Génie des Télécommunications et Réseaux (GTR) 3
TD et TP de Java Numéro 4
L'Héritage, les Interfaces, les Classes Abstraites et le Polymorphisme
Dr. KENMOGNE Edith Belise

Rappel de Cours :

1) Le concept d'héritage permets de modéliser les liens de parenté alors que le concept d'interface permets de modéliser des liens comportementaux.

2) Une interface contientdéclarations de méthodes abstraites et de constantes. Toutes les méthodes sont *abstract public* même si non-spécifié. Tous les champs sont *public final static* même si non-spécifié.

3) Une classe doit être déclarée abstraite si elle possède une méthode (introduite dans la dite classe ou acquise par héritage ou acquise par implémentation d'une interface) dont le corps n'est pas défini.

4) Une classe est dite concrête si toutes ses méthodes (introduites dans la dite classe ou acquises par héritage ou acquises par implémentation d'une interface) sont définies.

5) Seules les classes concrêtes sont instanciables. Les classes abstraites ne sont pas instanciables. Les interfaces ne sont pas instanciables.

6) Une reférence nulle n'a pas de membre. De ce fait, (1) une variable contenant une reference nulle (null) ne doit pas recevoir de message, et (2) on ne doit pas accéder à un champ (variable d'instance ou de classe) d'une variable contenant une reférence nulle.

7) La validation ou l'invalidation d'un envoi de message est statique (avant l'exécution du programme). Une variable contenant une référence non nulle (null) doit recevoir **uniquement** les messages prévus dans son type déclaré (classe déclarée ou interface déclaré ou classe de conversion en cas de *cast*). Autrement dit une variable contenant une référence non nulle ne reçoit pas les messages non définis dans son type déclaré.

8) Le traitement d'un envoi de message à un objet se fait en deux temps : **pendant la compilation** (c'est la phase statique du traitement) et **pendant l'exécution** (c'est la phase dynamique du traitement).

Pendant la compilation (phase statique) : Le compilateur s'assure que la classe de l'objet contient bien la méthode (soit directement, soit par héritage, soit par implémentation d'une interface) qu'on applique sur l'objet. Si ce n'est pas le cas, la compilation est arretée.

Pendant l'exécution (phase dynamique) : Le JVM (Java Virtual Machine) qui exécute le programme choisi le version de la méthode à appliquer en fonction du type réel de l'objet. C'est le polymorphisme.Ainsi on recherchera d'abord la version à exécuter dans la classe réelle de l'objet. Si la recherche échoue, on exécutera la version de la classe ascendante la plus proche.

9) Le choix du champs (la résolution des champs) est statique (avant l'exécution du programme). Il est effectuée par le compilateur, en fonction du type déclaré de la variable qui contient la référence.

10) En Java, l'héritage des classes est simple et pas multiple : Une classe fille a une seule classe mère.

11) En Java, l'héritage des interfaces est multiple. Une interface peut hériter de plusieurs autres interfaces. **Une interface ne peut pas implémenter une ou plusieurs interfaces.**

12) En Java, une classe peut implémenter plusieurs interfaces. Mais une classe ne peut hériter que d'une seule classe. Implementer une interface revient à définir (donner un corps à) toutes les méthodes non-définis de la dite interface. Implémenter une méthode abstraite revient à donner un corps à la dite méthode. Une méthode est dite abstraite si elle n'a pas de corps.

13) En Java, la première instruction d'un constructeur d'une classe fille doit être un appel à un constructeur de la classe mère.

14) Le transtypage implicite (automatique) consiste à convertir automatiquement une reférence d'une classe en une reférence d'une classe ascendante. L'échec de cette convertion provoque une levée d'exception de type ClassCastException.

15) Implémentation, re-définition et surchage de méthode : (1) implémenter une méthode abstraite revient à donner un corps à la dite méthode. (2) redéfinir une méthode héritée revient à donner un nouveau corps à la dite méthode. (3) surcharger une méthode revient à introduire une autre méthode de même nom ayant un prototype/entête différent(e).
16) Variables, méthodes et classes déclarées *final* : (1) Une variable (d'instance ou de classe) *final* doit être initialisée à la déclaration. C'est une constante. **(2)** Une méthode (d'instance ou de classe) déclarée *final* ne peut pas être redéfinie dans une classe descendante. **(3)** Une classe déclarée *final* ne peut pas avoir de classe fille.

Exercice 1 : Définir une classe EleveSalarie (élève salarié) comme une classe de la classe Eleve :

1. Introduire un champ pour le salaire.
2. Introduire des accesseurs sur le salaire : ***getSalaire()*** et ***setSalaire(int montant)***
3. Introduire un constructeur qui prend en entrée le nom, une note en mathématique, une note en physique, une note en français, une note en anglais et le salaire.
4. redéfinir le méthode ***toString()*** héritée de la classe mère
5. redéfinir la méthode ***clone()*** héritée de la classe mère.
6. Sachant que deux élèves sont égaux s'ils ont le même matricule, a t'on besoin de redéfinir la méthode héritée ***equals()*** ?
7. On considère le bout de code ci-dessous.
   (1) pourquoi la première instruction est-elle correcte ? **(2)** pourquoi la deuxième instruction est-elle correcte ? Quelle est la version de toString() exécutée ? **(3)** pourquoi la troisième instruction n'est elle pas correcte ? **(4)** Pourquoi la quatrième instruction n'est elle pas correcte ? **(5)** pourquoi la cinquième instruction est elle correcte ? **(6)** Pourquoi la sixième instruction est elle correcte ? **(7)** Pourquoi la septième instruction n'est pas correcte ? (8) Pourquoi la huitième instruction est-elle correcte ?
   1. Eleve e= new EleveSalarie(KINGUE, 12, 14, 15, 16, 700000) ;
   2. String s=e.toString() ;
   3. int val= e.getSalaire() ;
   4. e.setSalaire(1000000) ;
   5. int val1 = ((EleveSalarie) e).getSalaire() ;
   6. ((EleveSalarie) e).setSalaire(1000000) ;
   7. EleveSalarie es=e ;
   8. EleveSalarie es1= (EleveSalarie) e ;
8. Écrire une méthode de classe ***public static int nbreEleveSalarie(Eleve[] tab)*** de classe qui prend en entrée un tableau d'élèves et retourne le nombre d'élèves salariés. On utilisera ***instanceOf***.

