pointé. Ecrivez sur papier les trois diagramme d'objets suivants : 1- une liste de trois élements, 2- la liste mise a jour lorsque on rajoute un nouveau block b qui sera pointé par l'ancien dernier dernier block de la liste 3- la liste complétement mise à jour, avec son champs lastBlock qui pointe vers b. Vous indiquerez également les deux instructions qui font passer de 1- à 2- puis de -2 a 3-. Question 11 Programmer la méthode add.

A présent on veut programmer la méthode get. la méthode `get(int i)` doit renvoyer le contenu du premier bloc si `i` vaut `0`, le contenu du bloc suivant si `i` vaut `1`, de celui d'après si `i` vaut `2`, etc.

Question 12 Commencons par bien comprendre comment la liste est représentée en mémoire. Dessiner l'état de la mémoire correspondant à un liste de 3 élements "a", "b","c".

Question 13 Comment peut on accéder au troisième élement "c" ?

Question 14 Plus généralement comment faudra t-il faire pour récupérer l'élément d'indice i ?

Question 15 vous êtes mûr, programmez cette fonction get

Question-Test Votre classe MyLinkedList est prête, bravo ! Que proposez vous pour la tester ? Programmer le et . . .executer !

Question 16. Quel est le gros avantage de MyLinkedList ?

Question 17 Quel est le méchanisme qui donne cette propriété ?

Question 18 Comment appelle t-on une telle structure de donnée ?

2.3 Itérateurs

En Java, l'itération sur les éléments d'une collection se fait via la construction d'un objet itérateur qui est chargé de fournir un à un les éléments de la collection.

L'itérateur d'une collection d'éléments de type `T` a le type `Iterator<T>`, défini par l'interface suivante (importée avec la commande `import java.util.Iterator;`) :

```
interface Iterator<T> {
    /** Renvoie vrai s'il existe
     un prochain élément. */
    boolean hasNext();
    /** Donne le prochain élément
    et prépare le passage au suivant. */
    T next();
}
```

Chaque appel à la méthode `next()` fait avancer l'itération en renvoyant l'élément suivant.

On manifeste qu'il est possible d'itérer sur une collection en déclarant que cette collection implémente l'interface `Iterable<T>`, qui demande une méthode produisant un itérateur du type correspondant (importée avec la commande `import java.lang.Iterable;`).

```
interface Iterable<T> {
    Iterator<T> iterator();
}
```

Nous vous donnons la classe générique AscendingIterator permettant d'itérer sur nos List2 dans l'ordre croissant des indices. Nous vous donnons aussi une méthode printIterator qui utilise l'itérateur pour affichier une List2.a pluspart du temps l

Question 19 que faut il passer à printIterator pour afficher une List2 l

Question 20 utilisez printIterator pour afficher une FixedCapacityList

Question 21 On veut programmer un autre printIterator2 qui afficher deux fois chaque élément, comment s'y prendre ?

Question 22 Programmer printIterator2 et tester le.

Question 23 On veut programmer un autre printIterator3 qui n'afficher qu'un élément sur deux, comment s'y prendre ?

Question 24 Programmer printIterator3 et tester le.

A présent, on souhaite implémenter une classe génériqueDescendingIterator<T> implémentant l'interface `Iterator<T>` et représentant une itération sur un objet `List2<T>`, qui fournit les éléments dans l'ordre du dernier au premier.

Question 25 qu'est ce qui change ?

Question 26 Programmer la classe `DescendingIterator<T>`.

Question-test Tester DescendingIterator, comment faire ?

2.4 Iterable permet les boucles foreach

Question foreach : Foreach est une syntaxe qui exploite l'interface iterable pour simplifier l'écriture des boucles. C'est à elle seule, une justification pour utiliser Iterable. Consulter google pour voir cette syntaxe, et modifier votre `main()` pour l'utiliser. Attention, il vous faudra faire deux choses :

1. Commenter la déclaration des interface Iterator et Iterable, car il faut que ce soit les interfaces proposées par le système qui soient implémentée, même si elles ont le même nom, elle sont considérées distinctes par le compilo.
2. Ajouter les deux import :
   ```
   import java.lang.Iterable;
    import java.util.Iterator;
   ```

2.5 L'aspect performance

Question 27 Dans le cas de FixedCapacityList combien coute printIterator avec un itérateur ascendant si la taille est un nombre entier n ?

Question 28 Et pour MyLinkedList (même question) ?

Question 29 Comment peut on obtenir une complexité linéaire $O(n)$ i.e itérer sur une linkedList de taille n avec seulement de l'ordre de n opérations ?

Question 30 Programmer et tester cette solution.

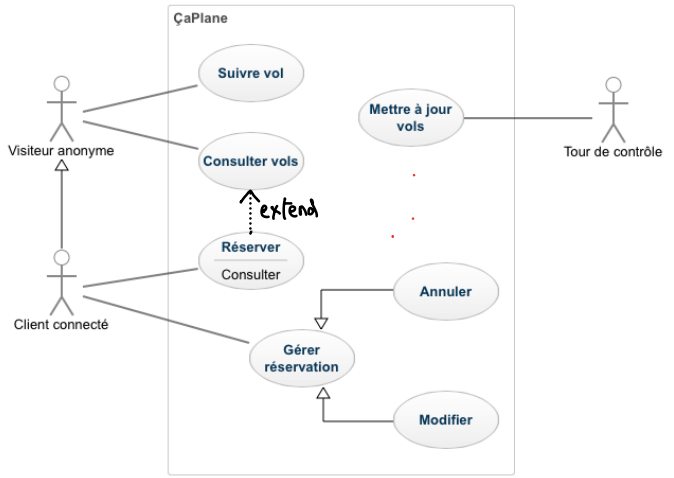
Question-Test Comment mettre en evidence que votre nouvelle solution va beaucoup plus vite ?

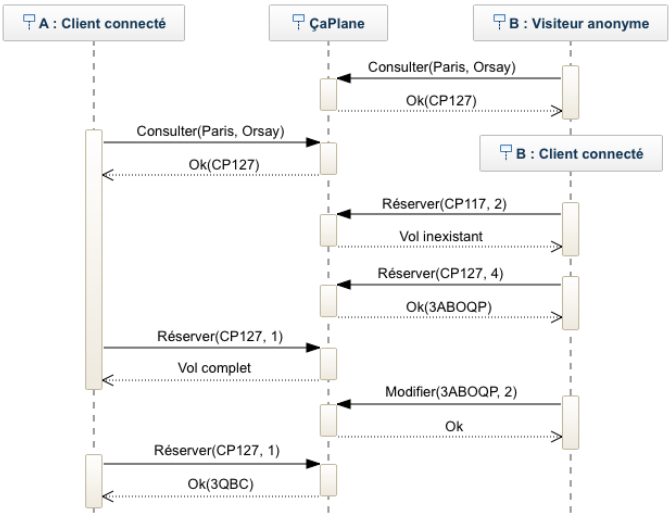
Question Philosophie : Quelle conclusion ?

3 TD1 : cas d'utilisation, scenario.

3.1 ÇaPlane pour moi.

Les diagrammes suivants donnent des éléments sur le système de réservation et de suivi des vols de la compagnie aérienne ÇaPlane.





Question 1 Comment s'appellent- t'il ?

Question 2 Identifier les différents acteurs extérieurs susceptibles d'interagir avec le système. Distinguer les acteurs humains des acteurs non humains, et identifier les éventuels cas où une même personne peut endosser plusieurs rôles.

Question 3 Que signigie la fleche de client connecté vers visiteur anonyme ?

question 4 pourquoi on a écrit "extends" sur la fléche qui relie consulter a reserver ?

Question 5 que signifie les fléches de annuler et modifier vers gérer ?

Question 6 pourquoi certaines fléches sont en pointillés dans le scenario ?

Question 7 Écrire des cas d'utilisation détaillés pour la recherche d'un vol et pour la réservation d'un vol, en utilisant au mieux les informations présentées dans le diagramme de cas d'utilisation et dans le scénario. Décrire tout les cas alternatifs.

3.2 Système de gestion de locaux

Ce cas d'étude concerne la gestion de locaux municipaux. Une commune souhaite informatiser la gestion de ses salles de réunion et de spectacle, et mettre en place un système de réservation par internet.

Le système doit permettre à un usager connecté via un compte personnel d'enregistrer de nouvelles réservations ou d'annuler des réservations en cours. Pour réserver une salle, l'usager doit fournir le nom de la salle ainsi qu'une date et une plage horaire. Si la salle est disponible sur le créneau demandé, le système indique le tarif de location et l'état des équipements de la salle, après quoi l'usager peut confirmer la réservation. Il reçoit en retour un numéro de réservation, qui identifie la réservation pour les autres opérations, notamment l'annulation, ainsi qu'un courrier électronique de confirmation. Lors de la réservation, l'usager a accès au calendrier des disponibilités de la salle qu'il souhaite réserver, ainsi qu'à la liste des équipements disponibles pour cette salle.

Un gestionnaire peut éditer les factures et enregistrer les paiements. En cas de retard de paiment, le système envoie chaque semaine un rappel par courrier électronique à l'usager retardataire. Après trois relances, le système envoie un message au gestionnaire qui peut envoyer un courrier recommandé à l'usager ou procéder à un recouvrement. Un usager peut également enregistrer une réservation directement auprès du gestionnaire, qui utilise pour ceci un compte générique.

Les agents de la commune chargés d'entrenir et d'apprêter les salles peuvent se connecter au système via un compte personnel pour consulter les calendriers de réservation et mettre à jour l'état des salles (disponibilité des différents équipements, entretien).

Toute personne souhaitant obtenir un compte pour se connecter au système doit en faire la demande auprès du gestionnaire et lui fournir une adresse électronique. Cette personne recevra par courrier électronique un mot de passe provisoire, qu'elle devra modifier pour finaliser la création du compte. L'envoi des courriers électroniques est fait en utilisant le serveur de courrier que la commune possède par ailleurs.

Question 8 Recenser les différent cas d'utilisation auquel l'ennoncé fait allusion

Question 9 L'un des cas étends un autre cas, lequel ?

Question 10 Y a t'il des relations de généralisation entre les cas ?

Question 11 Quel sont les différent acteurs (roles) dans ce système, préciser les relations de généralisation.

Question 12 Finir le diagramme de cas d'utilisation en reliant les cas aux acteurs.

Question 13 Préciser deux interaction dans cette enoncé qui sont hors du système

Question 14 Donner un scénario combinant au moins les éléments suivants (dans un ordre quelconque donnant un résultat cohérent) :

- un utilisateur crée un compte, fait une première demande de réservation qui n'aboutit pas et une deuxième qui aboutit,
- un agent met à jour l'état d'une salle,
- un gestionnaire édite la facture de la réservation.

4 TP3 Graphe, GL diagrammes UML de classes, et d'objets.

Il est important dans ce TP de faire les diagrammes de classes et d'objets demandés, sur papier. Il ne s'agit pas de juste programmer en java, il s'agit de d'abord faire ces diagrammes et ensuite les utiliser pour programmer. On veut construire une bibliothèque Java permettant de manipuler des graphes non orientés. Les principaux services que doit fournir la bibliothèque sont :

- Création d'un graphe : la classe `Graphe` doit permettre de créer un graphe vide, ajouter des sommets et des arêtes.
- Affichage d'une description d'un graphe : on veut pouvoir afficher dans un format textuel l'ensemble des sommets, chacun accompagné par l'ensemble des arêtes qui lui sont incidentes (la description correspond donc à un dictionnaire d'adjacence).
- Coloration des sommets de manière à ce que deux sommets adjacents aient des couleurs différentes. on ne demande pas une coloration optimale mais correcte. Une couleur sera simplement un entier positif, on veut minimiser le nombre de couleurs, donc choisir autant que possible des entiers petits. Ce problème est important ; il est utilisé en compilation pour minimiser le nombre de registres utilisés pour éxecuter une boucle.
- Distance entre deux sommets : étant donnés deux sommets on veut pouvoir connaître la longueur du chemin le plus court de l'un à l'autre (la longueur d'un chemin étant le nombre des arêtes).
- Routage d'un sommet à un autre : étant donnés deux sommets on veut pouvoir obtenir un chemin le plus court de l'un à l'autre.

L'analyse a inclus en outre les contraintes non fonctionnelles suivantes dans le cahier des charges :

- Contrainte mémoire faible : Chaque sommet peut stocker une quantité d'information proportionnelle à la taille du graphe.
- Contrainte de temps forte sur les requétes : La réponse aux requêtes de distance et de couleur doit se faire en temps constant. La réponse aux requêtes de routage doit se faire en temps proportionnel au nombre d'arêtes dans le chemin renvoyé.
- Contrainte de temps faible au moment de la construc-

tion : des calculs plus coûteux sont admissibles lors de la création du graphe.

4.1 Comprehension de l'ennoncé.

Question 0 : A quelle famille de contraintes appartiennent les contraintes de temps fort sur les requétes, de mémoire faibles ? Question 1 Comment satisfaire les contraintes de temps fortes aux requétes en exploitant les contraintes de temps faible a la construction ?

Question 2 : Comment les contraintes faible de mémoires vont elles, elles aussi, pouvoir être exploitée ?

4.2 Modélisation UML.

Question 3 : Il s'agit de faire les diagrammes de classe. Dessiner juste les rectangles pour les classes graphes, noeuds et arrêtes avec attributs et les operations qui permettrons de contruire un graphe et l'afficher. Important, passez du temps à essayer de répondre par vous même à cette question qui est du TD et pas du TP ! Dessinez/comprendre des diagramme UML est une partie importante de l'examen.

Question 4 A présent, ajouter les associations qui existent et leur multiplicité.

Question 5 Préciser les roles la ou cela permet de rendre plus lisible

Question 6 Préciser les agrégation/composition.

Question 7 Depuis un sommet, il faut pouvoir accéder aux sommets voisins, proposer une méthode qui le permet.

4.3 Prise en main squelette

Question 8 A présent, regarder votre squelette, et préciser quel sont les attributs utilisés pour traduire les différentes associations du diagramme UML, et quels sont leur type.

Question 9 Et comment le type associé a "*" est il implémenté ?

Question 10 Ce mapping association vers attributs pourrait il être généré automatiquement ?

Question 11 Executer le squelette qu'affiche t'il ?

Question 12 Qu'est ce qu'on voudrait aussi voir s'afficher, pour vérifier que on a bien le bon graphe ?

Question 13 Comment afficher cela ?

Question 14 Programmer-le

Question 15 Dessiner le diagramme d'objets correspondant à ce graphe, que java va instancier en mémoire. Important, ne brulez pas cette étape, dessiner des diagrammes d'objets vous sera demandé aux examens.

4.4 Coloration

On va donc colorer les sommets au fur et a mesure que l'on construit le graphe.

Question 16 on va coder une couleur comment ?

Question 17 Lorsqu'on crée un sommet et qu'on l'ajoute au graphe, comment peut on initialiser la couleur ? pourquoi ?

Question 18 Quand est ce que va falloir recalculer la couleur d'un sommet ?

Question 19 Comment recalculer au plus simple ?

Question 20 Programmer en décomposant si possible votre code en morceaux indépendants et cohérents.

Question 21 que proposez vous de faire maintenant ?

4.5 Distance

Question 22 Completer le diagramme UML pour pouvoir représenter les distances d'un sommet aux autres

Question 23 Quel sera le type de l'attribut distance ?

Question 24 Quel est le type en java qui implémente cela ?

Question 25 comment initialiser distances ?

Question 26 Lorsqu'on construit le graphe, quel est l'ajout qui nécessite le recalcule des distances ?

Question 27 Quelles sont potentiellement les distances modifiées ?

Question 28 Y a deux cas possibles, pour la mise a jour de distance, préciser les quels.

Question 29 comment récupérer les noeuds accessibles depuis un sommet ?

Question 30 comme on y accéde plusieurs fois, ca vaut le coup d'en faire une méthode, faite le.

Question 31 Programmer la mise a jour des distances.

Question 32 : Que faites vous a présent ?

4.6 Pour les "mordus" : Routage

Question 33 : Il faut a présent représenter des chemins. Rajouter le classe chemin dans le diagramme UML des classes, que faut il mettre dedans ?

Question 34 Comment trouver le routage optimal à partir des distances ?

Question 35 Pour quelle graphes "routage" respecte t'il le cahier des charges pour routage ? Donner des exemples positif, négatifs.

5 TP4 Circuits : Classes abstraites, héritage, redéfinition méthodes.

Dans ce TP, nous allons créer des circuits effectuants des opérations arithmétiques. Ils ont une particularité : ils ont une seule entrée. Ces circuits sont constitués de nœuds reliés entre eux, chaque nœud effectuant une opération élémentaire. Le squelette contient quelques définitions de base et une méthode `main`, qui elle-même contient une série d'exemples avec lesquels vous pourrez tester votre code. Pour l'instant, ces exemples sont placés en commentaire pour éviter les erreurs de compilation. Vous décommenterez chaque exemple dès que les opérations dont il a besoin auront été définies.

5.1 Construction simple de circuits.

Un objet de la classe circuit possède trois attributs privés :

- `int entree` : l'entrée unique du circuit,
- `ArrayList<Noeud> noeuds` : l'ensemble des nœuds du circuit, et
- `Noeud sortie` : le dernier nœud du circuit, qui donne le résultat final,

ainsi que les méthodes publiques suivantes

— `int litEntree()` : renvoie la valeur de l'entrée,
— `int calcule(int e)` : renvoie la valeur calculée par le nœud `sortie` avec une entrée valant `e`,
— des méthodes `Noeud cree*` pour chaque classe de nœud pouvant être ajoutée au circuit,

et un constructeur `Circuit()`.

Classes Abstraites Tous les nœuds que nous allons manipuler hériteront de la classe abstraite suivante :

```
abstract class Noeud {
  abstract public int valeur();
}
```

Un appel à la méthode `valeur()` renvoie le résultat calculé par ce nœud du circuit, et ce résultat sera un entier. Cette méthode doit donc être définie pour chaque classe concréte héritant de noeuds. Dans le cas ou la valeur est l'Entrée nous vous donnons la classe correspondante :

```
class Entree extends Noeud {
    private Circuit circuit;
    public Entree(Circuit c) {
this.circuit = c;
    }
    public int valeur() {
return this.circuit.litEntree();
    }
  }
```

Les nœuds pourront de plus posséder un certain nombre d'attributs privé nécessaire à leur calcul. Par exemple pour Entree, nous avons besoin d'un attribut pour pointer sur le circuit, de facon a pouvoir récupérer l'entrée via la méthode litEntree.

Losqu'on veut calculer la sortie du circuit, on passe l'entrée à la méthode calcule, qui le stoque dans l'attribut entrée du circuit, puis déclenche le calcul de la valeur sur le noeud sortie. Dans l'exemple très élementaire que nous vous donnont, comme nous ne disposont que de noeuds pouvant renvoyer l'entrée, nous ne pouvons pour le moment rien faire d'autre que de calculer la fonction identité $x \mapsto x$, i.e. de renvoyer l'entrée. Pour commencer a faire des circuits arithmetiques, on souhaite creer une classe `Addition` qui permet d'additionner les valeurs calculées par deux autre neouds.

Question 1 de quels attributs aura t'on besoin ?

Question2 Comment va t'on initialiser ces attributs ?

Question 3 Programmer la classe Addition.

Question 4 Comment faire pour que lors de la création d'un noeud addition, celui ci soit systematiquement ajouté dans les noeuds du circuit dont il fait partie ?

Question 5 Testez Addition.

Question 6 On souhaite utiliser des noeuds constantes pour pouvoir faire des calculs plus intéressants. De quels attributs aura t'on besoin ?

Question 7 Comment va t'on initialiser cet attribut ?

Question 8 Programmer la classe Constante.

Question 4 Comment faire pour que lors de la création d'un neoud constante, celui ci soit systematiquement ajouté dans les noeuds du circuit dont il fait partie ?

Question 5 Testez Addition avec constante

Question 6 Comment allez vous implémenter la soustraction et la multiplication ?

Question 7 Programmez la soustraction et la multiplication et testez.

5.2 Héritage : Hierarchiser pour simplifier.

Nous allons maintenant utiliser un hériage plus sophistiqué que simplement hériter d'une classe abstraite ancêtre, commune à tout le monde. Cela permettra de factoriser du code. Question 8 : Dessiner le diagramme des classes neouds, addition, soustraction et multiplication.

Question 9 Les classes addition, soustraction et multiplication se ressemblent, comment peut on exploiter ces ressemblances pour simplifier le code ? dessiner le nouveau diagramme correspondant.

Question 10 Modifier le code pour qu'il corresponde a ce nouvel arrangement, comment tester vous ?

5.3 Redéfinition méthodes pour Afficher.

On souhaite afficher l'expression arithmétique associée à un circuit c, ou l'entrée est traduite par l'inconnue x. Par exemple le circuit c.creeAdition(c.creeEntree(), c.creeConstante(1)) s'affichera par $x + 1$.

Question 11 Rappel : on peut appliquer system.out.print sur un objet dés l'instant ou sa classe implémente la méthode `String toString()`, qui renvoie la chaine de caractères à imprimer. Il suffit donc de re-définir cette méthode toString de maniére approprié, pour chaque type de noeuds. Faites le pour les noeuds entrée et constante qui n'ont pas de source.

Question 12 Comme pour le calcul de valeur(), afficher le circuit aura juste à "printer" le noeud de sortie. Ceci est évident dans le cas trivial ou il n'y a que un seul noeud de sortie : entrée ou constante. Programmer la méthode "afficher" du circuit, tester sur le circuit identité.

Question 12bis Dans le cas non trivial ou il y aussi des noeuds arithmetiques, ceux-ci appeleront recursivement toString sur leur deux opérandes. Préciser comment. On suivra un des objectifs de ce TP qui consiste à utiliser convenablement l'héritage, les méthodes abstraites et les redéfinitions pour écrire un minimum de code.

Question 13 programmer le et tester le.

5.4 Des outils de diagnostic, redéfinition bis

On veut pouvoir analyser le nombre de portes arithmétiques d'un circuit et l'utilisation de ses nœuds.

Question 14 Definissez une méthode pour savoir si un noeuds est arithmetique. Indice : if faudra la redéfinir dans NoeudBinaire.

Question 15 Comment faire pour compter les porte arithmétiques ?

Question 16 Programmez le.

