**Héritage, masquage de méthode**

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

**Objectif :**  
Identifier quelles méthodes de Rectangle sont héritées dans SlantedRectangle, et **les redéfinir si nécessaire**.

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)**
* 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) : le test d’appartenance ne marche plus en supposant que les côtés sont parallèles aux axes.
* contains(Rectangle r) : le test d’appartenance ne marche plus en supposant que les côtés sont parallèles aux axes.

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 ?

### ****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 SlantedRectanle. Ainsi les méthodes surface, contains et hull fonctionneront toujours correctement, car elles sont definies dans la class rectangles 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.

**Explication détaillée**

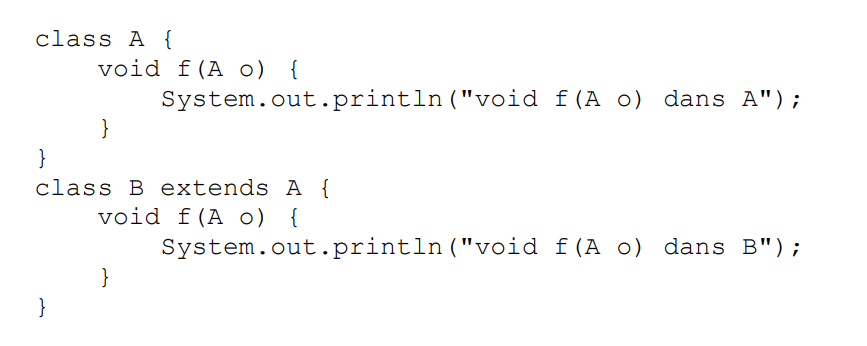
1. **Stockage des objets** : La classe Dessin utilise un tableau de Rectangle pour stocker les objets. Étant donné que SlantedRectangle est une sous-classe de Rectangle, les instances de SlantedRectangle peuvent être stockées dans ce tableau sans problème.
2. **Méthode surface()** : La méthode surface() dans Dessin appelle la méthode surface() de chaque Rectangle. Si SlantedRectangle redéfinit cette méthode pour tenir compte de l'inclinaison, alors la surface sera calculée correctement pour les rectangles inclinés. Sinon, la méthode héritée de Rectangle sera utilisée, ce qui pourrait ne pas refléter l'inclinaison.
3. **Méthode translate(float dx, float dy)** : Cette méthode appelle la méthode translate de chaque Rectangle. Si SlantedRectangle hérite ou redéfinit correctement cette méthode, la translation sera appliquée correctement aux rectangles inclinés.
4. **Méthode hull()** : La méthode hull() calcule le rectangle englobant de tous les rectangles dans le dessin. Si SlantedRectangle redéfinit correctement les méthodes nécessaires pour fournir ses dimensions et sa position, alors le calcul du rectangle englobant inclura correctement les rectangles inclinés.

**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**

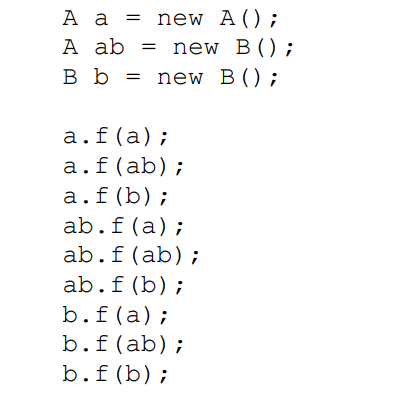
* La définition a été faite dans le code au niveau de la classe Rectangle.
* Il s'agit ici d'une **redéfinition** de la méthode héritée de Object, adaptée aux spécificités de la classe Rectangle.
* Concernant la classe SlantedRectangle, qui hérite de Rectangle et introduit une nouvelle propriété angle, il est **nécessaire** de **redéfinir** également la méthode toString(). Cette redéfinition permettra d'inclure la valeur de l'angle dans la représentation textuelle de l'objet, assurant ainsi une description complète et précise. (Nous l’avons fait dans le code de la classe SlantedRectangle)

