Classe abstraite et Interfaces

Jeudi 15 septembre

Classe abstraite

- Une classe abstraite est préfixée par le mot clef « abstract »
- Empêche la création d'instance : même avec des constructeurs, on ne pourra faire de « new A »,
- Permet quand même l'utilisation de méthode static

```
public abstract class A {

public void m1(){
System.out.println("m1");}
public static void m2(){
System.out.println("m2");}

}
```

```
public static void main(String[] args) {
A monA;
monA = new A();
A.m2();

}
```

possible

Pas possible

On ne peut pas créer

D'une classe abs

possible

Pourquoi ?

- Une classe possédant une (ou plus) méthode abstraite, la classe sera forcéme abstraite
- Méthode abstraite : méthode non-implémentée, préfixé du mot-clef abstract sans corps (finir la ligne avec un ;)

```
abstract public void m3();
```

- Sert pour l'héritage : les classe qui hérite d'une classe abstraite doivent implémenter les méthodes abstraites, et rend utilisable les méthodes implémentée dans la classe abstraites

Classe A :

```
public abstract class A {

public void m1(){
System.out.println("m1");}
public static void m2(){
System.out.println("m2");}
abstract public void m3();
}
```

Classe B :

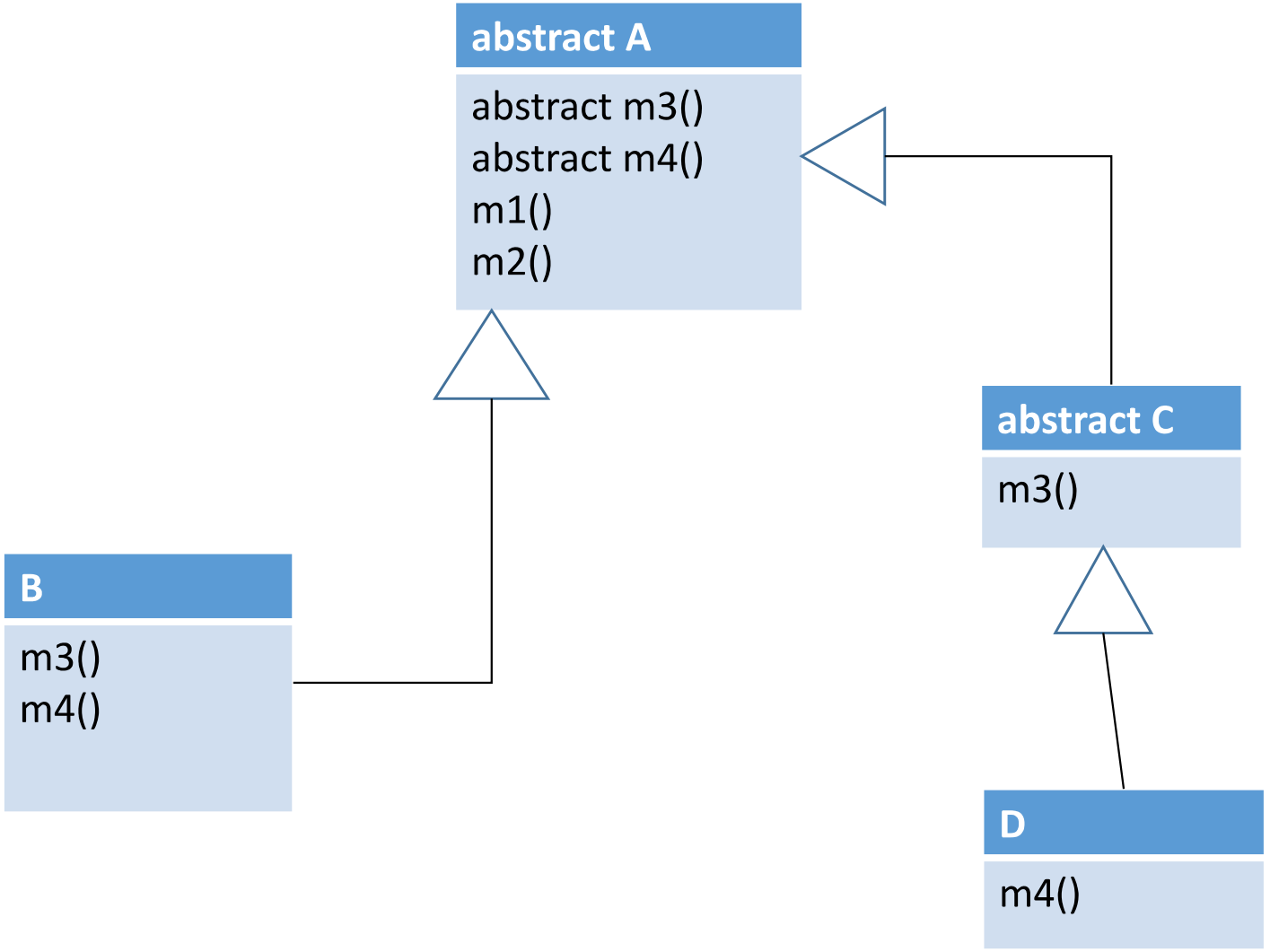
```
public class B extends A {

public void m3() {
System.out.println("m3");}
public void m4() {
System.out.println("m4");}
}
```

