Héritage

Exercice 1

1. Le code suivant provoque une erreur à la compilation. Laquelle ? Expliquer votre réponse.

```
class Mere {
private int x;
public Mere(int x) {this.x = x;}
}

class Fille extends Mere {
private int x;
public Fille(int x) {this.x = x;}
}
```

2. Le code suivant provoque une erreur à la compilation. Laquelle ? Expliquer votre réponse.

```
class Mere {
private int x;
public Mere() {x = 1;}
}

class Fille extends Mere {
public Fille(int x) {super.x = x;}
}
```

3. On définit deux classes :

```
class Mere {
public int x;
public Mere(int x) {this.x = x;}
}

class Fille extends Mere {
public int x;
public Fille(int x) {super(x);}
}
```

   Qu'affiche les instructions suivantes ?
   (a) `System.out.println(new Mere(1).x)`
   (b) `System.out.println(new Fille(1).x)`
4. Les affirmations suivantes sont-elles correctes ?
   (a) Une classe peut avoir plusieurs super-classes.
   (b) On peut définir des sous-classes de la classe `String`.
   (c) L'instruction `A[] t = new A[6]` définit un objet tableau dont le type est `A[]`.
   (d) Pour n'importe quelle classe `A`, le type `A[]` est un sous-type de `Object[]`.

Exercice 2

1. Créer une classe `Computer` modélisant un ordinateur. La seule propriété d'un objet `Computer` est la quantité de mémoire (en Go) et la capacité du disque dur (en GB). La classe possède un seul constructeur initialisant toutes les propriétés. L'interface de la classe propose des accesseurs en lecture vers chacune de ses propriétés.

2. Créer une classe `Laptop`. Un objet `Laptop` est un objet `Computer` avec une propriété additionnelle : la taille de l'écran (en pouces). La classe possède un seul constructeur initialisant toutes les propriétés. L'interface de la classe propose des accesseurs en lecture vers chacune de ses propriétés.
3. Ecrire une classe de démonstration.

Exercice 3

— Dans le cadre du développement d'une application de gestion de salles de cinéma, un développeur à défini deux classes : une classe `Cinema` représentant un cinéma et une autre classe `CinemaQuartier` représentant un cinéma de quartier.

— Le responsable de l'équipe de développement n'est pas satisfait de la proposition du développeur. Il souhaiterait qu'il existe une relation d'héritage entre les deux classes. Il demande donc au développeurs de modifier la définition de ces classes en établissant une relation d'héritage entre elles.

1. Proposer une réécriture des classes `Cinema` et `CinemaQuartier` utilisant l'héritage. L'exécution de la méthode `main` de la classe `DemoCinema` doit produire le même affichage qu'avec la définition originelle des classes :

```
Salle 1 : Titre 1
Salle 2 : Titre 2
Salle 3 : Titre 3
Titre
```

Exercice 4

1. Donner les attributs et les méthodes de la classe `Account`.
2. Écrire une classe de démonstration de la classe `Account`.
3. Écrire une sous-classe `SavingAccount` de la classe `Account` modélisant un compte rémunéré. En plus des propriétés et méthodes d'un objet `Account`, un objet `SavingAccount` possède un attribut privé indiquant le taux de rémunération du compte (`double`) et une méthode calculant la rémunération et l'ajoutant au solde du compte (la rémunération se calcule en multipliant le solde par le taux de rémunération).
4. Écrire une sous-classe `Currentccount` de la classe `Account` modélisant un compte limité. En plus des propriétés et méthodes d'un objet `Account`, un objet `CurrentAccount` possède un attribut privé indiquant le solde maximal d'un compte.
5. Écrire une classe de démonstration des classes `SavingAccount` and `CurrentAccount`.

