Environnement Client/Serveur
M1 Miage Classique

vendredi 18 décembre 2015

Examen

durée 2h

Consignes l'examen dure 2 heures et est noté sur 20 points. Il comporte quatre exercices. Les supports de cours et les notes manuscrites sont les seuls documents autorisés. Les barème n'est donné qu'à titre indicatif et reflète la difficulté des exercices (et donc le temps qu'il est conseillé d'y consacrer).

QCM (3 points)

Pour chacune des questions suivantes, reporter sur votre copie l'ensemble des lettres des réponses correctes (il peut y en avoir plusieurs ou aucune). Si aucune réponse n'est correcte selon vous, indiquez le par $\emptyset$. Un point est accordé si l'ensemble exacte des bonnes réponses est sélectionné, sinon aucun point (pas de points négatifs).

1. Le protocole de transport TCP :
   (a) se situe au même niveau que le protocole IP
   (b) garantit l'absence de perte de paquets
   (c) utilise un mécanisme de *handshake* (négociation) initial entre le client et le serveur
   (d) permet de faire de la diffusion
2. Soit une classe `A` contenant un attribut `x` et une classe `B` telle que `B extends A` :
   (a) On peut re-définir l'attribut `x` dans `B` s'il n'est pas `final` dans `A`
   (b) On peut re-définir l'attribut `x` dans `B` s'il n'est pas `private` dans `A`
   (c) On peut re-définir l'attribut `x` dans `B` s'il n'est pas `static` dans `A`
   (d) On ne peut pas re-définir l'attribut `x` dans `B`
3. La somme de contrôle d'un paquet IPv4 :
   (a) Prend en compte l'adresse source et l'adresse destination du paquet
   (b) Permet de vérifier l'intégrité des données transportées par le paquet
   (c) Permet de détecter les erreurs de transmission
   (d) Peut valoir 200000

Sept erreurs (4 points)

On considère le code Java présenté à la figure 1. Donner sept erreurs dans ce code Java. Les erreurs recherchées sont des erreurs *statiques* (i.e. qui empêchent le code Java de compiler correctement), mais ne sont **pas des erreurs de syntaxe** (i.e. le code est « bien écrit »). Si deux lignes Java sont incompatibles, alors c'est toujours la ligne avec le numéro le plus grand qui constitue l'erreur. Par exemple :

```
1 String x = "ABC";
2
3 int y = x - 10;
```

L'erreur est ici à la ligne 3 (`x` est de type `String` et est utilisée comme opérande pour la soustraction). **Donner le numéro de ligne** suivi d'une **courte description** de l'erreur.

```
1  import java.util.*;
2  import java.io.*;
3
4  interface Iface {
5     int t;
6     public Iface()
7
8     public void m() throws IOException;
9     public void f(int x);
10 }
```

```
10
11 class C implements  Iface {
12
13   int x = 0;
14   static int y = 0;
15   static final z;
16
17   C() { super(); }
18   C(int x, int y) { super(x,y); }
19
20   public void m() throws IOException, FileNotFoundException
21   {
22     x = 10;
23     z = 42;
24
25     return x + z;
26   }
27
28  static int g()
29    {
30      return x+y;
31    }
32 }
33
34
35
```

FIGURE 1 – Code Java contenant 7 erreurs

Compteur global atomique (7 points)

On souhaite implémenter un compteur *global* et *atomique*, en utilisant la technologie Java RMI. La classe faisant office de serveur, possède deux méthodes :

- `bool init(Integer i)` permettant d'initialiser la valeur du compteur. Cette méthode ne peut être appelée qu'une fois. Elle renvoie `true` la première fois qu'elle est appelée et positionne la valeur interne du compteur à `i`, et renvoie `false` les fois suivantes, sans changer le compteur.
- `Integer next()` qui renvoie la valeur du compteur et incrémente ce dernier (la valeur renvoyée et celle **avant** incrément)

Questions

1. Donner **le code** d'une interface Java `IGlobalCount` correspondant à la spécification ci-dessus et représentant des objets dont les méthodes peuvent être appelées à distances via le procédé RMI. Il n'est pas demandé de mettre les `import` nécessaire.
2. Donner **le code** d'une classe Java `GlobalCount` implémentant cette interface et pouvant être appelée à distance (il faut donc une méthode `main` permettant de démarrer la classe serveur). On prendra garde aux problèmes d'accès concurrents. On peut supposer que la machine sur laquelle on veut exécuter le serveur s'appelle `serveur.com`.
3. Donner un code client contenant uniquement une méthode `main` qui initialise le compteur à 42 et l'incrémente deux fois.
4. Expliquer **précisément** (mais sans donner de code) comment on peut modifier les serveurs et clients pour que le client puisse découvrir sur le réseau local un serveur implémentant le compteur global atomique.

JSP : authentification (6 points)

On souhaite mettre en place un système d'authentification pour un site Web en utilisant JSP. Ce système est similaire à celui que l'on voit sur beaucoup de sites :

— Une page `index.jsp` contient un formulaire permettant d'entrer son *login* et sont *mot de passe*.
— Une servlet `CheckServlet` est la cible du formulaire, elle vérifie auprès d'une base de données que le login et mot de passes sont corrects. Si c'est le cas, la servlet redirige vers une page `ok.jsp` Sinon elle redirige vers la page `index.jsp` de nouveau.
— La page `ok.jsp` contient un lien vers une servlet `ByeServlet` permettant de se déconnecter et qui redirige vers `index.jsp`

On suppose que l'on dispose d'une classe Java `CheckDB` représentant le modèle ayant le code suivant :

```
class CheckDB {
  private Connection conn;

  //Effectue la connexion à la base
  CheckDB() throws SQLException, ClassNotFoundException { ... }

  boolean check(String login, String password) { /* A COMPLETER */ }
}
```

Questions

1. Donner le code de la méthode `check` de la classe `CheckDB` sachant que l'on dispose d'une table `USERS` contenant deux colonnes de types `VARCHAR(20)` : `USERNAME` et `PASSWORD`. La méthode ne lève aucune exception mais renvoie simplement `false` en cas de problème. Si le login et le mot de passe correspondent à ceux de la base, la méthode renvoie `true` et `false` sinon.
2. Donner le code de la méthode `doGet` de la classe `CheckServlet`. Cette dernière doit récupérer un objet `CheckDB` dans une variable de session ou le créer si besoin, récupérer deux paramètres `usr` et `passwd` dans la requête et appeler `CheckDB.check` avec ses paramètres. Si la méthode renvoie vrai, rediriger vers `ok.jsp` sinon rediriger vers `index.jsp` et placer un attribut de requête `"state"` à `1`. **On ne gérera pas les exceptions dans cette méthode** (i.e. inutile de mettre un `try/catch` global).
3. Donner le code de la méthode `doGet` de la classe `ByeServlet`. Cette dernière détruit la session et redirige vers `index.jsp` en plaçant un attribut de requête `"state"` à `2`. **On ne gérera pas les exceptions dans cette méthode** (i.e. inutile de mettre un `try/catch` global).
4. Donner le code de la section `body` du fichier `index.jsp`. Ce dernier doit afficher le formulaire (avec deux champs texte et un bouton) permettant de se logguer. Il doit aussi vérifier l'attribut de session `"state"` et afficher le texte « problème avec vos identifiants » si `"state"` vaut `1`. et « Déconnexion réussie » si `"state"` vaut `2`. Vous devez utiliser des *tags* JSTL.