**Exercice 8 :** On considere les definitions de class suivantes :



**Consigne :**

Disons ce qu’affiche ce code



Apres avoir compile 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

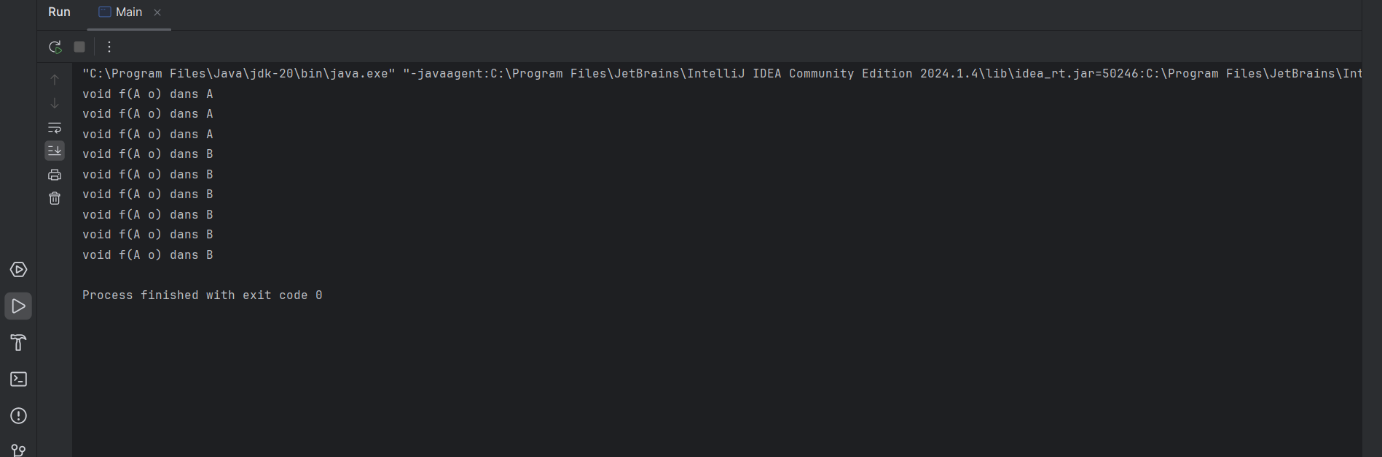
void f(A o) dans B

void f(A o) dans B

void f(A o) dans B

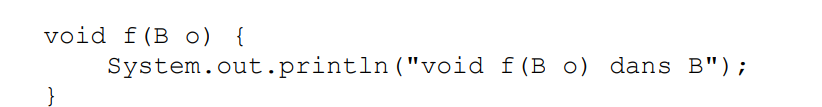
void f(A o) dans B

void f(A o) dans B



**Exercice 9**

En ajoutant la méthode suivante :



