**Examen : Programmation Orientée Objet JAVA**

Nom : Meggouri

Prénom : Ismail

**Session 1: 2022-2023**

**Exercice 1 :** (répondre sur la feuille)

|  |  |
| --- | --- |
|  | Si le programme ne contient pas d’erreur d’exécution  Résultat à afficher par le programme est :  Si le programme contient des erreurs  Lignes des erreurs sont :  **La ligne 13.**  Corrections à faire :  **Il faut initialiser :**  **points[0] = new Point(0,0);**  **points[1] = new Point(0,0);**  Résultat à afficher après correction est :  **(2,2) (3,3)**  **null** |
| **1.import** java.util.ArrayList;  **2.import** java.util.List;  **3.abstract** **class** Didi{  **4.** **public** String getDetail(){**return** "droite -";} }  **5.class** Fifi **extends** Didi{  **6.**@Override  **7.** **public** String getDetail() {**return** "gauche -";} }  **8.class** Nini **extends** Didi{  **9.** @Override  **10.** **public** String getDetail() {**return** "milieu -";} }  **11.public** **class** Question2 {  **12.public** **static** **void** main(String[] args) {  **13.**List<Didi> list=**new** List();  **14.**list.add(**new** Didi());list.add(**new** Fifi());list.add(**new** Nini());  **15.for**(Didi d:list) System.***out***.print(d.getDetail());  **16.**} } | **Réponse :**  Si le programme ne contient pas d’erreur d’exécution  Résultat à afficher par le programme est :  Si le programme contient des erreurs  Lignes des erreurs sont :  **Les lignes 13 et 14**  **On ne peut pas instancier des classes abstraites (List, Didi)**  Corrections à faire :  **List<Didi> list=new ArrayList();**  **Enlever list.add(new Didi());**  Résultat à afficher après correction est :  **gauche -milieu -** |
| **1.abstract** **class** Fa{  **2.protected** **int** value=5;  **3.abstract** **void** f();  **4.void** g(){System.***out***.println("Traitement 1 de g:"+value);}  **5.**@Override  **6.public** String toString() {**return** "("+value+")";} }  **7.class** Ma **extends** Fa{  **8.void** f(){System.***out***.println("Traitement de f");**this**.value++;}  **9.**@Override  **10.void** g(){System.***out***.println("Traitement 2 de g:"+value);**this**.value++;}  **11.void** h(){  **12.**System.***out***.println("je suis un traitement de h");**this**.value++;} }  **13.class** Ta **extends** Ma{}  **14.public** **class** Question3 {  **15.public** **static** **void** main(String[] args) {  **16.**Fa ma=**new** Ma(); ma.f(); ma.g();  **17.** ma.h();  **18.**System.***out***.print(ma);  **19.**Fa ta=**new** Ta(); ta.f(); ta.g();  **20.** ta.h();  **21.**System.***out***.print(ta); } } | **Réponse :**  Si le programme ne contient pas d’erreur d’exécution  Résultat à afficher par le programme est :  Si le programme contient des erreurs  Lignes des erreurs sont :  **Les Lignes 17 et 20**  Corrections à faire :  **Ajouter une methode abstraite**  **abstract void h();**  **à la class abstraite Fa**  Résultat à afficher après correction est :  **Traitement de f**  **Traitement 2 de g:6**  **je suis un traitement de h**  **(8)Traitement de f**  **Traitement 2 de g:6**  **je suis un traitement de h**  **(8)** |

**Exercice 2 :**

1. Quelles sont les différences entre une méthode surchargée (overriding) et une méthode redéfinie « overloading)

**La surcharge de méthode est utilisée dans une seule classe où vous avez le même nom de fonction mais un ensemble d’arguments différent pour chaque fonction.**

**La redéfinition de méthode est un concept que l’on rencontre lors de la création de sous-classes, elle est utilisée lorsque vous déclarez une sous-classe et créez une fonction avec le même nom et les mêmes arguments qu’une fonction dans la classe de base.**

1. Quelles sont les différences entre une classe abstraite et une interface ?

**Interface :**

* **Une interface peut hériter un nombre illimité d’interfaces à la fois**
* **Une interface ne peut avoir que des méthodes abstraites**

**Class Abstract :**

* **Une classe abstraite ne peut hériter qu’une classe ou une classe abstraite à la fois.**
* **Une classe abstraite peut avoir des méthodes abstraites et concrètes**

1. Étant donné une classe abstraite A qui implémente une interface F, est ce que A doit redéfinir toutes les méthodes de F ? justifier la réponse.