Question 17 Tester le comptage de noeuds arithmetiques en décommentant le prochaint test qui construit le circuit c_2 qui calcule $((x * x) * (x * x))$. Celui ci utilise 2 noeuds, vérifier le. Comment peut on faire trois multiplications avec

seulement deux noeuds. Y a til un bug ? Afin de comprendre ce qui se passe il vous faut dessiner le diagramme des objets représentant ce circuit, faites le sur papier, s'il vous plait.

Question 18 Comment compter non pas le nombre de noeuds, mais le nombre d'évaluations de noeuds arithmetiques ?

Question 19 Programmer ce nouveau comptage, tester le.

5.5 Mémoïsation, redéfinition ter.

On cherche à présent a minimiser le nombre de calcul pour évaluer un circuit dont les portes sont réutilisée, i.e. certaines portes ont plusieurs sorties.

Question 20 Pourrait t-on évaluer le circuit c_2 avec deux opérations au lieu de 3 ?

Mémoiser consiste pour chaque porte à stoquer le résultat d'une évaluation, de sorte que lors de l'évaluation suivante, on consulte le résultat précedement evalué au lieu de le réavaluer.

Question 21, nous alons nous limiter à mémoiser seulement les multiplications. Créer une classe `MultiplicationMemoisee` qui hérite de `Multiplication` et qui redéfinit sa méthode `valeur()` de sorte à ce que :

- le premier appel à la méthode `valeur()` d'un objet effectue le calcul comme un nœud `Multiplication` normal, mais enregistre en plus le résultat dans un nouvel attribut privé `mem`,
- tous les appels suivants se contentent de renvoyer la valeur enregistrée (qu'on qualifie aussi de *mémoïsée*) sans refaire le calcul. Ainsi, la méthode `nbOpEffectuees()` necomptera pas une nouvelle opération.

Question 22 Decommenter le test suivant pour vérifier que le circuit utilisant la mémoisation ne fait bien que deux évaluations au lieu de trois.

Question 23 Pour mieux mettre en évidence l'interêt de la mémoisation, construire un circuit "expRapide" qui calcule x^{2^n} en chainant n multiplication avec la méthode statique expRapide. Donner une version mémoisée de ce circuit, et tester le.

6 TP5, Formule 1 : Test avec Junit

Si vous avez le probléme que intelij ne trouve pas junit4, consulter le paragraphe "Utilisation des suggestions de Intellij" du tp1. Dans le jeu *FormuL*∞, vous guidez un bolide (classe `Bolide`) autour d'un circuit de course (classe `Circuit`), en lui indiquant à chaque seconde dans quelle direction accélerer.

Question 1 :Dans le dossier src, à la racine se trouve un fichier mc1.flinf, ouvrez le avec un éditeur de texte, que représente t'il ?

Dans le dossier src, à la racine, se trouve aussi l'éxecutable "FormuLInf.jar" lancer ou bien depuis une fenêtre de commande, avec la commande `java -jar FormuLInf.jar mc1.flinf`, ou bien depuis intelliji en faisant un click droit plus run sur "FormuLInf.jar", mais ca ne vas marcher tout de suite, car le programme a besoin du nom du circuit, mc1.flinf. Pour lui fournir ce nom, cliquer sur la petite fléche vers le bas juste a droite de la ou est écrit "FormuLInf.jar", selectionner "edit configuration", et écrivez "mc1.flinf" dans la case program argument. Attention, cela fonctionne seulement si le fichier mc1.flinf est dans le même répertoire que FormuLInf.jar. Passez quelque minutes à tester vous-même le jeu sur le circuit exemple et en apprendre plus sur son fonctionnement. Vous cliquez devant la voiture, pour la faire avancer vers la case cliquée.

Question 2 : Que se passe t-il si la voiture rencontre le mur ?

Petit rappel de physique. Une accélération est une modification de la vitesse, et une vitesse est donnée par un vecteur (classe `Vect`). Dans notre version nous considérons uniquement des positions et des vecteurs à coordonnées entières dans un espace à deux dimensions. Nous restreignons de plus l'intensité des accélérations, de sorte qu'une accélération (x, y) aura toujours une norme L∞ égale à 0 ou 1. Plus concrétement, cela signifie que les coordonnées x et y ne pourront valoir que 0, 1 ou -1.

Question 3 Qu'indique les cases vertes ?

Question 4 Si la voiture est lancée a fond dans une direction, c'est difficile de lui faire changer de direction, pourquoi ? :

Dans le dossier src, nous vous fournissons l'interface graphique IG (la même que pour Jdemine et Nreines) pour avoir des case cliquable, le package formule1 contient la classe principale `FormuLInf`, et les squelettes des trois classes `Vect`, `Bolide` et `Circuit` que vous devez compléter.

Nous ne dirons rien de plus sur les règles du jeu. En revanche, dans le dossier tests/formule1 nous vous fournissons un jeu de tests unitaires pour chacune des méthodes à compléter, qui vous montrent les comportements attendus et qui devront vous guider dans l'écriture du code.

Votre objectif : Vous aller expérimenter "la programmation guidée par les tests". Il vous faut déduire des différents tests, pour chaque méthode du squelette :

1. Sa spécification, les tests sont choisis pour la définir.
2. son code, qui va faire réussir les tests.

A la fin de ce processus, votre programme se comportera comme la version de référence que vous avez déjà exécutée.

Mise en route Intellij. Il faut que java puisse accesder aux fichiers hamcrest-core2.2-sources.jar junit-4.13.2-sources.jar, dans les librairies. Ces fichiers se trouvent à la racine. Normalement ca devrait fonctionner direct, mais il se peut que vous deviez maniper : un clic droit et selectionner "add as library". Normalement le dossier tests devrait apparaitre en vert, c'est la couleur des tests, sinon Faire un clic droit sur tests et selectionner "add as test source root" voire aussi un clic droit sur src et selectionner "add as source root",

Autre méthode possible en cas de problémes : Ouvrez par exemple vect.test, et cliquez sur le "junit" qui apparait en rouge dans la ligne au tout début "import org.junit.*;", une ampoule rouge va s'allumer. Cliquez sur cette ampoule, l'idea vous suggére alors (en bleu) de cliquer sur add Junit 4 to class Path, vous cliquez la dessus, puis OK sur le bandeau suivant qui apparait, et ca résoud le problème.

Gérer la correction et le squelette. Votre archive contient à la foit le squelette complété et le squelette à compléter c'est pratique. Si Intelij se met à crier que les classes sont définies deux fois. Il faut sélectionner celui des dossier src et test dont vous ne voulez pas, faire un clique droit, "mark directory as", "excluded".

6.1 VectTest.java

JUnit est un outil permettant d'écrire et d'exécuter des tests unitaires sur des programmes Java.

Il est disponible à l'adresse `http://www.junit.org/`. Il peut aussi s'utiliser avec Intelij. Un test en JUnit 4 est une méthode annotée par `@Test`. Les méthodes de test sont généralement regroupées en une classe dédiée aux tests. Ouvrez le fichier VectTest.java du repertoire tests. Si votre installation est correcte vous devez pouvoir éxecuter ce fichier, ce qui provoquera l'execution de tout les tests contenu dans la classe VectTest,

Question 5 Sur quoi porte les teste de VectTest ?

Question 6 Pourquoi sont il pour la pluspart marqué en rouge avec une mention "échec".

Question 7 Est-ce que les méthode de Vect, d'instantiation, add,sub, et egale de Vect sont compliquées ? que font elles ?

Question 8 Pourquoi vous a t'on mis cette quantité impressionantes de tests ?

Question 9 A quoi servent tout les attributs initialisés par la classe VectTest ?

Question 10 Comment faire passer les tests du rouge au vert ?

Question 11 Que pensez vous de l'utilité des tests sur normalize, et pourquoi sont ils tous devenus vert sans corrections supplémentaires ?

6.2 BolideTest.java

Question 12 : Que testent les méthodes void bolideZero(), void bolidePos() , void bolideNeg() ? Verdir !

Qu'est ce que les "Preambule" Le corps d'une méthode de test comporte en général trois parties :

- le *préambule*, qui permet de créer les objets et de les amener dans l'état nécessaire pour le test ;
- le *corps de test*, dans lequel la méthode à tester est appelée sur les objets créés ;
- l'*identification*, qui permet de délivrer le verdict du test (succès ou échec) en vérifiant un ensemble de propriétés (assertions) sur l'état des objets après le test.

Il est possible de grouper les tests ayant un préambule commun (c'est-à-dire devant être exécutés dans le même état) en une classe et de définir une méthode qui exécutera ce préambule avant chacun des tests de la classe. Cette méthode doit être annotée par `@Before`.

Question 12 bis A quoi sert le préambule de BolideTest, on aurait pu se passer de préambule ? pourquoi le préambule ?

Question 13 Que teste les 9 methodes suivantes dont le nom commence par calculeAcceleration ? En déduire une spécification, verdir.

Question 14 Que teste les 2 methodes suivantes dont le nom commence par AccelereDe ? En déduire une spécification, verdir.

Question 15 Que teste les 2 methodes suivantes dont le nom commence par stop ? En déduire une spécification, verdir.

Question 16 Idem pour les derniéres méthodes

6.3 CircuitTest

Question 17 Que déduisez vous des test Circuit simple ?

Question 18 Que déduisez vous des test "HorsLimite"

Question 19 Que déduisez vous des test "CarteTropGrande", "CarteTropPetite" ?

Question 19 bis Programmer le constructeur de Circuit.

Question 20 Que déduisez vous des test DeplaceBolide

Question 20 Programmer deplacementBolide.

Question 21 Finalement, que fait gereClick ? Programmez.

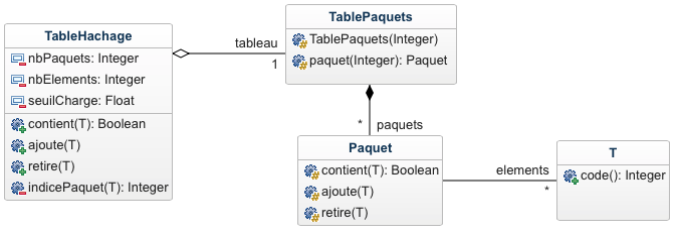
Question 22 Pour les "mordus" : colorier en vert le trajet juste effectué, comme ca se passe dans la démo que vous avez fait au début. C'est pas très compliqué, si vous le faites vous allez vous sentir super pro. :

7 TD2 Hashtable : pre/post condition, invariant et Sequence Diagram

Dans ce TD, nous nous intéressons à la conception d'une structure de table de hachage, et en particulier à la manière dont les fonctionnalités principales se décomposent en enchaînements d'opérations des différentes classes composant la structure. Le thème abordés et la spécification des opérations. Il s'agira de préciser les valeurs que peuvent prendre les paramétres d'une méthode (préconditions) et les valeurs que prennent les valeurs retournées (postconditions). Ce TD illustre aussi comment la notion diagrammes de séquence introduite dans les diagrammes de cas peut aussi être utilisée pour la conception.

7.1 Tables de hachage

Voici un diagramme de classes représentant les différentes entités entrant dans la conception de notre table de hachage. Les classes TablePaquet et Paquet sont des classes internes privées utilisable seulement depuis "l'intérieur" de la table de hashage.



Dans la classe `TablePaquets`, les `paquets` sont stockés dans un tableau auquel on peut accéder par des indices. Dans la classe `Paquet`, les `elements` sont stockés dans une liste chaînée. Les invariants associés à ces classes sont :

— l'attribut `nbPaquets` est égal au nombre de paquets associés au `tableau`,
— l'attribut `nbElements` est égal au total des nombres d'éléments présents dans chaque paquet,
— le `tableau` ne contient aucun paquet égal à `null`, et aucun paquet ne contient l'élément `null`,
— tout élément `elt` de la table est dans le paquet d'indice `elt.code() % nbPaquets`. Ici, la fonction "code" renvoie un entier, et doit être définie.

On considére une table qui code des entiers, et que l'opération `code` renvoie la multiplication par 3 du nombre à coder.

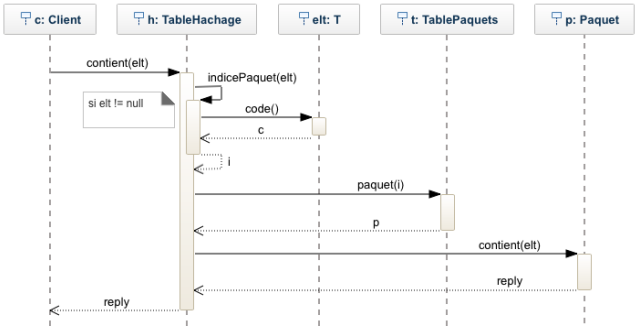
Question 1 : Dans quel paquet est stoqué -2

Considérons à présent une table de hachage à 8 paquets, contenant les entiers -2, -3, 5, 7, 11, 13 et 17,

Question 2 : Dessiner un diagramme d'objets représentant cette table. On fera en sorte de mettre en évidence que à l'intérieur d'un paquet, les elements sont stoqués dans une liste chainée.

7.2 Test d'appartenance

L'une des fonctionnalités de notre table de hachage est le test d'appartenance d'un élément à l'ensemble représenté par la table, réalisé par l'opération `TableHachage::contient(T elt)` qui déclenche l'enchaînement d'opérations suivant dans le cas où le paramètre `elt` n'est pas `null` : Le diagramme suivant utilise l'extension aux objets de la notion de scénarios entre agents. Cela a été abordée en cours.



Les spécifications des méthodes utilisées sont les suivantes :

— `TableHachage::contient(T elt)`
  — Précondition : néant
  — Résultat : renvoie `true` si `elt` est dans la table, et `false` sinon ou si `elt` est `null`
  — Postcondition : néant
— `TableHachage::indicePaquet(T elt)`
  — Précondition : `elt` n'est pas `null`
  — Résultat : renvoie l'indice du paquet susceptible de contenir `elt`
  — Postcondition : le résultat est compris entre `0` inclus et `nbPaquets` exclu
— `TablePaquets::paquet(int i)`
  — Précondition : `i` est compris entre `0` inclus et `nbPaquets` exclu
  — Résultat : renvoie le paquet d'indice `i`
  — Postcondition : le résultat n'est pas `null`
— `Paquet::contient(T elt)`
  — Précondition : néant
  — Résultat : renvoie `true` si `elt` est dans le paquet, et `false` sinon
  — Postcondition : néant
— `T::code()`
  — Précondition : néant
  — Résultat : renvoie le code de hachage de l'élément
  — Postcondition : néant

Question 3 : Décrivez la séquence des opérations effectuées. Détaillez en particulier les informations circulant d'un objet à l'autre par l'intermédiaire des paramètres et résultats des appels de méthodes.

Question 3 bis, que fait [Paquet : :contient] ?

7.3 Préconditions, Postconditions

Nous allons nous pencher dessus pour mieux comprendre leur raison d'être.

Question 4 : Pourquoi la méthode hashtable : :contient n'a pas de précondition précisant que elt ne doit pas être null ?

Question 4bis : Pourquoi la méthode Paquet : :contient n'a pas de précondition précisant que elt ne doit pas être null ? ?

Question 5 : Pourquoi la méthode T : :code n'a pas de précondition ?

La question suivante illustre une technique objet importante : Pour les méthodes privées, on cherche à poser des préconditions sur les paramétres qui simplifie les calculs en évitant des vérifications, car vu que les appels peuvent seulement venir de la classe elle même, (et non pas d'un client externe) on peut s'assurer que elle sont toujours vraie.

Question 6 : Pour chacunes des deux méthodes TableHachage : :indicePaquet, et TablePaquets : :paquet répondre a ces deux questions : 1- En quoi la précondition choisie permet-elle de simplifier le code ? 2- Ces précondition peuvent elle être fausses ?

Question 7 : Puisque les préconditions sont toujours vraies, qu'en déduit on ?

Question 8 :Pour simplifier plus encore, quelqu'un propose de donner comme précondition à l'opération `TableHachage::contient(T elt)` que `elt` ne soit pas `null`. Discuter.

7.4 Ajouter et retirer des éléments

On fixe les spécifications suivantes pour les opérations de la classe `Paquet` :

— `Paquet::ajoute(T elt)`
  — Précondition : `elt` n'est pas `null`
  — Résultat : néant
  — Postcondition : une occurrence de `elt` est ajoutée au paquet
— `Paquet::retire(T elt)`
  — Précondition : néant
  — Résultat : néant
  — Postcondition : une occurrence de `elt` est retirée du paquet s'il y en avait au moins une, aucun effet sinon

Pour la méthode `TableHachage::retire(T elt)` On veut donner à la spécification suivante :

— Précondition : néant
— Résultat : néant
— Postcondition : toutes les occurrences de `elt` sont retirées de la table et `nbElements` est décrémenté de 1 si l'élément était présent, aucun effet si `elt` est `null`

Question 9 Que faudrait-il faire pour réaliser une telle opération en ne supposant rien de plus que les invariants et spécifications actuelles ?

Question 10 Quel invariant sur la table de hachage permettrait de faire plus simple ?

Question 11 Proposer une spécification (postcondition) pour `TableHachage::ajoute(T elt)` qui assure le maintient de cet invariant. Donner un diagramme de séquence qui met en evidence cette nouvelle spécification.

7.5 Limiter la taille moyenne des paquets

Pour limiter la taille moyenne des paquets, on veut appliquer la stratégie habituelle suivante : lorsque le rapport du nombre d'éléments sur le nombre de paquets dépasse le seuil de charge (dont une valeur typique est 0.75), doubler le nombre de paquets et réinsérer chaque élément dans le paquet qui convient.

Question 12 Quelle opération modifier pour introduire ce nouveau comportement ? Donner des versions mises à jour de sa spécification et de son diagramme de séquence.

8 TD3 :Projet

Il y a un TD entier consacré au projet. Il consiste principalement a mettre en place le diagramme des classes du projet. C'est un travail réalisé avec tout le groupe, chaque groupe aura potentiellement un diagramme assez différent, suivant la compréhension plus au moins préciser qu'ils on du sujet, et que leur prof également a du sujet. On pourra commencer par répondre aux questions des étudiants sur la compréhension du projet, s'il y en a. Je met un certain nombre d'infos dans un howto générique. Je parle de ces info en cours donc y a pas trop besoin de passer dessus durant le TD projet.

8.1 Howto générique

Je met un certain nombre d'infos ici, pour que mes chargés de TP les lisent aussi, en plus du cours. Je parle de ces info en cours donc y a pas trop besoin de passer dessus durant le TD projet.

Calcul de la note Pour la session1, le projet compte pour 25% , partiel 25% et examen 50%. Pour la session2, il compte pour 33% et examen rattrapage 66%. Notez bien que si vous ne travaillez pas sur le projet, votre note de projet reste, et risque de vous plombez pour le passage. Comme le partiel ne compte plus, ce qu'on doit réviser pour le rattrapage inclus le programme du partiel.

8.1.1 Déroulement

Le projet démarre avec une semaine qui lui est dédiée : cours + TD, deux semaines aprés les partiels. Je prie le groupe dont le TD est décalé de une semaine, de ne pas se plaindre d'avoir moins de temps, en effet, il peut tout à fait commencer aprés le cours, avant le TD dedié au projet, car le tout début du projet porte sur la prise en main de swing, avec simplement l'affichage du plateau de jeu, tandisque le TD dedié, lui, a pour but de produire avec les étudiants présents, une ébauche du diagramme de classe global.

Vous avez a peu près 5 semaines jusqu'à la soutenance. Le cours est entiérement structuré pour maximiser ce laps de temps. Il n'est malgré tout pas énorme, et vous aurez d'autres projets avec d'autre matières, il est donc essentiel de démarrer immédiatement. Le projet est à réaliser en binome de deux étudiants d'un même groupe à peu près du même niveau [1]. Il est déconseillé de le faire tout seul, c'est moins drôle, et vous n'apprenez pas à travailler en groupe. Il n'est pas autorisé de faire le projet à trois, car sur trois il y en a souvent un qui ne fais rien. Vous saisirez vos binomes dans ce fichier partagé :

`https://docs.google.com/document/d/1a0a7xA-j2NIUAr1qwjue_VL1KtK9hsH01K8oEsOZqqQ/edit`

Celui-ci servira également pour vous informer des heures de passage, pendant les soutenances.

Votre projet doit être hébergé sur un dépôt git. Vous n'êtes pas autorisé à le faire sur github. On vous demande de le prendre chez nous sur le gitlab de l'université `https://gitlab.lisn.upsaclay.fr/users/sign_in` surlequel vous avez un compte.

8.1.2 Intruction pour le dépot git

Création du projet à faire par l'un des membres du binôme.

1. Allez sur l'interface web du gitlab de l'université : https ://gitlab.dsi.universite-paris-saclay.fr/ et connectez-vous (avec vos identifiants habituels de l'université).
2. Cliquez en haut à droite : new project, puis sélectionnez create blank project.
3. Donnez un nom à votre projet.
4. Assurez-vous que le niveau de visibilité private est sélectionné, puis confirmez la création.

Ajouter son binomes et son prof de TP, à réaliser dans l'interface web du projet.

1. Dans le menu, sélectionnez “project information”, puis “members”.
2. Cliquez sur invite member et intégrez le deuxième membre du binôme, avec le rôle maintainer.

1. La formation des binomes est supervisée par votre TPman, celui ci peut autoriser un regroupement intergroupe, pour éviter de faire le projet seul, ou parceque vous avez un bon ami dans un autre groupe

3. Recommencez et intégrez votre encadrant(e) de TP, avec un rôle reporter (ou supérieur), puis de même avec frederic.gruau, je pourrai venir visiter en cas de problème.

Note : vous ne pouvez inviter que quelqu'un qui s'est déjà connecté à ce gitlab.

Connecter Git et IntelliJ, Import du projet, à faire par chaque membre du groupe et sur chaque ordinateur utilisé.

1. lorsque vous faites "New-¿Project From Version Control" dans IntelliJ
2. Une fenêtre s'ouvre, dans la barre latérale à gauche cliquer sur GitLab. Dans la barre prévue à cet effet entrer l'URL suivante : `https://gitlab.dsi.universite-paris-saclay.fr/`
3. Cliquer sur Generate Token, se connecter à GitLab et accepter ensuite les autorisations souhaitées sur GitLab.
4. Copier la clé générée et l'insérer dans IntelliJ dans le champ de texte prévu à cet effet.
5. Cliquer sur clone et normalement ça fonctionne !
6. Il est possible qu'IntelliJ vous redemande le token lors de votre premier push

Autre possibilité de problémes :

1. Dans Version control : Git, il est possible que le message rouge Git is not installed apparaisse. Dans ce cas, systématiquement à droite sur la même ligne il y a un lien cliquable Download and install. Cliquer dessus et l'installation se fait automatiquement. Après quelques secondes de patience, reprendre les étapes du TP.
2. lors de la création de la clé Token, sélectionner role as Developer / Maintener et non pas Guest comme par défaut.

