# TP 2 — Interfaces

## 1 Notion d'interface

Une *interface* en Java est un *contrat*. Une interface définit un ensemble de signatures de méthodes mais sans préciser le contenu, et les classes qui implémentent l'interface sont tenues d'implémenter ces méthodes.

Comme pour une classe, une interface a une visibilité, un nom et un bloc de déclaration. Une interface décrit un ensemble de méthodes en fournissant uniquement leur signature (mais pas leur corps). Comme pour les classes, une interface est déclarée dans son propre fichier qui porte le même nom que l'interface. Pour l'exemple ci-dessus, le fichier doit s'appeler `Compte.java` :

Comptes.java
```java
public interface Compte {
    void deposer(float montant);
    float retirer(float montant);
    float getBalance();
}
```

Une classe qui implémente une interface est indiquée par le mot-clé **`implements`**. Une classe concrète doit fournir une implémentation pour toutes les méthodes d'une interface, soit dans sa déclaration, soit parce qu'elle en hérite. Par exemple, la classe `CompteBancaire` donnée en annexe A page 4 implémente l'interface `Compte`.

On y retrouve les définitions *et une implémentation* de toutes les méthodes demandées par l'interface, à savoir : `void deposer(float montant)`, `float retirer(floant montant)` et `float getBalance()`. On retrouve également un constructeur, ce qui n'est pas le cas pour une interface. Et la classe `CompteBancaire` contient l'attribut supplémentaire `private final String numero`, qui n'est pas exigé par l'interface.

**1)** Intégrer ces deux fichiers sources dans un projet java nommé `r201_tp2_<nomDeFamille>`.

**2)** Compiler le code et ajouter un fichier source `Main.java` contenant une méthode `Main`. On rappelle qu'en Java, la méthode `Main` a la signature suivante :

Main.java
```java
public static void Main(String args[])
```

Dans la classe principale, instancier un objet de classe `CompteBancaire` en affectant le numéro `31415`.

**3)** Dans le fichier `CompteBancaire.java`, placer en commentaire la méthode `retirer` et compiler. Que remarque-t-on ?

**L'annotation `@override`**

On rappelle que l'annotatio `@override` indique au compilateur qu'il s'agit d'une redéfinition de méthode : cela permet d'identifier quelles sont les méthodes héritées d'une classe mère.

Cela améliore la lisibilité du code et les performances du programme.

On considère maintenant une *deuxième* classe `CompteEpargne`, qui implémente également l'interface `Compte`. Le code est disponible en annexe B page 5. Observez que les méthodes de l'interface sont redéfinies de manière différente que celles de la classe `CompteBancaire`.

Ainsi les classes `CompteBancaire` et `CompteEpargne` implémentent la même interface `Compte`. De ce fait les objets instanciés de l'une ou l'autre des deux classes pourront, notamment, être rassemblés dans un même conteneur (un tableau d'objets de classe `Compte` par exemple) et manipulés de la même manière. L'exemple donné en annexe C page 6 illustre cette possibilité.

# 2 À vous de jouer !

L'objectif est de créer un petit programme, capable de gérer des dessins constitués de formes géométriques élémentaires. On se base sur le travail réalisé lors du TP 1 avec les points du plan et les rectangles.

On reprend donc les classes `Point2D` et `Rectangle` vues au séances précédentes. On s'intéresse maintenant aux disques. On considère qu'un disque est déterminé par son centre et son rayon.

## 2.1 L'interface `Formes`

**1)** Reprendre les classes `Point2D` et `Rectangle` des séances précédentes et les insérer dans le projet du TP 2.

**2)** Définir une classe `Disque` avec des constructeurs appropriés et que des méthodes `translate`, `surface` et `contains`.

On souhaite à présent définir une classe `Dessin`. Un dessin pourra contenir des rectangles et des disques. Comme les disques et les rectangles peuvent se retrouver dans un même conteneur, il est logique de définir une interface pour ces deux classes.

**3)** Définir une interface `Forme` qui devra être implémentée par les classes `Rectangle` et `Disque`. Cette interface devra contenir les signatures des méthodes `translate`, `surface` et `contains`.