Classe de Test :

```
public class Main {
public static void main(String[] args)
A monA;
A.m2();
B monB = new B();
monB.m1();
monB.m3();
monB.m4();
}
}
```

Exemple de diagramme



C est forcément abstraite :
Elle n'a pas implémenté
toute les méthodes
abstraites

D n'est pas forcément abstra
Elle a implémenté toutes l
méthodes abstraites
(directement ou par héritag

Motivations

- Une classe abstraite permet l'utilisation de méthode déjà codée à condition d'implémenter certaines méthodes, donne une contraintes sur ses classes dérivées
- Une classe fille bénéficie des méthode de la classe abstraite, avec pour contrepartie de devoir implémenter les méthodes abstraites.
- Avec l'exemple du diagramme précédent, on peut par exemple faire :

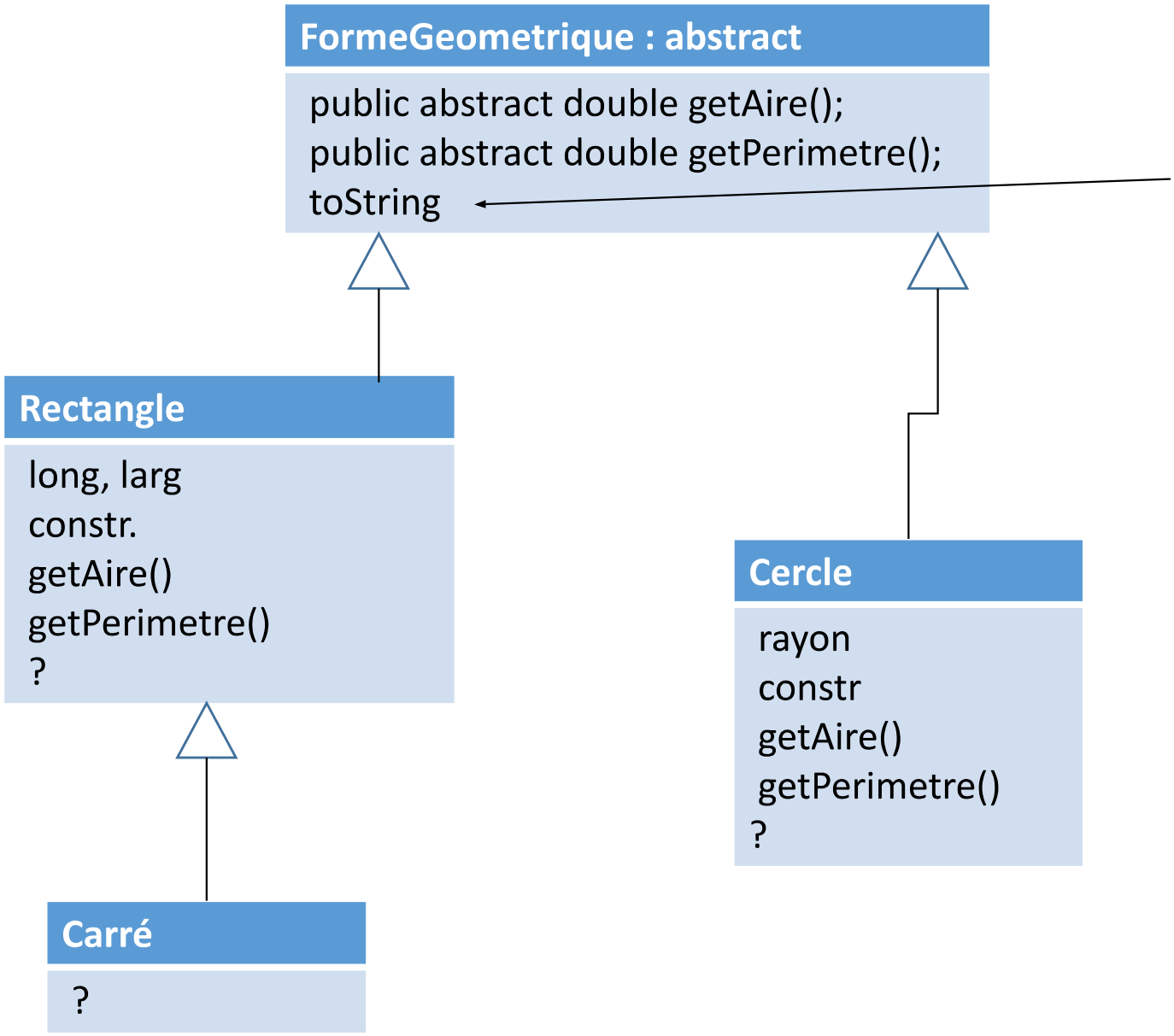
```
A[] tab = new A[3];
tab[0] = new B();
tab[1] = new D();
tab[2] = new B();
for (A e : tab) e.m3();
```