Initialisation des fichiers, à faire par l'un des membres du groupe.

1. Dans le projet IntelliJ, faites un clic droit new, directory, et nommez src le dossier créé.
2. Faites un clic droit sur src, puis sélectionnez mark directory as, et sources root.
3. Ajoutez dans le dossier src vos fichiers .java. Si l'interface vous propose des les ajouter à Git, acceptez.
4. Poussez les nouveaux fichiers sur le dépôt Git : dans le menu git, allez chercher les actions commit et push (l'action commit ouvrira dans l'interface une fenêtre où vous devez écrire un message décrivant le contenu de la modification que vous enregistrez).

Travailler avec git À réaliser par chaque personne travaillant sur le projet.

1. Au début de chaque session de travail, récupérez les fichiers du dépôt partagé pour mettre à jour vos fichiers locaux avec pull. Pour cela : cliquez dans le menu git, puis pull. Recommandation : c'est généralement une bonne idée, dans modify options, d'allez sélectionner –rebase.
2. Après chaque modification significative, enregistrez (localement) votre travail avec commit. Dans le menu git, sélectionner commit, et entrer un message décrivant la modification.
3. À la fin de chaque séance (voire plus souvent), publiez vos commits sur le dépôt partagé avec push. Dans le menu git, sélectionner push, et valider.

Pour éviter les interférences entre les modifications faites par plusieurs personnes travaillant en parallèle sur le même projet, il vaut mieux souvent publier ses modifications (commit/push) et toujours commencer par récupérer les modifications des partenaires (pull) avant de commencer à coder.

Intellij propose une interface "code with me" qui permet de programmer simultanément à plusieurs en temps réel. Elle a été utilisé avec succés par certains de mes meilleurs étudiants, en 2022, qui m'en ont dit le plus grand bien.

8.1.3 Consignes

Faites-le dépot git dès le début, ce afin que nous puissions nous assurer que vous avancez. La mise à jour régulière de la version sur git fait partie des critères dans la note finale.

Il n'y a pas à proprement parler de "rendu" , votre rapport sera dans le README.txt de votre git, excepté le diagramme de classes que vous pouvez publier séparément, dans le git, sous forme d'un PDF. Il est préférable que vous dessinez un résumé du diagramme, qui explique vos choix de structures. Le diagramme généré automatiquement est trop complexe à lire et à comprendre. Vous pouvez aussi mettre une ou deux impressions ecran de votre rendu visuel.

Les soutenances ont lieu à votre horaire de TP, dans votre salle habituelle de TP, la semaine d'avant les examen De cette manière, la maitrise java que vous aurez acquis grâce au projet, vous servira aussi pour réussir l'examen : coup double. Nous assignerons un horaire précis dans le fichier des binomes. Les soutenances sont rapides, entre 10 et 15 minutes. Pendant qu'un groupe passe, le suivant s'installe, et les autres attendent dans une autre salle à côté. L'un des objectifs principal de la soutenance est de vérifier que vous êtes bien l'auteur de votre projet, vous serez noté sur les parties que vous maitrisez. Si vous ne pouvez pas expliquer le code, on considére que ce n'est pas votre code, vous avez 2 sur 20, et cela arrive. En effet l'expérience montre qu'il est en fait impossible de simuler être l'auteur d'un code qu'on a pas produit.

Le README.txt détaille :

1. Les parties du sujet que vous avez traitées.
2. Vos choix d'architectures.
   Les choix d'architecture font partie de ce que nous allons évaluer : donnez un peu de détail sur l'organisa-

tion de vos classes et le partage des rôles. Utiliser les différentes techniques vues en cours et en TP.

3. Les problèmes qui sont présents et que vous n'avez pas pu éliminer.
4. Les morceaux de code écrits à plusieurs ou empruntés ailleurs.
   Sans cette information, cela s'appelle du plagiat, et est considéré comme une fraude par le réglement de l'université.

Intellij dans sa version "ultimate" [2], vous permet de directement pondre tout le diagramme de classes, mais c'est dur à lire pour le correcteur. Si vous souhaitez avoir tout les points pour le diagramme de classe, il faut faire l'effort d'en faire une version résumée plus lisible. Nous allons tester votre application : le code source doit impérativement compiler sans erreur. *Si des morceaux de code ne sont pas finis, vous pouvez les mettre en commentaire. Une méthode non implémentée peut avoir un code vide, ou renvoyer une valeur arbitraire comme* `null`.

Présence à la soutenance Elle est obligatoire pour tous les membres du groupe car il nous faut valider l'implication réelle de chaque participant, et cela implique du présentiel ! Vous pourrez peut être difficilement profiter de la semaine de révisions pour faire le voyage chez vos parents, si ceux ci habitent trop loin.

8.1.4 L'évaluation

Les trois consignes (format et contenu du `README`, code source qui compile, presence) sont impératives ; leur non-respect équivaut à la note zéro. La note finale sera basée principalement sur l'avancement de votre projet, le premier but de votre présentation est donc de démontrer jusqu'ou vous êtes allé. Si rien ne s'éxecute vous avez 2 sur 20. Compte aussi : la qualité de la présentation, l'implication respective des deux membres, votre `README`, la maitrise du code dont vous prétendrez être l'auteur (comment vous répondez aux questions), la qualité du code, la présence de tests junit, la clarté de la présentation, l' habillage graphique, les commentaires et en particulier la description des paramètres de chaque méthode, le fait de répartir les classes dans trois packages : modele, vue et controlleur.

8.1.5 Cause principale d'échec

De commencer a s'y mettre trop tard. Vers la fin du semestre vous êtes assommé de trucs à faire, car il n'y a pas seulement le projet de POGL. Pour réussir sans douleur, et profitez de cette opportunité que vous avez de devenir des programmeurs "pros", commencez à travailler exactement maintenant en imaginant que la soutenance est la semaine prochaine. Chaque année, un bon quart des binomes plantent pour cause de mauvais timing. Je garanti que vous êtes tous capable de réussir au moins 12 sur 20 si vous démarrez à temps.

2. cette version est payante, mais avec votre email a paris-saclay vous y avez accés gratuitement

8.2 The Projet "Colt Express"

Ce projet de programmation vous invite à construire une version informatique et un peu simplifiée du jeu *Colt Express* [3].

Les indications de rendu et consignes générales sont groupées dans la section précédente.

8.2.1 Aperçu des règles du jeu

Le jeu se déroule à bord d'un train, composé d'une locomotive et d'un certain nombre de wagons. Les joueurs incarnent des bandits qui ont sauté à bord pour détrousser les passagers. Objectif : récupérer le plus de butin possible, chacun pour soi. Il s'agit d'un jeu de programmation, dans lequel on alterne entre deux phases :

Planification : chaque joueur décide secrètement un certain nombres d'actions, que son personnage va effectuer dans l'ordre.

Action : on effectue toutes les actions numéro 1, puis toutes les numéro 2, et ce jusqu'au bout.

Les bandits peuvent se trouver dans les wagons ou la locomotive, et pour chacun de ces éléments soit à l'intérieur soit sur le toit. Dans cet énoncé, par abus de langage on désignera par « wagon » un élément quelconque du train, qui peut être la locomotive. Les actions possibles pour les bandits sont :

- Se déplacer d'un wagon en avant ou en arrière, en restant au même étage.
- Aller à l'intérieur ou grimper sur le toit de leur wagon actuel.
- Braquer un voyageur pour récupérer du butin (ou simplement récupérer un butin qui a été abandonné là).
- Tirer sur un autre bandit proche pour lui faire lâcher son butin.

Les butins récupérables à bord du train sont :

- Des bourses valant entre 0 et 500$, auprès des passagers, à l'intérieur des wagons.
- Des bijoux valant 500$, auprès des passagers, à l'intérieur des wagons.
- Un magot valant 1000$, à l'intérieur de la locomotive, sous la garde du Marshall.

Un Marshall est présent à bord du train et peut se déplacer entre la locomotive et les wagons, en restant toujours à l'intérieur. Il tire sur tous les bandits qui se trouvent à la même position que lui et les force à se retrancher sur le toit.

8.2.2 Organisation

L'application sera organisée selon une architecture Modèle-Vue-Contrôleur (MVC), et vous pouvez vous inspirer du fichier Conway.java disponible dans votre archive java. Les différentes sections du sujet vous proposent de mettre progressivement en place les différents éléments du jeu, et suggèrentun ordre dans lequel les traiter. Précisons toutefois qu'il est toujours utile de réfléchir en amont à la manière dont les éléments suivants pourront s'insérer dans votre code.

3. De Christophe Raimbault, chez Ludonaute.

8.2.3 Un modèle réduit

Un élément central du modèle est le train lui-même, qui sert de plateau de jeu. Déterminez les différents éléments qui le composent (il pourra être utile de les organiser en diagramme de classes) et écrivez les classes correspondantes. Ajoutez une classe pour les bandits et une action de déplacement.

Note : pour l'instant, on ne s'intéresse pas aux autres actions (braquage, tir), ni aux butins ni au Marshall.

Quelques précisions :

- Le nombre des wagons est donné par une constante `NB_WAGONS` que vous devez définir dans la classe principale. Vous pourrez par exemple la fixer à `4`.
- On placera pour l'instant un unique bandit à bord du train, sur le toit du dernier wagon. Le nom de ce bandit est donné par une constante `NOM_BANDIT_1`.
- Un déplacement vers l'arrière lorsque l'on est dans le dernier wagon ou en avant lorsque l'on est dans la locomotive n'a pas d'effet. De même pour un déplacement vers le toit ou vers l'intérieur lorsque l'on y est déjà.
- Vous devez écrire des tests unitaires pour les actions des bandits.
- À chaque action du bandit, affichez un compte rendu sur la sortie standard. Par exemple :

```
Wyatt grimpe sur le toit.
Wyatt est déjà sur le dernier wagon.
```

Indication : pour définir les directions, vous pouvez utiliser une énumération Java, par exemple

```
enum Direction { AVANT,
                 ARRIERE,
                 HAUT,
                 BAS }
```

On peut alors désigner une direction sous la forme `Direction.AVANT`. Cela sera utilie aussi bien pour les déplacements, que pour les tirs.

8.2.4 Une belle vue

Pour commencer à voir le modèle s'animer, ajoutez une vue comportant deux parties :

- Un panneau d'affichage dans lequel on voit les différents éléments du train et la position du bandit.
- Un bouton `Action !` qui effectue la prochaine action du bandit.

Quelques précisions :

- Pour l'instant, le bandit n'effectue qu'une suite d'actions prédéterminées (écrites en dur dans le modèle).
- L'affichage doit être mis à jour à chaque déplacement du bandit.
- Continuez à afficher les compte-rendus sur la sortie standard.

Indications :

- Vous pouvez soit récupérer les définitions de `Observer` et `Observable` vues en cours soit utiliser celles de la bibliothèque standard. Dans ce deuxième cas : les appels à la méthode `notifyObservers` devront être précédés d'un appel à `setChanged()`.
- Vous pourrez avoir l'usage de nouvelles fonctions de la classe `Graphics`, comme `drawString` ou `drawRect`. Consultez leur documentation.

8.2.5 Une poignée de dollars

Ajoutez dans le modèle les butins et le Marshall.

Quelques précisions :

- À l'intérieur de chaque wagon se trouvent initialement entre 1 et 4 butins (de type bourse de valeur aléatoire, ou bijou). La nature des butins présents à chaque position doit apparaître sur la vue, mais pas leur valeur.
- Le magot et le Marshall sont placés à l'intérieur de la locomotive.
- Avant chaque action du bandit, le Marshall se déplace avec une probabilité p dans une direction aléatoire (mais toujours à l'étage inférieur, c'est-à-dire à l'intérieur des wagons). La probabilité p est définie par une constant `NERVOSITE_MARSHALL`, qu'on pourra fixer à `0.3`.
- Le bandit a maintenant accès à une nouvelle action : braquage. Cette action lui fait récupérer un butin au hasard parmi ceux présents sur sa position.
- Si le Marshall atteint ainsi la position du bandit, le bandit lâche un des butins qu'il a ramassés tiré au hasard (s'il en a) et se déplace immédiatement vers le toit. Le butin est ajouté à l'ensemble des butins de la position dont le bandit vient d'être chassé, et peut être récupéré à nouveau. Continuez à afficher sur la sortie standard un compte-rendu de tous ces événements.
- Une fois toutes les actions effectuées, affichez le montant total des butins possédés par le bandit.

8.2.6 Tout est sous contrôle

Toujours en considérant un seul bandit, ajoutez maintenant de nouveaux boutons permettant au joueur de planifier les actions de son personnage. Le contrôleur doit alterner entre deux états :

1. Un état de planification, dans lequel le joueur doit donner un certain nombre d'ordres en cliquant sur les boutons correspondants. Le bouton `Action !` est inopérant pendant cette phase.
2. Un état d'action, dans lequel chaque clic sur le bouton `Action !` effectue la prochaine action, les autres boutons étant inopérants.

Quelques précisions :

- Le panneau de commande doit afficher une ligne de texte indiquant quelle est la phase en cours (voir par exemple la classe `JLabel`).
- Le nombre d'actions que doit indiquer le joueur lors d'une phase de planification est une constante `NB_ACTIONS` que vous pouvez fixer par exemple à `4`.

Indication : Définissez une classe abstraite "Action", avec une méthode abstraite "executer" qui modifie le modéle, et qui sera définie différement pour chaque type d'action possible. Cela vous évitera de faire un gros IF au moment d'exécuter l'action, et c'est bien la une des caractéristiques importante

de la programmation objet que nous souhaitons vous inculquer.

8.2.7 Echanges de plombs entre amis

Étendez maintenant le jeu pour pouvoir y jouer à plusieurs. Une vue supplémentaire devra montrer l'état de chaque bandit, en particulier les butins qu'il porte et le nombre de balles qu'il a encore à sa disposition. C'est le moment également d'ajouter aux bandits l'action de tir.
Quelques précisions :

- Le nombre de joueurs est donné par une constante `NB_JOUEURS`.
- Les joueurs jouent toujours dans le même ordre, laissé à votre discrétion (dans la vue montrant les états des bandits, respecter cet ordre).
- Lors de la phase de planification, chaque joueur à son tour donne tous ses ordres à la fois.
- Lors de la phase d'action, on effectue d'abord toutes les premières actions (dans l'ordre des joueurs), puis toutes les deuxièmes actions (dans le même ordre), et ainsi de suite.
- Un tir est fait dans l'une des quatre directions, qu'on interprète de la même manière que pour les déplacements : en avant ou en arrière au même étage, ou vers le haut ou vers le bas dans le même wagon. Un tir vers le haut (resp. vers la bas) lorsqu'un bandit se trouve sur le toit (resp. à l'intérieur) vise la position occupée par le bandit.
- Le tir touche un bandit tiré au hasard parmi les occupants de la position visée, excepté le tireur lui-même.
- Un bandit touché par un tir lâche l'un de ses butins tiré au hasard. Le butin est ajouté à ceux présents à la position du bandit touché et pourra être récupéré à nouveau.
- Chaque tir utilise une balle. Chaque bandit possède à l'origine un nombre de balles donné par une constante `NB_BALLES`, que vous pourrez par exemple fixer à `6`.
- Le compte-rendu textuel doit également mentionner ces actions de tir et leurs conséquences.

8.2.8 Expression libre

Ajoutez pour finir au moins une fonctionnalité supplémentaire à votre application.
Voici une mini-liste de suggestions d'ajouts. La liste n'est évidemment pas limitative et a simplement pour rôle de vous donner un point de départ.

- Récupérer tous les paramètres du jeu sur la ligne de commande plutôt que d'utiliser des valeurs arbitraires, ou les récupérer dans une fenêtre de dialogue s'ouvrant au début de la partie.
- Gérer plus finement les vues pour ne redessiner que les positions qui ont effectivement été modifiées.
- Lors de la planification, afficher la séquence d'ordres donnée par le joueur courant et lui donner la possibilité de revenir sur ses choix tant qu'il n'a pas cliqué sur un bouton de validation (le bouton `Action !` peut être utilisé pour la validation).
- Introduire des raccourcis clavier pour les différentes commandes du jeu.
- Afficher le compte-rendu de la partie non sur la sortie standard mais dans une zone dédiée de la fenêtre graphique.
- Ajouter une intelligence artificielle pour faire jouer certains bandits par l'application.
- Transformer votre application en une application Android.

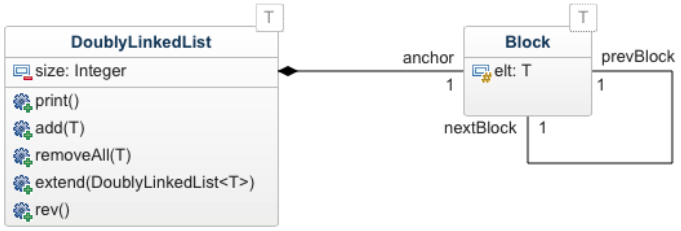
9 TP6, Listes doublement chaînées : Invariants, assertions, et lambda-expressions

Dans ce TP, nous allons réaliser une structure de liste doublement chaînée, et utiliser des assertions pour vérifier que les invariants de cette structure sont toujours valides. Ce sera aussi l'occasion d'utiliser les lambda-expressions.

9.1 Listes doublement chaînées

Une liste doublement chaînée est constituée d'une séquence de blocs contenant chacun une valeur (définie), dans laquelle chaque bloc possède un pointeur vers son successeur et un pointeur vers son prédécesseur.

Pour réaliser une telle structure, on propose l'organisation présentée par le diagramme suivant :



Pour que chaque bloc, y compris le premier ou le dernier, ait à la fois un prédécesseur et un successeur on aura recours à l'astuce suivante : chaque liste doublement chaînée contiendra, en plus de ses blocs normaux contenant les éléments de la liste, un bloc spécial appelé *ancre* (`anchor`). L'ancre est le prédécesseur du premier bloc utile, et aussi le successeur du dernier bloc utile. Dans le cas d'une liste vide, l'ancre est son propre prédécesseur et son propre successeur. L'attribut `elt` de l'ancre vaut systématiquement `null`.

Intêret Une liste doublement chaînée permet de gagner en complexité sur l'insertion et la suppression par rapport à la liste simplement chaînée, en plus de permettre un parcours des elements à l'envers. Mais on utilise un pointeur supplémentaire par élément. Il faut donc maintenir cohérentes deux fois plus de variables que dans une liste simplement chaînée.

9.2 Prise en main, diagramme d'instances

Question 1. Récupérer l'archive, executer votre squelette, et parcourez le. Que fait-il ?

Question 2 Pour bien comprendre ce qui se passe, Donner des diagrammes d'instances pour la liste vide et pour la liste contenant les éléments 4, 2 et 1 dans cet ordre.

Question 3 : Comment sont implémentées les classes Block et DLLiterator, par rapport a la classe DLL ? Expliquer l'utilité de ce choix d'implémentation pour DLLiterator, puis pour Block.

9.3 Vérifier les invariants par des assertions

Question 4 : Énoncer les invariants de notre structure de liste chaînée. Regarder notamment les aspects suivants :

— définition ou non-définition des champs `elt`,
— symétrie des liens prédécesseur-successeur,
— valeur du champ `size`.

9.4 Assertions java

Pour rattraper les erreurs aussi tôt que possible, nous allons écrire du code chargé de vérifier les invariants. Les tests correspondants seront intégrés dans des instructions `assert`. Une assertion java ne sera exécutées que lorsque l'option `-ea` est activée. Il faut donc, dans intellij, cliquer sur la petite flèche qui pointe vers le bas, en haut a droite, juste a droite de DLL, selectionner "edit configuration" puis "modify option" puis "add vm option", et saisir "-ea" dans la case "VM option". Pour vérifier que cela marche faite un assert(false), et vérifier que le programme s'arrête bien.

Question 5, pour vérifier l'invariant sur la taille, on a besoin de calculer quoi ? Comment pouvez vous le calculer facilement (inspirez vous de la méthode print) ? Calculer le !

Question 6 En déduire une première version d'une méthode `private boolean checkInvariants()` qui teste si la taille de la liste correspond au nombre de ses éléments.

Question 7 il s'agit maintenant de "tester le test" checkInvariants (vérifier que ce test teste bien ce que on souhaite tester). Comment faite vous ?

9.5 Lambda-expressions.

Pour les invariants 2 et 3, il va falloir à présent appliquer une condition sur chaque block. C'est l'occasion d'utiliser des lambdas expressions.

Question 8 : Allez sur internet pour vous renseigner sur l'interface `java.util.function.Predicate<T>` . Que dit elle ? Importer la.

Question 9 Sous quelle condition une lambda expression peut implementer l'interface `Predicate<T>`

Question 10 Définir une méthode qui renvoie `true` si tous les blocs de la liste, ancre comprise, vérifient le prédicat `p`. `private boolean forAllBlocks(Predicate<Block> p)` On commencera par définir une méthode qui renvoie un itérateur sur les block, en créant une classe interne anonyme. Le fait que c'est une classe interne permet d' avoir accés aux infos dont on a besoin sur la liste.

```
private Iterator<Block> blockIterator() {
  return new Iterator<Block>() {
    private Block currentBlock = A compléter;
    public boolean hasNext(){return A compléter};
    public Block next() { A compléter  }; }
```

Question 11 Compléter la méthode `checkInvariants()` de manière à ce qu'elle vérifie tous les invariants 2 et 3, en utilisant deux lambda expressions.

Question 12 Comment vérifier sur les exemples précédents que les instances correctes sont validées et que les instances incorrectes sont rejetées.

9.6 Manipuler des listes double chaînées [4]

Nous allons maintenant définir des opérations pour modifier des listes doublement chaînées. Vous utiliserez des assertions pour vous assurer que les modifications respecte les invariants.

Question 13 On veut programmer une méthode removeAllBlock qui suprime tout les block contenant un élement donné, que peut on commencer par programmer qui serait plus simple ? programmer-le, et utiliser une assertion pour serrez les boulons.

L'interface `Consumer<T>` ressemble à `Test<T>`, elle déclare une méthode accept, `void  accept(T t)` qui prends un elément de type T et ne renvoie rien.

Question 14 Programmer une méthode forAllBlock2 qui itérer un `Consumer<Block>` sur chaque block d'une liste.

Question 15 Utilisez à présent ce forAllBlock2 pour programmer notre removeAllBlock avec une lambda expression.