**A ne doit pas redéfinir toutes les méthodes de F parce qu’on ne peut pas instancier une classe abstraite.**

1. Quelles sont les différences entre le mot clé this et le mot clé super ?

**Le mot-clé *this* est couramment utilisé lorsqu'une variable d'instance est masquée par un paramètre d'une méthode.**

**Le mot-clé *super* est utile lorsqu'une classe remplace une méthode de sa classe parente et que nous devons invoquer la méthode remplacée.**

1. Pourquoi utiliser les interfaces ?

**Les interfaces permettent de créer des spécifications qui devrons être utilisé quand une classe implémente cette interface.**

1. Est-ce qu’une classe abstraite peut contenir un constructeur ? si oui, à quoi peut servir ? si non, pourquoi ?

**Oui par ce que le constructeur est utilisé pour initialiser un objet et n’est pas utilisé pour construire un objet.**

1. Quand peut-on utiliser l’héritage ?

**L'héritage est utilisé lorsque nous avons une relation entre des objets.**

1. Qu’est-ce que le polymorphisme ? une méthode polymorphe ? donner un exemple

**Le polymorphisme permet de modifier le comportement d'une classe fille par rapport à sa classe mère.**

**Une méthode polymorphe est une méthode qui peut être écrit différemment aux classes filles.**

**Ex : toString()**

1. Dans quels cas utiliser LinkedList à la place de ArrayList ? (faire une petite recherche)

**Utiliser LinkedList pour manipuler des données (supprimer, ajouter).**

1. Dans quels cas utiliser ArrayList  à la place de LinkedList? (faire une petite recherche)

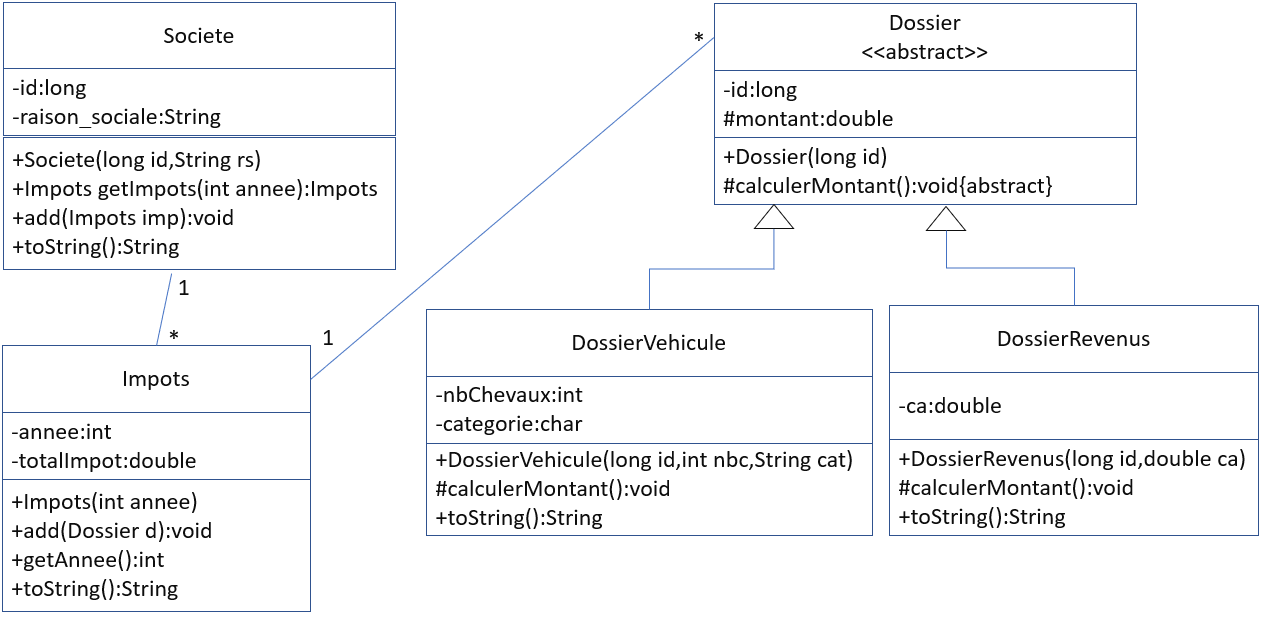
**ArrayList est préférable pour stocker et accéder aux données (rechercher).**