Comme la classe A impose à ses dérivées d'implémenter la métho
on peut utiliser le polymorphisme pour exécuter la méthode sur d
réels différents.

Pourquoi utiliser abstract ?

- Une méthode est abstraite quand elle a pas vraiment de sens dans la méthod elle est codée, que sont fonctionnement dépend intrinsèquement de la classe fille pour laquelle elle sera appelée
- Les classes abstraites : ça n'a pas vraiment de sens de les instancier, pas de ré concrète, une généralité pour laquelle il nous faut une spécialisation pour que ait un sens concret utilisable. N'a pas lieu d'être créée, n'a pas de sens propre logique.
- Garanti une utilisation normale/un modèle cohérent
- Permet de grouper des classes indépendantes pour pouvoir utiliser du polymorphisme afin par exemple de faire un tableau pour un traitement sur différents types de données, crée de la généricité

TP



Donne par ex :
« Rectangle d'aire 20 et de périmètre

A tester avec :

```
public static void main(String[] args) {
FormeGeometrique[] tab = new FormeGeometr
tab[0]=new Rectangle(10,5);
tab[1]=new Carre(4.5);
tab[2]=new Cercle(3);
for (FormeGeometrique f :tab)
System.out.println(f);
}
```

Qui renvoie :

```
Rectangle d'aire 50.0 et de périmètre 30.0
Carre d'aire 20.25 et de périmètre 18.0
Cercle d'aire 28.274333882308138 et de périmètre
```

Classe et méthodes scellées

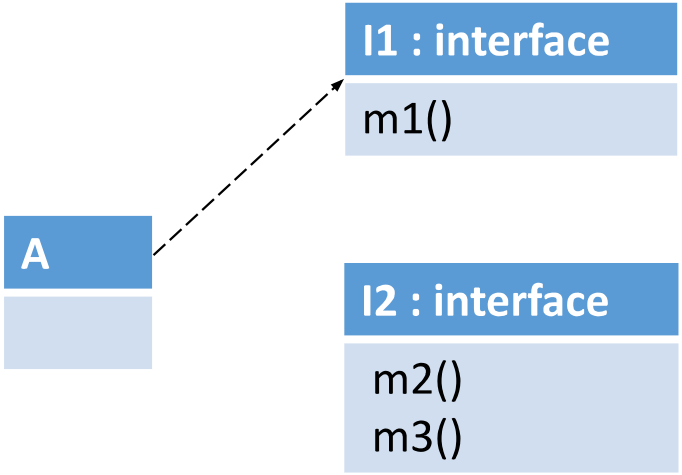
- Classe préfixée par **final** : empêche l'héritage
- Méthode **final** : empêche la surcharge dans les classes dérivées
- Rappel : attribut final = constante
- Exemple : Integer est une classe finale qui hérite de la classe abstraite Numbe
  (qui possède des méthodes abstraites et non abstraites, on voit que les abstra
  sont bien implémentée dans Integer)

Interface

- Une entité qui définit un nouveau type
- Contient **uniquement des méthodes non implémentée** (pas préfixée par abstract) : pas d'attributs, pas de méthodes implémentées

```
public interface I2 {
public void m2();
public void m3();
}
```

```
public interface I1 {
public void m1();
}
```




```
public class A implements I1 {

public void m1() {
System.out.println("M1 de A");
}
}
```