Question 16 Nous vous donnons le code d'une méthode swap12 sensée échanger le premier block avec le second. Exécuter la sur smallList et vérifier que le test plante, pourquoi ? corriger !

9.7 Pour les "mordus"

Question 17 Définir la méthode `void concat(DLL<T> l)` qui accole les blocs d'une liste donnée à la fin de la liste courante.

Question 18 Définir une méthode `public void rev()` qui renverse la liste (les prédécesseur et successeur de chaque bloc sont inversés).

10 TD4 Train : Liaison dynamique et interface.

Ce TD est centré principalement sur la notion de liaison dynamique, mais aussi sur les interfaces. Il présente une situation simple ou une classe étends une autre classe et simultanément, implémente une interface. Il illustre comment on peut à partir d'une interface, définir des méthode qui travaillent sur des instances de ces interfaces.

On va construire une hiérarchie de classes représentant les éléments roulants d'un hypothétique compagnie ferroviaire. Les éléments décrits seront les suivants.

4. Cela devient palpitant, malheuresement l'expérience montre que peu d'étudiants ont le temps d'aborder ceci durant le TP, vous devrez donc terminer chez vous.

— La classe générale `EltTrain` regroupe tous les éléments de train (on peut imaginer qu'un train est une liste d'objets de cette classe). Chaque élément possède un nom (type `String`) et a un certain poids.
— Une `Locomotive` est un élément de train pouvant tracter d'autres éléments dans une certaine limite de poids.
— Un `Wagon` est un élément de train pouvant accueillir un certain nombre de passagers.
— Un `WagonBar` est un wagon ne pouvant pas accueillir de passagers (les passagers doivent avoir un siège réservé dans un autre élément adapté).
— Un `Autorail` est un élément de train pouvant tracter d'autres éléments dans un certaine limite de poids et pouvant accueillir un certain nombre de passagers.

On veut de plus avoir les comportements suivants.

— Il est impossible de créer un élément directement avec le constructeur de la classe `EltTrain` (on peut seulement passer par l'un des quatre autres types d'éléments).
— Chaque élément pouvant accueillir des passagers possède une méthode `boolean reserver()` qui renvoie `true` et enregistre qu'une place supplémentaire est accupée si la capacité maximale n'est pas encore atteinte, et renvoie `false` sinon. On alloue les places dans l'ordre.
— Tous les éléments possèdent une méthode `int calculerPoids()` qui estime le poids total de l'élément, passagers compris (on comptera 75kg par passager).
— Chaque nouvel élément créé reçoit un nom, qui est différent des noms des autres éléments déjà créés.

10.1 Organiser les classes en hiérarchies

Question 1 Deux propriétés caractérise les eléments de train, lesquelles ?

Question 2 Certain élements ont l'une, d'autres ont l'autre, y en a qu'on les deux. lesquelles ?

Question 3 Laquelle des deux propriétés est plus intéressante a hériter pour factoriser le plus de code.

Question 4, Comment faire pour que les élements qui doivent en hériter le puisse ?

Question 5 La la deuxieme proprité est caractérisée par la présence de quelle méthode ?

Question 6 Ah bon, et pour la deuxieme propriété comment l'implémenter de facon a aussi pouvoir regrouper du code qui voudrait utiliser cette méthode

Question 7 Dessiner le diagramme UML de la hierarchie obtenue, àl'exeption de la classe wagonBar.

Question 7bis Il est impossible de créer directement un element de train, ca implique quoi ?

Question 8 Quels sont parmi les attributs ceux dont la valeur ne bougera pas après leur initialisation ?

Question 9 Comment peut on garantir que les valeurs ne bougeront effectivement jamais ?

10.2 Constructeurs aussi en hierarchie.

Question 10 Ou c'est qu'on initialise l'attribut nom ?

Question 11 Comment faire pour facilement donner un nom différent a chaque element du train ?

Question 12 Ou c'est qu'on initialise capacité, reservation ?

Question 13 Ecrire le constructeur de eltReservable et wagon

Question 14 On voudrait quand même pouvoir dire que un wagon-bar est un wagon, donc hérite de wagon, cependant on ne peut par réserver dans un wagon bar comment faire ?

10.3 Redefinition de méthodes

Question 15 Dans quelle classe se trouve la méthode réserver ? qui la réutilise ? Programmer la.

Question 16 En ce qui concerne la méthode calculerPoids(), peut on s'en tirer avec une seule définition ? Programmer.

10.4 Que se passe-t-il ? Liaison dynamique.

Devinez le type réel et le type apparent d'un objet a été fait en cours, et ce n'est pas du tout aussi évident qu'il n'y parait au premier abord. Considérons la séquence d'instructions ci-dessous. Indiquez quelles instructions sont correctes ou incorrectes. Pour les instruction incorrectes, précisez si l'erreur est détectée à la compilation ou à l'exécution. Pour les instructions correctes, précisez le résultat obtenu. Pour l'exécution des instructions qui viennent après une instruction incorrecte, on ignorera l'effet de cette instruction incorrecte. C'est aussi un classique testé à l'examen.

```
EltTrain e;
Wagon w;
WagonBar wb;
Locomotive l;
Autorail a;
e = new EltTrain(100);
l = new Locomotive(1000, 10000);
w = new Wagon(1000, 20);
wb = new WagonBar(1000);
a = new Autorail(1000, 5000, 10);
System.out.println(w.capacite);
System.out.println(wb.capacite);
System.out.println(e.nom);
System.out.println(wb.nom);
a.reserver();
w.reserver();
wb.reserver();
l.reserver();
wb = (WagonBar) w;
System.out.println(wb.capacite);
l = (Locomotive) a;
e = wb;
e.reserver();
System.out.println(e.capacite);
w = (Wagon) e;
w.reserver();
System.out.println((EltTrain) w).capacite);
System.out.println((WagonBar) w).capacite);
```

10.5 Train complet, instanceOf, cast.

On étudie comment la primitive "instanceof" et le cast de type permettent de maniper des instances de l'interface elemenMoteur. C'est un classique qui sera donc testé en exam.

Question 17 : Créez une classe `Train` représentant les trains. Un train sera caractérisé par une liste d'éléments `ArrayList<EltTrain>`. Définir des méthodes ayant les effets suivants.

- Ajouter un élément à un train.
- Calculer le poids total du train.
- Vérifier la bonne formation du convoi :
  - le poids total doit être inférieur aux capacité de traction cumulées des éléments tracteurs,
  - les éléments tracteurs sont toujours aux extrémités.

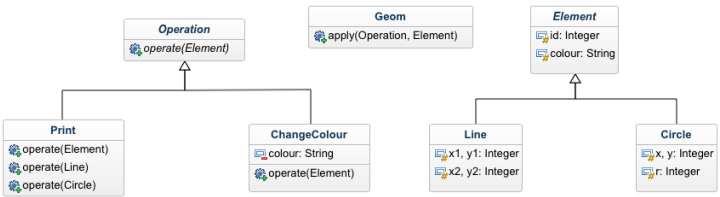
On mettra en commentaire, un entête qui décrit ce que font les méthodes, et aussi les paramètres et résultats.

11 TP7 Dessin vectoriel : visiteurs.

Dans ce TP, nous allons utiliser des techniques objet pour représenter et manipuler des dessins vectoriels. Ce TP illustre le design pattern appelé "visiteurs", ainsi que les notions de surcharge, et étudie également la liaison dynamique. Il y aura des questions sur ce TP à l'examen, donc ne le ratez pas même si vous êtes en retards sur votre projet.

11.1 Surcharge statique et liaison dynamique

Le squelette de code réalise le diagramme de classe suivant, dans lequel on décrit plusieurs types d'éléments graphiques et d'opérations pouvant s'y appliquer.



Qui

un fichier SVG correctement formaté doit inclure une balise de fermeture ¡/svg¿ à la fin.

La classe `Geom` fournit une méthode `apply` qui applique une opération à un élément graphique. L'opération Print affiche le code SVG correspondant à l'élement. Le format SVG est un format texte. Il permet de spécifier des figures géométrique en vectoriel, et de facilement zoomer ou translater les figures. `https://developer.mozilla.org/fr/docs/Web/SVG/Tutoriel`

Question 1 Dans la classe `Geom`, la méthode `main` qui crée un objet `o` de classe `Print`, un objet `e` de classe `Line`, et réalise les deux appels `o.operate(e)` et `apply(o, e)` Dans chacun des deux appels il s'affiche un truc différent. Vérifiez le. C'est en fait trés étonnant, car le deuxiéme appel `apply(o, e)` ne fait rien d'autre que de recopier le premier. Utilisez le debugger pour tenter de comprendre ce qui se passe. Les questions suivantes vont apporter un éclairage.

Question 2 : Quels sont les types réels de `o` et `e` ?

Question 3 : Dans chacun des deux appels à `operate`, quels sont les types apparents de `e` et `o` au niveau de l'appel ?

Question 4 : Quelle méthode `operate` est sélectionnée ?

Question 5 Qu'en déduire sur la manière dont Java utilise les types apparents et types réels pour déterminer la méthode à appeler ?

La surcharge de fonction (overloading en anglais) est une possibilité offerte par certains langages de programmation de définir plusieurs fonctions ou méthodes de même nom, mais qui diffèrent par le nombre ou le type des paramètres effectifs.

La surcharge est statique si le choix de la version est fait en fonction du nombre d'arguments et de leur type apparent, déterminé à la compilation

Question 7, Est ce que la surcharge statique fonctionne pour selectionner quelle opération appliquer sur quel objet, parmis un ensemble d'opération et d'objets ?

Question 8 : On va donc renoncer à utiliser la surcharge. Modifiez la classe `Print` en renommant `operateLine` la méthode `operate` attendant un paramètre de type `Line` et `operateCircle` la méthode `operate` attendant un paramètre de type `Circle`.

11.2 Sélection en fonction du type réel de tous les paramètres

Comment est ce que la méthode `operate` de Print va pouvoir sélectionner operateLine, ou operateCircle en fonction du types réels du paramètres explicite `e` ? Nous allons considérer deux solutions en expliquant pourquoi elle ne sont pas très satisfaisantes. Notre troisiéme solution introduit le design pattern "visiteurs" qui fonctionne bien.

Solution 1 : test de sous classe, et transtypage explicite. La solution la plus brutale consiste à tester quelle est le type réel, qui est une sous-classe, en utilisant `instanceof` . Puis, faire un appel a operateLine en transtypant préalablement sur Line, ou operateCircle en transtypant sur circle.

Question 9 Modifier le operate de Print, en suivant cette stratégie et tester que l'execution de `apply(o, e)` affiche bien "Line".

Cette solution est cependant lourde car on doit faire soi-même le test et la sélection d'un comportement.

Solution 2 : liaison dynamique double. Pour éviter de faire soi-même le test d'instance, on peut demander à l'opération de faire appel à des méthodes déclarées dans les `Element` eux même et (re)définies dans chaque type d'éléments.

Question 10 Déclarer un print() abstrait dans Element, implémenter le dans Line, et Circle, définisser une nouvelle version de operate dans Print qui appeller `e.print()`, tester que l'execution de `apply(o, e)` affiche bien "Line".

Question 11 pourquoi appelle t-on cela liaison dynamique double ?

Cette solution est lourde parce qu'elle demande, lorsqu'on veut une nouvelle opération, d'écrire du code dans chaque

classe d'`Element`. On préfére regrouper ensemble tout les traitements relatif a une opération. Ce point deviendra plus évident lorsq'on abordera les structures composites.

Solution 3 : Visiteurs. Le *design pattern* Visiteur donne une solution générique qui ne demande de modifier les classes d'`Element` qu'une seule fois. Le principe est le suivant :

— La classe `Operation` implémente l'interface `Visitor` suivante, dans laquelle un traitement est prévu pour chaque type d'élément.

```
interface Visitor {
    public void visitLine(Line l);
    public void visitCircle(Circle c);
}
```

Les méthodes `visitLine` et `visitCircle` sont à définir dans chaque classe concrète d'`Operation`.

— La classe `Element` implémente l'interface `Visitable` suivante.

```
interface Visitable {
    public void accept(Visitor v);
}
```

Pour cela, chaque classe concrète d'`Element` fournit la définition

```
public void accept(Visitor v) {
    v.visit***(this);
}
```

en remplaçant `visit***` par la désignation de la méthode `visit` adaptée à la classe concrète [5].

Ainsi, c'est au niveau des classes implémentant `Visitor` que l'on place tout le code à appliquer aux différents `Element` : les définitions sont groupées par opération plutôt que par objet cible.

Question 12 Modifiez les classes d'`Operation` pour qu'elles implémentent les interfaces `Visitor`

Question 13 Modifiez les classes d'`Element` pour qu'elles implémentent les interfaces `Visitable`.

Question 14 tester ce nouveau code en l'utilisant pour modifier la couleur de notre ligne puis pour l'afficher.

Pour expérimenter le fait que les définitions sont groupées par opération plutôt que par objet cible nous allons introduire deux nouvelles opérations.

Question 15 Créer une nouvelle classe d'opération `Translate`, qui applique une translation de coordonnées `dx, dy` à l'élément auquel elle est appliquée.

Question 16 Tester dans le main, sur la ligne qui existe déja

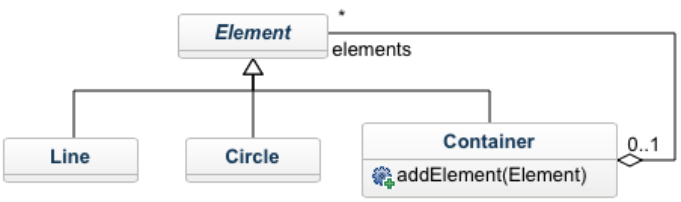
Question 17 Créer une nouvelle classe d'opération `Zoom`, qui multiplie les coordonnées et les distances par un facteur `z`, à l'élément auquel elle est appliquée.

Question 18 Tester sur un cercle.

5. Ce *design pattern* est généralement présenté en donnant le même nom `visit` à toutes les méthodes `visit***`. On remarquerait alors ici que la méthode `accept` est recopiée à l'identique dans chaque classe concrète (mêmes nom, signature et code) plutôt que factorisée dans la classe abstraite mère. Si vous pensez que c'est bizarre, revenez à la question 1.

11.3 Structure composite

Nous allons maintenant tenir compte du fait qu'un dessin vectoriel peut contenir plusieurs éléments, et que les éléments peuvent être groupés. On obtient alors un structure de donnée récursives, et le nom de "visiteurs prends tous son sens, car on visite quelque chose de arbitrairement grand au lieu d'un seul élement borné. Pour ceci on ajoute une nouvelle sous-classe d'`Element` appelée `Container`, qui contient une liste d'éléments. Notez qu'il est tout à fait possible qu'un `Container` contienne d'autres `Container`, et ce jusqu'à une profondeur arbitraire. Nous utilisons là le *design pattern* Composite.



Question 19 Ajoutez une classe `Container` telle que décrite ci-dessus, pour l'instant, ne faite rien dans la méthode accept().

Les visiteurs donnent une manière agréable d'itérer une opération sur tous les éléments d'une collection, même si cette collection a une structure potentiellement complexe comme peut l'être un enchevêtrement d'`Element` et de `Container`. Le principe est de séparer deux aspects :

— L'opération à effectuer, qui est définie dans le `Visitor` comme on l'a déjà vu.

— Le parcours de la collection d'objets, qui est défini une fois pour toutes dans les `Visitable`.

Ainsi, c'est la méthode `accept` de la classe `Container` qui va se charger de propager l'opération aux `Element` qu'elle contient, en faisant appel à leurs propres méthodes `accept`.

Question 20 Définissez sa méthode `accept` de sorte à ce qu'un visiteur atteignant un `Container` soit propagé aux `Element` qu'il contient et qu'on puisse zoomer, translater, afficher toute la hiérarchie. Tester qqc.

En SVG, lorsqu'on définit un groupe, le début du groupe est indiqué par le markup `"<g>"`, et la fin par le markup `"</g>"`. Ce markup permet de définir des propriétés pour tout les élements contenu dans le groupe. Par exemple, on peut changer la couleur de remplissage de ces élements avec `"<g fill= "red" >"`

Question 21 Qui doit afficher ces markup, et comment ?

Question 22 programme me-le.

Question 23 A présent crer une petite hierarchie de container, afficher le code SVG correspondant.

Question 24 Copier coller le code svg dans un fichier dont le nom se termine par .svg et ajouter au début du fichier le markup suivant :

```
<svg version="1.1"
    baseProfile="full"
    xmlns="http://www.w3.org/2000/svg">
```

Rajouter la balise de fin `</svg>` Ensuite, vous pourrez l'ouvrir le fichier avec un browser internet, on le faisant glisser dessus par exemple, et vous verrez s'afficher des cercles et des lignes.

En conclusion remarquez deux caractéristiques des visiteurs :

— il est facile d'ajouter une nouvelle opération, puisque cet ajout se limite à la création d'une classe,

— il est plus laborieux d'ajouter un nouveau type d'élément, car il faut alors compléter tous les visiteurs déjà écrits. Toutefois, de passer par des redéfinitions peut permettre de ne pas toucher à tout les visiteurs.

11.4 Comparaison avec Caml

En Caml, on peut directement déclarer des types récursif comme par exemple le type ASA, utilisé dans le cours de PIL des L2 info, pour déclarer des arbres de syntaxe abstaites. Ils combinent les unions et les produits, de tel types sont appelée algébriques. On peut ensuite facilement se promener, i.e. faire des calculs sur cet arbres en utilisant le pattern matchin de caml. Par exemple on peut ainsi associer un type a chaque noeud de cet ASA, ou calculer un environement courant,également pour chaque noeuds, pour vérifier des régles de portée.

En java, il n'y a pas ni ces types recursifs, ni ce pattern matching, mais le design pattern de visiteurs permet également de regrouper plusieurs bout de code pour se promener dans des structures hierarchiques tout en faisant au fur et a mesure, des calculs qui dépendent des différents type de noeuds. On peut donc dire que il remplace le pattern matching. C'est un peu moins concis, car il comporte une certaine dose de code redondant, mais tout de même comparable.

12 CoursTD exception

Nous allons construire des classes pour la gestion d'un emploi du temps hebdomadaire, dont plusieurs méthodes seront susceptibles de lever des exceptions. L'objectif est double : Apprendre les exceptions, et s'entraîner en prévision de l'examen à écrire du code simple sur papier.

12.1 Déclarer une exception - Cours

En Java, tout est objet. En particulier, les erreurs qu'on appelle en réalité des *exceptions*, sont des objets comme les autres, qui appartiennent à des classes. Ces classes sont inscrites dans une hiérarchie de sous-classes, montrée figure 1, et on peut en définir de nouvelles en utilisant l'héritage.

Tous les objets de la classe `Throwable`, et donc toutes les exceptions que nous pouvons définir, possèdent un certain nombre de champs et de méthodes, qui nous renseignent sur l'erreur qui a eu lieu. On peut par exemple récupérer un message associé à l'exception (`public String getMessage()`) ou obtenir des indications sur l'état de la pile d'appels. Comme `Throwable` hérite de `Object`, nous avons également une méthode `public String toString()`. Enfin, la classe `Throwable` possède deux constructeurs :

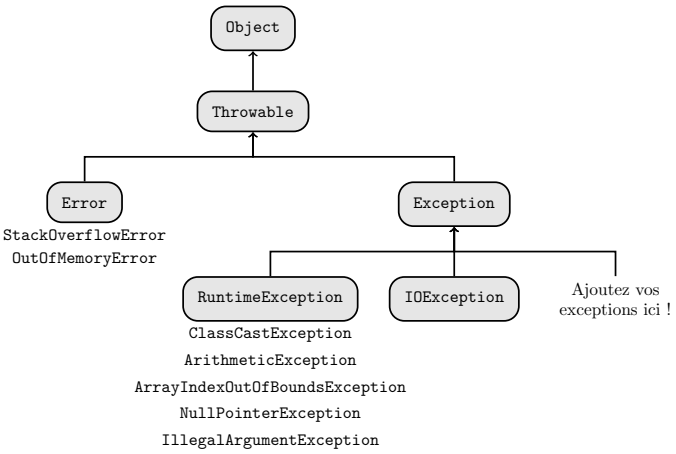


FIGURE 1 – Hiérarchie des exceptions en java

— `public Throwable()` crée une exception sans message associé.

— `public Throwable(String s)` crée une exception avec le message `s`.

Par les mécanismes d'héritage habituels, toutes les classes d'exceptions possèdent aux moins ces deux constructeurs.

Les exceptions appartenant à des sous-classes de `Error` ou de `RuntimeException` correspondent en général à des erreurs de programmation. Les `Error`s sont liées à l'environnement d'exécution (dont, par exemple, on aura saturé la mémoire avec une chaîne sans fin d'appels récursifs d'une méthode), tandis que les `RuntimeException`s sont déclenchées par des opérations illicites (mauvais transtypage, opérations sur un pointeur nul, accès en dehors d'un tableau, etc.). Ces exceptions peuvent survenir dans n'importe quel programme, et sont dites pour cela *hors contrôle* (*unchecked*).

Toutes les autres exceptions sont dites *sous contrôle* (*checked*). Les autres exceptions sont en général liées à une application ou une classe particulière, et correspondent à des appels de méthode avec de mauvais arguments ou dans un contexte inadapté. Par exemple : tentative d'ouverture d'un fichier inexistant, ou initialisation d'une classe avec des paramètres illégitimes. Java demande que toutes les exceptions sous contrôle pouvant être levées par une méthode donnée soient explicitement citées dans la signature de cette méthode avec le mot-clé `throws` (voir section suivante).

12.2 Déclarer une exception - Exercice

On considére le code suivant qui déclare une classe Horaire possédant deux attributs jour et heure, qui sont deux entiers dans des intervalles bien choisis (lundi sera représenté par 0 et dimanche par 6).

```
class Horaire  {
 private int jour, heure;
 public int getJour() { return jour; }
 public int getHeure() { return heure; }
 public Horaire(int j, int h) {
this.jour = j; this.heure = h;
```

```
} }
```

Telle que cette classe est écrite, il est possible de créer des horaires invalides, par exemple en initialisant jour avec un numéro négatif ou heure avec 25. On souhaite utiliser les exception pour réperer la création d' horaire invalides.

Question 1 : Déclarer une exception HoraireInvalide qui étends IllegalArgumentException.

Question 2 : Est-on obligé de définir les constructeurs de l'exception HoraireInvalide ?

