

TP 2

Formes**1 – Présentation du TP***Objet du TP*

Ce TP a pour objectif de modéliser, manipuler et dessiner des formes géométriques en Java.

Compétences travaillées dans ce TP

- Ecriture d'une classe de zéro
- Héritage
- Polymorphisme d'héritage
- Classes abstraites
- Interfaces

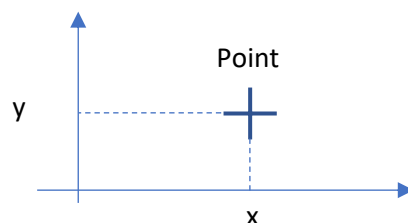
2 – Démarrage du TP

Suivez la procédure de démarrage du TP 1, en remplaçant le nom du projet « tp1 » par « tp2 ».
Toutes les autres actions sont identiques.

3 – Concept de Point

Nous allons commencer par implémenter une classe très simple modélisant un point dans un espace à 2 dimensions.

Un point est caractérisé par deux coordonnées, x et y.

*Création de la classe*

Ajoutez une classe appelée « Point » à votre projet. Pour rappel, ceci peut être fait par : clic droit sur le répertoire « src » de la partie gauche, puis sélection de New > Java Class.

Définition des attributs

A l'intérieur de la classe Point, déclarez les attributs suivants, avec une visibilité privée et un type adéquat.

Nom	Type	Description
x	Nombre entier	Coordonnée du point suivant l'axe x, en pixels
y	Nombre entier	Coordonnée du point suivant l'axe y, en pixels

Implémentation du constructeur

Implémentez le constructeur :

```
public Point(int x, int y)
```

qui initialise les attributs x et y à partir de la valeur des paramètres.

Rappel : pour éviter toute confusion entre les paramètres et les attributs préfixez les attributs de la classe par le désignateur « this » dans le code des méthodes.

Implémentation des accesseurs

Implémentez les accesseurs suivants :

Attribut	Accesseur en lecture	Accesseur en écriture
x	getX	setX
y	getY	setY

Comme vous pouvez le constater, la convention de nommage pour les accesseurs est la suivante :

- « get + nom de l'attribut » pour les accesseurs en lecture,
- « set + nom de l'attribut » pour les accesseurs en écriture.

Le tout écrit en « camel case ».

Test des accesseurs

Le code suivant :

```
Point p = new Point(100, 200);  
System.out.println(p.getX());  
p.setX(300);  
System.out.println(p.getX());
```

doit produire l'affichage :

```
100  
300
```

Faites ce test dans la méthode main().

De manière similaire, testez le bon fonctionnement de getY() et setY().

4 – Surcharge de la méthode toString()

Exécutez le code suivant dans le main() :

```
Point p = new Point(100, 200);  
System.out.println(p);
```

Vous devriez voir un affichage de la forme "Point@" suivi de chiffres et de lettres.

Explication : Tous les objets Java possèdent une méthode toString(), héritée de la classe Object.

Cette méthode renvoie la représentation sous la forme d'une chaîne de caractères d'une instance. Elle est appelée à chaque fois que l'on tente de convertir une instance en chaîne, par exemple lors de la concaténation d'une instance avec une chaîne ou alors quand on appelle System.out.println() sur cette instance.

De base, cette méthode renvoie une chaîne de la forme "Nom de la classe@[hashcode](#)", ce qui pourrait donner ici quelque chose comme "Point@4c873330". Pas très parlant !

Surcharger cette méthode dans une classe peut s'avérer bien pratique pour l'utilisateur de cette classe. Illustrons ceci sur la classe Point.

Ajoutez la surcharge de toString() suivante dans la classe Point :

```
public String toString(){  
    return "(" + x + ", " + y + ")";  
}
```

Relancez le code précédent. Il devrait maintenant produire l'affichage :

```
(100, 200)
```

On peut à présent très facilement incorporer la représentation d'un point dans une chaîne. Par exemple, si on exécute le code suivant :

```
Point p = new Point(100, 200);  
String s = "Point coordinates : " + p;
```

Alors la chaîne s contiendra : "Point coordinates : (100, 200)". Vérifiez ceci.