Pas de l'héritag
(peut hériter d'autre
Mot clé impleme

Doit implémer
toutes les mét
de l'interface

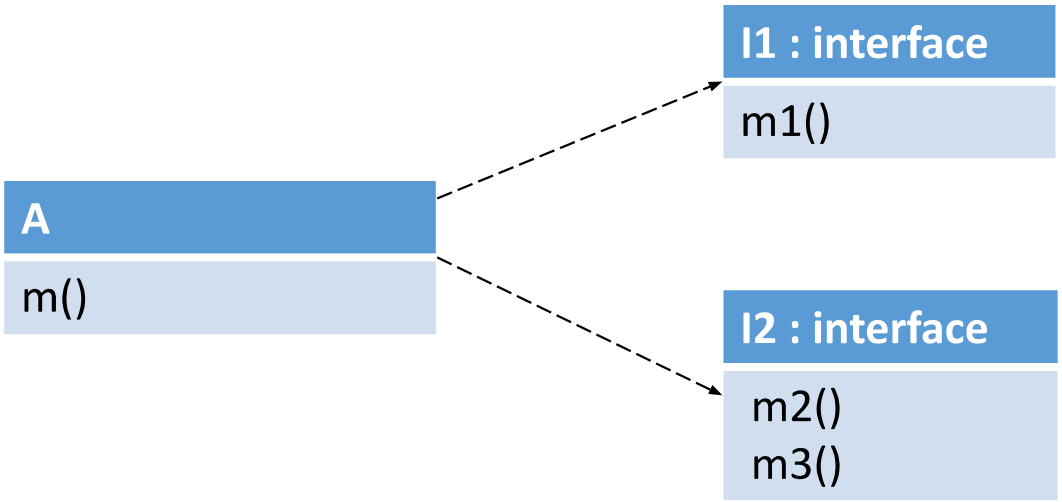
Une interface = un contrat

- L'interface donne un contrat aux classes qui l'implémente
- Une classe qui implémente une interface doit « adopter son comportement » doit implémenter toutes ses méthodes
- Exemple :
  une interface cryptable possédant des méthodes crypter et décrypter
  implémenter par une classe DocumentTexte
- On peut typer avec l'interface,
  visiblité = visibilité des méthodes de l'interface

```
Avec le schéma précédent
I1 a =new A();
a.m1();
```

Multiple implémentation d'interface

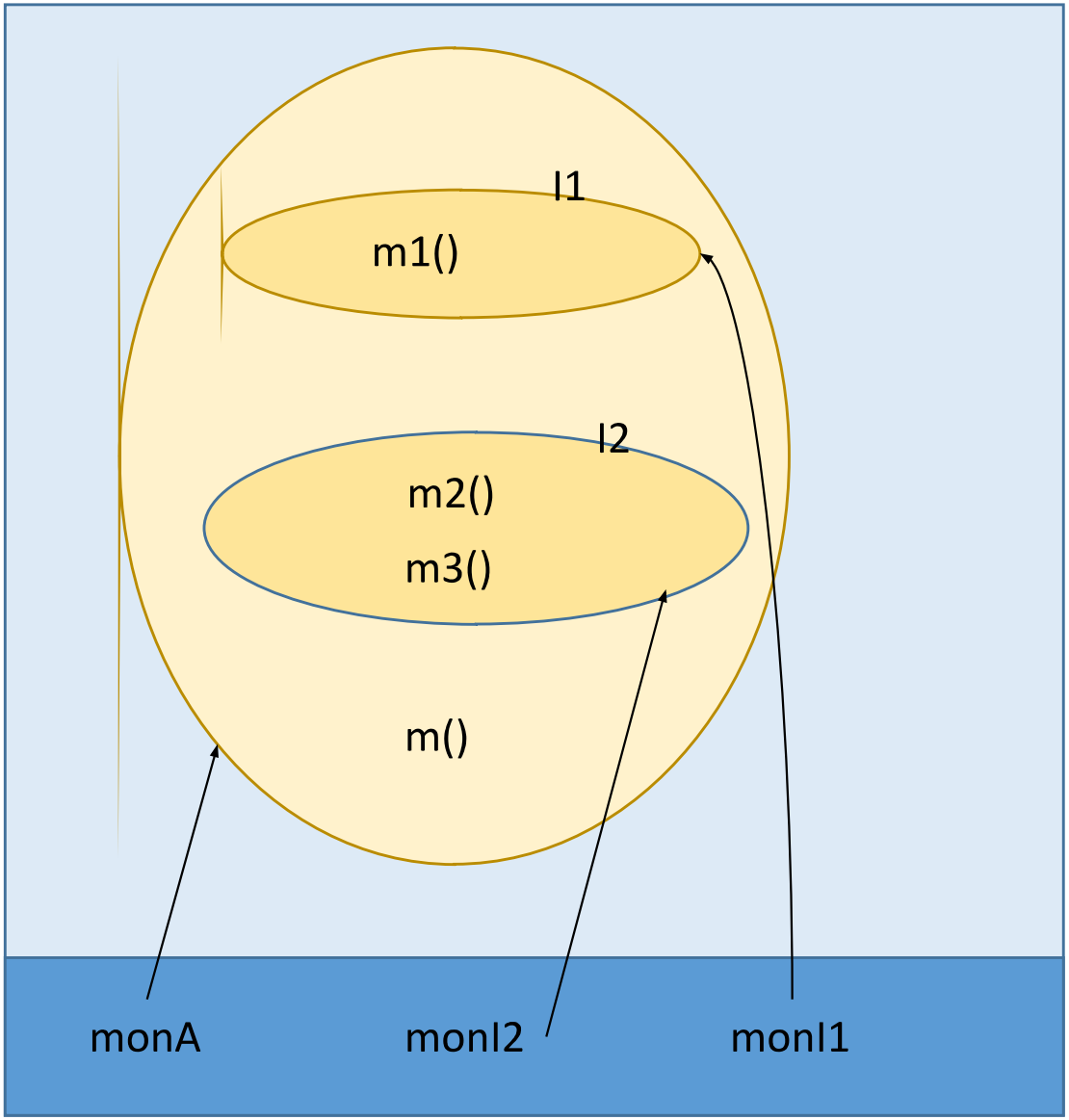
- Une clase peut implémenter plusieurs interfaces :




```
public class A implements I1, I

public void m1() {…}
public void m2() {…}
public void m3() {…}
public void m() {…}
}
```

