DS OBJET

Durée : 2h – Tous documents autorisés

Cet examen comporte deux parties : une partie de questions de cours et une partie de réflexion et d'écriture de code en langage Java sur une application imaginaire. Nous vous conseillons de lire l'intégralité du sujet avant de débuter.

Questions de cours

Question 1 : Classes abstraites/Interfaces

Expliquez quel(s) intérêt(s) vous voyez à utiliser soit une classe abstraite, soit une interface.

Question 2 : Exceptions

Étant donné le code Java suivant (voir Figure 1) : quel est le résultat d'exécution de la classe `WhatAMess` ?

```
public class WhatAMess {

    public static void main(String[] args) {
        System.out.println("1");

        try {
            System.out.println("2");
            if(true) {
                throw new Exception();
            }
            System.out.println("3");
        } catch(Exception e1) {
            try {
                System.out.println("4");
                if(true){
                    throw new Exception();
                }
                System.out.println("5");
            } catch(Exception e2) {
                try {
                    System.out.println("6");
                    if(false) {
                        throw new Exception();
                    }
                    System.out.println("7");
                } catch(Exception e3) {
                    System.out.println("8");
                } finally {
                    System.out.println("9");
                }
                System.out.println("10");
            }
        }
        finally{
            System.out.println("11");
        }

    }
}
```

Figure 1 : Classe WhatAMess

Question 3 :

En vous basant sur la classe `Smarties`, présentée en Figure 2, et sur vos connaissances de Java : quel est le résultat de l'exécution du code Java écrit dans la classe `TestSmarties` (et présenté dans la Figure 3) ?

```
1   package ex1617;
2
3   public class Smarties {
4       private String couleur;
5
6       public Smarties(String couleur) {
7           this.couleur = couleur;
8       }
9       public String getCouleur() {
10          return couleur;
11      }
12      public void setCouleur(String couleur) {
13          this.couleur = couleur;
14      }
15      public void affiche() {
16          System.out.println("Smarties de couleur : "+this.couleur);
17      }
18  }
```

Figure 2 : Classe Smarties

```
1   package ex1617;
2
3   public class TestSmarties {
4
5       public static void m2(Smarties s1, Smarties smartB) {
6           s1.setCouleur("vert");
7           smartB = new Smarties("jaune");
8       }
9
10      public static void main(String[] args) {
11          // Creation des objets
12          Smarties smartR = new Smarties("rouge");
13          Smarties smartB = new Smarties("bleu");
14          // Affichage des smarties
15          smartR.affiche();
16          smartB.affiche();
17          // Appel de la méthode
18          m2(smartR,smartB);
19          // Affichage des smarties
20          smartR.affiche();
21          smartB.affiche();
22      }
23  }
```

Figure 3 : Classe TestSmarties

Question 4 :

En vous basant sur le code (voir Figure 4) des classes `Truc`, `Machin`, remplissez le tableau suivant lorsque l'instruction proposée est insérée à la ligne 28 de la classe `Test`.

Instruction proposée	Erreur compilation	Erreur exécution	OK (compilation et exécution)	Affichage résultat
`x.tutut(y) ;`	☐	☐	☐	
`y.tutut(y);`	☐	☐	☐	
`x.tutut(z);`	☐	☐	☐	
`z.tutut(y);`	☐	☐	☐	
`x.vroumvroum(z);`	☐	☐	☐	
`x.vroumvroum(y);`	☐	☐	☐	
`y.vroumvroum(y);`	☐	☐	☐	
`z.vroumvroum(y);`	☐	☐	☐	

```
1   class Truc {
2       void tutut(Truc t) {
3           System.out.println("tutut de Truc");
4       }
5
6       void vroumvroum(Truc t){
7           System.out.println("vrouvroum de Truc");
8       }
9   }
10
11
12  class Machin extends Truc{
13      void tutut(Truc t) {
14          System.out.println("tutut de Machin");
15      }
16
17      void vroumvroum(Machin t){
18          System.out.println("vrouvroum de Machin");
19      }
20  }
21
22  public class Test {
23      public static void main(String[] args){
24          Truc x = new Machin();
25          Machin y = new Machin();
26          Truc z = new Truc();
27
28          // l'instruction est à mettre ici
29      }
30  }
```