12.3 Générer une exception : Cours

Pour générer une exception, il suffit d'utiliser le mot clé throw, suivi d'un objet dont la classe dérive de Throwable.

12.4 Générer une exception :TD

Question 3 Modifier la classe Horaire pour que cette exception soit utilisée pour réperer la création d' horaire invalides. On pourra associer un message à l'exception pour préciser si le problème vient du jour ou de l'heure.

Question 4 A présent que il peut lever l'exception HoraireInvalide, le constructeur de Horaire doit il le signaler dans son en-tête ?

On demande de plus que les horaires puissent être comparés entre eux via une méthode de signature public int compareTo(Horaire hor), telle que x.compareTo(y) renvoie : — Un nombre négatif si x est avant y. — Zéro si x et y sont égaux. — Un nombre positif si x est après y.

Question 5 Modifier l'entête de la classe Horaire pour spécifier que les horaires sont comparables.

Question 6 implémenter compareTo

12.5 Exception sous controle, héritage : TD

La classe Creneau représente un créneau horaire par un Horaire de début et un Horaire de fin.

```
class Creneau {
 private Horaire debut, fin;
 public Horaire getDebut() { return debut; }
 public Horaire getFin()   { return fin; }
 public Creneau(Horaire debut, Horaire fin)
{this.debut = debut; this.fin = fin; }
 }
```

Question 7 Ecrire un deuxiéme constructeur de creneau qui prends en argument un horaire et une durée en heures et qui utilise la méthode ajouteDurée qui peut ajouter des heures a un horaire, et dont voici le code :

```
public Horaire ajouteDuree(int duree)
{ int heure2finJour = 24 - this.heure;
  int d = duree - heure2finJour;
  if (duree < heure2finJour)
      return new Horaire(this.jour,
             this.heure + duree);
  else return new Horaire(this.jour+d/24, d%24);}
```

Question 8 On part à la chasse des erreurs possible sur les creneaux. Définir une nouvelle classe d'exceptions CréneauInvalide. Attention, cette fois ci, on souhaite utiliser des exceptions sous-controle (checked).

On souhaite distinguer les erreurs de creation de créneau intitulées “CréneauVide” qui identifie le cas ou l'horaire de fin n'est pas strictement après l'horaire de début. et celles intitulée “Dépassement” utilisé pour le second constructeur, si la durée est trop grande causant un créneau qui s'étend au-delà de dimanche 23h59.

Question 9 Definir les exceptions correspondantes : CreneauVide et Depassement

Question 10 Lever l'exception CreneauVide dans le premier constructeur lorsque c'est nécessaire.

12.6 Rattraper les exceptions :Cours

Flot de contrôle

Lors de l'exécution d'un programme, chaque instruction est visitée zéro, une ou plusieurs fois, selon un ordre qui dépend du *flot de contrôle* de ce programme. Quelques règles sont :

- Si deux instructions sont en séquence, séparées par `;`, on passe de l'une à l'autre.
- Lors d'un appel de méthode, on *saute* au début du code de cette méthode.
- À la fin de l'exécution d'une méthode, on revient au point où la méthode avait été appelée.
- En cas de branchement (`if`), on saute au début du code de la branche choisie.
- À la fin d'un tour de boucle, on revient au test du début de la boucle.

```
1   public static void main(String[] args) {
2     int k   = 4;
3     int res = f(k);
4     int k   = 2;
5     int res = f(k);
6     System.out.println(res);
7   }
8   public static int g(int a, int b) {
9     int res = a / b;
10    return res;
11  }
12  public static int f(int n) {
13    int res = g(n, n-2);
14    return res;
15  }
```

Dans cet exemple, l'exécution passe par la séquence de lignes `2-3-13-9-10-14-4-5-13-9`. Le processus s'arrête ensuite à cet endroit car une erreur va survenir à la ligne `9` (division par zéro) et interrompre l'exécution du programme.

Pile d'appels

À chaque étape de la séquence précédente, nous sommes en train d'exécuter l'appel à une certaine méthode, qui peut lui même avoir été déclenché par un autre appel de méthode. La *pile d'appels* table 1 enregistre ces informations sur le contexte de l'exécution : elle énumère l'ensemble des appels

de méthode dont l'exécution est en cours, en plaçant le plus récent (ou le plus local) au sommet et le plus ancien (ou le plus global) au fond.

Au moment où l'erreur de division par zéro survient dans notre programme, la machine virtuelle Java nous indique donc une erreur (`java.lang.ArithmeticException: / by zero`), et précise que l'erreur est apparue lors de l'exécution de la ligne `9` pendant l'appel de la méthode `g`, appel déclenché à la ligne `13` lors de l'exécution d'un appel de la méthode `f`, lui-même déclenché à la ligne `5` lors de l'exécution de la méthode `main`.

```
Exception in thread "main"
 java.lang.ArithmeticException: / by zero
        at Exemple.g(Exemple.java:9)
        at Exemple.f(Exemple.java:13)
        at Exemple.main(Exemple.java:5)
```

Rattraper les exceptions

On peut insérer dans le code des mécanismes de rattrapage, qui indiquent le comprtement à adopter en cas d'erreur (au lieu de simplement arrêter l'exécution d'un programme). Ce mécanisme prend la forme d'une instruction `try`, couplée à une ou plusieurs instructions `catch`.

- `try { ... }` introduit un bloc de code qui peut produire des erreurs.
- `catch (E e) { ... }` introduit un bloc de code à exécuter en cas d'occurrence d'une erreur de type `E` dans le bloc `try`.

```
 1   public static void main(String[] args) {
 2     int res;
 3     try {
 4       res = g(2, 0);
 5       System.out.println("Pas d'erreur !");
 6     }
 7     catch (java.lang.ClassCastException e) {
 8       res = -1;
 9     }
10     catch (java.lang.ArithmeticException e) {
11       res = 0;
12     }
13     System.out.println(res);
14   }
15   public static int g(int a, int b) {
16     int res = a / b;
17     return res;
18   }
```

La séquence des lignes visitées est maintenant `2-4-16-11-13`. Comme précédemment, l'instruction `return` après l'erreur (ligne `17`) n'est pas exécutée, et la ligne suivant l'appel de méthode erroné (ligne `5`) non plus. En revanche, l'erreur est advenue à l'intérieur d'un bloc `try`, et peut donc être rattrapée dans l'un des deux blocs `catch` associés. Le premier est ignoré, car l'erreur n'est pas un problème de transtypage (`java.lang.ClassCastException`), et la ligne `8` n'est donc pas visité. Au contraire, la ligne `catch (java.lang.ArithmeticException e)` intercepte l'erreur de division, et l'exécution reprend donc au point `11`. Après l'exécution sans erreur du bloc `try` ou, en cas d'erreur rattrapée, après l'exécution sans erreur du bloc `catch` correspondant, l'exécution reprend normalement au niveau du code qui suit les blocs `catch` (ici à la ligne `13`) et se poursuit jusqu'à la fin du programme. Si l'erreur n'avait pas été rattrapée par l'un des blocs `catch`, alors comme dans la partie précédente l'exécution aurait été simplement interrompue (le cas échéant, après vérification des autres paires `try/catch` présentes dans la pile d'appels).

12.7 Rattraper les exceptions : TD

Question 11 Le deuxiéme constructeur peut lever Depassement, mais. . .dans quelle méthode au juste, est il en fait naturel de lever cette exception ? levez la dans la dite méthode !.

Question 12 Au final, qu'est ce qui change dans le deuxiéme constructeur du fait que les exceptions son levées ?

On veut ajouter une méthode statique `static Créneau creneauSafe(Horaire, Horaire)` à la classe Créneau. Elle doit créer un nouveau créneau à coup sûr, en :

1. remettant les horaires de début et de fin dans le bon ordre s'ils ont été fournis en ordre inversé.
2. définissant la durée du creneau à 1 heure si égalité.

Question 13 Programmer creneauSafe. Vous devrez rattraper l'exception CréneauVide. Pouvez-vous éviter la déclaration d'exceptions dans l'en tête ?

12.8 Exception = outil de programmation.

Un emploi du temps (classe EDT) sera caractérisé par une liste dynamique de créneaux (`(ArrayList<Creneau>`).

```
public class EdT {
 private ArrayList<Creneau> liste;
 public EdT()
   {liste = new ArrayList<Creneau>();}
 }
```

Pour ajouter ou enlever des creneaux, on utilisera les deux méthodes de List suivantes :

- add(int i, Creneau c) ajoute un créneau c à l'indice i dans une liste et décale les éléments suivants.
- remove(int i) retire l'élément d'une liste à l'indice i et décale les éléments suivants.

Dans le cas ou le créneau ajouté recouvre des créneaux déjà présents, on prévoit de lever une exception Conflit, qui contient comme attributs le créneau que l'on a tenté d'ajouter, ainsi que l'indice du premier créneau de la liste qui est en conflit avec lui.

Question 14 Dans un premier temps, déclarer la classe de l'exception Conflit avec ses attributs et son constructeur.

Les créneaux d'un EdT sont rangés par ordre chronologique suivant les indices croissants. Pour ajouter au bon endroit un creneau dans l'EdT, on utilisera les méthodes suivantes définies dans la classe créneau :

2	3	13	9	10	14	4	5	13	9
			g :9	g :10					g :9
		f :13	f :13	f :13	f :14			f :13	f :13
m :2	m :3	m :3	m :3	m :3	m :3	m :4	m :5	m :5	m :5

TABLE 1 – pile d'appels

```
 public boolean precede(Creneau c) {
return this.fin.compareTo(c.debut) < 0;
 }
 public boolean suit(Creneau c) {
return this.debut.compareTo(c.fin) > 0;
 }
```

Question 15 Programmer ajouteCréneau. On utilisera la ruse suivante pour simplifier le code et mettre à profit nos toutes nouvelles connaissance des exceptions : Dans la boucle qui cherche l'endroit ou insérer le nouveau créneau on ne considérera pas la sortiedu tableau. Ensuite, l'erreur IndexOutOfBoundsException sera rattrappée dans un deuxième temps.

Il s'agit maintenant de résoudre le conflit potentiel crée par un ajout de créneau c à l'indice i. On utilise pour cela des CrenauCours qui héritent de Crenau, et qui utilise un nouvel attribut entier "priorite", et la méthode

```
public boolean estPrioritaire(CreneauCours c) {
  return (this.priorite > c.priorite);}
```

La stratégie suivante permet garder les créneaux de plus grande priorité :

1. Regarder parmi les créneaux à partir de l'indice i s'il existe un créneau c' qui à la fois a une priorité plus élévée que c et est en conflit avec c.
2. Si on a trouvé un tel c', abandonner l'ajout de c. Sinon, enlever tous les créneaux en conflit avec c et ajouter c à la place.

Question 16 Dans la classe EdT, programmer la méthode ajouteCreneauPriorite implementant cette stratégie.

13 The Corrigés

TP1 N-reines.

Question 2 Il faut : créer le tableau de case, parcourir avec un boucle imbriquée toutes les coordonnées, pour 1- creer chaque case, et 2- l'ajouter avec this.ajouteElement qui vient de IG.grille, pour que la case s'affiche.

Question 3

```
  // Initialisation du tableau de cases
this.plateau = new Case[taille][taille];
for(int i=0; i<taille; i++) {
for (int j = 0; j < taille; j++) {
// Création d'une case du plateau
Case cas = new Case(this);
plateau[i][j] = cas;
// chaque case doit être ajoutée
// à l'affichage graphique.
// La méthode vient de IG.Grille.
this.ajouteElement(cas); } }
```

Question 4 Le ajouteElement se fait dans la grille pour les cases, c'est un element graphique configuré précisément pour placer les element ajouter suivant une grille 2D, de gauche a droite puis de haut en bas, au fur et a mesure qu'on les ajoute. C'est l'instruction `setLayout(new GridLayout(hauteur, largeur, 5, 5));` du constructeur de grille, qui produit cet effet, on peut le vérifier en le mettant en commentaire, car alors, l'échiquier se transforme en ligne de cases.

En revanche, les boutons indice et valider eux, sont ajouté dans la fenetre principale.

Question 5 il y a deux états qu'on représente par un attribut booleen occupé, vrai si une dame occupe la case. On l'initialise a faux. C'est plus propre de déclarer l'attribut privé, auquel cas, il faut une méthode estOCcupé, qui renvoie ce booleen.

```
class Case extends ZoneCliquable {
    // Attributs
    private boolean occupee= false;
  // Pour permettre à un objet [Plateau]
  // de consulter l'état d'une case.
    public boolean estOccupee() {
    return this.occupee;}

   Question 6

class Case extends ZoneCliquable {
// Action à effectuer lors d'un clic gauche.
// Ceci utilise [IG.ZoneCliquable].
    public void clicGauche() {
this.occupee = !this.occupee;
if (this.occupee) {
    setBackground(Color.BLACK);
} else {
    setBackground(Color.WHITE);
}
    }
}
```

Question 7 Dans la classe plateau, car celle ci a accés au tableau de cases
Question 8 On compte pour chaque diagonale, le nombre de reines présente, puis on vérifie que il y en a au plus une, idem pour les antidiagonales.

Question 9

```
 public boolean verifieConfiguration() {
return (verifieDiagonales()
&& verifieAntidiagonales());
    }
```

Question 10 Il faut juste comme pour les case, changer la couleur de fonds.

```
class Validation extends ZoneCliquable {
     public void clicGauche() {
if (plateau.verifieConfiguration()) {
    setBackground(Color.GREEN);
} else {
    setBackground(Color.RED);
}
    }
}
```

Question 11 On peut zapper les colonnes pour aller plus vite, parceque cela ressemble pas mal au lignes.

```
 private int compteLigne(Case[] l) {
    // La variable [nb] est un accumulateur,
    // qui est incrémenté à chaque
    // nouvelle reine trouvée.
int nb = 0;
for (Case cas : l) {
    if (cas.estOccupee()) { nb++; }
}
return nb;
    }
    // La méthode [verifieLignes]
    //vérifie qu'aucune ligne ne contient deux
    // reines ou plus.
    private boolean verifieLignes() {
 // Stratégie avec accumulateur :
 //une variable booléenne [ok] qui est
```

```
 // mise à jour après analyse de chaque ligne,
 // et est renvoyée à la fin.
boolean ok = true;
for (Case[] l : this.plateau) {
    ok = ok && (compteLigne(l) < 2);
}
return ok;
    }

 // Vérification des colonnes (avec un [for] ).
 // La méthode [compteColonne] prends en
 // paramètre l'indice d'une colonne de
 // l'échiquier et renvoie le nombre de reines
 // présentes sur cette colonne.
    private int compteColonne(int j) {
 // La variable [nb] est  un accumulateur.
int nb = 0;
for (int i=0; i<this.taille; i++) {
  if (plateau[i][j].estOccupee()) { nb++; }
}
return nb;
    }
 // La méthode [verifieColonnes] vérifie que
 // aucune  colonne ne contient >  2 reines.
    private boolean verifieColonnes() {
 // Stratégie avec court-circuit :
 // on renvoie le résultat définitif
  // [false] dès qu'une colonne fautive
  // est trouvée. Sinon, si toutes
  // les colonnes ont été vérifiées
  // sans souci, on renvoie [true].
for (int j=0; j<this.taille; j++) {
    if (compteColonne(j) > 1)
       { return false; }
}
return true;    }
```

Question 12

```
  // Méthode de vérification générale.
    public boolean verifieConfiguration() {
return (verifieLignes() && verifieColonnes() &&
verifieDiagonales() && verifieAntidiagonales());
    }
```

Question 13 On cherche la premiére ligne non occupée, et on essaie les cases de cette lignes les unes aprés les autres.
Question 14

```
   private int prochaineLigne() {
int k=0;
while (k<this.taille &&
   compteLigne(this.plateau[k]) != 0)
    { k++; }
return k;
    }
```

Question 15 Il renvoi vrai si l'appel a prochaine ligne renvoie taille. Il renvoie faux, si la configuration actuelle n'est pas valide, ou Si, aprés avoir tester toutes les cases de la prochaineLigne, aucun appel récursif ne renvoie vrai.
Question 17 Non, cela serait lourd pour deux raisons : 1- Cela prendrai de la place sur la pile d'appel, 2- Java utilise le passage par réference, de base. Passer par valeur obligerait a construire un nouveau plateau a chaque appel récursif.
Question 18 : A chaque fois qu'on tente d'occuper une case, et que l'on fait un appel récursif, aprés le retour de cet appel récursif, il faut surtout ne pas oublier de libérer cette case, afin de récuperer le plateau dans l'état ou il se trouvait avant l'appel courant.
Question 19 :

```
  public boolean verifieResolubilite() {
// Si la configuration n'est pas valide,
//on retournera [false].
if (!verifieConfiguration()) return false
else {
// Si la configuration est valide,
// on regarde la prochaine  ligne non occupée.
//  Si toutes les lignes sont occupées
// alors une solution a été trouvée
// et on retourne [true].
    int l = prochaineLigne();
    if (l==taille) {
return true;
    } else {
  // On explore toutes les cases de la ligne [l]
  // considérée, jusqu'à atteindre le bout
  // de la ligne ou  que le booléen
  // [solutionTrouvee] passe  à [true].
boolean solutionTrouvee = false;
for (int c=0; c<this.taille
          && !solutionTrouvee; c++) {
    plateau[l][c].occupe();
    solutionTrouvee = verifieResolubilite();
    plateau[l][c].libere();
}
return solutionTrouvee;
    }
}  }
```

Question 20

```
 public void clicGauche() {
//Test de résolubilité de la config  actuelle.
        if (plateau.verifieResolubilite()) {
    setBackground(Color.GREEN);
   else setBackground(Color.RED);
    }
```

Question 21 A priori le premier.
Question 22 Ces deux indices sont stoqués comme des attributs de plateau, et mis a jour a la fin de VerifieRésolubilité, de sorte que c'est bien le premier appel de VerifieRésolubilité qui mettra a jour en dernier.

```
     // Variables locales destinées à
     // recevoir les coordonnées de
    // l'éventuelle proposition de coup.
    private int indiceL, indiceC;
    public int getIndiceL() { return indiceL; }
    public int getIndiceC() { return indiceC; }
```

Question 23

```
  for (int c=0; c<this.taille
         && !solutionTrouvee; c++) {
 plateau[l][c].occupe();
 solutionTrouvee = verifieResolubilite();
 plateau[l][c].libere();
 indiceC = c;
}
indiceL = l;
```

Question 24

```
    public void clicGauche() {
 if (plateau.verifieResolubilite()) {
  setBackground(Color.GREEN);
  int indiceL = plateau.getIndiceL();
  int indiceC = plateau.getIndiceC();
  Case cas = plateau.getCase(indiceL, indiceC);
  cas.setBackground(Color.BLUE);
} else { setBackground(Color.RED); }
    }
```

TP2 itérateurs

Question 1 : Il utilise la classe ArrayList de String, qui implemente déjà l'interface List, ajoute 4 String et les affiche, avec la méthode get Question 2 : la classe implémente l'interface List2<Object>
Question 3 : oui car objet est une classe ancetre de chaine, une chaine est un objet
Question 4 : renvoyer l'element i du tableau. On suppose que elle n'a pas besoin de vérifier les accès hors limites.
Question 5 :

```
  public Object get(int i) {
        return this.elements[i];
    }
```

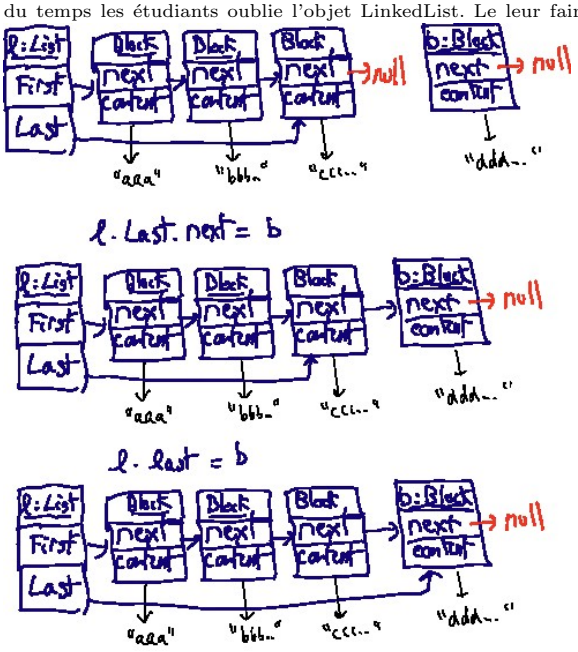
L'usage de this est facultatif.
Question 6 : Vérifier si il y a la place. Ranger la chaine dans la prochaine case libre du tableau dont l'index est size, et augmenter size.
Question 7 :

```
  public void add(Object elt) {
        if (this.size < this.capacity)
           this.elements[this.size++] = elt;
    }
```

Question-Test : Dans le programme principal, au lieu d'une arrayList on instancie dans l une FixedCapacityList. Ca doit faire le même effet que avant.

```
FixedCapacityList l = new FixedCapacityList(10);
```

Question 8 faut tester si la liste est vide ou non.
Question 9 Creer un bloc est dire que c'est le nouveau dernier
Question 10 Si la liste est vide faut dire que c'est le premier, sinon faut dire que c'est le suivant de l'ancien dernier Question 10 bis La pluspart du temps les étudiants oublie l'objet LinkedList. Le leur faire remarquer.



