

RÉPUBLIQUE DU CAMEROUN

Paix – Travail – Patrie

MINISTRE DE L'ENSEIGNEMENT
SUPERIEUR

UNIVERSITÉ DE MAROUA

ECOLE NATIONAL SUPERIEUR
POLYTECHNIQUE DE MAROUA

DEPARTEMENT D'INFORMATIQUE ET
TELECOMMUNICATION



REPUBLIC OF CAMEROUN

Peace – Work – Fatherland

MINISTRY OF HIGHER EDUCATION

THE UNIVERSITY OF MAROUA

NATIONAL ADVANCED SCHOOL OF
ENGINEERING OF MAROUA

DEPARTMENT OF COMPUTER
SCIENCE AND
TELECOMMUNICATION

TRAVAIL PERSONNEL DE L'ETUDIANT

UE 316 : Programmation Orientée Objet

TD : HÉRITAGE, MASQUAGE DE MÉTHODE

Membres du groupe

Noms et prénom	Matricule
KAMDEM DEFO BAKAM ONESIME	22ENSPM474
NOUGA MFANGNIA ANGE MERVEILLE	24ENSPM340
TONFEU NJODIEU STYVE FRIEDEL	22ENSPM490

Niveau et filière : IC-3 Informatique et télécommunication (INFOTEL)

Option : Génie Logiciel

Sous la direction de : M. HAMAYADJI ABDOUL AZIZ

Année académique: 2024-2025

Table des matières

Table des matières	i
Exercice 3 – Quelles méthodes hérite-t-on ? Que faut-il redéfinir ?	1
Exercice 4 : Pour chacun des appels de méthode ci-dessous, disons s'il va être compilé correctement et auquel cas, quelle méthode est appelée effectivement à l'exécution ?.....	2
Exercice 5 : Disons si la classe Dessin définie précédemment peut contenir des rectangles inclinés et si les méthodes surface, contains et hull de la classe Dessin fonctionnent encore correctement ?.....	4
Exercice 6 : Définir une méthode String toString() dans la classe Rectangle ? Est-ce en fait une définition ou une redéfinition ? Est-il nécessaire de la redéfinir dans la classe SlantedRectangle ?	4
Exercice 8 : On considère les définitions de class suivantes :	5
Exercice 11 : Qu'affiche le code suivant :	8
Exercice 12 : Dans la classe Rectangle a été définie une méthode boolean contains(Rectangle). Cette méthode doit-elle être redéfinie dans la classe SlantedRectangle ? Quels cas ne sont pas couverts par cette redéfinition ? On ajoute alors une méthode boolean contains(SlantedRectangle) dans les classes Rectangle et SlantedRectangle. Quels cas ne sont toujours pas couverts par ces ajouts ?.....	9
Exercice 13 : On considère les définitions de classes suivantes :	10

Exercice 3 – Quelles méthodes hérite-t-on ? Que faut-il redéfinir ?

Solution

En Java, une classe fille hérite de toutes les méthodes publiques et protégées de sa superclasse. Ainsi, `SlantedRectangle` hérite des méthodes suivantes de `Rectangle` :

- Accesseurs et mutateurs : `getPointbasgauche()`, `setPointbasgauche()`, `getLargeur()`, `setLargeur()`, `getHauteur()`, `setHauteur()`
- Méthodes utilitaires : `surface()`, `translate(float x, float y)`, `sameAs(Rectangle r)`
- Méthodes de test : `contains(Point p)`, `contains(Rectangle r)`
- Méthodes statiques : `getNbr()`, `setNbr(int n)`, `hull(Rectangle[] rec)`

Mais : certaines méthodes ne sont plus correctes avec un rectangle incliné.

Par exemple :

- `contains(Point p)` - Car le calcul doit prendre en compte la rotation
- `contains(Rectangle r)` - Pour prendre en compte l'angle
- `sameAs(Rectangle r)` - Pour vérifier l'égalité des angles
- `hull (Rectangle[])` - Pour prendre en compte les rectangles inclinés dans le calcul du rectangle englobant

Donc :

- On va **redéfinir (override)** les méthodes qui ne fonctionnent plus correctement dans un rectangle incliné.

Pour le deuxième volet de la question, qui consiste à redéfinir les méthodes qui ne sont plus correctes, nous les avons implémentées dans la classe **`SlantedRectangle`**.

Exercice 4 : Pour chacun des appels de méthode ci-dessous, disons s'il va être compilé correctement et auquel cas, quelle méthode est appelée effectivement à l'exécution ?

Solution

1. Déclarations des objets

Point p = new Point(1, 2);

Rectangle r = new Rectangle(p, 2, 3);

Rectangle t = new SlantedRectangle(p, 2, 3);

SlantedRectangle s = new SlantedRectangle(p, 2, 3);

- **p** : Instance de la classe Point.
- **r** : Instance de la classe Rectangle.
- **t** : Référence de type Rectangle pointant vers une instance de SlantedRectangle.
- **s** : Instance de la classe SlantedRectangle.