Figure 4 : Classes Machin, Truc et Test

Question 5 :

En vous basant sur les exemples fournis dans les vidéos additionnelles au cours, et sur le code suivant (également disponible en **annexe** à la fin du devoir en plus grand), réalisez des schémas de la mémoire (états de la pile et du tas) lors de l'exécution de la méthode `main()` de la classe principale `Main`. Vous ne devez réaliser ces schémas que pour les endroits indiqués par le marqueur suivant :

```
// POINT_X draw memory scheme at this point
```

Vous préciserez également l'affichage obtenu sur la sortie standard (résultats des `System.out.println(…)`) pour l'ensemble de l'exécution de cette application.

```java
package pizza;

public class Ingredient {

    private String name;
    private int quantity;

    public Ingredient(String name, int quantity) {
        this.name = name;
        this.quantity = quantity;
    }

    public String getName() {
        return this.name;
    }

    public int getQuantity() {
        return this.quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public Ingredient duplicate() {
        return new Ingredient(this.name, this.quantity);
    }

    public boolean equals(Object o) {
        if (o != null && o instanceof Ingredient) {
            Ingredient ing = (Ingredient) o;
            return (this.name.equals(ing.getName())
                    && this.quantity == ing.getQuantity());
        } else {
            return false;
        }
    }

    public String toString() {
        return this.name + " " + this.quantity + " unit(s)";
    }
}
```

```java
package pizza;

public class Main {

    public static void main(String[] args) {
        Pizza mg = new Pizza("Margherita");
        Ingredient tom = new Ingredient("Tomatoes", 150);
        Ingredient mozz = new Ingredient("Mozzarella", 100);
        mg.addIngredient(tom);
        mg.addIngredient(mozz);

        Pizza mg2 = new Pizza("Margherita");
        mg2.addIngredient(new Ingredient("Mozzarella", 100));
        mg2.addIngredient(new Ingredient("Tomatoes", 150));

        System.out.println("POINT 1");
        System.out.println(mg);
        System.out.println(mg2);
        System.out.println(mg == mg2);
        System.out.println(mg.equals(mg2));
        // POINT_1 draw memory scheme at this point

        Pizza np = mg.duplicate();
        Ingredient ingFromMG = mg.getIngredient(0);
        Ingredient ingFromNP = np.getIngredient(0);

        System.out.println("POINT 2");
        System.out.println(np);
        System.out.println(mg);
        System.out.println(np == mg);
        System.out.println(np.equals(mg));
        System.out.println(ingFromMG == ingFromNP);
        System.out.println(ingFromMG.equals(ingFromNP));
        // POINT_2 draw memory scheme at this point

        Ingredient ingFromMG2 = mg2.getIngredient(1);
        ingFromMG.setQuantity(ingFromMG.getQuantity() + 10);
        ingFromMG2.setQuantity(ingFromMG2.getQuantity() + 10);
        ingFromNP.setQuantity(ingFromNP.getQuantity() + 10);
        System.out.println("POINT 3");
        System.out.println(mg);
        System.out.println(mg2);
        System.out.println(np);
        // POINT_3 draw memory scheme at this point
    }
}
```

```java
package pizza;

import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

public class Pizza {

    private String name;
    private List<Ingredient> ingredients;

    public Pizza(String name) {
        this.name = name;
        this.ingredients = new ArrayList<>();
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }

    public void addIngredient(Ingredient ing) {
        this.ingredients.add(ing);
    }

    public Ingredient getIngredient(int index) {
        return this.ingredients.get(index);
    }

    public List<Ingredient> getIngredients() {
        return this.ingredients;
    }

    public int getNbIngredients() {
        return this.ingredients.size();
    }

    public Pizza duplicate() {
        Pizza d = new Pizza(this.getName());
        Iterator<Ingredient> it = this.ingredients.iterator();
        while (it.hasNext()) {
            d.addIngredient(it.next());
        }
        return d;
    }

    public boolean equals(Object o) {
        if (o == null || !(o instanceof Pizza)) {
            return false;
        }

        Pizza p = (Pizza) o;
        if (!this.getName().equals(p.getName())
                || this.getNbIngredients() != p.getNbIngredients()) {
            return false;
        }

        Iterator<Ingredient> it = p.getIngredients().iterator();
        while (it.hasNext()) {
            if (!this.ingredients.contains(it.next())) {
                return false;
            }
        }
        return true;
    }

    public String toString() {
        String s = this.name + "[";
        for (int i = 0; i < this.ingredients.size(); i++) {
            if (i > 0) {
                s += ",";
            }
            s += this.ingredients.get(i);
        }
        s += "]";
        return s;
    }
}
```