Formes.java

```
1 /* Interface Forme ***********************************/
2 public interface Forme {
3     /* … */
4 }
```

**4)** Modifier les classes `Rectangle` et `Disque` pour qu'elles implémentent l'interface `Formes`.

## 2.2 La classe `Dessin`

L'étape suivante consiste à définir la classe `Dessin`. Chaque objet de cette classe devra avoir comme attribut un tableau pouvant contenir aussi bien des rectangles que des disques. On vous donne les spécifications suivantes :

- Un dessin peut être considéré comme une forme.
- Un des attributs de la classe est un tableau de formes (rectangles et disques).
- La taille du tableau est fixée à la construction de l'objet. Au départ, un dessin ne contient aucun rectangle ni disque.
- Un attribut de la classe permet de comptabiliser le nombre de formes contenues dans le tableau.

**5)** Intégrer le fichier `Dessin.java` à votre projet et ajouter les attributs, getters et setters nécessaires, en accord avec les spécifications.

**6)** Doter la classe `Dessin` des méthodes suivantes :

`add` qui permet d'ajouter une forme au dessin ;
`translate` qui permet de translater chacune des formes du dessin ;
`translate` qui calcule la somme des surfaces des formes ;
`contains` qui permet de savoir si un point est dans une des formes du dessin.

**7)** Tester toutes ces classes et leurs méthodes depuis le programme principal `Main.java`

## 2.3 Dessins extensibles

Le problème est qu'un dessin tel qu'il est défini jusqu'à présent, ne permet pas d'ajouter plus de formes que le nombre passé au constructeur lors de l'instanciation d'un objet de classe `Dessin`. Tout ajout supplémentaire au-delà de la limite du tableau des formes, provoque une erreur de dépassement de capacité. Pour pallier ce problème, on se propose de disposer d'une classe de dessins extensibles.

**8)** Proposer la définition d'une classe nommée `DessinExtensible` qui hérite de la classe `Dessin` et qui permet de redéfinir la méthode `add` de manière à pouvoir, le cas échéant, augmenter la taille du tableau, en le réallouant en mémoire, si le nombre de formes a atteint la limite du tableau.

**9)** Tester cette nouvelle classe et plus particulièrement sa méthode `add`.

# 3 Annexes

## A Classe `CompteBancaire.java`

CompteBancaire.java

```java
public class CompteBancaire implements Compte {
    private final String numero;
    private float balance;

    /* Constructeur */
    public CompteBancaire(String numero) {
        this.numero = numero;
    }

    /* Getters */
    @Override
    public float getBalance() {
        return this.balance;
    }

    public String getNumero() {
        return this.numero;
    }

    /* Méthodes */
    @Override // Notez bien qu'il s'agit d'une redéfinition de méthode
    public void deposer(float montant) {
        this.balance += montant;
    }

    @Override
    public float retirer(float montant) {
        return this.balance -= montant;
    }
}
```

## B Classe `CompteEpargne.java`

CompteEpargne.java

```java
public class CompteEpargne implements Compte {
    private final String numero;
    private final float maxEpargne = 15000;
    private float balance;

    /* Constructeur */
    public CompteEpargne(String numero) {
        this.numero = numero;
    }

    /* Getters */
    @Override
    public float getBalance() {
        return this.balance;
    }

    public String getNumero() {
        return this.numero;
    }

    /* Méthodes */
    @Override
    public void deposer(float montant) {
        if (this.balance+montant <= maxEpargne) {
            this.balance += montant;
        }
        else {
            System.out.println("Montant maximum atteint pour un compte épargne.")
        }
    }

    @Override
    public float retirer(float montant) {
        System.out.println("Impossible de retirer sur un compte épargne avant échéance.");
        return this.balance;
    }
}
```

## C Implémentations différentes de l'interface `Comptes`

Main.java

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Exemple d'utilisation des interfaces");
        CompteBancaire cptb = new CompteBancaire("B1234XYZ");
        cptb.deposer(10);
        System.out.println("balance : "+cptb.getBalance());
        cptb.retirer(5);
        System.out.println("balance : "+cptb.getBalance());

        CompteEpargne cpte = new CompteEpargne("AA34672ET");
        cpte.deposer(100);
        cpte.retirer(5);

        // Tableau d'interfaces de Compte
        Compte[] comptes = new Compte[2];
        comptes[0] = cptb;
        comptes[1] = cpte;

        comptes[0].retirer(10);
        System.out.println(comptes[0].getBalance());

        comptes[1].retirer(10);
        System.out.println(comptes[1].getBalance());
    }
}
```