2. Appels de méthodes

a. *System.out.println(r.surface());*

- **Compilation** : Réussie.
- **Exécution** : Appelle la méthode surface() définie dans Rectangle.
- **Sortie** : Affiche la surface du rectangle r.

b. *r.rotate(2);*

- **Compilation** : Erreur.
- **Raison** : La méthode rotate(float) n'est pas définie dans la classe Rectangle.
- **Solution** : Ajouter une méthode rotate(float) dans Rectangle ou effectuer un transtypage si l'objet est effectivement une instance de SlantedRectangle.

c. *System.out.println(r.contains(p));*

- **Compilation** : Réussie.
- **Exécution** : Appelle la méthode contains(Point) définie dans Rectangle.
- **Sortie** : Affiche true ou false selon que le point p est contenu dans le rectangle r.

d. System.out.println(t.surface());

- **Compilation** : Réussie.
- **Exécution** : Appelle la méthode surface() de Rectangle, car SlantedRectangle n'a pas redéfini cette méthode.
- **Sortie** : Affiche la surface du rectangle t.

e. t.rotate(2);

- **Compilation** : Erreur.
- **Raison** : La référence t est de type Rectangle, qui ne possède pas de méthode rotate(float).
- **Solution** : Effectuer un transtypage vers SlantedRectangle :

f. System.out.println(t.contains(p));

- **Compilation** : Réussie.
- **Exécution** : Appelle la méthode contains(Point) redéfinie dans SlantedRectangle, grâce au polymorphisme dynamique.
- **Sortie** : Affiche true ou false selon que le point p est contenu dans le rectangle incliné t.

g. System.out.println(s.surface());

- **Compilation** : Réussie.
- **Exécution** : Appelle la méthode surface() héritée de Rectangle, car SlantedRectangle ne l'a pas redéfinie.
- **Sortie** : Affiche la surface du rectangle incliné s.

h. s.rotate(2);

- **Compilation** : Réussie.
- **Exécution** : Appelle la méthode rotate(float) définie dans SlantedRectangle.
- **Effet** : Modifie l'angle d'inclinaison du rectangle s.

i. `System.out.println(s.contains(p));`

- **Compilation** : Réussie.
- **Exécution** : Appelle la méthode `contains(Point)` redéfinie dans `SlantedRectangle`.
- **Sortie** : Affiche `true` ou `false` selon que le point `p` est contenu dans le rectangle incliné `s`.

Exercice 5 : Disons si la classe `Dessin` définie précédemment peut contenir des rectangles inclinés et si les méthodes `surface`, `contains` et `hull` de la classe `Dessin` fonctionnent encore correctement ?

Solution

La class `Dessin` peut contenir des `SlantedRectangle`. Ainsi les méthodes `surface`, `contains` et `hull` fonctionneront toujours correctement, car elles sont définies dans la class `Rectangle` et d'après l'exercice 3 ces méthodes ont été redéfinies dans la class `SlantedRectangle`. Donc en conclusion peuvent être utilisées par les objets de type `SlantedRectangle`.

Pour les méthodes :

- **surface()** : fonctionne correctement car elle fait appel à la méthode `surface()` de chaque rectangle, et le polymorphisme assure que la méthode appropriée est appelée.
- **translate()** : fonctionne correctement car la méthode `translate()` est héritée sans modification.
- **hull()** : pourrait ne pas fonctionner parfaitement car elle ne prend pas en compte la rotation des rectangles inclinés. Le rectangle englobant sera calculé en utilisant les dimensions et positions sans considérer l'angle d'inclinaison.

Exercice 6 : Définir une méthode `String toString()` dans la classe `Rectangle` ? Est-ce en fait une définition ou une redéfinition ? Est-il nécessaire de la redéfinir dans la classe `SlantedRectangle` ?

Solution

- La définition a été faite dans le code au niveau de la classe `Rectangle`.

- La méthode **toString()** dans la classe Rectangle est une redéfinition de la méthode **toString()** de la classe Object, qui est la classe mère de toutes les classes en Java.
- Oui, il est nécessaire de redéfinir **toString()** dans la classe SlantedRectangle pour inclure l'information sur l'angle de rotation qui n'est pas présente dans la classe Rectangle.

