Licence M I 2 E

Java Objet - TD 1

Le programme Java suivant utilise quatre classes pour décrire un parc de véhicules réparti en véhicules de tourisme et en utilitaires. Une classe décrit les acheteurs, la classe TestTD1 contient la méthode main.

```
package tdr.td1;

public class TestTD1 {
	public static void main(String[] args){
		Tourisme t1 = new Tourisme("Clio","Renault",120);
		TourismeEco t2 = new TourismeEco("106","Peugeot",210);
		Utilitaire u1 = new Utilitaire("Kangoo","Renault",700);
		Client c1 = new Client("Dupont","Paris");
		Client c2 = new Client("Durand","Lyon");
		c1.achat(t1);
		c2.achat(u1);
		System.out.println(c1);
		System.out.println(c2);
	}
}
abstract class Vehicule{
	final int noSerie;
	 static int NoSerieCourant = 100;
	String marque, modele;
	Client proprietaire;
	Vehicule(String marque, String m){
		noSerie = NoSerieCourant++;
		System.out.println("Appel constr. Vehicule " + noSerie);
		marque = marque;
		modele = m;
	}
}
class Tourisme extends Vehicule{
	int emissionCO2;
	Tourisme(String m, String mod, int e){
		System.out.println("Appel constr. Tourisme");
		super(m,mod);
		emissionCO2 = e;
	}
	public String toString(){
		return "Vehicule " + marque + " " + modele;
	}
}
class TourismeEco extends Tourisme{
	int prime;
	int calculPrime(){
		if (emissionCO2<=100)prime=1000;
		if(emissionCO2>100 && emissionCO2<=120)prime=700;
		if(emissionCO2>120 && emissionCO2<=130)prime=200;else
		  {System.out.println("Erreur sur emissionCO2: " + emissionCO2);prime=0;}
		return prime;
	}
	TourismeEco(String m, String mod, int e){
		prime =calculPrime();
		super(m,mod,e);
	}
}
```

```
class Utilitaire extends Vehicule{
     int capacite;
     Utilitaire(String m, String mod, int cap){
           super(m,mod);
           capacite = cap;
     }
}
class Client{
     static Vehicule[] listeVehicules = new Vehicule[100];
     final int noClient;
     static int noClientCourant = 10;
     String nom;
     String adresse;
     Vehicule vehiculePossede;
     Client(String n,String a){
           noClient = noClientCourant; noClientCourant++; nom = n; adresse = a;
     }
     static void achats(){
                                    Compléter
     }
     boolean achat(Vehicule v){
           boolean res;
           if (vehiculePossede != null) return false;
           vehiculePossede = v;
           v.proprietaire = this;
           return true;
     }
     boolean revente(Client acheteur){
           boolean res;
                                    Compléter
           return res;
     }
     public String toString(){
           return noClient + " " + nom + " possede: " + vehiculePossede;
     }
}
```

Questions:

1. Le constructeur de la classe abstraite **`Vehicule`** est-il licite?
2. Proposer des modifications pour que l'appel des constructeurs des classes **`Vehicule`**, **`Tourisme`**, **`TourismeEco`** et **`Utilitaire`** fonctionnent.
3. On veut ajouter à la classe **`TourismeEco`** un constructeur à deux arguments **`TourismeEco(String marque,String mod)`** : dans ce cas l'attribut **`emissionCO2`** est initialisé à 120. Décrire ce constructeur (plusieurs solutions).
4. Que provoque l'ajout dans la méthode **`main`** de l'instruction **`TourismeEco t3 =new TourismeEco() ?`** Modifier le programme pour rendre licite cette instanciation.
5. Quel affichage produit le programme?
6. On veut mémoriser la liste des véhicules vendus dans une variable de classe **`ListeVehicules`** de la classe **`Client`**. On écrira une méthode de classe **`Achats()`** de la classe **`Clients`** qui affichera cette liste, elle sera appelée de la méthode **`main`**. Indiquer les modifications à apporter à la méthode **`main`** et à la classe **`Client`**.
7. Compléter la méthode **`revente(Client acheteur)`** de la classe **`Client`**. Elle réalise la vente du véhicule du client à un autre client (n'étant pas considérée comme une vente, l'actualisation de la variable **`ListeVehicules`** n'est pas nécessaire).
8. Proposer une autre formulation des 3 premières instanciations de la méthode **`main`**.

CORRIGE

Remarques générales:

Un client possède au plus un seul véhicule, les n° Client et Vehicule ne peuvent être modifiés. **`ListeVehicules`** contient les véhicules vendus en première main, il ne prend pas en compte les reventes envisagées question 7. Insister sur l'ordre d'appel des constructeurs.

Préciser l'intérêt (dans cet exemple) du **`final`** et du **`static`**.

Expliquer les différences d'affichages par la non REDEFINITION de **`toString()`** dans **`Utilitaire`** (préparer éventuellement à l'introduction du polymorphisme de méthode).

Q1 :

Oui, une classe abstraite se comporte comme toute autre classe à part l'impossibilité d'instancier.

Q2

L'appel du constructeur de la classe mère (**`super`**) doit précéder TOUTES les autres instructions. Absence de **`this`** dans le constructeur de **`Vehicule`**.

Remarque : l'ordre des opérations lors d'une instanciation n'a pas été vu dans le premier cours.