Question 11

```
public void add(T elt) {
     // On crée un nouveau bloc [b]
     Block<T> b = new Block<T>(elt);
     // Si la liste est vide
     // [b] devient le premier bloc,
     // sinon [b] devient le successeur
     //du dernier bloc courant
     if (this.firstBlock == null) {
         this.firstBlock = b;
     } else {
         this.lastBlock.nextBlock = b;
     }
     // Dans tous les cas,
     // [b] devient le dernier bloc
     this.lastBlock = b;
 }
```

Question 12 Faut dessiner 3 bloc de deux cases, la premiére contient un pointeur vers une chaine la deuxiéme un pointeur ver le bloc suivant. Les premiers est dernier block sont pointés depuis la liste.
Question 13 Il faut suivre les liens depuis le premier élements, en remontant deux fois.
Question 14 Il faut suivre les liens depuis le premier élements, en remontant i fois.
Question 15

```
public T get(int i) {
     // On mémorise un bloc courant,
     Block<T> currentBlock = this.firstBlock;
     // on avance [i] fois,
     for (int j = 0; j < i; j++)
        currentBlock = currentBlock.nextBlock;
     // et on renvoie la valeur trouvée.
     return currentBlock.contents;
 }
```

Question-Test : Dans le programme principal, au lieu d'une arrayList on instancie une MyLinkedList

```
MyLinkedList l = new MyLinkedList();
```

Question 16 On n'a pas besoin de savoir à l'avance combien d'élements il y aura au maximum.
Question 17 En creant un nouveau block on alloue de la mémoire au fur et a mesure que on en a besoin.
Question 18 Une structure de donnée dynamique.
Question 19 l.iterator()
Question 20 Dans le main faut déclarer l a nouveau comme une FixedCapacityList et mettre

```
printIterator(l.iterator());
```

Question 21 Faut utiliser un objet auxiliaire pour stoquer next, qu'on pourra afficher deux fois
Question 22

```
public static void printIterator2(
            Iterator<Object> i) {
    while (i.hasNext()) {
        Object o = i.next();
        System.out.println(o);
        System.out.println(o);
    }
}
```

Question 23 Faut faire deux next, mais tester qu'il y a bien un next avant de faire le deuxieme.
Question 24

```
public static void
 printIterator3(Iterator<Object> i) {
    while (i.hasNext()) {
        System.out.println(i.next());
        if (i.hasNext()) i.next();
    }
}
```

Question 25 L'indice est decrémenté au lieu d'être incrementé, et faut donc initialiser à la taille de la liste, y a un hasNext si on n'est pas à zéro.
Question 26

```
/** Itérateur descendant. */
class DescendingIterator<T> implements Iterator<T> {
    // Similaire au précédente,
    //mais la position initiale est la dernière de
    // la liste, et chaque appel
    // à [next] décrémente la position.
    private List2<T> list;
    private int currentIndex;

    public DescendingIterator(List2<T> l) {
        this.list = l;
        this.currentIndex = l.size();
    }

    public boolean hasNext() {
        return this.currentIndex > 0;
    }

    public T next() {
        return this.list.get(--currentIndex);
    }
}
```

Question-test : Pour tester, il faut changer la méthode iterator() de FixedCapacityList pour que elle renvoie un descendingIterator
Question foreach : `for(Object s: l ) System.out.println(s);`
Question 27 Chaque get i coute un nombre fixe d' opérations, en tout c'est de l'ordre de $O(n)$
Question 28 Chaque get i coute i opérations, la somme des n premiers entier vaut $n*(n+1)/2$ c'est de l'ordre de $O(n^2)$
Question 29 Il faut programmer un itérateur qui stoque un block au lieu d'un indice
Question 30

```
/** Itérateur de liste chaînée */
class LinkedListIterator<T>
   implements Iterator<T> {
    // On mémorise uniquement le bloc courant,
    // les champs [nextBlock]
    // suffiront à atteindre les autres
    private Block<T> currentBlock;

    // À la construction, on fournit un bloc
    // (a priori le premier)
    public LinkedListIterator(Block<T> block) {
        this.currentBlock = block;
    }

    // Il y a un prochain élément
```

```
    // tant qu'il y a un bloc
    public boolean hasNext() {
        return currentBlock != null;
    }

    // Extrait l'élément du bloc courant,
    // puis change le bloc courant. Si le
    // bloc courant était le dernier,
    // [currentBlock] prendra la valeur [null].
    public T next() {
        T elt = currentBlock.contents;
        this.currentBlock =
            this.currentBlock.nextBlock;
        return elt;
    }
}
```

Question-Test Faut construire une liste de grande taille en utilisant une boucle puis iterer dessus sans rien faire (pas d'affichage !). 30.000 ca devrait ramer avec la solution lente, et être immédiat avec la solution efficace

Question Philosophie : La connaissance plus détaillé de l'implémentation d'une collection permet de définir des méthode plus performantes.

TD1 Cas d'utilisation.

Question 1 Le premier s'appelle les cas d'utilisations, et le deuxième un scenario

Question 2 Acteurs humains : Visiteur anonyme, client connecté. La même personne peut tenir les deux rôles, la bascule se faisant via des opérations de connexion/déconnexion pas présentées dans les diagrammes.

Autre acteur : tour de contrôle. On n'en sait pas assez pour dire s'il est humain ou non, on peut discuter des situations concrètes qui correspondraient à l'un ou à l'autre.

Question 3 c'est un "extends" comme pour les classes il se note pareil, un client connectés étends un client anonyme, ou bien encore est un cas particulier de client anonyme, il peut donc faire les trucs que fait l'anonyme plus d'autre trucs. Attention, on s'est trompé dans le sens de la flèche elle doit aller de reserver vers consulter.

question 4 Reserver est une suite logique de consulter, logique mais pas obligatoire, car consulter tout seul est aussi possible, par contre, on peut pas réserver si on n'a pas consulter avant.

Question 5 Ce sont des cas particulier de gestion

Question 6 ce sont des messages "réponses"

Question 7

Recherche d'un vol
Nom : Consulter
Description : Identifie un vol en fonction d'un point de départ et d'un point d'arrivée
Acteur : visiteur anonyme ou client connecté
Entrées/préconditions : deux noms de villes valides
Résultat/postconditions : si un vol existe, son numéro (le cas contraire n'apparaît pas, on peut supposer par exemple une réponse "pas de vol").
Procédure normale :
1. L'acteur saisit les noms des villes et demande une recherche
2. Le système renvoie le numéro de vol
Procédure alternative (pas montrées dans les diagrammes, on imagine)
2a. Le système indique qu'un tel vol n'existe pas
Autre alternative
2b. Le système indique que les villes n'existent pas

Réservation d'un vol
Nom : Réserver
Description : réserve un certain nombre de place à bord d'un vol
Acteur : client connecté
Entrées/préconditions : un numéro de vol et un nombre de places demandées
Résultat/postconditions : confirmation avec référence de réservation, ou message d'erreur si impossibilité.
Procédure normale :
1. Le client saisit le numéro de vol et le nombre de places demandées
2. Le système réserve les places et renvoie une référence de réservation
Alternative :
1a. Le client appelle le cas "consulter"
puis retour à 1.
Alternative :
2b. Le système indique que le numéro de vol est invalide
Alternative :
2c. Le système indique qu'il n'y a pas assez de places disponibles et ne fait aucune réservation.

Remarque : le schéma ne parle pas de date. On peut supposer par exemple que dans ce cas on s'intéresse à la prochaine occurrence du vol concerné.

Question 8 creer Compte, gérer les paiements, envoyer une relance, consulter disponibilité, reserver une salle, annuler une réservation, mettreAjourEtatSalle

Question 9 La flèche "extends" indique qu'une "consultation" peut contenir une "reservation", mais ce n'est pas systématique. une réservation elle ne peut exister sans consultation préalable.

Question 10 Pour la gestion des réservations faite par le gestionnaire (édition de factures et enregistrement de paiements), on évite de mentionner un par un tous les cas qui ne sont que différentes facette d'une même activité. Dans ce cas je préfère donc un cas d'utilisation plus général "gestion". Pour faire apparaître tout de même ces cas particuliers, voir commentaire suivant.

On peut utiliser dans les cas d'utilisation la flèche de généralisation. Comme ailleurs, cela signifie qu'un cas est plus général qu'un autre. Ici, ce serait le meilleur moyen de faire apparaître les cas spécifiques "édition de facture" et "enregistrement de paiment" : comme spécialisations du cas plus général "gestion".

De la même façon, les cas "réservation" et "annulation" pourraient n'apparaître que comme deux spécialisations d'un même cas "gérer une réservation" de l'usager. Seule chose à laquelle faire attention : l'extension "consultation" ne concerne que l'un des deux cas, fait qu'il vaut mieux laisser visible même si les deux cas sont indiqués comme des spécialisations d'un cas plus général.

Question 11 Lorsqu'on est en présence de deux acteurs pouvant tous les deux déclencher un cas d'utilisation, comme les usagers et les agents municipaux pour la consultation du planning, il faut introduire un rôle supplémentaire regroupant les deux autres (les "connectés"), qui devient le seul acteur principal du cas d'utilisation consultation.

On a donc trois acteurs : Usager, Agent, connectés qui généralise usager et agent. La flèche de généralisation, comme dans les diagrammes de classe, signifie qu'un acteur en étend/spécialise un autre. En plus des siens propres, l'acteur spécialisé est considéré comme connecté à tous les cas d'utilisation de l'acteur général.

Le gestionnaire représente un quatriéme role, car il a des droit supplémentaire pour voir les factures.

Dans le cas d'une action périodique comme l'envoi de relances, on peut ajouter un acteur extérieur représentant une horloge. Il ne s'agit pas nécessairement d'un élément extérieur au système (et donc pas d'un acteur au sens strict), mais c'est une manière répandue de représenter ce déclenchement périodique

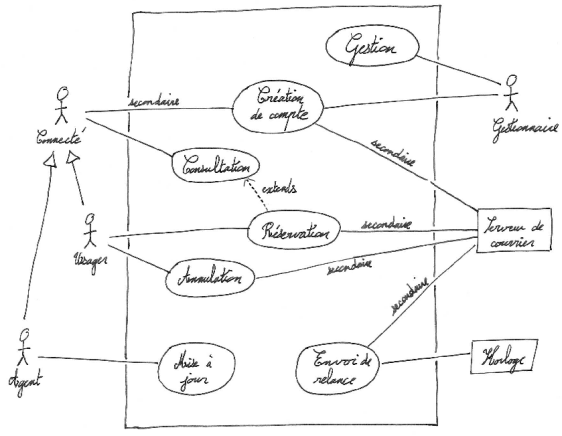
Question 12 Quand on connecte plusieurs acteurs à un même cas d'utilisation, il est sous-entendu qu'ils y participent tous.

Quand un usager (personne physique) demande au gestionnaire (personne physique) d'enregistrer pour lui une réservation, il ne s'agit pas d'ajouter un lien entre le gestionnaire (acteur logique) et le cas d'utilisation "réservation" : dans cette situation le gestionnaire (personne physique) se connecte en utilisant un compte d'usager générique et endosse le rôle d'usager (acteur logique) qui est naturellement relié au cas "réservation".

Question 13 1-l'evenement "usager qui récoit un email de relance", 2- la demande de création de compte au gestionnaire

Quand un acteur est concerné par un cas d'utilisation mais n'y participe pas vraiment, comme l'usager à qui est envoyé un courrier de relance, on peut se demander si l'acteur doit être indiqué comme acteur secondaire. Mon choix est ici de ne pas indiquer ce lien, car il n'y a aucune interaction entre le système et l'usager : le système rédige un courrier et le transmet au serveur de courrier, qui se charge ensuite de la transmission (qui a lieu en dehors de et indépendamment du système).

De même, pour la création de compte : l'usager (ou l'agent) concerné fait sa demande au gestionnaire de vive voix ou par un autre moyen indépendant du système. C'est ensuite le gestionnaire qui initie l'action au niveau du système. L'usager apparaît ensuite comme acteur secondaire en raison de l'action de validation (changement de mot de passe) qui lui est demandée à la fin de la procédure.



Question 14

.

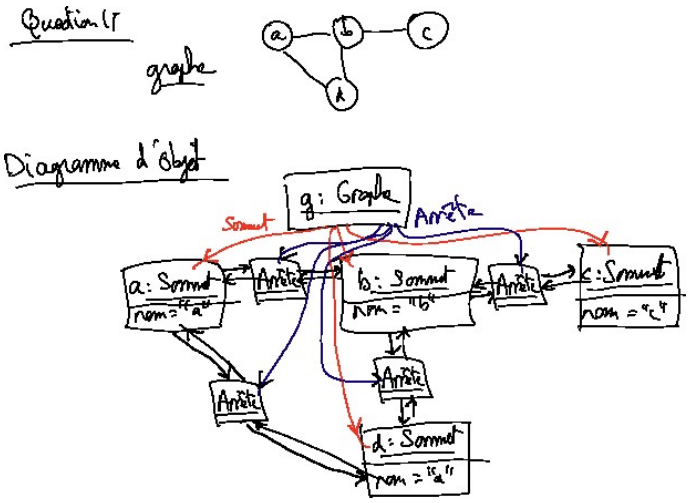
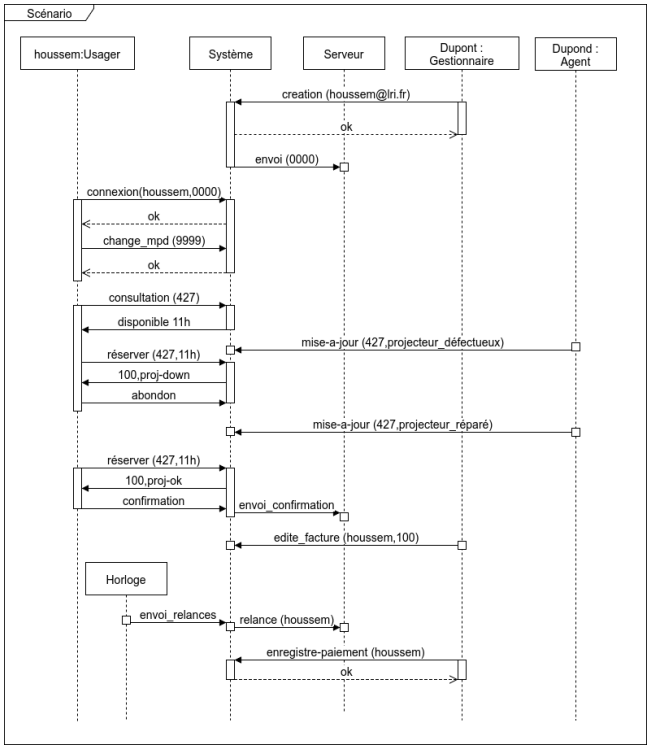


FIGURE 2 – Diagramme d'objets



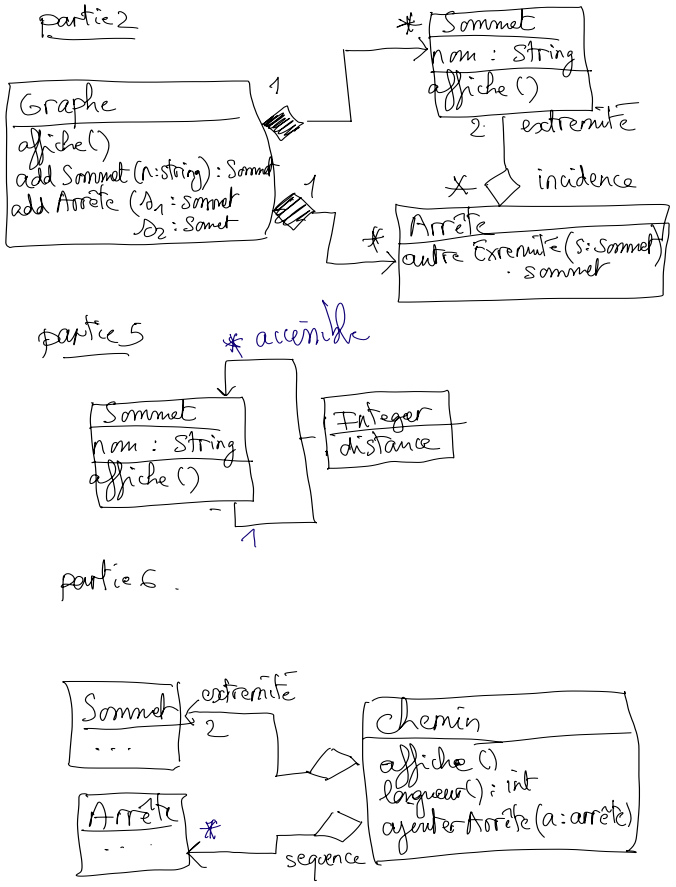


FIGURE 3 – Diagramme de classe

TP3 Graphe

question 0 Ce sont des contraintes non fonctionnelles, elles ne porte pas sur un output, mais sur une facon de générer cet output Question 1 On calculera les info a fournir : couleurs, distance .. au moment de la construction du graphe, i.e. en même temps que l'on ajoute des noeuds et des arretes

Question 2 : On calcule tout au fur et a mesure de la construction, puisque on a la place pour le mémoriser. Pour chaque noeuds faudra stoquer la distance vers tout les autres noeuds accessibles, ca reste proportionnel à la taille du graphe.

Question 3 Les diagramme UML se trouvent figure 3. Attention, les attributs représentés en UML sont seulement ceux qui ont un type simple, string ou int, les attributs qui sont des classes seront représenté par des associations. Y a un seul attribut nom :string dans Sommet pour identifier les sommets. Les deux opérations nécessaires pour contruire, sont ajouterSommet(nom :String) :Sommet, et ajouterArrete(s1,s2 : Sommet). AjouterSommet doit renvoyer un sommet pour qu'on puisse ensuite s'en servir pour ajouter des arrêtes. Affiche est défini pour graphe et sommet, y a rien à afficher pour une arrête. J'ai souvent la question, est ce que un sommet est une composition forte de ses arrêtes, la réponse est non, parceque une arrête est à cheval sur deux sommets, on ne peut donc pas considérer que un seul sommet contient ses arrêtes.

Question 4 Un graphe a plusieurs sommets et arrêtes. Un sommet à plusieurs arrêtes, une arrête a deux sommets. Dans ce TP, On n'est pas obligé d'accéder au graphe depuis une arrête ou un sommet, donc dans le corrigé, on oriente les assocations sortant du graphe vers arrêtes et sommet. Mais on peut aussi partir du principe que un noeud ou une arrête doit appartenir à un graphe et mettre l'association bidirectionnelle. .

Question 5 Pour l'association neouds-sommet, les arrêtes d'un sommet on peut les appeler "incidentes", les deux sommets d'une arrête "extremité"

Question 6 Lorsque le graphe est détruit, les arrêtes et sommets qui le composent aussi, on a donc une relation de composition. Une arrête peut être vue comme l'agregation de deux sommets, mais ceux ci ne disparaissent pas si l'arrête est détruite. On ne peut pas dire que un sommet est une composition d'arrêtes, car une arrête est partagée par deux sommets

Question 7 Faut une méthode autreExtremité dans la classe arrête, qui prends un sommet et qui renvoie celle des deux extremitée qui n'est pas celle la. Pour récupérer les voisins depuis un sommet, on commencera par récupérer les arrêtes incidentes, puis via cette méthode, les sommets de l'autre cote

Question 8 les '*' scorrespondent a des attributs de type Set. Set est une interface. Pour les extrémités on utilise un tableau de taille 2.

Question 9 Pour construire l'ensemble vide initial, on utilise la classe HashSet qui implement les Set efficacement, avec une table de hashage (on verra ca plus tard.

Question 10 Oui, si vous editez votre diagramme UML avec un outil pro, miracle, vous aller pouvoir générer les trois classes avec leurs attributs, juste en cliquants sur un bouton style "generateJava". De plus, vice versa, il pourra ausssi construire le diagramme de classe à partir d'un source java.

Question 11 Il affiche 4 sommets

Question 12 On veut voir les arrêtes, mais afficher juste les arrêtes ne convient pas, car il se peut que il y ait des sommets sans arrêtes. On veut donc afficher chaque sommets, puis après chaque sommets, la liste des sommets voisin. (c'est plus court que les arrêtes qui nécessitent deux sommets).

Question 13 La méthode affiche de sommets, doit iterer sur les voisins et afficher leur noms, va falloir une méthode afficheNom pour sommet.

Question 14

```
public void affiche() {
     System.out.print(nom   + ": [");
     for(Arete a:incidences)
        a.autreExtremite(this).afficheNom();
System.out.println("]");
}
```

Question 15 Voir figure 2

Question 16 Par un attribut supplémentaire de Sommet contenant un entier positif qui réprésente une couleur.

Question 17 On l'inialise à 0, faut garder les couleurs le plus petit possible pour minimiser le nombre de couleur totale, et comme y a pas encore de voisin, on peut prendre 0

Question 18 Si on ajoute une arrête entre deux sommets ayant la même couleur, faut recalculer la couleur d'un des deux sommets

Question 19 On parcours les couleurs depuis 0, et on choisit la premiére couleur non présente chez les voisins

Question 20 Faut ajouter deux méthodes à la classe Sommet. Sommets : :colorie sera appellé par le constructeur de arrêtes, sur l'une des extrémités

```
// Un sommet est bien coloré s'il n'a pas
// la même couleur que ses voisins.
   public boolean bienColore() {
for (Arete a : incidences) {
Sommet autre=a.autreExtremite(this);
    if (autre.couleur == this.couleur)
     return false;
}
return true;
    }

    // Affecte à un sommet la plus petite
    // couleur pas déjà prise par ses
    // voisins.
    public int colorie() {
this.couleur = 0;
while (!this.bienColore())
this.couleur++;
return this.couleur;
    }
```

Question 21 Faut tester ce code en affichant la couleur des sommets, zyva.

Question 22 Pour mémoriser les distances, un sommet maintient un ensemble de sommets atteignables, (association reflexive de sommet vers sommet) et pour chacun des atteignables, la distance du chemin le plus cours qui y méne. cela est représenté en UML par une "classe-association" : la classe Integer est reliée à l'association distance. Faut faire une association "distance" reflexive de sommet vers lui-même étiquetée par la classe Integer

Question 23 Faut associer des distances aux sommets accessibles `Map<Sommet, Integer>() distances;`

Question 24 La hashmap, c'est comme hashset, sauf que on associe des valeurs
`distances = new HashMap<Sommet, Integer>();`

Question 25 La distance a soi même vaut 0 ; `distances.put(this, 0);`

Question 26 L'ajout d'une arrête a entre s1 et s2.

Question 27 Les distances entre un accessibles s'_1 de s_1 et un accessible s'_2 de s_2, car un chemin de s'_1 vers s'_2 peut passer par a

Question 28 Y a le cas ou s'_1 n'etait pas relié a s'_2 et le cas ou il l'etait mais avec un chemin de taille plus grande.

Question 29 Ce sont les clefs de la map distances

Question 30

```
  Set<Sommet> accessibles() {
       return new HashSet<>(distances.keySet());
    }
```

Faut recréer le HashSet sinon lorsqu'on va modifier distance ca bugge a cause d'un java.util.ConcurrentModificationException, car on va itérer sur une map qu'on modifie en même temps. Vous pourrez ne pas le recréer pour observer ce bug qui est intéressant.