Exercice 8 : On considère les définitions de class suivantes :

```
class A {
    void f(A o) {
        System.out.println("void f(A o) dans A");
    }
}
class B extends A {
    void f(A o) {
        System.out.println("void f(A o) dans B");
    }
}
```

Consigne :

Disons ce qu'affiche ce code

```
A a = new A();
A ab = new B();
B b = new B();

a.f(a);
a.f(ab);
a.f(b);
ab.f(a);
ab.f(ab);
ab.f(b);
b.f(a);
b.f(ab);
b.f(b);
```

Après avoir compilé ce code ci-dessus en créant une class A, une class B et une classe Main nous avons eu le résultat suivant :

void f(A o) dans A

void f(A o) dans A

void f(A o) dans A

void f(A o) dans B

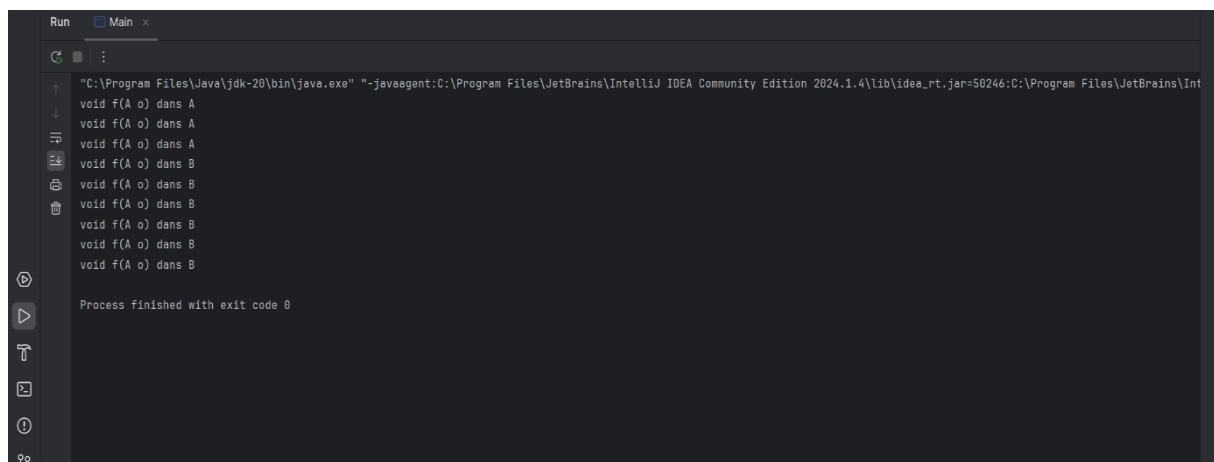
void f(A o) dans B

void f(A o) dans B

void f(A o) dans B

void f(A o) dans B

void f(A o) dans B



```
Run Main x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.4\lib\idea_rt.jar=50246:C:\Program Files\JetBrains\Int
void f(A o) dans A
void f(A o) dans A
void f(A o) dans A
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
Process finished with exit code 0
```

Exercice 9

En ajoutant la méthode suivante :

```
void f(B o) {
    System.out.println("void f(B o) dans B");
}
```

dans la classe B S'agit-il d'une redéfinition ou d'une surcharge ? qu'affiche alors le fragment du programme de l'exercice 8

Solution

1. Il s'agit d'une **surcharge**. La méthode `f(B o)` a le même nom que `f(A o)` mais un paramètre de type différent (B au lieu de A). En Java, la surcharge permet d'avoir plusieurs méthodes avec le même nom mais des signatures différentes au sein de la même classe.
2. Le fragment de programme de l'exercice 8 affiche donc :

```
void f(A o) dans A
void f(A o) dans A
void f(A o) dans A
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(B o) dans B
```



```
Run Main
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.4\lib\idea_rt.jar=50248:C:\Program Files\Jet
void f(A o) dans A
void f(A o) dans A
void f(A o) dans A
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(A o) dans B
void f(B o) dans B
Process finished with exit code 0
```

Exercice 10

On ajoute finalement à la classe A la méthode suivante :

```
void f(B o) {
    System.out.println("void f(B o) dans A");
}
```

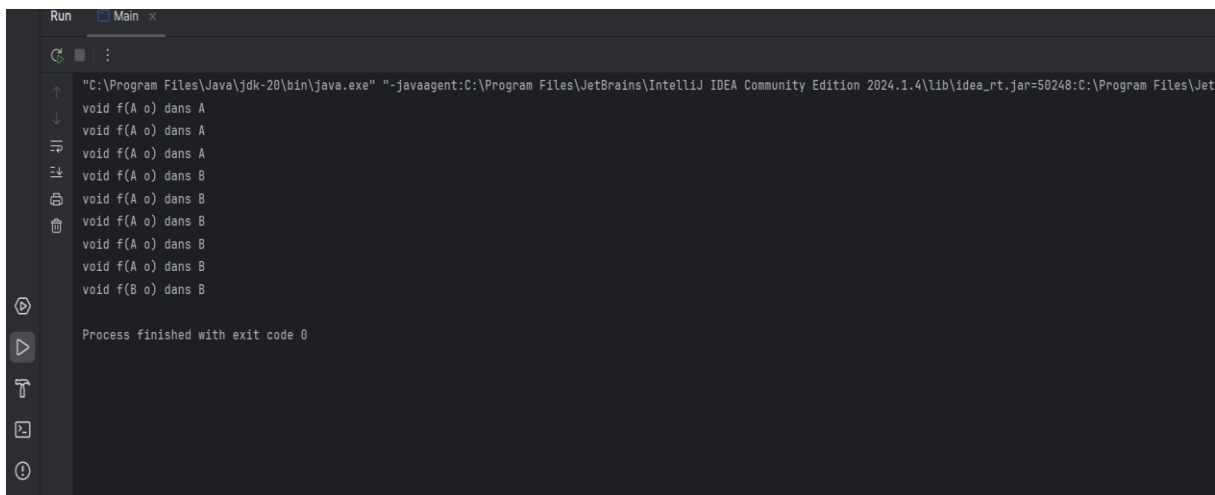
Consigne :