1. Pourquoi le concept de généricité est utilisé ?

**La généricité est un concept utilisé pour développer des objets ou des algorithme travaillant avec plusieurs types de données.**

**Exercice 3 :**

Soit le diagramme de classes suivant :



Implémenter les classes suivantes :

* Classe Societe:
  + La méthode getImpots(int annee) retourne un enregistrement Impots qui correspond à annee à partir de la liste des impôts
  + La méthode add(Impots imp) ajoute un enregistrement Impots à sa liste des impôts
  + La méthode toString() retourne l’id et la raison sociale de la société
* Classe Impots   :
  + La méthode add(Dossier d) ajoute un dossier à la liste des dossiers d’un enregistrement Impots
  + La méthode toString() retourne le détail de la liste des dossiers (voir exécution)
* Classe Dossier   :
  + La méthode calculerMontant() est absraite
* Classe DossierVehicule   :
  + La méthode toString retourne le détail d’un dossier sous la forme : Vehicule E 6 CH, montant:350.0
  + La méthode calculerMontant() calcule le montant selon le tableau ci-dessous
* Classe DossierRevenus   :
  + La méthode toString retourne le détail d’un dossier sous la forme : 15% de revenus 1000000.0: 150000.0
  + La méthode calculerMontant() calcule le montant des impôts qui représente 15% du chiffre d’affaire ca

Le montant des impôts des véhicules dépend du nombre de chevaux et de la catégorie du véhicule selon le tableau suivant :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Catégorie/ nb Chevaux | Inférieur à 8 CH | De 8 à 10 CH | De 11 à 14 CH | Sup ou égal à 15 CH |
| Véhicule à moteur Essence | 350 DH | 650 DH | 3000 DH | 8000 DH |
| Véhicule à moteur Gasoil | 700 DH | 1500 DH | 6000 DH | 20.000 DH |

**Classe Program et son exécution**

|  |  |
| --- | --- |
| **public** **class** Program { **//à ne pas modifier**  **public** **static** **void** main(String[] args) {  Societe societe=**new** Societe(1, "SDCC");  Impots imp1=**new** Impots(2022);  societe.addImpots(imp1);  imp1.add(**new** DossierRevenus(111, 1000000));  imp1.add(**new** DossierVehicule(222, 6, 'E'));  imp1.add(**new** DossierVehicule(222, 8, 'G'));  imp1.add(**new** DossierVehicule(222, 11, 'E'));  imp1.add(**new** DossierVehicule(222, 15, 'G'));  System.***out***.println(societe);  System.***out***.println(societe.getImpots(2022));  }  } | Scoiété:1 - SDCC  Liste des impôts 2022  15% de revenus 1000000.0, Montant:150000.0  Vehicule E 6 CH, montant:350.0  Vehicule G 8 CH, montant:1300.0  Vehicule E 11 CH, montant:3000.0  Vehicule G 15 CH, montant:20000.0  Total:174650.0 |