Q3

Voir programme complet. (en particulier ajout constructeur à 2 places dans **`Tourisme`**)

Autre sol.
```
TourismeEco(String m, String mod){
      super(m,mod,120);
      prime =calculPrime();

      }
```

Autre sol.
```
TourismeEco(String m, String mod){
      this(m,mod,120);

      }
```

Q4

Erreur d'instanciation car la création d'un constructeur explicite supprime le constructeur implicite crée par la JVM, il faut donc en rajouter un dans **`TourismeEco`**, **`Tourisme`** et **`Vehicule`**. Attention! **`Vehicule`** contient un attribut **`final`**, il doit donc être initialisé dans le constructeur car il ne serait plus modifiable ensuite , soit:

```
Vehicule(){noSerie = NoSerieCourant++;}
```

Q5

```
Appel constr. Vehicule 100
Appel constr. Tourisme
Appel constr. Vehicule 101
Appel constr. Tourisme
Erreur sur emissionCO2: 210
Appel constr. Vehicule 102
10 Dupont possede: Vehicule Clio Renault
11 Durand possede: tdr.td1.Utilitaire@7a8913
```

Remarque : affichage Utilitaire !

Q6

Voir programme corrigé :

Initialisation de **`static int i`** dans **`Client`**

Ajout de v dans **`ListeVehicules`** (méthode **`achat`**)

Appel dans **`main`**

Q7

Idem

Remarque : ne pas faire appel à **`achat()`** car cela comptabiliserait une nouvelle vente dans **`ListeVehicules`**.

Q8

Typer avec Vehicule (généralisation)

PROGRAMME COMPLET

```
package tdr.td1;

public class TestTD1 {
	public static void main(String[] args){
		Tourisme t1 = new Tourisme("Clio","Renault",120);
		TourismeEco t2 = new TourismeEco("106","Peugeot",210);
		Utilitaire u1 = new Utilitaire("Kangoo","Renault",700);
		Client c1 = new Client("Dupont","Paris");
		Client c2 = new Client("Durand","Lyon");
		c1.achat(t1);
		//c2.achat(u1);
		System.out.println(c1);
		System.out.println(c2);
		c1.revente(c2);
		System.out.println(c1);
		System.out.println(c2);
		c1.achat(u1);
		Client.achats();
	}
}
abstract class Vehicule{
	final int noSerie;
    static int NoSerieCourant = 100;
	String marque, modele;
	Client proprietaire;
	Vehicule(String marque, String m){
		noSerie = NoSerieCourant++;
		System.out.println("Appel constr. Vehicule " + noSerie);
		this.marque = marque;
		modele = m;
	}
}
class Tourisme extends Vehicule{
	int emissionCO2;
	Tourisme(String m, String mod, int e){
		super(m,mod);
		System.out.println("Appel constr. Tourisme");
		emissionCO2 = e;
	}
	Tourisme(String m, String mod){
		super(m,mod);
	}
	public String toString(){
		return "Vehicule " + marque + " " + modele;
	}
}
class TourismeEco extends Tourisme{
	int prime;
	int calculPrime(){
		if (emissionCO2<=100)prime=1000;
		if(emissionCO2>100 && emissionCO2<=120)prime=700;
		if(emissionCO2>120 && emissionCO2<=130)prime=200;else
			{System.out.println("Erreur sur emissionCO2: " +
emissionCO2);prime=0;}
		return prime;
	}
	TourismeEco(String m, String mod, int e){
		super(m,mod,e);
		prime =calculPrime();
	}
	TourismeEco(String m, String mod){
		super(m,mod);
		emissionCO2 = 120;
		prime = calculPrime();
	}
}
class Utilitaire extends Vehicule{
	int capacite;
	Utilitaire(String m, String mod, int cap){
```

```
            super(m,mod);
            capacite = cap;
      }
}
class Client{
      static Vehicule[] listeVehicules = new Vehicule[100];
    static int i = 0;
      final int noClient;
      static int noClientCourant = 10;
      String nom;
      String adresse;
      Vehicule vehiculePossede;
      Client(String n,String a){
            noClient = noClientCourant; noClientCourant++; nom = n; adresse = a;
      }
      static void achats(){
            System.out.println("Liste des véhicules achetés");
            for(int j=0;j<i;j++)
            System.out.println(listeVehicules[j]);
      }
      boolean achat(Vehicule v){
            boolean res;
            if (vehiculePossede != null) return false;
            vehiculePossede = v;
            v.proprietaire = this;
            listeVehicules[i] = v; i++;
            return true;
      }
      boolean revente(Client acheteur){
            boolean res;
            if(acheteur.vehiculePossede != null && vehiculePossede == null)
res=false;//Acheteur possédant déjà une voiture et vendeur ss voit.
            else {acheteur.vehiculePossede = vehiculePossede;
vehiculePossede.proprietaire= acheteur;vehiculePossede = null; res = true;}
            return res;
      }
      public String toString(){
            return noClient + " " + nom + " possede: " + vehiculePossede;
      }
}
```

Résultats :

```
Appel constr. Vehicule 100
Appel constr. Tourisme
Appel constr. Vehicule 101
Appel constr. Tourisme
Erreur sur emissionCO2: 210
Appel constr. Vehicule 102
10 Dupont possede: Vehicule Clio Renault
11 Durand possede: null
10 Dupont possede: null
11 Durand possede: Vehicule Clio Renault
Liste des véhicules achetés
Vehicule Clio Renault
tdr.td1.Utilitaire@7a8913
```