5 – Concept de forme

Nous allons maintenant implémenter le concept de forme.

Une forme possède un centre, modélisé par un point.

Implémentation

- Créez une classe Shape qui possède un attribut de type Point appelé center.
- Implémentez le constructeur :

```
public Shape(Point center)
```

- Implémentez les accesseurs getCenter() et setCenter()

- Implémentez une méthode `print()` qui affiche :

```
Shape - center : ({x}, {y})
```

où `{x}` et `{y}` sont à remplacer par les coordonnées du centre.

Indication : souvenez-vous que `toString()` est surchargée dans `Point` !

Test

Le code suivant :

```
Point center = new Point(100, 200);  
Shape s = new Shape(center);  
s.print();
```

doit produire l’affichage :

```
Shape - center : (100, 200)
```

Note : Si le point `center` n’est utilisé que dans le constructeur de `Shape`, il est inutile de créer une variable intermédiaire. On peut instancier directement le point dans l’appel au constructeur, comme ceci :

```
Shape s = new Shape(new Point(100, 200));  
s.print();
```

Testez cette nouvelle version du code et vérifiez que vous obtenez le même résultat.

6 – Concept de cercle

Un cercle est une forme caractérisée par son centre et son rayon.

Nous allons modéliser ceci par une classe `Circle` qui hérite de `Shape` et qui possède un attribut `radius` modélisant son rayon (nombre entier, en pixels).

Implémentation de la classe `Circle`

- Créez la classe `Circle`
- Indiquez que `Circle` hérite de `Shape`
- Implémentez le constructeur :

```
public Circle(Point center, int radius)
```

Contrainte : Ce constructeur doit uniquement initialiser l’attribut `radius`, et déléguer l’initialisation de `center` à la classe `Shape`.

- Implémentez les accesseurs `getRadius()` et `setRadius()`.

Test de la méthode print()

A votre avis, si on exécute le code suivant :

```
Circle c = new Circle(new Point(100, 200), 50);  
c.print();
```

Que va-t-on obtenir ? Une erreur ? Un affichage ? Lequel le cas échéant ?

Surcharge de la méthode d'affichage

Nous allons redéfinir la méthode d'affichage dans la classe Circle, afin d'afficher le rayon en plus des coordonnées du centre.

Dans la classe Circle, implémentez une méthode :

```
public void print()
```

qui affiche :

```
Circle - center : ({x}, {y}), radius : {r}
```

où {x} et {y} sont à remplacer par les coordonnées du centre du cercle et {r} par son rayon.

Élément de réflexion : comment rendre l'attribut center accessible à la classe Circle ?

Test de la surcharge de l'affichage

Relancez le code précédent. Vous devriez observer le résultat suivant dans la console :

```
Circle - center : (100, 200), radius : 50
```



Point clé

Si une surcharge existe dans une sous-classe, alors elle sera appelée à la place de la méthode de la super-classe.

Modifiez le rayon du cercle et vérifiez qu'elle a bien été effectuée en appelant la méthode print().

Transtypage ascendant (upcast)

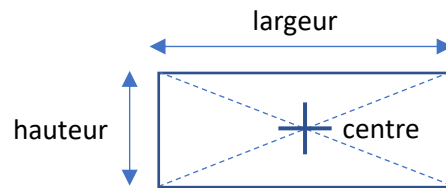
A votre avis, si on change le type de la variable c de Circle à Shape :

```
Shape c = new Circle(new Point(100, 200), 50);  
c.print();
```

Va-t-on obtenir un affichage du type "Shape - ..." ou "Circle - ..." ? Vérifiez votre réponse en modifiant et en exécutant le code.

7 – Concept de rectangle

Un rectangle est une forme caractérisée par son centre, sa largeur (dimension selon l'axe x) et sa hauteur (dimension selon l'axe y).



