THEME 8 : Héritage et interface

TD 53, 54, 55 et TME 57, 60

Exercice 53 – Redéfinition de la méthode equals

1)
Ligne 08 : déclaration de la variable p1 de type Point qui référence en mémoire
un objet Point de coordonnées (5, 2)
Ligne 09 : déclaration de la variable p2 de type Point qui référence en mémoire
un objet Point de coordonnées (5, 2)
Ligne 10 : déclaration de p3 qui reçoit une copie de la référence contenue dans p1
Ligne 11 : déclaration de p4 de type Point qui référence en mémoire
un objet Point de coordonnées (1, 1)

Ligne 12 : Affichage 'p1 = p2 : false'
Ligne 13 : Affichage 'p1 = p3 : true'
Ligne 14 : Affichage 'p1 = p4 : false'

2)
Ajouter la redéfinition de la méthode equals() dans la classe Point

```
public boolean equals(Object o){
        Point p = (Point) o ;
        return (this.x == p.x) && (this.y == p.y) ;
}
```

En utilisant cette redéfinition de la méthode equals(), on aura les affichages ci-dessous
Ligne 12 : Affichage 'p1 = p2 : true'
Ligne 13 : Affichage 'p1 = p3 : true'
Ligne 14 : Affichage 'p1 = p4 : false'

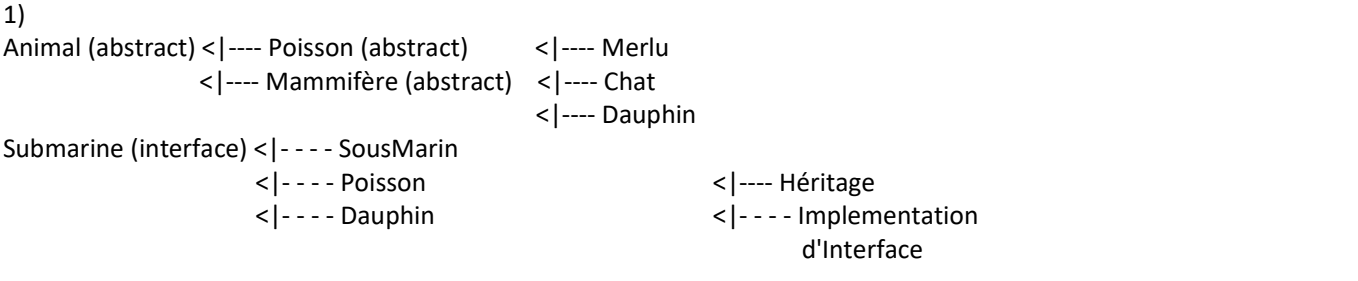
3)
A l'exécution on aura une erreur car on ne peut pas appliquer un cast
sur un objet String pour obtenir un Point
Il y aura une erreur java.lang.ClassCastException

```
public boolean equals(Object o){
        if (o == null) return false ; // pour éviter l'erreur NullPointerException car nous allons utiliser une méthode de o
        if (this.getClass() == o.getClass()){
                Point p = (Point) o ;
                return (this.x == p.x) && (this.y == p.y) ;
        }
        return false ;
}
```

En utilisant la nouvelle redéfinition de la méthode equals() on aura l'affichage suivant
Affichage 'p1 = s1 : false'

Exercice 54 – Interface Submarine

1)



2)
```
public interface Submarine {
	public void seDeplacer() ;
}
```

3)
```
public abstract class Poisson extends Animal implements Submarine {
	public void seDeplacer(){
		System.out.println(''Le poisson nage'') ;
	}
}
```

4)
Oui, un Merlu a la propriété de nager par héritage de la classe Poisson

5)
```
public class Dauphin extends Mammifere implements Submarine, Echolocation {
	public void seDeplacer() { System.out.println(''Le Dauphin nage'') ; }
	public void envoyerSon() { System.out.println(''Envoyer son Dauphin'') ; }
	public void ecouterSon() { System.out.println(''Ecouter son Dauphin'') ; }
}
```

En JAVA, une classe étend au maximum une classe
Mais une classe peut implémenter plusieurs interfaces (c'est le cas de la classe Dauphin)

6)
```
public class SousMarin implements Submarine, Echolocation {
	public void seDeplacer() { System.out.println(''Le sous-marin se déplace'') ; }
	public void envoyerSon() { System.out.println(''Envoyer son sous-marin'') ; }
	public void ecouterSon() { System.out.println(''Ecouter son sous-marin'') ; }
}
```

7)
```
import java.util.ArrayList ;

public class Mer {
	private ArrayList<Submarine> liste = new ArrayList<Submarine>() ;

	public void ajouter(Submarine s){ liste.add(s) ; }

	public void deplacer(){
		for(Submarine s : liste){
			s.seDeplacer() ;
		}
	}
}
```

8)
```
Public class TestSubmarine {
	public static void main(String[] args){
		Mer m = new Mer() ;
		m.ajouter(new Merlu()) ;
		m.ajouter(new Dauphin()) ;
		m.ajouter(new SousMarin()) ;
		m.deplacer() ;
	}
}
```

9)
Non, on ne peut pas ajouter un chat dans Mer, car Chat n'implémente pas Submarine

10)
Pour définir la nouvelle espèce de chat on aura une définition proche de celle ci-après

```
public class ChatSub extends Chat implements Submarine {
        public vois seDeplacer() { System.out.println(''Le chat nage'') ; }
}
```

Pas besoin de modifier la classe Mer
On pourra ajouter un chat de type ChatSub en utilisant la méthode ajouter() de la classe Mer

exemple : m.ajouter(new ChatSub()) ;

Exercice 55 – Interface Motorise

1)
La méthode remplirReservoir(Vehicule v)() de la classe StationService n'est pas bien programmée, car si on avait un grand nombre de véhicules différents, on devrait ajouter autant de if() qu'il y a de sous classe de Vehicule
Dans l'idéal on ne devrait pas modifier la méthode remplirReservoir() à chaque fois qu'une nouvelle classe Vehicule est définie

remplirReservoir() s'applique uniquement aux vehicules qui ont un moteur
On peut donc créer une classe abstraite VehiculeMoteur qui étend Vehicule

```
public abstract class VehiculeMoteur extends Vehicule {
        public abstract void faireLePlein() ;
}
```

Dans StationService
La méthode remplirReservoir() est modifiée

```
public void remplirReservoir(Vehicule v){
        if (v instanceof VehiculeMoteur) {
                ((VehiculeMoteur) v).faireLePlein() ;
        }else{
                System.out.println(''Inutile, pas de réservoir'') ;
        }
}
```

2)
Les lignes de 22 à 30 constituent la réponse à la question Q-55.2