Est-ce une redéfinition ou une surcharge ? Qu'affiche alors le fragment de programme de l'exercice 8 ?

Solution :

1. Cette méthode constitue une **surcharge** de la méthode f(A o) existante dans la classe A
2. Le fragment du code affichera :

```
void f(A o) dans A  
void f(A o) dans A  
void f(A o) dans A  
void f(A o) dans B  
void f(A o) dans B  
void f(A o) dans B  
void f(A o) dans B  
void f(A o) dans B  
void f(A o) dans B  
void f(B o) dans B
```



```
Run Main x  
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.4\lib\idea_rt.jar=50248:C:\Program Files\Jet  
void f(A o) dans A  
void f(A o) dans A  
void f(A o) dans A  
void f(A o) dans B  
void f(A o) dans B  
void f(A o) dans B  
void f(A o) dans B  
void f(A o) dans B  
void f(A o) dans B  
void f(A o) dans B  
void f(B o) dans B  
Process finished with exit code 0
```

Exercice 11 : Qu'affiche le code suivant :

```
System.out.println(a instanceof A);  
System.out.println(ab instanceof A);  
System.out.println(b instanceof A);  
  
System.out.println(a instanceof B);  
System.out.println(ab instanceof B);  
System.out.println(b instanceof B);
```

Solution

Après ajout de ce code dans la classe main nous avons ce résultat :

```

C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.4\lib\idea_rt.jar=50260:C:\Program Files\JetBrains\Int
true
true
true
true
false
true
true

Process finished with exit code 0

```

Exercice 12 : Dans la classe Rectangle a été définie une méthode boolean contains(Rectangle). Cette méthode doit-elle être redéfinie dans la classe SlantedRectangle ? Quels cas ne sont pas couverts par cette redéfinition ? On ajoute alors une méthode boolean contains(SlantedRectangle) dans les classes Rectangle et SlantedRectangle. Quels cas ne sont toujours pas couverts par ces ajouts ?

Solution

Oui, la méthode boolean `contains(Rectangle)` doit être redéfinie dans la classe `SlantedRectangle` car le calcul pour déterminer si un rectangle est contenu dans un rectangle incliné est différent.

Les cas non couverts par cette redéfinition sont les interactions entre différents types de rectangles :

- Un rectangle normal contenant un rectangle incliné
- Un rectangle incliné contenant un rectangle normal

En ajoutant la méthode boolean `contains(SlantedRectangle)` dans les classes `Rectangle` et `SlantedRectangle`, nous couvrons ces cas supplémentaires. Cependant, il reste des cas non couverts :

- Si de nouvelles sous-classes de Rectangle sont créées à l'avenir (par exemple, ColoredRectangle), il faudrait également ajouter des méthodes spécifiques pour elles.

Exercice 13 : On considère les définitions de classes suivantes :

```
class C {  
    char ch = 'C';  
    char getCh() { return ch; }  
}  
class D extends C {  
    char ch = 'D';  
    char getCh() { return ch; }  
}
```

Consigne :

Disons ce qu'affiche ce code :

```
C c = new C();  
C cd = new D();  
D d = new D();  
System.out.println(c.ch);  
System.out.println(c.getCh());  
System.out.println(cd.ch);  
System.out.println(cd.getCh());  
System.out.println(d.ch);  
System.out.println(d.getCh());
```

Solution :

Après ajout de ce code dans la classe main nous avons ce résultat :

```
PS D:\ENSPM\S6\P00\TPE_POO_GLO_GROUPE2\Exercice8_9_10_11> javac MainExercice8_9_10_11_13.java  
PS D:\ENSPM\S6\P00\TPE_POO_GLO_GROUPE2\Exercice8_9_10_11> java MainExercice8_9_10_11_13  
C  
C  
C  
D  
D  
D  
PS D:\ENSPM\S6\P00\TPE_POO_GLO_GROUPE2\Exercice8_9_10_11> █
```