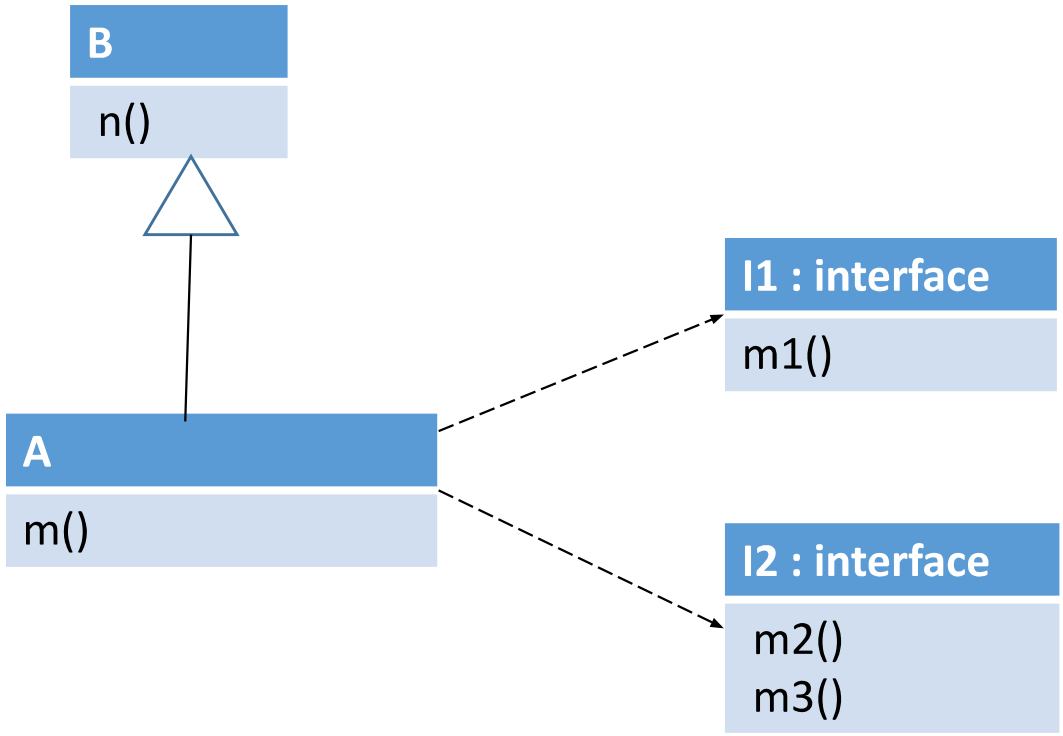
Il faut alors implémenter tous les méthodes de chacune des interface

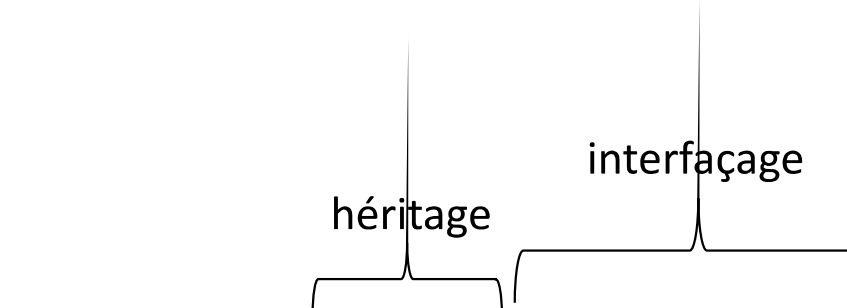



```
A monA =new A();
I1 monI1 = (I1) monA;
I2 monI2 = (I2) monA;
```

Interface et héritage

- L'héritage et l'interface sont compatible :