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

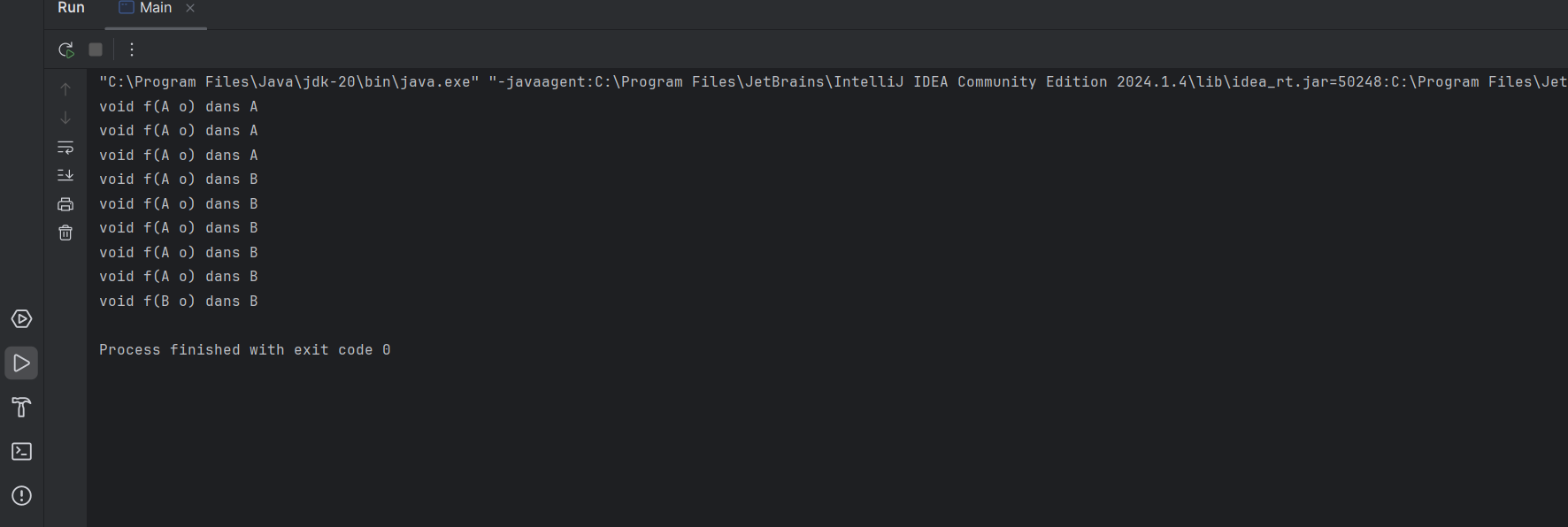
void f(A o) dans B

void f(A o) dans B

void f(A o) dans B

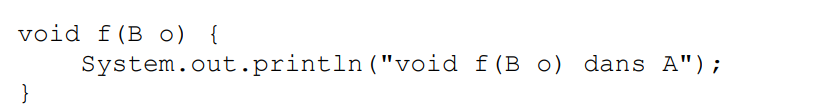
void f(A o) dans B

void f(B o) dans B



**Exercice 10**

On ajoute finalement à la classe A la méthode suivante :