Question 31

```
//a rajouter Dans la création d'une arrête:
       s1.updateDistances(s2, this);
       s2.updateDistances(s1, this);

  void updateDistances(Sommet s, Arete a) {
       for (Sommet s1 : accessibles())
           for (Sommet s2 : s.accessibles()) {
             int newDist =  distances.get(s1)
                       + a.getPoids()
                       + s.distances.get(s2);
           if (!s1.distances.containsKey(s2))
              s1.distances.put(s2, newDist);
           else if (s1.distances.get(s2) >
                   newDist)
            s1.distances.replace(s2, newDist);
                           }
    }
```

Question 32 Vous modifier afficher de sommet pour afficher aussi les distances

Question 33 Un chemin est composé d'une sequence d'arrêtes. Comme un chemin est orienté, on ajoute ausssi les deux sommets extrémités en distinguant source et destination. La sequence d'arrête démarre à la source et termine a la destination.

Question 34 On regarde lequel des voisins est le plus proche du sommet vers lequel on veut router, voir le squelette complété.

Question 35 Pour les graphes de degré borné, comme la grille 2D, le cout de contruction reste linéaire en la taille du chemin qui est <= que la taille du graphe. Pour le graphe complet il devient quadratique.

TP4, circuits.

Question 1 deux attributs source1 et source2 qui pointent vers les deux sources en question. La méthode valeur() doit renvoyer la somme des valeurs calculées par les deux sources.

Question2 En les passant dans le constructeur qui aura donc pour signature Addition(Noeud n1, Noeud n2) .

Question 3

```
class Addition extends Noeud  {
    private Noeud source1, source2;
    public Addition(Noeud e1, Noeud e2)
    { source1=e1 ;source2=e2 }
    public int valeur() {
        return (source1.valeur()
        + source2.valeur());
    }
}
```

Question 4 Faut une méthode creeAddition dans circuit, qui encapsule la création et rajoute le noeuds, comme pour creeEntree

Question 5 on construit le circuit qui calcule $x- > x+1$, en décommentant la ligne suivante.

Question 6 Un attribut qui stocque cette constante

Question 7 En les passant dans le constructeur qui aura donc pour signature Constante(Int c) .

Question 8

```
class Constante extends Noeud {
    private int cst;
    public Constante(int c) { this.cst = c; }
    public int valeur() { return this.cst; }
}
```

Question 4 Faut une méthode creeConstante dans circuit, qui encapsule la création et rajoute le noeuds, comme pour les autres

Question 5 on construit le circuit qui calcule $x- > x + x$

Question 6 On copie colle la classe de addition et on remplace + par − puis une autre fois par $*$, on copie colle creerAddition et on adapte

Question 8 Voir figure 4

Question 9 Elle utilisent toute la même paire d'attributs source1, source2, on déclare cette paire dans une classe "NoeudBinaire" intermediaire dont vont hériter les trois. Cette classe n'implémente pas valeur() donc elle est aussi abstraite. Si on veut garder les attribut private, alors faut aussi programmer valeurSource1() et valeurSource2() qui factorisent egalement l'appel a valeur sur les sources.

Question 10

```
abstract class NoeudBinaire extends Noeud {
    private Noeud source1, source2;
    public NoeudBinaire(Noeud s1, Noeud s2) {
this.source1 = s1;
        this.source2 = s2;
    }
    public int valeurSource1() {
return this.source1.valeur();
    }
    public int valeurSource2() {
return this.source2.valeur();
    }
}

class Addition extends NoeudBinaire {
    public Addition(Noeud e1, Noeud e2)
     { super(e1, e2); }
    public int valeur() {
        return (this.valeurSource1() +
           this.valeurSource2());
    }
}
```

Pour tester, on refait tourner le même programme.

Question 11

```
//classe entrée
    public String toString() {
        return "x";
    }
//classe constante
public String toString() {
        return String.valueOf(this.cst);
    }
```

Question 12 Dans la classe circuit :

```
 public void affiche() {
System.out.println(this.sortie);
    }
```

Question 12 bis On implémente toString au niveau de noeudBinaire, en déclarant une nouvelle méthode abstraite de signature : String opString() qui renvoie une chaine de caractères avec un seul caractère représentant un operateur, à insérerer entre les deux appels récursifs de toString sur chaque source.

Question 13 On met systematiquement des parenthéses.

```
   abstract public String opString();
    public String toString() {
        return "(" + this.source1 +
         this.opString() + this.source2 + ")";
    }
```

Question 14 On utilise une méthode estArithmetique dans Noeuds qui renvoie faux, et on la redéfini dans NoeudBinaire pour que elle renvoie vrai

Question 15 On parcourt les neouds et on incrémente un compteur à chaque fois qu'on trouve un neoud arithmetique.

Question 16

```
  public int nbNoeudsArith() {
        int nb=0;
        for (Noeud n : this.noeuds) {
            if (n.estArithmetique)) { nb++; }
        }
        return nb;  }
```

Question 17 Non, c'est simplement que la porte n5 est évaluée deux fois. Le diagramme est dans figure 4.

Question 18 On utilise un attribut compteur pour chaque porte, que seul les noeuds arithmetiques incrémentent, lorsque leur méthode valeur() est appelée.

Question 19 la méthode valeur appelle incrNbopp().

```
//dans NoeudbBinaire
 private int nbOp;
    public int nbOpEffectuees()
      { return this.nbOp; }
    public void incrNbOp() { this.nbOp++; }

  //dans circuit
  public int nbOpEffectuees() {
        int nb=0;
        for (Noeud n : this.noeuds) {
            nb += n.nbOpEffectuees();
        }
        return nb;
    }
```

Question 20 Oui, il suffirait de stoquer le résultat de n5 lors de sa premiére évalution, car la deuxième donne un résultat identique.

Question 21

```
class MultiplicationMemoisee
    extends Multiplication {
    // Par défaut, un nœud n'est pas déjà évalué
    //, et {mem} n'est pas défini.
    // Invariant des objets de cette classe :
    // si {dejaEvalue} vaut {true},
    // alors {mem} contient une valeur.
    private boolean dejaEvalue=false;
    private int mem;
    public MultiplicationMemoisee
             (Noeud e1, Noeud e2)
    { super(e1, e2); }
    // Redéfinition de {valeur()}
    public int valeur() {
        if (!this.dejaEvalue) {
   // Si pas déjà évalué, on appelle la méthode
   // {valeur()} de la classe mère {Multiplication}
   //  pour définir  l'attribut {mem}.
            this.mem = super.valeur();
            this.dejaEvalue = true;
        }
        // Dans tous les cas, à ce stade {mem}
        //  est défini on renvoie sa valeur.
        return this.mem;
    }
```

Question 23 Faut créer un méthode statique dans la classe Circuit.

```
 public static Circuit expRapide(int p) {
        // On crée d'abord un circuit vide...
        Circuit c = new Circuit();
        Noeud n=c.creeEntree();
        for(int i=0;i<p;i++)
            n=c.creeMultiplication(n,n);
        c.sortie=n;
        return c;
    }
 public static Circuit expRapideMemoise(int p) {
        Circuit c = new Circuit();
        Noeud n=c.creeEntree();
        for(int i=0;i<p;i++)
          n=c.creeMultiplicationMemoisee(n,n);
        c.sortie=n;
        return c;
    }
```

TP5 Tests avec Junit

Question 1 :Un circuit de course, les '.' sont la route, et les diéses représentent les bords de la route.

Question 2 :La voiture s'arrête dans la case juste avant le mur, elle ne peut donc pas aller sur une case de mur.

Question 3 Le trajet parcouru par la voiture durant la seconde précédente

Question 4 Un clic donne une accélération unitaire dans la direction de la souris, et donc modifie la vitesse seulement un peu, si la vitesse est trop grande, et le mur et pas loin, on peut plus corriger.

Question 5 Sur les méthodes de la classe Vect !

Question 6 Parceque les méthodes de Vect ne sont pas programmées.

Question 7 Elle sont super simple, à la lumiére des test : elle initialise,calcule l'addition (add) la soustration (sub) de vecteur, le test d'égalité (egale)

Question 8 Pour faire pro, et pour vous faire un peu peur.

Question 9 Ce sont de vecteurs qu'on réutilisera éventuellement pour vérifier les résultat d'opérations de vecteurs.

Question 10 Vous commencez par simplement lire les tests juste pour comprendre ce que doit faire chaque méthode. Après cela vous implementer

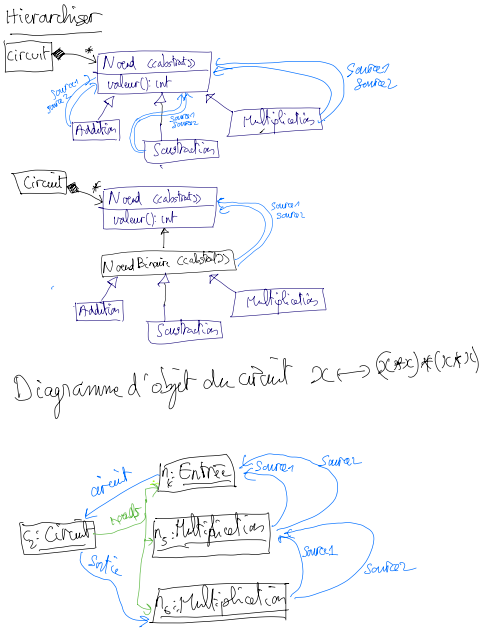


FIGURE 4 – Diagramme de classes et d'objet de circuits.

la méthode testée, et vous constater que les tests correspondant passent au vert. Une source d'erreur commune est de zapper l'instanciation qui initialiser les valeur de x et y a avec les paramètres, au lieu de zèro.

Question 11 "normalize" est la seule méthode réélement compliquée. C'est la normalisation d'un vecteur. On va même pas avoir à porgrammer, puisqu'elle l'est déja. Les tests sont quand même utile pour comprendre ce qu'elle fait : qui est de renvoyer le vecteur de norme infinie 1 dont l'orientation est la plus proche. Ils sont devenus vert parceque on a implementé egale et c'était cela qui coincait auparavant.

Question 12 : L'instanciation de bolide, qui doit enregistrer la position initiale, et une vitesse nulle. Question 12bis Il sert à instancier le bolide b et à lui donner une vitesse. On va ensuite réutiliser b. On aurrait pu aussi le programmer dans l'intanciation de la classe BolideTest, c'est plus clean de mettre cela en préambule.

Question 13 Elle teste la méthode calculeAccélération. Spec : Calcule la direction en faisant cible - position, puis Traduit la direction en une accélération élémentaire, en prenant le vecteur de norme 1 approchant au mieux la direction.

Question 14 Elle teste accelereDe qui Ajoute l'acceleration à la vitesse

Question 15 Elles teste la méthode stop qui met la vitesse a zéro.

Question 16 Elles teste la méthode CalculeDeplacement qui renvoie une liste de déplacements de norme 1, dont la somme est égale la vitesse. Stratégie : on part d'un vecteur cible égal à la vitesse, qu'on normalise pour obtenir le premier déplacement, puis on retire ce premier déplacement du vecteur cible et on normalise à nouveau pour obtenir le deuxième déplacement, etc.

```
List<Vect> calculeDeplacements() {
    Vect deplacementCible = this.vitesse;
    List<Vect> deplacements = new MyLinkedList<>();
    while (!deplacementCible.equals(Vect.ZERO)) {
      Vect deplacement = deplacementCible.normalise();
        deplacements.add(deplacement);
      deplacementCible =
          deplacementCible.sub(deplacement);
    }
    return deplacements;  }
```

Question 17 La valeur true correspond à traversable, le circuit crée consiste en un petit trou en forme de "L", les deux derniers paramètres sont la position initiale du bolide dont la vitesse est zéro.

Question 18 Si le bolide démarre en dehors de la carte ca génére une exeption.

Question 19 Que ca dérange pas si la carte est trop petite ou trop grande par rapport a la taille du circuit, on suppose que ya des murs si c'est pas définit par la carte.

Question 20 Le bolide exécute les déplacement unitaire jusqu'a ce qu'on lui demande de rentrer sur une case ou y a un mur, auquel cas il s'arrête. et ca renvoie Faux, sinon ca renvoie vrai D'ou le code :

Question 20 bis

```
boolean deplaceBolide(List<Vect> deplacements) {
      Vect positionCourante = bolide.position;
      for (Vect depl: deplacements) {
        Vect positionCible =
                bolide.position.add(depl);
        if(getCase(positionCible).traversable)
            {bolide.position = positionCible;
          } else {
              bolide.stop();
              return false;
          }
      }
      return true;
  }
```

Question 21 Met tout ensemble : en calcule l'accélération, update la vitesse, calcule les déplacements, et déplace le bolide.

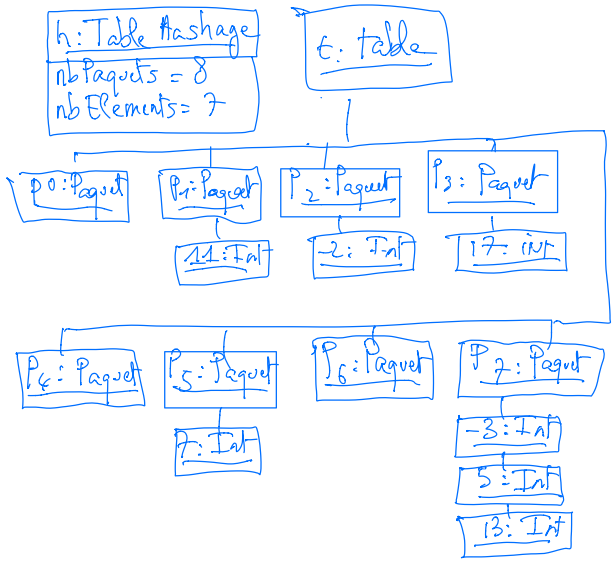
```
void gereClic(Vect cible) {
    bolide.accelereVers(cible);
   <List<Vect> deplacements =
          bolide.calculeDeplacements();
    deplaceBolide(deplacements)  }
```

:

TD2 : Hashtable, spécification

Question 1 : $-2 * 3 \bmod 8 = 2$, donc dans la case 2.

Question 2 : On fait apparaitre les liens représentant les elements de la liste chainée du paquets 7 pour souligner que l'accés nécessite de parcourir la liste.



Question 3 : La séquence décrite ne concerne que le cas où le paramètre [elt] est différent de [null] (dans le cas contraire la spécification indique que le résultat doit être [false], et aucun calcul n'est nécessaire). On suppose donc que le paramètre a déjà été testé, et on se place dans le cas non [null].

Exemple A : elt = 11

Exemple B : elt = 8

Dans ce cas, la table de hachage utilise son opération d'obtention d'un indice de paquet [TableHachage : :indicePaquet]. Pour cela, elle appelle l'opération [T : :code] d'obtention du code de l'élément [elt] passé en paramètre et obtient un code [c], dont elle déduit un indice [i] dans le bon intervalle avec une opération de modulo.

Exemple A : c = 11*3 = 33 i = 33

Exemple B : c = 8*3 = 24 i = 24

Puis utilisation de l'opération de récupération d'un paquet du tableau [TablePaquets : :paquet], à laquelle on passe le paramètre [i]. On obtient

en retour un paquet [p], et on utilise l'opération de test d'appartenance [Paquet : :contient] d'un élément à ce paquet [p].

Exemple A : p = t.paquet(1) = p1

Exemple B : p = t.paquet(0) = p0

Enfin, on renvoie le résultat donné par cette dernière opération.

Exemple A : resultat = p1.contient(11) = true

Exemple B : resultat = p0.contient(8) = false

Question 3 bis, Elle itére sur la liste chainée associée au paquer pour chercher l'élements.

Question 4 : Pour contient, On a prévu de répondre faux dans ce cas.

Question 4bis On peut aussi répondre faux, c'est ce qui se passe dans l'interface List de java

Question 5 : On peut toujours calculer le hashcode. Toutes les classes héritent d'un schéma basique de hachage de la classe de base java.lang.Object.

Question 6 : TableHachage : :indicePaquet(T elt)

La précondition permet d'appeler directement le calcul de hashcode de [elt] sans tester au préalable qu'il n'est pas [null] Cette condition est toujours satisfaite à chaque appel, parceque indicePaquet est une méthode privée, elle n'est appellée que depuis l'intérieur de la classe TableHashage, et donc avec des elements non null.

TablePaquets : :paquet(int i)

La précondition permet d'éviter de tester le respect des bornes du tableau et de prévoir un cas d'erreur. Elle est toujours satisfaite car TablePaquet est une classe interne privée , et le paramètre fourni est le résultat d'un appel à [TableHachage : :indicePaquet], dont la postcondition assure le respect des bornes.

Question 7 : Puisque les méthodes ne sont jamais appelées dans des contextes dans lesquels les préconditions ne seraient pas valides, il n'y a pas besoin de les tester avec un assert, même a des fins de débogages.

Question 8 : Nous ne maîtrisons pas le contexte d'appel de [TableHachage : :contient] par le client. Il faut donc prendre des mesures pour éviter qu'un mauvais paramètre ne perturbe le reste de la chaîne. Notamment :
1- la précondition doit être documentée publiquement
2- on peut aussi, et c'est même recommandé, vouloir faire un test avec un assert pour déclencher une exception en cas de paramètre illégal plutôt que d'attendre que cela ait une conséquence imprévue ailleurs ; cette exception doit également être documentée

Question 9 Selon la spécification actuelle, un paquet peut contenir plusieurs occurrences de l'élément à retirer. L'opération [Paquet : :retire] ne retirant qu'une occurrence, et ne faisant pas de retour particulier en cas d'absence de l'élément, on a besoin de tester la présence de l'élément, et d'itérer jusqu'à sa disparition complète.

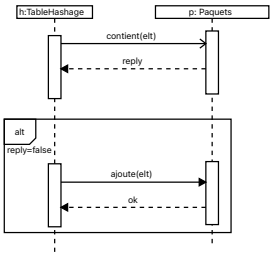
Question 10 Si on avait un invariant selon lequel les paquets ne contiennent pas de doublons, on pourrait se contenter d'un seul appel à [Paquet : :retire].

Question 11 Pour garantir l'absence de doublons, on inclus dans la postcondition de [TableHachage : :ajoute] la contrainte que cette opération ne modifie pas la table lorsque l'élément est déjà présent. On a donc :

Nouvel invariant - Aucun élément n'apparaît deux fois dans un paquet

Spécification de TableHachage : :ajoute(T elt) - Précondition : [elt] n'est pas [null] - Postcondition : [elt] est ajouté à la table s'il n'y était pas déjà ; dans ce cas l'attribut [nbElements] est incrémenté, et sinon la table n'est pas modifiée

Diagramme de séquence correspondant : On précise déjà que la séquence d'opérations n'est exécutée que lorsque [elt] n'est pas [null] (si [elt] est [null] on s'arrête immédiatement). Comme pour [contient] on commence par calculer l'indice [i] du paquet et récupérer le paquet [p]. Ensuite on utilise [Paquet : :contient] pour tester la présence de l'élément. Jusqu'ici, c'est le même diagramme que pour contient. À partir de là, deux possibilités : 1/ si l'élément était présent on s'arrête 2/ sinon (avec une boite ALT) on utilise `[Paquet::ajoute]` (et on incrémente nbElements mais cela n'apparaît pas vraiment dans le diagramme comme une opération, on peut se contenter de mettre un commentaire)



question 12 On modifie l'opération `[TableHachage::ajoute]`. On ajoute en postcondition que si [nbElements]/[nbPaquets] est plus grand que [seuilCharge], alors [nbPaquet] est doublé, [tableau] est remplacé par un nouveau tableau de la bonne taille, et de nouveaux paquets sont formés, qui contiennent exactement les éléments précédents, plus l'élément qui vient d'être ajouté.

Sur le diagramme de séquence correspondant on peut observer la création d'une nouvelle instance de [TablePaquet] et la disparition de l'instance d'origine, comme dans le fragment suivant.

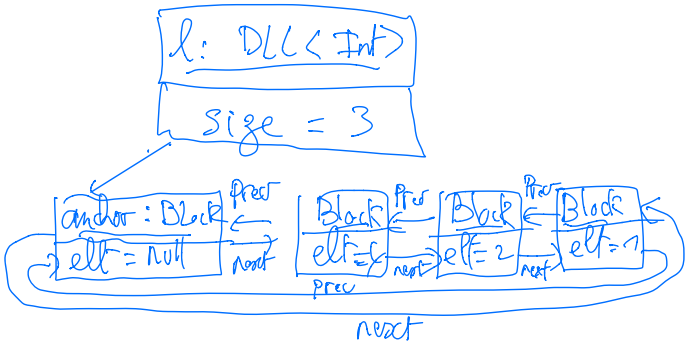
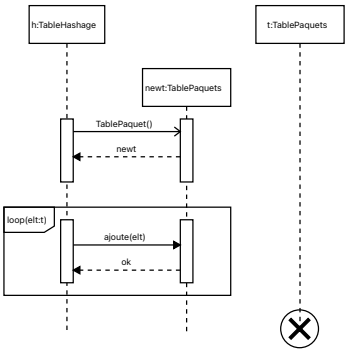


FIGURE 5 – Diagramme d'objet



TP6 : Invariants, assertions, et lambda-expr

Question 1. Il construit une liste vide, puis une liste smallList contenant trois élements, 4,2,1, et l'affiche.

Question 2 Voir figure 13.

Question 3 : Ce sont des classes internes de DLL, elle peuvent accéder aux attributs de la liste dont elle dépendent. La classe DLLiterator peut ainsi accéder a l'attribut anchor deux fois, pour initialiser et pour tester la fin de l'itération. La class Block n'a pas besoin, mais la rendre interne souligne le fait que elle n'est utilisée que par DLL.

Question 4 :Les invariants sont :

1. la valeur de l'attribut [size] est le nombre d'éléments.
2. l'attribut [elt] vaut [null] pour l'ancre, et est différent de [null] pour tous les autres blocs d'une liste
3. le prédécesseur du successeur d'un bloc est lui-même, le successeur du prédecesseur d'un bloc est lui-même

Remarque : l'organisation circulaire des listes fait que les deux lignes du 3/ sont redondantes l'une avec l'autre

Question 5 : On a besoin de calculer le nombre de blocks, pour cela on utilise un foreach, car notre DLL est iterable.

```
private int countElements() {
      int count = 0;
      for (T elt: this) { count++; }
      return count;
  }
```

Question 6

```
private boolean checkInvariants() {
  return  this.size == countElements()  }
```

Question 7 Il faut des test positifs, on appelle checkInvariant, sur les deux listes crées, listeVide et listeSmall, on vérifie que ca passe avec l'option de compilation -ea. Après cela on introduit un bug pour vérifier que ca ne passe plus ; par exemple on oublie d'incrémenter size dans add. On vérifier que le bug est signalé.

