|  |  |  |
| --- | --- | --- |
|  | **Année universitaire : 2023 - 2024**  **Examen Écrit**  **Semestre : 1** | |
| **Module :** Conception Orienté Objet et Programmation Java  **Classes :**3A  **Durée :** 1h30  **Nombre de pages :** 7 | | **Documents autorisés :** Non  **Internet autorisé :** Non  **Date :** 12/01/2024  **Heure :** 14H00 |
|  |
|  |
| **Nom et prénom :** | | **Classe :** |  |

**Exercice 1 (4 pts) :**

**Une seule réponse est correcte**

1. Étant donné que la classe Dog hérite de la classe Animal, que sera le résultat d'exécution de ce code ?



* 1. Le code fonctionne parfaitement
  2. Erreur d’exécution
  3. Erreur de compilation
  4. Le code fonctionne parfaitement à condition de redéfinir toutes les méthodes de la classe mère Animal dans la classe Dog

1. Comment le compilateur réagira-t-il face à ce bout de code lors de la compilation ?



* 1. La surcharge est valide et la compilation s'effectue correctement
  2. Des exceptions sont générées
  3. Les compilations se déroulent correctement car il ne s'agit pas d'une surcharge
  4. La surcharge est invalide et la compilation échoue

1. Quels sont les modificateurs d'accès qu'on ne peut appliquer à une classe en Java ?
   1. public et private
   2. public et default
   3. default et protected
   4. private et protected
2. Considérez le code suivant :



Quel sera le résultat de l'exécution du programme ?

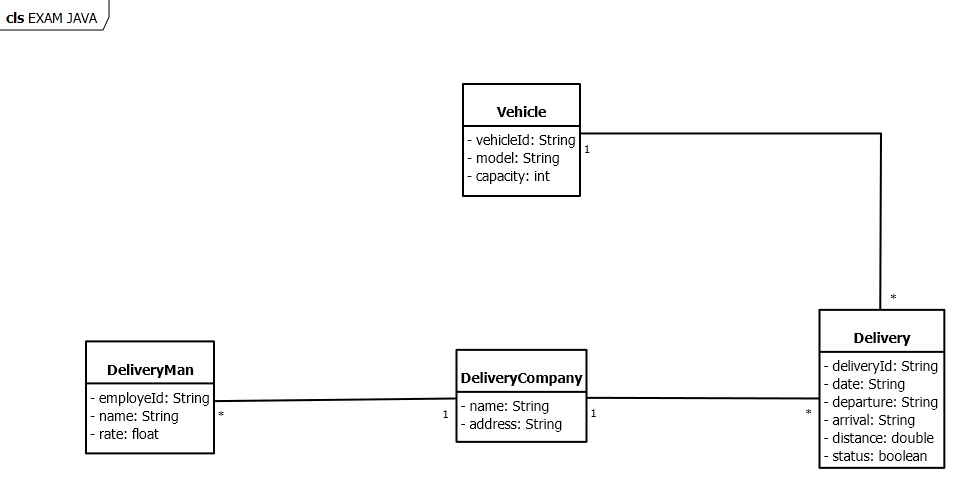
* 1. "Cannot divide by zero" suivi de "Finally block executed" et une exception sera levée avec le message "Additional exception"
  2. "Result : 0" suivi de "Finally block executed" et une exception sera levée avec le message "Additional exception"
  3. "Cannot divide by zero" suivi de "Finally block executed" sans autre exception
  4. "Result : 0" suivi de "Finally block executed" sans autre exception

**Exercice 2 (16 pts) :**

Dans le cadre d'un projet académique, vous êtes chargé de développer un programme en Java qui gère une entreprise de livraison selon le diagramme de classe suivant :

On suppose que les classes Vehicle, Delivery et DeliveryMan sont déjà implémentées.

**Nb : Respecter les Règles (Conventions) de Nommage Java.**



|  |
| --- |
| **Classe Vehicle** |
| public class Vehicle {  private String vehicleId;  private String model;  private int capacity;  public Vehicle() {  }  public Vehicle(String vehicleId, String model, int capacity) {  this.vehicleId = vehicleId;  this.model = model;  this.capacity = capacity;  }  */\*\*  \* Les getters, setters, equals, hashCode to toString sont déjà implémentées  \*/*  } |
| **Classe Delivery** |
| public class Delivery {   private String deliveryId;  private String date;  private String departure;  private String arrival;  private Vehicle vehicle;  private boolean status;   public Delivery() {  }   public Delivery(String deliveryId, String date, String departure, String arrival, Vehicle vehicle, boolean status) {  this.deliveryId = deliveryId;  this.date = date;  this.departure = departure;  this.arrival = arrival;  this.vehicle = vehicle;  this.status = status;  }  */\*\*  \* Les getters, setters, equals, hashCode to toString sont déjà implémentées  \*/*  } |
| **Classe DeliveryMan** |
| public class DeliveryMan {   private String employeId;  private String name;  private float rate;   public DeliveryMan() {  }   public DeliveryMan(String employeId, String name, float rate) {  this.employeId = employeId;  this.name = name;  this.rate = rate;  }  */\*\*  \* Les getters, setters, equals, hashCode to toString sont déjà implémentées  \*/*  } |
| **Classe DeliveryCompany** |
| public class DeliveryCompany {  private String name;  private String address;  private Map<DeliveryMan, List<Delivery>> deliveries;  // TODO1 (1,5pts) : implémentez un constructeur paramétré sachant que l’ensemble de livraisons doit être triée selon la note de livreur (rate).    // TODO2 (1pts) : Implémentez la logique pour retourner une Map qui regroupe la liste des livreurs selon leurs notes (rate).    // TODO3 (2,5pts) : Implémentez la logique pour vérifier si un véhicule spécifique est disponible à une date donnée.    // TODO4 (2,5pts) : Implémentez la logique pour affecter une livraison pour un livreur spécifique si le véhicule est disponible :   * Si le véhicule est disponible, ajoutez la livraison à l’ensemble des véhicules correspondant du livreur. * Si le véhicule n'est pas disponible, affichez un message indiquant que le véhicule n'est pas disponible.     // TODO5 (1,5pts) : Implémentez la logique pour vérifier si au moins un livreur a une note inférieure à la note spécifiée en paramètre. (Avec l’API Stream seulement)    // TODO6 (2pts) : Implémentez la logique pour compter les livraisons par départ pour un livreur spécifique. (Avec l’API Stream seulement)    // TODO7 (2pts) : Implémentez la logique pour afficher les modèles de véhicules pour un livreur spécifique à une date donnée. (Avec l’API Stream seulement)    // TODO8 (3pts) : Implémentez la logique pour afficher les livraisons pour une date spécifique (Avec l’API Stream seulement)  } |