Nous allons modéliser ceci par une classe Rectangle qui hérite de Shape et qui possède les attributs suivants :

- width : modélisant la largeur, en pixels
- height : modélisant la hauteur, en pixels

En vous inspirant de ce qui a été fait pour la classe Circle, implémentez la classe Rectangle possédant les méthodes suivantes :

- un constructeur :

```
public Rectangle(Point center, int width, int height)
```

- tous les accesseurs en lecture et écriture,
- une méthode print() produisant l'affichage :

```
Rectangle - center : ({x}, {y}), width : {r}, height : {h}
```

où {x} et {y} sont à remplacer par les coordonnées du centre du rectangle, {w} par sa largeur et {h} par sa hauteur.

Testez le bon fonctionnement de votre classe en instanciant un rectangle, et en appelant la méthode print() avant et après modification de ses dimensions.

8 – Affichage d'un ensemble de formes

Ajoutez la méthode statique suivante à votre classe Shape :

```
public static void printShapes(Shape[] shapes)
```

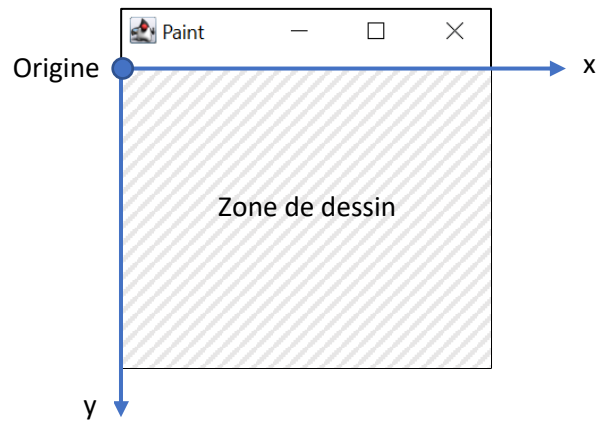
qui appelle la méthode print() sur toutes les formes du tableau shapes.

Testez le bon fonctionnement de cette méthode en l'appelant sur un tableau contenant un Cercle et un Rectangle.

9 – Récupération et test de la classe Paint

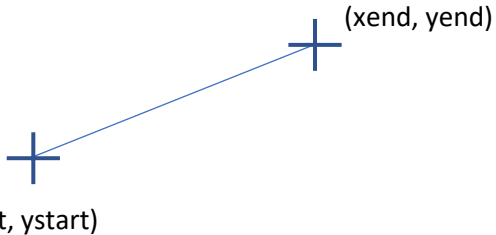
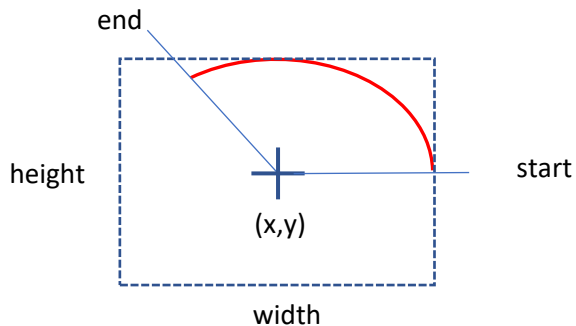
La classe de dessin Paint

Il est à présent temps de dessiner nos formes ! Pour cela, nous vous fournissons une classe Paint qui permet de dessiner dans une fenêtre. Dans la zone de dessin, les points sont repérés avec le système de coordonnées suivant :



Attention : Comme vous pouvez le voir, l'origine est située dans le coin supérieur gauche et l'axe y est dirigé vers le bas, ce qui est très fréquent quand on manipule des bibliothèques graphiques (pas uniquement en Java).

La classe Paint se manipule de la manière suivante :

Code	Effet
<code>Paint p = new Paint(width, height)</code>	Crée une zone de dessin de largeur width et de hauteur height, affichée dans une fenêtre.
<code>p.drawLine(xstart, ystart, xend, yend)</code>	Dessine une ligne telle qu'illustrée ci-dessous : 
<code>p.drawArc(x, y, width, height, start, end)</code>	Dessine un arc tel qu'illustré ci-dessous en rouge :  Les angles start et end sont exprimés en degrés.