Exercice 2 : Analyse de Code
Nous considérons ici les définitions de classes et d'interfaces suivantes.

```
public class A implements I1, I2 {
   String a=new String("Classe A") ;
   int mi1() { ... }
   int mi2() { ... }
   int ma() { ... }
   ...
}
```

```
public class B extends A implements I {
   String a=new String("Classe B") ;
   int b=20 ;
   int mi2() { ... }
   int mi() { ... }
   int mi3(){ ... }
   int mi4() { ... }
   int mb() { ... }
}
```

public class C extends B { *String a=new String("Classe C") ;* *String b=new String("C") ;* *float c ;* *int mi1() { ... }* *int mi4() { ... }* *int mc() { ... }* *int ma() { ... }* *int mb() { ... }* *...* *}*	*public interface I1 {* *int mi1() ;* *}*
public interface I2 { *int mi2() ;* *}*	*public interface I extends I3, I4 {* *String a=new String("Interface I") ;* *int mi() ;* *}*
public interface I3 { *int mi3() ;* *}*	*public interface I4 {* *int mi4() ;* *}*

1) Notion de diagramme de classe et d'interface : Donner le diagramme de classe et d'interface en faisant apparaître clairement les liens d'héritage (entre classes et entre interfaces) et d'implémentation d'interface. Sachant que **les interfaces introduisent une notion de typage**, répondre aux questions suivantes :
(1.1) Un objet de A est-il un I1 et un I2 ?
(1.2) Un objet de B est-il un A, un I1, un I2, un I, un I3, un I4 ?
(1.3) Un objet de C est-il un A, un B, un I1, un I2, un I, un I3, un I4 ?
(1.4) Un objet de A est-il un I ?, un I3 ?, un I4 ?
(1.5) Un I est-il un I1, un I2 ?

2) Notion de Classe Concrête et Notion de Classe Abstraite :
(2.1) Pourquoi peut-on dire que les classes A, B et C sont concrêtes ?
(2.2) La classe A reste-elle concrête si on supprime dans A la définition de la méthode *mi1()* ? Même question pour *mi2()*.
(2.3) La classe B reste-elle concrête si on supprime dans B la définition de la méthode *mi()* ? Même question pour *mi3(), mi4()* et *mi2()*.
(2.4) Peut-on rendre la classe C abstraite en supprimant une des méthodes définies dans C ?

3) Notion d'héritage de méthode, notion de redéfinition de méthode et notion d'implémentation de méthode :
(3.1) Un objet de la classe B a combien de version de champ ayant le nom a ?
(3.2) Un objet de la classe C a combien de version de champ ayant le nom a ?
(3.3) Un objet de la classe C a combien de versions de champ ayant le nom b ?
(3.4) Quelles sont les méthodes (1) introduites dans B ? (2) héritées par B ? (3) implémentées par B via une interface ?, et (4) redéfinies dans B ?
(3.5) Même question qu'en (3.1) pour la classe C.
(3.6) Même question qu'en (3.1) pour la classe A.
(3.7) Quelles sont les champs hérités par chacune des classes B et C ?

4) Traitement d'un envoi de message ou résolution de méthode

(4.1) Dans le bout de code qui suit, dire si chaque envoi de message est correct ou incorrect (contrôle statique pendant la compilation). Dans le cas où c'est correcte, préciser la version de la méthode qui est exécutée.

1. A a = new C() ;
2. a.ma() ;
3. a.mb() ;
4. a.mc() ;
5. a.mi1() ;
6. a.mi2() ;
7. a.mi3() ;
8. a.mi4() ;

(4.2) Dans le bout de code qui suit, dire si chaque envoi de message est correct ou incorrect **(contrôle statique pendant la compilation).**

1. I1 a1=new A() ;
2. a1.mi1() ;
3 a1.mi2() ;
4 a1.ma() ;
5 ((A) a1).ma() ;

(4.3) Dans le bout de code qui suit, dire si chaque affectation est correct ou incorrect **(contrôle statique pendant la compilation).** Justifiez votre réponse.

1. I1 i1=new A() ;
2. I i = new A() ;
3. I ii = new C() ;
4. I3 i3=ii ;
5. I4 i4=ii ;

5) La résolution du champ (variable d'instance ou variable de classe) à accéder : Dans le bout de code qui suit dire pour chaque accès au champ a est correct ou incorrect. Dans la cas où c'est correct dire ce qui esr affiché.

1. I1 i1 = new A() ;
2. A a1 = new A() ;
3 A a2= new new C() ;
4. System.out.println(**i1.a**) ;
5. System.out.println(**((A) i1).a**) ;
6. System.out.println(**a1.a**) ;
7. System.out.println(**a2.a**) ;
8. System.out.println((**(B) a2).a**) ;
9. System.out.println((**(A) a2).a**) ;