Étude d'une application imaginaire : Système d'Information (SI) de la société « MyDesignFactory »

Dans cette partie, nous vous demandons de vous intéresser à la création du Système d'information minimaliste pour la société « MyDesignFactory » qui fabrique des meubles et produits de décoration d'intérieur design.

NB : cette application et le descriptif qui suit est parfaitement fictif et ne reflète pas ce que l'on vous demanderait pour vraiment créer un système d'information même minimaliste. Nous vous demandons de ne pas en tenir compte et de répondre aux questions demandées sans relever les imprécisions ou les invraisemblances.

Votre client la société « MyDesignFactory » vous a contacté pour développer une nouvelle version minimaliste de leur SI. Ils vous fournissent seulement une description textuelle de leurs attentes et ce sera à vous de proposer des solutions et une implémentation du SI minimaliste.

Le descriptif de la première version fournie par la société « MyDesignFactory » est représenté dans la Figure 8. Assurez-vous bien de lire attentivement ce descriptif avant de répondre aux questions.

Question 6.1 : Diagramme de classes UML

Proposez un diagramme de classes et une description brève de vos choix lors de la réalisation de ce diagramme et des relations entre les classes et/ou leurs attributs.

Ne faites apparaître que les attributs que vous jugez utiles et les méthodes les plus importantes (**pas de getters/setters, ni tous les constructeurs**).

Si vous voulez représenter des classes ou méthodes abstraites, veuillez adopter une notation facilement lisible (p. ex. en couleur, souligné, etc.) et **précisez quelle notation vous avez choisi !**

Ajoutez à votre diagramme une justification de vos choix (par ex. si vous n'avez pas représenté dans votre diagramme une partie des informations fournies par la société).

Question 6.2 : Magasins

La société « MyDesignFactory » est satisfaite de votre proposition, elle vous demande maintenant de pouvoir rajouter à votre SI leurs magasins qui existant un peu partout en France.
Un magasin possède un certain nombre d'articles à vendre qui peuvent ou non être expédiés aux clients. Quand ils ne peuvent pas être expédiés, les clients doivent venir les retirer en magasin.

Modifiez votre diagramme et/ou ajoutez des classes permettant de prendre cette modification en compte. Implémentez le constructeur par défaut de la classe Magasin.

Nous souhaitons une nouvelle version de notre SI, minimale dans un premier temps. Dans cette première version nous souhaitons pouvoir modéliser les différents articles design que nous produisons et vendons.

Tous nos articles ont en commun d'avoir une référence (sous forme de chaine de caractères), un prix hors-taxe (HT, nombre réel), un prix toutes taxes comprises (TTC, nombre réel) et un nom/description succincte (chaine de caractères).

Nos articles sont de différents types : meubles, luminaires ou accessoires. Chaque type a plusieurs sous-types :

- Canapé, fauteuil, table, rangement et lit pour les meubles.
- Lampadaires, lampes, abat-jour ou suspension pour les luminaires.
- Tapis, Salle de bains, décoration et textile pour les accessoires.

Enfin chaque sous-type a plusieurs modèles, par exemple il existe un modèle de canapé « Rochester » qui peut avoir plusieurs couleurs différentes et existe en deux tailles différentes : 3 ou 5 personnes. Tous les modèles de tous les sous-types ont les mêmes caractéristiques, par exemple tous les modèles de canapés sont caractérisés par une couleur et une taille (décrite comme un nombre de personnes pouvant s'assoir sur le canapé).