`p.clear()`

Efface le contenu de la fenêtre de dessin.

Récupération de la classe *Paint*

Récupérez le fichier `Paint.java` sur la page Moodle du cours et ajoutez-le à votre projet par l'un des moyens suivants (choisissez celui que vous préférez) :

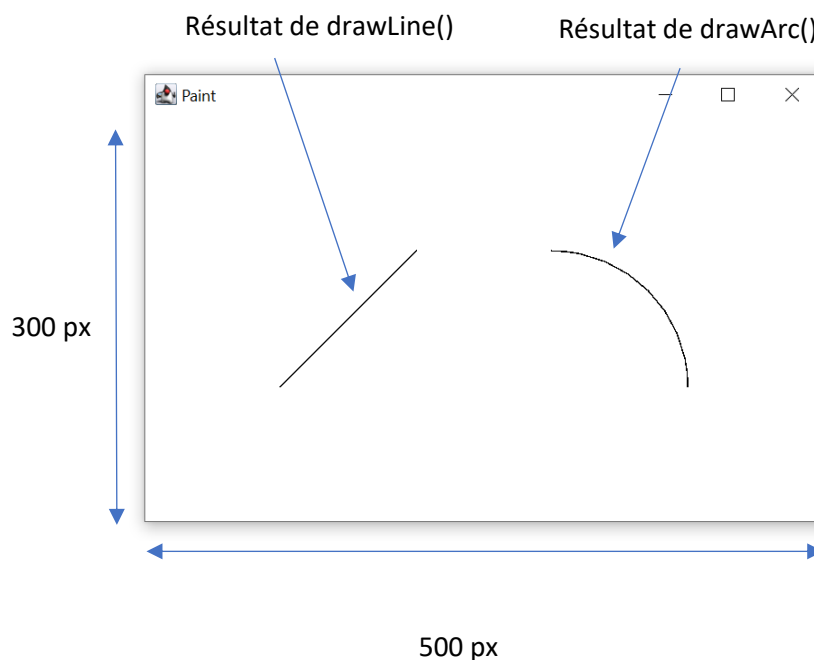
1. Glisser-déposer le fichier depuis l'explorateur de fichiers vers le répertoire « `src` » de votre projet dans IntelliJ
2. Sélectionnez le fichier dans l'explorateur, appuyez sur `CTRL + C`, sélectionnez ensuite le répertoire « `src` » dans de votre projet dans IntelliJ et appuyez sur `CTRL + V`
3. Ouvrez le répertoire de votre projet dans l'explorateur de fichiers, et copiez le fichier dans le répertoire « `src` » ; le projet se mettra à jour automatiquement dans IntelliJ.

Test de la classe *Paint*

Exécutez le code suivant :

```
Paint p = new Paint(500, 300);  
p.drawLine(100, 200, 200, 100);  
p.drawArc(300, 200, 200, 200, 0, 90);
```

Vous devriez voir apparaître zone de dessin de 500 x 300 px, contenant une ligne et un arc de cercle :



10 – Méthodes de dessin

Dessin d'un cercle

Dans la classe Circle, implémentez une méthode :

```
public void draw(Paint paint)
```

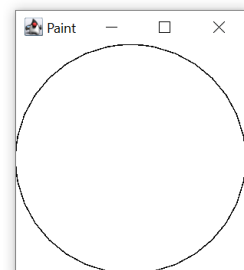
qui dessine l'instance courante de Circle dans la fenêtre de dessin `paint`.

Exécutez le code suivant :

```
Paint p = new Paint(200, 200);  
Circle c = new Circle(new Point(100, 100), 100);  
c.draw(p);
```

Vous devriez obtenir le résultat ci-contre :

Si votre code fonctionne correctement, le cercle doit être centré et tangenter les bords de la fenêtre.



Dessin d'un rectangle

Dans la classe Rectangle, implémentez une méthode :

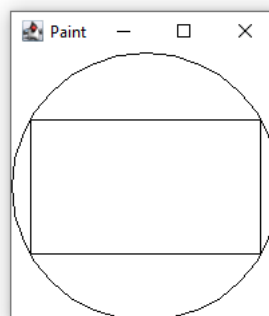
```
public void draw(Paint paint)
```

qui dessine l'instance courante de Rectangle dans la fenêtre de dessin `paint`.