Une classe abstraite peut hériter d'une i
Elle peut alors déclarer tout ou partie de
méthodes à implémenter comme abstra
(l'implémentation est alors déléguée au
dérivées)

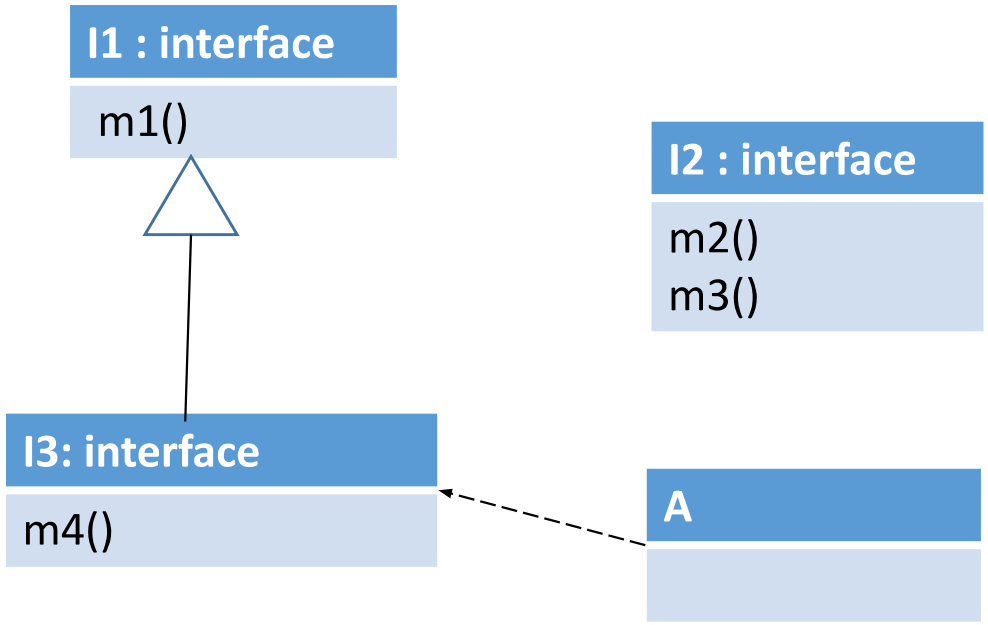
Exemple : Integer hérite de Number et implémente Comparable
Number (classe abstraite) implémente Serializable

Pas de multiple héritage

- Plusieurs solutions pour y palier, différents types de pattern pour s'y substitue dans différents cas.
  Typiquement le pattern Facade
- Multiple implémentation permet de récupérer l'aspect appartient à plusieurs famille, mais ne permet pas de récupérer des méthodes implémentées
- En java, pas possible de coder plusieurs versions d'une méthode en fonction d l'interface par laquelle elle est appelé.

Héritage entre interfaces

- Une interface peut hériter d'une autre : on dit plutôt **étendre** qu'hériter de (n
  c'est le même principe et mot clef)




```
public interface I3 extends I1 {
public void m4();
}
```

A doit implémenter les méthodes de I3 et I1, et ses instances peuvent être castées comme l'un où l'autre

Cast

```
A monA =new A();
I1 monI1 = (I1) monA;
```

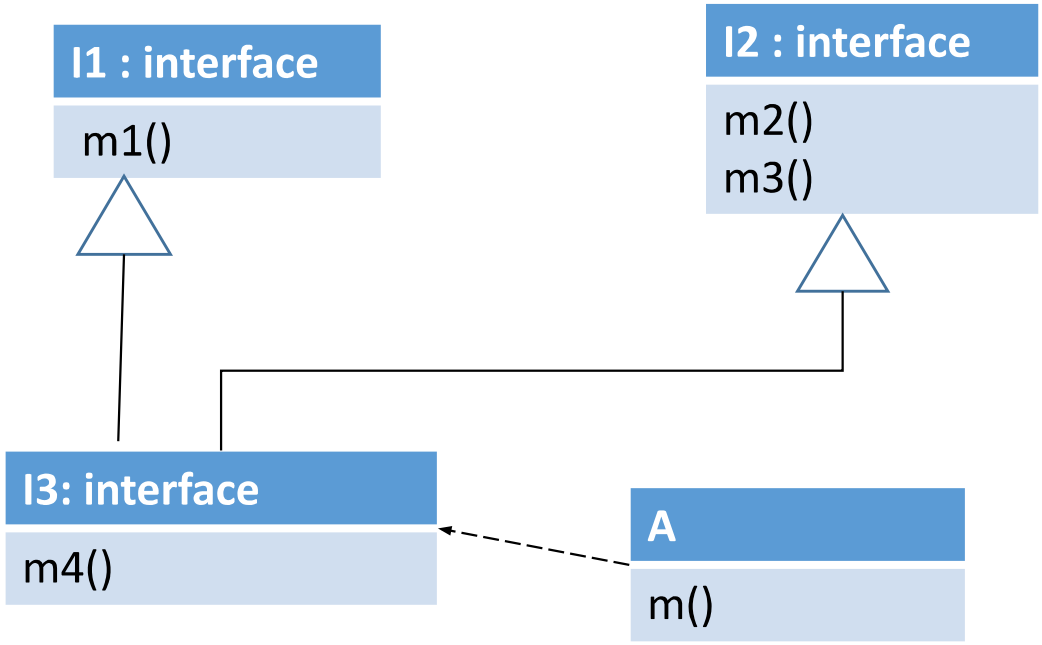
Pas très propre : plante à l'éxécution si en fait A n'implémente pas I1