**Classe Société :**

package meggouri;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Societe {  
  
 private long id;  
 private String raisonSociale;  
 List<Impots> impots = new ArrayList<>();  
  
 public Societe(long id, String rs) {  
 this.id = id;  
 this.raisonSociale = rs;  
 }  
  
 public void addImpots(Impots imp) {  
 this.impots.add(imp);  
 }  
  
 public Impots getImpots(int annee) {  
 for (Impots imp : impots) {  
 if (imp.getAnnee() == annee) {  
 return imp;  
 }  
 }  
 return null;  
 }  
  
 @Override  
 public String toString() {  
 return "Societe : " + id + " - " + raisonSociale;  
 }  
}

**Classe Dossier:**

package meggouri;  
  
public abstract class Dossier {  
  
 private long id;  
 protected double montant;  
  
 public Dossier(long id) {  
 this.id = id;  
 }  
  
 protected abstract void calculerMontant();  
  
 public abstract String toString();  
}

**Classe Impots:**

package meggouri;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Impots {  
 private int annee;  
 private double totalImpot;  
 public List<Dossier> dossiers = new ArrayList<>();  
  
 public Impots(int annee) {  
 this.annee = annee;  
 }  
  
 public void add(Dossier d) {  
 this.dossiers.add(d);  
 }  
  
 public int getAnnee() {  
 return annee;  
 }  
  
 public double getTotalImpot() {  
 for (Dossier d : dossiers) {  
 totalImpot += d.montant;  
 }  
 return totalImpot;  
 }  
  
 @Override  
 public String toString() {  
 for (Dossier imp : this.dossiers) {  
 imp.toString();  
 }  
 return "Liste des impots " + annee + "\n" + this.dossiers + "\n" + "Total : " + totalImpot;  
 }  
  
  
}

**Classe DossierVehicule:**

package meggouri;  
  
public class DossierVehicule extends Dossier {  
 private int nbChevaux;  
 private char categorie;  
  
 public DossierVehicule(int id, int nbc, char cat) {  
 super(id);  
 this.nbChevaux = nbc;  
 this.categorie = cat;  
 }  
  
 @Override  
 protected void calculerMontant() {  
 switch (this.categorie) {  
 case 'E':  
 if (this.nbChevaux >= 15) {  
 montant = 8000;  
 } else if (this.nbChevaux >= 11) {  
 montant = 3000;  
 } else if (this.nbChevaux >= 8) {  
 montant = 650;  
 } else montant = 350;  
 break;  
 case 'G':  
 if (this.nbChevaux >= 15) {  
 montant = 20000;  
 } else if (this.nbChevaux >= 11) {  
 montant = 6000;  
 } else if (this.nbChevaux >= 8) {  
 montant = 1300;  
 } else montant = 700;  
 }  
 }  
  
 @Override  
 public String toString() {  
 this.calculerMontant();  
 return "Vehicule " + categorie + " " + nbChevaux + " CH, montant : " + montant + "\n";  
 }  
}

**Classe DossierRevenus:**

package meggouri;  
  
public class DossierRevenus extends Dossier {  
  
 private double ca;  
  
 public double getCa() {  
 return ca;  
 }  
  
 public void setCa(double ca) {  
 this.ca = ca;  
 }  
  
 public DossierRevenus(long id, double ca) {  
 super(id);  
 this.ca = ca;  
 }  
  
 @Override  
 protected void calculerMontant() {  
 montant = 0.15 \* ca;  
 }  
  
 @Override  
 public String toString() {  
 return "15% de revenus " + ca + ", Montant : " + this.montant + "\n";  
 }  
  
  
}

**Classe Programme:**

package meggouri;  
  
public class Program {  
 public static void main(String[] args) {  
 Societe societe = new Societe(1, "SDCC");  
 Impots imp1 = new Impots(2022);  
 societe.addImpots(imp1);  
  
 imp1.add(new DossierRevenus(111, 1000000));  
 imp1.add(new DossierVehicule(222, 6, 'E'));  
 imp1.add(new DossierVehicule(222, 8, 'G'));  
 imp1.add(new DossierVehicule(222, 11, 'E'));  
 imp1.add(new DossierVehicule(222, 15, 'G'));  
  
 System.*out*.println(societe);  
 System.*out*.println(societe.getImpots(2022));  
 }  
}