Exécutez le code suivant (nouvelles lignes en bleu) :

```
Paint p = new Paint(200, 200);  
Circle c = new Circle(new Point(100, 100), 100);  
c.draw(p);  
  
Rectangle r = new Rectangle(new Point(100, 100), 173, 100);  
r.draw(p);
```

Vous devriez obtenir le résultat ci-après :



Polymorphisme d'héritage

Dans la classe Shape, ajoutez la méthode statique :

```
public static void drawShapes(Shape[] shapes, Paint paint)
```

qui dessine toutes les formes présentes dans le tableau `shapes` dans la fenêtre de dessin `paint`.

Ecrivez le code suivant dans le `main()` :

```
Paint p = new Paint(200, 200);  
  
Circle c = new Circle(new Point(100, 100), 100);  
Rectangle r = new Rectangle(new Point(100, 100), 173, 100);  
  
Shape[] shapes = new Shape[2];  
shapes[0] = c;  
shapes[1] = r;  
  
Shape.drawShapes(shapes, p);
```

Ce code ajoute un cercle et un rectangle dans un tableau de formes, puis demande à la méthode `drawShapes()` d'afficher les formes présentes dans ce tableau.

Exécutez ce code. Vous obtiendrez une erreur. Analysez cette erreur et faites les modifications nécessaires dans la classe Shape. Si elles sont correctes, alors l'exécution du `main()` produira l'affichage de la même fenêtre que dans la partie précédente.

11 – Un peu de style

Attributs de style

Ajoutez les attributs de style suivants dans la classe Shape :

Nom	Type	Description
<code>lineWidth</code>	<code>int</code>	Epaisseur du trait en pixels
<code>lineColor</code>	<code>Color</code>	Couleur du trait.

[Color](#) est une classe de l'API Java modélisant une couleur. Vous pouvez créer des couleurs quelconques en spécifiant des composantes RGB, ou utiliser les couleurs prédéfinies sous la forme de membres statiques, telles que `Color.BLUE` ou `Color.RED` par exemple.

Implémentez les accesseurs associés, et ajoutez les paramètres au constructeur de Shape permettant d'initialiser le style de la forme.

Ajoutez des paramètres aux constructeurs de Circle et Rectangle permettant d'initialiser le style.

Rappel : tout comme le centre, l'initialisation du style doit être délégué au constructeur de Shape.

Prise en compte du style dans le dessin

Modifiez l'implémentation de `draw()` dans les classes `Shape`, `Circle` et `Rectangle` pour tenir compte des attributs de style. Pour ce faire, les méthodes suivantes de la classe `Paint` vous seront utiles :

Code	Effet
<code>p.setColor(Color color)</code>	Affecte la couleur de trait du <code>Paint p</code> à <code>color</code> .
<code>p.setLineWidth(int linewidth)</code>	Affecte l'épaisseur de trait du <code>Paint p</code> à <code>linewidth</code> pixels.

Élément de réflexion : comment faire en sorte que l'initialisation du style ne soit pas dupliquée dans chaque surcharge de la méthode `draw()` ?

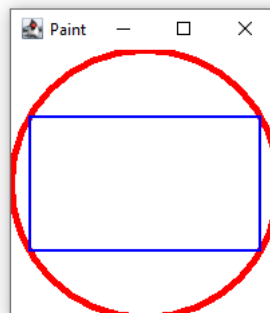
Test

En supposant que les 2 derniers paramètres des constructeurs permettent d'initialiser la couleur et l'épaisseur de trait, alors le code suivant :

```
Paint p = new Paint(200, 200);
Circle c = new Circle(new Point(100, 100), 100, Color.RED, 5);
c.draw(p);

Rectangle r = new Rectangle(new Point(100, 100), 173, 100, Color.BLUE, 2);
r.draw(p);
```

doit aboutir à l'affichage :



Style par défaut

Faites en sorte que l'utilisateur puisse instancier des formes sans fournir de paramètre de style. Dans ce cas, le style par défaut suivant sera appliqué :

- Couleur de ligne : noir
- Epaisseur du trait : 1 px.