Question 8 : Elles dit que pour implémenter `boolean Predicate(T t)`, il faut définir `boolean test(T t)`

Question 9 Une lambda-expression peut être reconnue comme instanciant n'importe quelle interface fonctionnelle, pour peu qu'elle soit cohérente avec le type de l'unique méthode abstraite de cette interface. Pour implémenter `test<T>` il suffit d'avoir un paramètre de type T est un résultat booléen

Question 10 On écrit donc d'abord l'itérateur sur les blocs (initialisation, test, passage d'un bloc à son successeur) et la logique (test du prédicat, et résultat). Une variable [currentBlock] mémorise le bloc qu'on est en train de

considérer. On commence par le block qui suit l'ancre, et on s'arrete quand on retrouve l'ancre

On définit ensuite la méthode s'occupant de la logique, qui considère un à un les blocs fournis par l'itérateur, renvoie faux si l'un des blocs ne vérifie pas le prédicat, et vrai si tous les blocs ont été testés avec succès. Mais l'itérateur utilise l'ancre pour tester la fin de l'itération, il ne peut donc itérer sur le bloc qui contient l'ancre. Pour pouvoir traiter le block qui contient l'ancre, la solution la plus simple est de le faire séparément.

```
private Block currentBlock = anchor.nextBlock;
public boolean hasNext(){
  return this.currentBlock != anchor;};
public Block next() {
  Block b = this.currentBlock;
  this.currentBlock = this.currentBlock.nextBlock;
  return b;  }; }

private boolean forAllBlocks(Predicate<Block> p) {
  Iterator<Block> it = this.blockIterator();
  if (!p.test(anchor)) return false; //il faut tester l'ancre séparément.
  while (it.hasNext()) {
    if (!p.test(it.next())) return false;
  }
   return true;}
```

Question 11

```
private boolean checkInvariants() {
  return   this.size == countElements()
 && forAllBlocks(
   b -> (b==anchor)?b.elt==null:b.elt!=null)
 && forAllBlocks(
   b -> b.nextBlock.prevBlock == b);   }
```

Question 12 On refait le test pour emptylist et smallList, et on modifie les add de smallList pour qu'elle contienne 4,null,1, au lieu de 4,2,1, le test doit alors planter.

Question 13 On programme d'abord removeBlock qui suprime un bloc. L'assertion permet de vérifier qu'on ne vire pas l'ancre.

```
 private void removeBlock(Block b) {
  assert b != this.anchor: "Enlève ancre";
  b.prevBlock.nextBlock=b.nextBlock;
  b.nextBlock.prevBlock=b.prevBlock;
   this.size--;
    assert checkInvariants() :
          "Problème removeBlock"; }
```

Question 14 Cette fois, on va appliquer accept.

```
private void forAllBlock2(Consumer<Block> f) {
        Block currentBlock = this.anchor;
        do {
            f.accept(currentBlock);
            currentBlock = currentBlock.nextBlock;
        } while (currentBlock != this.anchor);
    }
```

Question 15

```
 public void removeAll(T elt) {
  this.forAllBlock2(b ->
   { if (b.elt == elt) removeBlock(b);});
     }
```

Question 16 Parceque on met a jour seulement nextBlock, faut aussi s'occuper de prevBlock ;

Question 17 On fait un cas particulier si la liste à ajouter est vide, et on se donne deux méthode auxiliaires pour accéder au premier et au dernier blocs utiles d'une liste.

```
    public void concat(DLL<T> l) {
        this.size += l.size;
        if (!l.isEmpty()) {
            l.firstPayloadBlock().prevBlock
             = this.lastPayloadBlock();
            l.lastPayloadBlock().nextBlock
              = this.anchor;
            this.lastPayloadBlock().nextBlock
               = l.firstPayloadBlock();
            this.anchor.prevBlock =
             l.lastPayloadBlock();
        }
        assert checkInvariants() :
          "Problème après concat";
    }
    private boolean isEmpty() {
        return this.anchor.nextBlock ==
           this.anchor;
    }
    private Block firstPayloadBlock() {
        return this.anchor.nextBlock;
    }
    private Block lastPayloadBlock() {
        return this.anchor.prevBlock;
    }
```

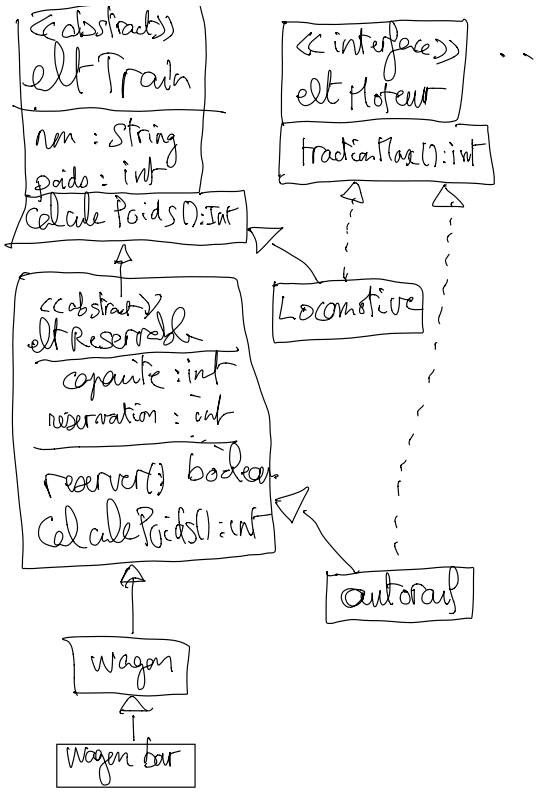


FIGURE 6 – Diagramme de classes

Question 18 Une proposition de solution, en deux passes : 1 -une itération en suivant les pointeurs [nextBlock] 2 -une itération en suivant les pointeurs [prevBlock]

```
public void rev() {
    forAllBlock2(b -> { b.prevBlock = b.nextBlock; });
    forEachBlockDescending(
       b -> { b.prevBlock.nextBlock = b; });
    assert checkInvariants() : "Problème après rev";
  }
private void forEachBlockDescending(
             Consumer<Block> f) {
        Block currentBlock = this.anchor;
        do {f.accept(currentBlock);
            currentBlock = currentBlock.prevBlock;
        } while (currentBlock != this.anchor);
    }
```

TD3 Train : liaison dynamique.

Question 1 Si on peut reserver une place dedans, et si y a un moteur

Question 2 wagon, autorail sont réservable, locomotive, autorail sont motorisés.

Question 3 La propriété réservable car elle met en jeu deux attributs capacité et reservations.

Question 4, faut définir une classe intermédiaire eltReservabe dont hérite wagon et autorail.

Question 5 La présence de la méthode tractionMax() qui renvoie un boolean.

Question 6 On définit une interface eltMoteur qui impose de définir tractionMax()

Question 7 Voir figure 13.

Question 7bis Que eltTrain est abstrait, eltReservable peut l'être aussi.

Question 8 le poids, le nom, la capacité

Question 9 On précéde la déclaration de l'attribut par le mot clef final

Question 10 Dans le constructeur de la classe eltTrain, mais si c'est abstrait, on peut quand même construire.

Question 11 On déclare un champs de classe "compteur" `static int cpt=0` Le nom est généré automatiquement `nom = "Elt "+ cpt; incr(cpt);`

Question 12 dans le constructeur de eltReservable. Une classe abstraite peut quand même avoir un constructeur.

Question 13

```
 public EltReservable(int poids, int capacite){
        super(poids);
        this.capacite = capacite;
        this.reservations = 0;
    }
 public Wagon(int poids, int capacite) {
        super(poids, capacite);
    }
}
```

Question 14 On met la capacité à zéro dès la création.

```
class WagonBar extends Wagon {
    public WagonBar(int poids) {
        super(poids, 0);
    }
}
```

Question 15 Dans la classe EltReservable. Tout les descendant la réutilise : wagon, autorail. Pour wagonbar elle existe mais renvoie toujours faux.

```
 public boolean reserver() {
        if (this.reservations >= this.capacite)
            return false;
        this.reservations++;
        return true;
    }
```

Question 16 Il faut la définir pour eltTrain, mais la rédéfinir pour EltReservable, pour prendre en compte le poids des passagers.

```
public int eltTrain::calculerPoids(){ return poids;}
public int EltReservable::calculerPoids() {
  return this.poids + 75 * this.reservations;
    }
```

Questions 17 train complet On définit une méthode intérmédiaire tractionMax().

```
class Train {
      private ArrayList<EltTrain> elements;

    /**
     * Initialement, un train ne
     * contient aucun élément.
     */
    public Train(){
        this.elements = new ArrayList<>();
    }

    /**
     * Ajout d'un élément au train.
     * @param elt  l'élément à ajouter.
     */
    public void ajouterElt(EltTrain elt){
        this.elements.add(elt);
    }

    /**
     * Calcul du poids total du train
     * et de son chargement.
     * @return  le poids total.
     */
    public int calculerPoids(){
        int poids = 0;
        for (EltTrain elt: this.elements) {
            poids += elt.calculerPoids();
        }
        return poids;
    }

    /**
     * Calcul de la capacité de traction
     * de l'ensemble des éléments
     * moteurs du train.
     * @return  la capacité de traction totale.
     */
    public int tractionMax() {
        int traction = 0;
        for (EltTrain elt: this.elements) {
            if (elt instanceof EltMoteur) {
                traction +=
                  ((EltMoteur)elt).tractionMax();
            }
        }
        return traction;
    }

    /**
     * Vérification de la bonne formation du train.
     * @return  true si et seulement si
     * le train est bien formé.
     */
    public boolean bienForme() {
        for(EltTrain elt:
           elements.subList(1,elements.size()-1))
            if(elt instanceof EltMoteur) return false;
       return calculerPoids()<=tractionMax();
    }
}
```

Question Que se passe-t-il ?

```
EltTrain e;
Wagon w;
WagonBar wb;
Locomotive l;
Autorail a;

e = new EltTrain(100);
-> erreur compilation, on ne peut pas instancier
 une classe abstraite

l = new Locomotive(1000, 10000);
w = new Wagon(1000, 20);
wb = new WagonBar(1000);
a = new Autorail(1000, 5000, 10);

System.out.println(w.capacite);
-> 20

System.out.println(wb.capacite);
-> 10

System.out.println(e.nom);
-> NullPointerException
(l'élément n'a pas pu être défini)

System.out.println(wb.nom);
-> Elt 3

a.reserver();
-> true

w.reserver();
-> true

wb.reserver();
-> false

l.reserver();
-> erreur compilation (méthode n'existe pas)

wb = (WagonBar) w;
-> erreur exécution (type réel de w
       pas compatible avec WagonBar)

System.out.println(wb.capacite);
-> 0

l = (Locomotive) a;
-> erreur compilation (type Autorail et Locomotive
 incompatibles a priori)

e = wb;
e.reserver();
-> erreur compilation (type apparent EltTrain
 ne connaît pas reserver)

System.out.println(e.capacite);
-> erreur compilation (pareil)

w = (Wagon) e;
-> (transtypage ok, car type réel WagonBar
```

```
        =  sous-type de Wagon)

  w.reserver();
  -> false

  System.out.println((EltTrain) w).capacite);
  -> erreur compilation (type EltTrain
    ne connaît pas capacite)

  System.out.println((WagonBar) w).capacite);
  -> 0
```

TP7 Dessin vectoriel : visiteurs.

Question 1 L' appel `o.operate(e)` affiche "Line" tandisque 'appel `apply(o, e)` affiche "Element". Etonnant, non ?

Question 2 : ce sont bien sur Print et Line

Question 3 : Pour e dans le premier cas l'appel se fait depuis le main, donc c'est Line, mais dans le deuxieme, l'appel se fait depuis apply, pour qui le type apparent de e est Element. On peut faire le même raisonnement pour o ; sont type apparent est Print pour l'appel depuis main, et Operateur pour l'appel depuis apply.

Question 4 : L' appel `o.operate(e)` affiche "Line"C'est donc la méthode operate(Line) qui est appelée, tandisque 'appel `apply(o, e)` affiche "Element". C'est donc le operate(Element) qui est appelée,

Question 5 Le type apparent des paramètres explicites corresponds à ce qui est affiché. Java selectionne la méthode appelée en construisant la signature. Pour contruire cette signature, java considére le type apparent pour les paramétres explicites, celui qui est dispo au moment de la compilation.

- Pour le premier appel o.operate(e), le type apparent de e est bien Line et c'est le operateLine qui est appellé.
- Dans le 2eme appel avec apply, à l'intérieur de apply, le type apparent de e est Element et c'est donc le operate de Element qui est appelé.

Dans le main, faites l'expérience déclarer e comme un Element au lieu d'une ligne. Vous constaterez que à nouveau ce sera le operate de Element qui sera appelé.

Question 7 Ces premiéres questions démontrent justement que non ca ne fonctionne pas. Dorénavant on évitera donc systématiquement la surcharge en introduisant différent noms, suivant le type du paramétre explicite.

Question 8 : On peut rappeller operate pour ne pas réecrire le print, car cette fois le type réel sera egal au type apparent.

```
public void operateLine(Line l) {operate(l);}
public void operateCircle(Circle c) { operate(c);}
```

Question 9

```
    public void operate(Element e) {
        if (e instanceof Line) operateLine((Line) e);
        else operateCircle((Circle) e);
    }
}
```

Question 10

```
//dans Element
  public abstract void print();
//dans Line
  public void print() {
     System.out.println("<line x1=\"" +
         x1 + "\" x2=\"" +  x2 +
        "\" y1=\"" +  y1 + "\" y2=\"" +  y2 +
        "\" stroke=\"" +  colour + "\"/>");
          }
//dans Circle
   public void print() {
        System.out.println("<circle cx=\"" +
           x + "\" cy=\"" + y +
          "\" r=\"" + r +
          "\" stroke=\"" + colour + "\"/>");
    }
//nouvelle version de Print::operate()
 public void operate(Element e) { e.print(); }
```

Question 11 La premiére liason faite pas apply selectionne quelle operation choisir, et la seconde faite par l'operation elle même selectionne sur quel element on l'applique.

Question 12

```
//dans Print
    public void visitLine(Line l)
          {operateLine(l);}
    public void visitCircle(Circle c)
          {operateCircle(c);}
//dans change color ca dépends pas de l'élement.
    public void visitLine(Line l)
        { operate(l) ;;  }
    public void visitCircle(Circle c)
        { operate(c) ;  }
```

Question 13

```
//dans Line
    public void accept(Visitor v) {
        v.visitLine(this);
    }
//dans Circle
    public void accept(Visitor v) {
        v.visitCircle(this);
    }
```

Question 14

```
//on rajoute dans le main
     Operation o2=new ChangeColour("red");
      e.accept(o2);
      e.accept(new Print();
```

Question 15

```
class Translate extends Operation {
    private int dx, dy;

    public Translate(int dx, int dy) {
        this.dx = dx; this.dy = dy;
    }

    public void operate(Element e) {  }

    public void visitLine(Line l) {
        l.x1 += dx; l.y1 += dy;
        l.x2 += dx; l.y2 += dy;
    }
    public void visitCircle(Circle c) {
        c.x += dx; c.y += dy;
    }
}
```

Question 16

```
        e.accept(new Translate(20,30));
        e.accept(o);
    }
```

Question 17

```
class Zoom extends Operation {
    private int z;
    public Zoom(int f) {z=f;}
    public void operate(Element e) {    }

    public void visitLine(Line l) {
        l.x1 *= z; l.y1 *= z;
        l.x2 *= z; l.y2 *= z;
    }
    public void visitCircle(Circle c) {
        c.x *= z; c.y *= z;
        c.r *= z;
    }
}
```

Question 18

```
       Circle c=new Circle(0,0, 10,"green");
        c.accept(o);
        c.accept(new Zoom(10));
        c.accept(o);
```

Question 19

```
class Container extends Element {
    private ArrayList<Element> elements;

    public Container(String c) {
        super(c);
        this.elements = new ArrayList<Element>();
    }
    public void addElement(Element e) {
        elements.add(e);
    }

    public void print() { }
    public void accept(Visitor v) { }
 }
```

Question 20

```
public void accept(Visitor v)   {
  for(Element e : elements) { e.accept(v); }
        }
```

Question 21 C'est l'opération Print qui doit le faire, lorsque elle visite un container. Faut donc un visitContainer() dans l'interface visitor. Comme c'est juste l'operation Print qui en a besoin, on peut dire que elle ne font rien pour toutes les operation et la redéfinir dans Print. Attention, cependant, car il y a deux print à faire, un avant la visite des elements pour afficher `"<g>"` et un aprés autre pour afficher `"<\g>"`. Il faut donc non pas un, mais deux visitContainer : un visitContainerIn et un visitContainerOut, et appeler ceux-ci, dans le accept de Container.

Question 22

```
// Rajout Interface Visitor
   public void visitContainerIn(Container ct)
    public void visitContainerOut(Container ct)
//rajout dans Operation
    public void visitContainerIn(Container ct) {  }
    public void visitContainerOut(Container ct) { }
//rajout dans Print
    public void visitContainerIn(Container ct) {
        System.out.println(
          "<g fill=\"" + ct.colour + "\">");
    }
    public void visitContainerOut(Container ct) {
        System.out.println("</g>");
    }
 //modif de accept(v:Visitor) dans Container
     public void accept(Visitor v)   {
        v.visitContainerIn(this);
        for(Element e : elements) { e.accept(v); }
        v.visitContainerOut(this);
    }
```

Question 23 On ajoute dans ceci le main :

```
 Container g = new Container("blue");
 Container g2 = new Container("yellow");
 g.addElement(g2);
 g2.addElement(new Circle(0, 0, 200, "green"));
 g.addElement(e);
 g.addElement(c);
 g.accept(o);
```

Question 24

```
 <svg version="1.1"
     baseProfile="full"
     xmlns="http://www.w3.org/2000/svg">
```

TD4 Emploi du temps, exception

Question 1 :

```
class HoraireInvalide
       extends IllegalArgumentException {
 public HoraireInvalide() {}
 public HoraireInvalide(String message)
   { super(message); }
}
```

Question 2 : On pourrait ne pas définir les constructeurs explicitement ce qui laisserai seulement le constructeur par défaut (sans arguments.))

Question 3 Le constructeur doit lever HoraireInvalide si les arguments fournis ne correspondent pas à un horaire valide. Il faut rajouter deux tests dans le contructeur

```
if ((j < 0 || (j > 23)
throw new HoraireInvalide("Jour invalide.");
if ((h < MIN_H) || (h > MAX_H))
throw new HoraireInvalide("Heure invalide.")
```

Question 4 Non car la classe IllegalArgumentException est une sous-classe de RuntimeException, et c'est donc une exception hors-contrôle (unckecked).

Question 5

```
class Horaire implements Comparable<Horaire>
```

Question 6

```
 protected int nbHeures()
  {return this.jour*24 + this.heure; }
 public int compareTo(Horaire hor)
  {return this.nbHeures()-hor.nbHeures(); }
```

Question 7

```
public Creneau(Horaire debut, int duree)
  {this(debut, debut.ajouteDuree(duree));}
```

Question 8

```
class CreneauInvalide extends Exception {}
```

Question 9

```
class CreneauVide extends CreneauInvalide {}
class Depassement extends CreneauInvalide {}
}
```

Question 10

```
public Creneau(Horaire debut, Horaire fin)
                     throws CreneauVide {
    if (!(debut.compareTo(fin) < 0))
       { throw new CreneauVide(); }
this.debut = debut; this.fin = fin;
 }
```

On remarque que comme l'exception levée est sous controle, il faut maintenant ne pas oublier de la mentionner dans l'en-tête de la méthode.

Question 11 Dans la méthode ajouteDurée, on obtiendra un dépassement si jour devient plus grand que 6, cela sera déja détecté comme un horaireInvalide, il suffit donc de rattrapter horaireInvalide pour ensuit léver Depassement !

```
public Horaire ajouteDuree(int duree)
   throws Depassement
{   ...
  if (duree >= heure2finJour)
    try { return new Horaire(
        this.jour + d/24, d%24);}
catch (HoraireInvalide e)
 { throw new Depassement(this); }
} }
```

Question 12

```
 public Creneau(Horaire debut, int duree)
  throws CreneauVide, Depassement
 { this(debut, debut.ajouteDuree(duree)); }
```

Les exceptions creneauVide et Depassement sont levées par les appels au premier constructeurs, et à ajouteDurée. Il faut juste les mentionner dans l'en-tête du constructeur.

Question 13 Comme lors du rattrapage on réutilise le constructeur, les déclarations d'exceptions doivent rester dans l'en tête même si elles ne peuvent pas être levées dans les faits.

```
 public static Creneau
     creneauSafe(Horaire hd, Horaire hf)
       throws CreneauVide, Depassement {
  try { return new Creneau(hd, hf); }
  catch (CreneauVide e) {
    try{ return new Creneau(hf, hd); }
catch (CreneauVide e2) { // debut == fin
return new Creneau(hd, 1);
 } } }
```

Question 14

```
class Conflit extends Exception {
 public Creneau c;
 public int i;
 public Conflit(Creneau c, int i)
   { this.c = c; this.i = i; }
}
```

Question 15

```
 public void ajouteCreneau(Creneau c)
                throws Conflit {
  int i=0;
  try {
    while (this.liste.get(i).precede(c))  i++;
     if (this.liste.get(i).suit(c))
         this.liste.add(i, c);
     else  throw new Conflit(c, i);
  } catch (IndexOutOfBoundsException e) {
  this.liste.add(c);// ajouter à la fin
 } }
```

Dans la boucle b on ne regarde jamais si [i] reste dans le tableau. On intercepte l'éventuelle sortie, en rattrapant l'exception IndexOutOfBoundsException qui correspond au cas où le créneau qu'on veut ajouter est postérieur à tous les créneaux déjà présents.

Question 16

```
public void ajouteCreneauPriorite(CreneauCours c)
  throws Conflit {
   try { this.ajouteCreneau(c); }
   catch (Conflit e) {
 if (this.liste.get(e.i).estPrioritaire(c))
     throw e;
 boolean conflit_prioritaire = false;
 int j = e.i;
 while ((j < this.liste.size())
        && !this.liste.get(j).suit(c)
        && !conflit_prioritaire)
 {  conflit_prioritaire =
          this.liste.get(j).estPrioritaire(c);
j++;
 }
 if (conflit_prioritaire) { throw e; }
 for (int i=e.i; i<j; i++)
       {this.liste.remove(i); }
 this.liste.add(e.i, c);
}
 }
```