TEK-UP University

Ecole Supérieure Privée Technologies & Ingénierie

Type d'épreuve	: ☐ Devoir	■ Examen SESSION PRINCIPALE	
Enseignant	: Slah Bouhari, Oussama Mansouri, Abdelaziz Menchari		
Matière	: Programmation JAVA		
Année Universitaire	: 2023-2024	**Semestre**	: 2
Classe	: TIC-1		
Documents	: ☐ Autorisés	■ Non autorisés	
Date	: 27/05/2024	**Durée :**	1h30mn
Nombre de pages	: 5		

Questions QCM (6 pts):

Q1. Quelle est la sortie de ce code Java ?

```
public static void main(String[] args) {
     int x = 5;
     System.out.println(x++); // Quelle est la sortie ?
     System.out.println(++x); // Et ici ?
}
```

A. 5 et 7
B. 5 et 6
C. 6 et 6
D. 6 et 7

Q2. Quelle est la syntaxe pour déclarer une méthode abstraite en Java ?

A. `void methodName() {}`
B. `abstract void methodName();`
C. `void methodName() abstract;`
D. `abstract methodName();`

Q3. Quelle est la différence entre == et. equals() en Java lors de la comparaison d'objets ?

A. == compare les références mémoire des objets, tandis que .equals() compare leurs contenus.
B. == compare les contenus des objets, tandis que .equals() compare leurs références mémoire.
C. == et .equals() effectuent la même opération de comparaison.
D. Aucune des réponses ci-dessus.

Q4. Quelle est la signification du mot-clé « final » en Java ?

A. Il indique que la classe ne peut pas être héritée.
B. Il indique que la méthode ne peut pas être redéfinie dans les classes filles.
C. Il indique que la variable ne peut pas être modifiée après son initialisation.
D. Toutes les réponses ci-dessus.

Q5. Quelle est la différence entre break et continue en Java ?

A. break termine complètement la boucle, tandis que continue passe à l'itération suivante.
B. break termine l'itération actuelle et passe à la suivante, tandis que continue termine complètement la boucle.
C. break termine complètement la boucle, tandis que continue arrête l'exécution de la boucle.
D. break et continue ont le même effet dans une boucle en Java.

Q6. Étant donné le fragment de code suivant, quelle modification permet au code d'imprimer "54321" :

```
3. public static void main(String[] args) {
4.       int x = 5;
5.       while (isAvailable(x)) {
6.           System.out.print(x);
7.
8.       }
9. }
10.
11. public static boolean isAvailable(int x) {
12.     return x-- > 0 ? true : false;
13. }
```

A. Remplacer la ligne 6 par `System.out.print(-x);`
B. Insérer `x--;` à la ligne 7
C. Remplacer la ligne 6 par `-x;` et insérer `System.out.print(x);` à la ligne 7
D. Remplacer la ligne 12 par `return (x > 0) ? false : true;`

Exercice 1 (4 pts):

Soit le code de la classe CompteBancaire suivant :

```
public class CompteBancaire {
     private String numeroCompte;
     private double solde;
```

```
    public CompteBancaire(String numeroCompte, double solde) {
        ……….
        ……..
    }
    public void setSolde(double montant) {
        ……..……. ;
    }
    public double getSolde() {
        return …… ;
    }
    public void retrait(double montant) {
        if (montant > solde) {
        System.out.println("Fonds insuffisants.");
        }
        else {
        solde -= montant;
        System.out.println(montant + " retiré avec succès.");
        }
    }
    public void depot(double montant) {
        solde += montant;
        System.out.println(montant + " déposé avec succès.");
    }
}
```

1. Ajoutez une classe d'exception personnalisée nommée **`SoldeInsuffisantException`**.
2. Compléter le code manquant de la classe `CompteBancaire` et modifier la méthode `retrait` de cette classe pour lancer une exception de type `SoldeInsuffisantException` **si le montant du retrait dépasse le solde du compte**.
3. Ajouter la méthode main a votre programme qui permet de :
    a. Créer un compte **CB1** avec le numéro **126699**, et un montant de **100 DT**
    b. Faire un retrait de **80 DT**

Exercice 2 (10 pts):

Une entreprise de location de véhicules électriques souhaite développer un système pour gérer ses véhicules disponibles à la location. Chaque véhicule a un numéro d'identification unique, une marque, un modèle et une date de location et une date de retour. La société souhaite également suivre les locations de chaque véhicule, y compris le client qui l'a loué et la date de location. La société a également besoin d'une fonctionnalité pour rechercher des véhicules disponibles pour la location.

Partie n°1 :

- Développez une interface `IVehicule` avec les méthodes suivantes :
  - **`boolean estDisponible()`** : Vérifie si le véhicule est disponible pour la location.
  - **`void louer (String client, Date date)`** : Permet de louer le véhicule à un client à la date spécifiée.
  - **`void retourner (Date date)`** : Permet de marquer le véhicule comme retourné à la date spécifiée.

- Écrire la classe abstraite `Vehicule` qui implémente l'interface `IVehicule`. Cette classe doit contenir les attributs privés suivants :
  - `id` : le numéro d'identification unique du véhicule `(String)`
  - `marque` : la marque du véhicule `(String)`
  - `modele` : le modèle du véhicule `(String)`
  - `date_location` : la date de location du véhicule `(Date)`
  - `date_retour` : la date de retour du véhicule `(Date)`
  - `etat_vehicule` : un indicateur indiquant l'état actuelle du véhicule qui peut être : Disponible, En_Charge , Loué `(Etat)`

Nous définissions un type énuméré Etat comme suit :

```
public enum Etat { Disponible, En_Charge , Loué };
```

Un constructeur avec trois paramètres doit être implémenter pour cette classe. Par défaut l'état d'un nouveau véhicule est "Disponible". ***Un véhicule qui est "En_Charge" ne peut pas être loué***.

Par défaut la date de location et la date de retour sont égaux à la date du jour de la création du nouveau véhicule.

On suppose et les méthodes accesseurs (getter et setter) sont fournies dans toutes les classes.

Les véhicules électriques disponibles pour location peuvent être soit :

- Une trottinette électrique (avec un attribut couleur `(String)` et un poids maximal`(int)`)
- Une moto électrique (avec un attribut paissance moteur `(int)`)

Écrire ***uniquement*** le code de la classes `TrotinetteElectrique` qui héritent de la classe Véhicule (fournir une implémentation pour les méthodes de l'interface `IVehicule`).

`Partie n°2 :`

Écrivez une classe de gestion `GestionLocation` avec les fonctionnalités suivantes :

- Une liste de véhicules pour enregistrer tous les véhicules disponibles à la location.
- **`ajouterVehicule (vehicule V)`** : méthode permet d'ajouter un nouveau véhicule à la liste des véhicules.
- **`rechercherVehiculesDisponibles ()`** : méthode pour rechercher et afficher tous les véhicules disponibles pour la location.
- **`louerVehicule (String id, String client, Date date)`** : méthode pour louer un véhicule spécifique à un client à une date spécifiée.
- **`retournerVehicule (String id, Date date)`** : méthode pour marquer un véhicule comme retourné à une date spécifiée.