Rappel : dans un constructeur, on peut appeler un autre constructeur de la classe avec le mot-clé `this`.

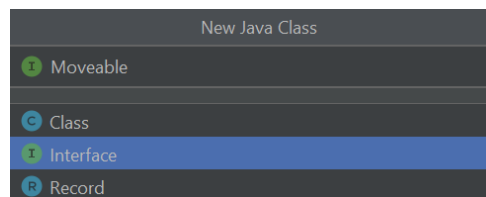
12 – Déplacement d'un objet

L'interface suivante modélise la notion d'objet déplaçable :

```
public interface Moveable{  
    void moveTo(int x, int y);  
}
```

Après appel à la méthode `moveTo()`, l'objet qui implémente cette interface se trouvera aux coordonnées spécifiées.

Ajoutez l'interface `Moveable` à votre projet. Pour cela, avoir cliqué sur `New > Java Class`, entrez le nom « `Moveable` » et sélectionnez « `Interface` » au lieu de « `Class` » :



Nous allons maintenant modéliser qu'un `Point` est un objet déplaçable. Pour cela, nous allons écrire que `Point` *implémente* l'interface `Moveable` comme ceci :

```
public class Point implements Moveable{  
    ...  
}
```

Modifiez la définition de la classe `Point` comme indiqué ci-dessus, vous obtiendrez alors une erreur. Pourquoi ? Faites les modifications nécessaires pour corriger cette erreur.

Modifiez la classe `Shape` pour modéliser qu'une forme est un objet déplaçable.

Testez le bon fonctionnement de la méthode `moveTo()` sur un point ou sur une forme. Vérifiez qu'il est possible de transtyper ces instances en `Moveable`.

13 – [Facultatif] Boîte englobante

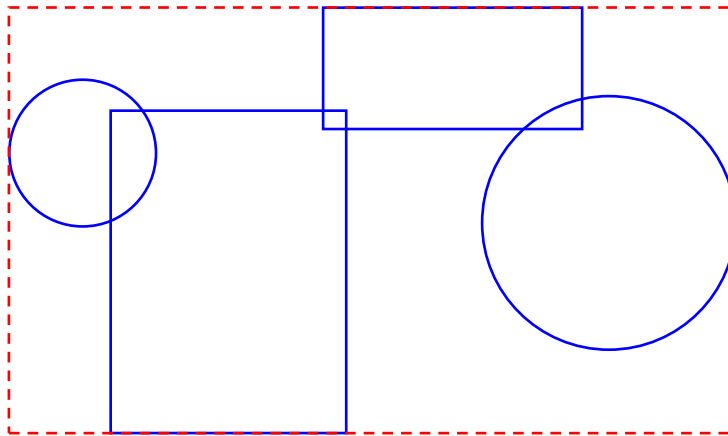
Implémentation

Dans la classe `Shape`, implémentez une méthode :

```
public static Rectangle getBoundingBox(Shape[] shapes)
```

qui calcule la boîte englobante d'un ensemble de formes, c'est-à-dire un rectangle de dimensions minimales contenant toutes les formes.

Exemple : ci-dessous, la boîte englobante des formes bleues est illustrée en pointillés rouges :



Contrainte : la méthode `getBoundingBox()` ne doit pas (et n'a pas besoin de) tester le type concret des formes présentes dans le tableau fourni.

Aide :

- La boîte englobante d'un cercle de rayon r est un rectangle de hauteur et de largeur $2r$.
- La boîte englobante d'un rectangle est le rectangle lui-même.
- Pour calculer la boîte englobant toutes les formes, il faut calculer les coordonnées minimales et maximales des boîtes englobantes associées.

Test

Vous pouvez dessiner les formes présentes dans le tableau `shapes` ainsi que le rectangle retourné par `getBoundingBox()` pour vérifier son bon fonctionnement.