**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

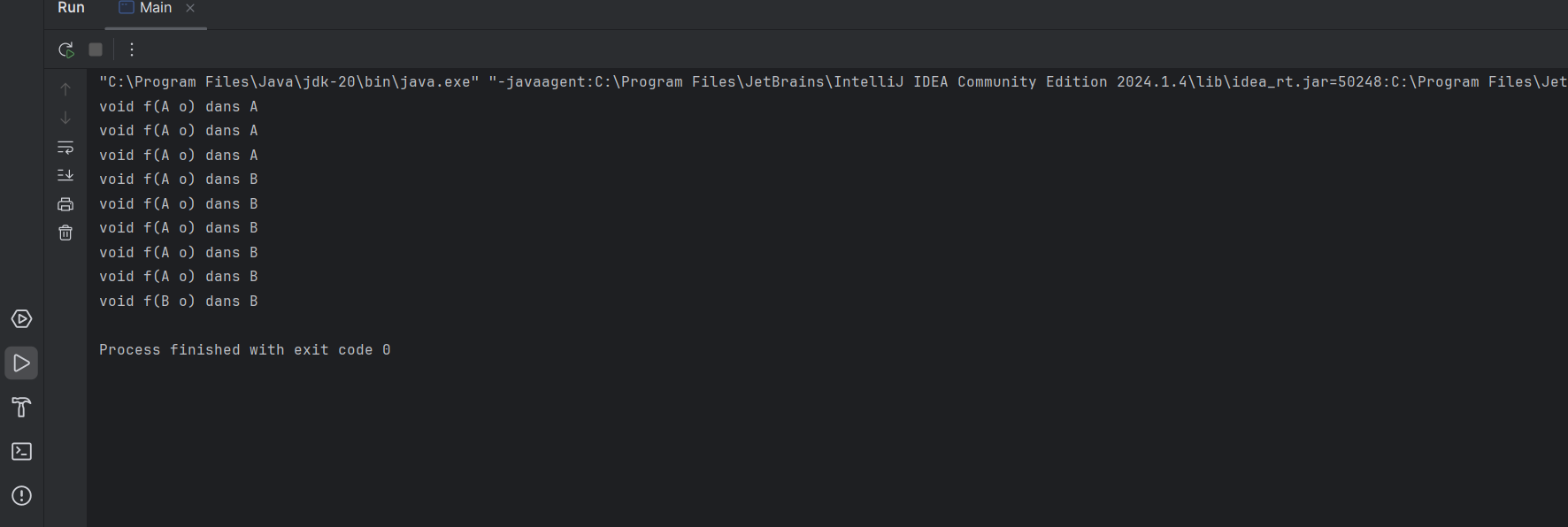
void f(A o) dans B

void f(A o) dans B

void f(A o) dans B

void f(A o) dans B

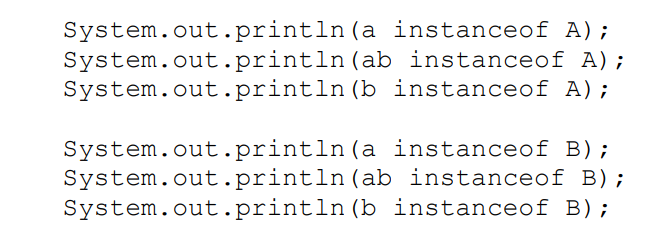
void f(B o) dans B



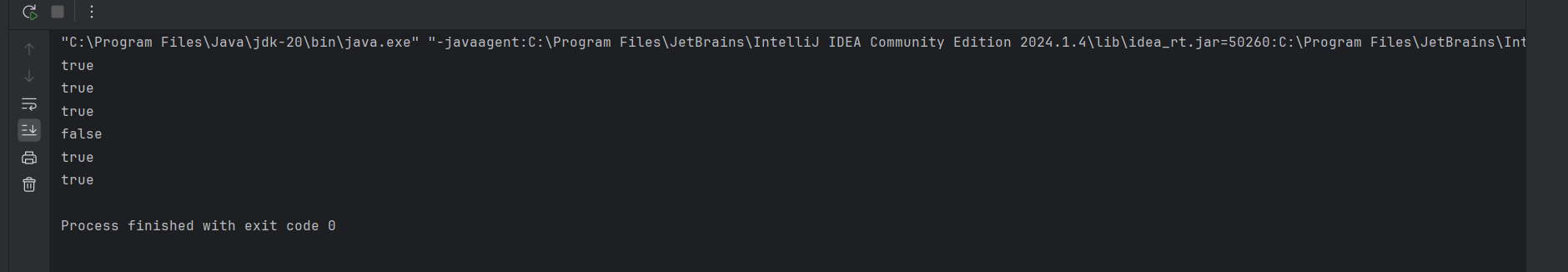
**Exercice 11 :**

**Consigne :**

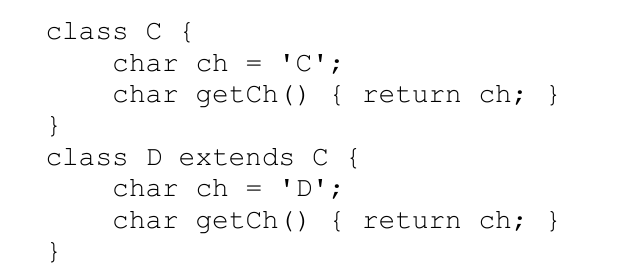
**Qu’affiche le code suivant :**

****

**Solution**

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

**Exercice 13 :** On considère les définitions de classes suivantes



**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 :**

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