Exercice 5

1. Créer deux sous-classes `Student` et `Teacher` de la classe `Person` modélisant un étudiant et un enseignant. Les deux classes ont une propriété supplémentaire : la référence d'un objet `Course`. Un objet `Course` permet d'enregistrer les participants à un cours (enseignant et étudiants). Le programme ci-dessous

```
Student etud1 = new Student("Lucy");
Student etud2 = new Student("Richard");
Teacher prof1 = new Teacher("James");
Teacher prof2 = new Teacher("Paul");

Course  cours = new Course();
etud1.follows(cours);
prof1.teaches(cours);
etud2.follows(cours);
prof2.teaches(cours);

System.out.println(cours.participants());
```

affiche dans la console :

```
Lucy (Student), James (Teacher), Richard (Student), Paul (Teacher)
```

Exercice 6

On définit les classes ;Flower et `Rose` :

```
public class Flower {public void fragrance() {System.out.println("Fleur");}}
public class Rose extends Flower {public void fragrance() {System.out.println("Rose");}}
```

Qu'affiche le programme suivant ? Expliquer votre réponse.

```
public class Programme {
public static void main(String[] args) {
    Flower flower1 = new Rose();
    flower1.fragrance();
}
```

Exercice 7

On a commencé à développer une classe modélisant une date. Il a été choisi d'enregistrer le jour, le mois et l'année dans un tableau :

```
class Date {
    private int[] t;

    /* à compléter */
}
```

1. Proposer une définition du constructeur `new Date(2,1,2023)`.
2. Proposer une rédéfinition de la méthode `toString` permettant de retourner la date sous la forme d'une chaîne de caractères au format donné en commentaires :

   ```
   System.out.println(d1.toString()) // 02/01/2023
   ```

3. Proposer une définition de la méthode `equals` permettant de tester l'égalité de deux dates :

   ```
   Date d1 = new Date(2,1,2023), d2 = new Date(2,1,2023);
   System.out.println(d1.equals(d2)) // true
   ```

4. On souhaite développer une nouvelle classe modélisant une date d'anniversaire. Une instance de la classe encapsule le prénom d'une personne (`String`) en plus de la date. Proposer une solution minimale telle que le programme suivant produise l'affichage donné en commentaire :

   ```
   DateAnniversaire d3 = new DateAnniversaire("Paul",2,1,2023);
   System.out.println(d3.equals(d1)); // true
   System.out.println(d3.toString()); // 02/01/2023, anniversaire de Paul
   ```

   Définir la classe `DateAnniversaire`.

Exercice 8

On modélise les fruits par la classe `Fruit` et les différents types de fruits par des sous-classes de la classe `Fruit` : la classe `Pomme` pour les pommes, la classe `Orange` pour les oranges, etc (nous ne listons pas toutes les classes modélisant les fruits mais il en existe d'autres que celles données dans l'énoncé). Toutes les classes modélisant les fruits doivent implémenter la méthode `String nom()` qui retourne le nom du fruit (`"pomme"` pour la classe `Pomme`, `"orange"` pour la classe `Orange`, etc). Un panier de fruits sera défini par une classe `Panier` qui permettra d'enregistrer les références de tous types de fruits. Un panier peut contenir un nombre illimité de fruits.

1. Donner la déclaration de la classe `Panier` en donnant uniquement ses attributs et un constructeur. Donner un exemple de création d'une instance de la classe `Panier`.
2. On souhaite pouvoir enregistrer les références des fruits du panier à l'aide d'une méthode `ajouter` qui retourne un booléen indiquant si l'ajout du fruit au panier était possible ou impossible (panier plein). Un exemple illustrant les comportements des méthodes d'ajout d'un fruit au panier et d'affichage du panier

   ```
   System.out.println(panier.ajouter(new Pomme()));
   System.out.println(panier.ajouter(new Orange()));
   System.out.println(panier.ajouter(new Pomme()));
   System.out.println(panier.ajouter(new Orange()));
   ```

   Proposer une implémentation de cette méthode.
3. On souhaite pouvoir afficher les noms des fruits contenues dans un panier à l'aide d'une méthode `afficher` :

   ```
   panier.afficher(); // pomme orange pomme orange
   ```

4. On souhaite pouvoir compter le nombre de pommes contenus dans un panier à l'aide d'une méthode `compterPommes` :

   ```
   System.out.println(panier.compterPommes() + " pommes"); // 2 pommes
   ```

   Proposer une implémentation de cette méthode.
5. Si on définit une sous-classe de la classe `Pomme` qu'on appelle `PommeGolden`. Que devraient afficher les instructions suivantes ?

   ```
   System.out.println(panier.ajouter(new Pomme()));
   System.out.println(panier.ajouter(new Orange()));
   System.out.println(panier.ajouter(new PommeGolden()));
   System.out.println(panier.ajouter(new Orange()));
   System.out.println(panier.compterPommes() + " pommes");
   ```