Solution : on peut vérifier avant : via instanceof (pas seulement pour les interfaces)

```
if(monA instanceof I1 ){
    I1 monI1 = (I1) monA;
}
```

Héritage multiple entre interface

- On peut faire de l'héritage multiple **entre interfaces** (multiple extension)

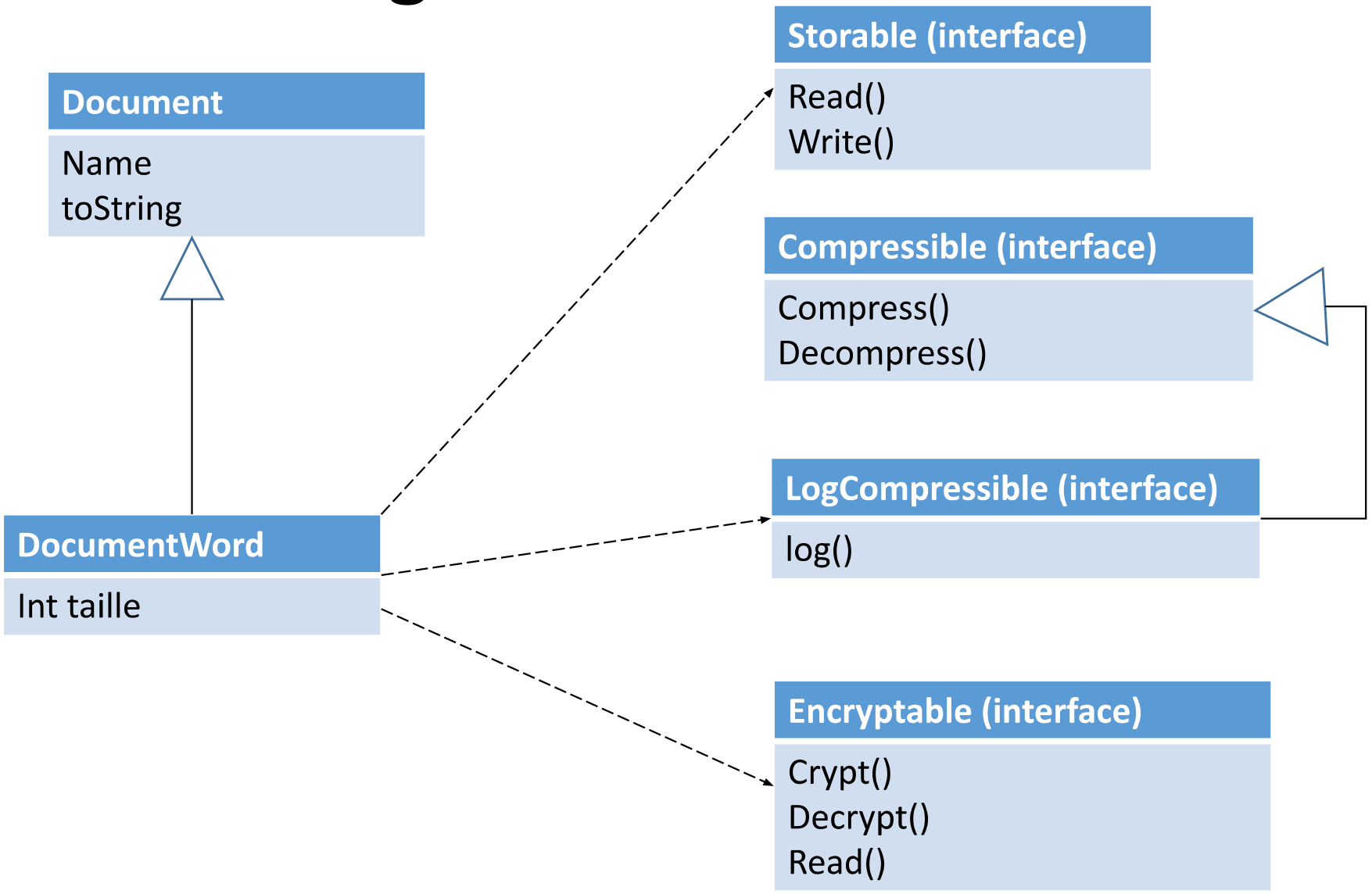


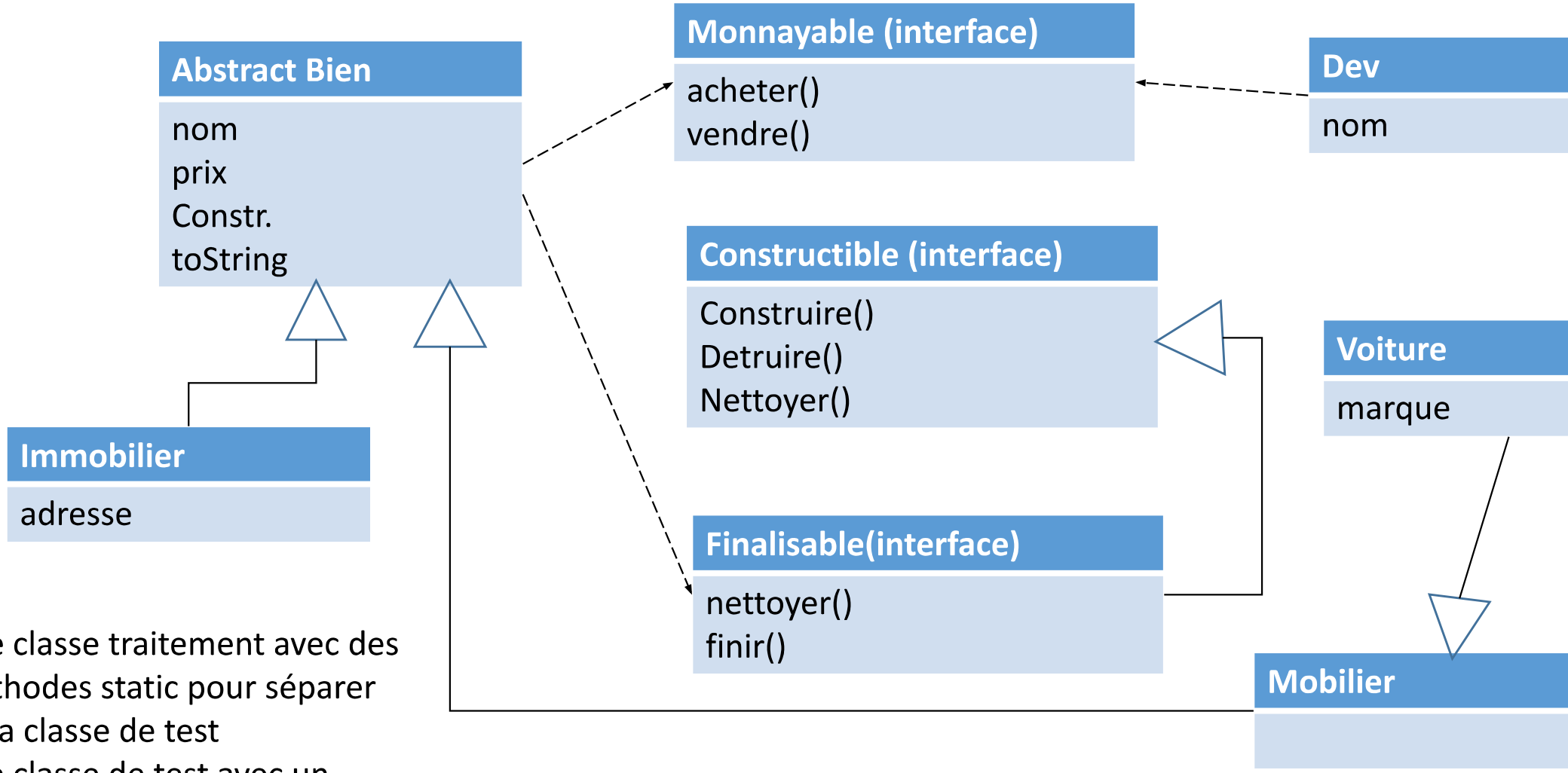

```
public class A extends B implements I3{

public void m1() {…}
public void m2() {…}
public void m3() {…}
public void m4() {…}
public void m() {…}
}
```

```
public interface I3 extends I1, I2 {
public void m4();
}
```

Ex à coder : gestion de documents





Une classe traitement avec des méthodes static pour séparer de la classe de test
Une classe de test avec un main