Certains articles peuvent être expédiés et d'autres non car ils sont trop volumineux. Pour les articles ne pouvant pas être expédiés les clients doivent venir les chercher en magasin. Les articles ne pouvant être expédiés sont les suivants :

- Canapés,
- Tables,
- Tapis,
- Lit,
- Lampadaires.

Les articles expédiés ont des frais de livraison qui seront ajoutés au prix TTC. Les articles les plus chers (prix TTC > 1000€) peuvent être payés en 3 fois sans frais.

Figure 5 : Description fournie par "MyDesignFactory" sur leur mode de fonctionnement.

Question 6.3 : Limites ?

Quelles sont les limites au schéma que vous proposez, et comment pourriez-vous les résoudre.

Annexe 1 : Code du paquetage pizza.

```
package pizza;

public class Ingredient {

    private String name;
    private int quantity;

    public Ingredient(String name, int quantity) {
        this.name = name;
        this.quantity = quantity;
    }

    public String getName() {
        return this.name;
    }

    public int getQuantity() {
        return this.quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public Ingredient duplicate() {
        return new Ingredient(this.name, this.quantity);
    }

    public boolean equals(Object o) {
        if (o != null && o instanceof Ingredient) {
            Ingredient ing = (Ingredient) o;
            return (this.name.equals(ing.getName())
                    && this.quantity == ing.getQuantity());
        } else {
            return false;
        }
    }

    public String toString() {
        return this.name + " " + this.quantity + " unit(s)";
    }
}
```

```java
package pizza;

import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

public class Pizza {

    private String name;
    private List<Ingredient> ingredients;

    public Pizza(String name) {
        this.name = name;
        this.ingredients = new ArrayList<>();
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }

    public void addIngredient(Ingredient ing) {
        this.ingredients.add(ing);
    }

   public Ingredient getIngredient(int index) {
        return this.ingredients.get(index);
   }

   public List<Ingredient> getIngredients() {
        return this.ingredients;
   }

   public int getNbIngredients() {
        return this.ingredients.size();
   }

    public Pizza duplicate() {
        Pizza d = new Pizza(this.getName());
        Iterator<Ingredient> it = this.ingredients.iterator();
        while (it.hasNext()) {
            d.addIngredient(it.next());
        }
        return d;
    }
    public boolean equals(Object o) {
        if (o == null || !(o instanceof Pizza)) {
            return false;
        }

        Pizza p = (Pizza) o;
        if (!this.getName().equals(p.getName())
                || this.getNbIngredients() != p.getNbIngredients()) {
            return false;
        }

        Iterator<Ingredient> it = p.getIngredients().iterator();
        while (it.hasNext()) {
            if (!this.ingredients.contains(it.next())) {
                return false;
            }
        }
        return true;
    }

    public String toString() {
        String s = this.name + "[";
        for (int i = 0; i < this.ingredients.size(); i++) {
            if (i > 0) {
                s += ",";
            }
            s += this.ingredients.get(i);
        }
        s += "]";
        return s;
    }
}
```

```java
package pizza;

public class Main {

    public static void main(String[] args) {
        Pizza mg = new Pizza("Margherita");
        Ingredient tom = new Ingredient("Tomatoes", 150);
        Ingredient mozz = new Ingredient("Mozzarella", 100);
        mg.addIngredient(tom);
        mg.addIngredient(mozz);

        Pizza mg2 = new Pizza("Margherita");
        mg2.addIngredient(new Ingredient("Mozzarella", 100));
        mg2.addIngredient(new Ingredient("Tomatoes", 150));

        System.out.println("POINT 1");
        System.out.println(mg);
        System.out.println(mg2);
        System.out.println(mg == mg2);
        System.out.println(mg.equals(mg2));
        // POINT_1 draw memery scheme at this point

        Pizza np = mg.duplicate();
        Ingredient ingFromMG = mg.getIngredient(0);
        Ingredient ingFromNP = np.getIngredient(0);

        System.out.println("POINT 2");
        System.out.println(np);
        System.out.println(mg);
        System.out.println(np == mg);
        System.out.println(np.equals(mg));
        System.out.println(ingFromMG == ingFromNP);
        System.out.println(ingFromMG.equals(ingFromNP));
        // POINT_2 draw memery scheme at this point

        Ingredient ingFromMG2 = mg2.getIngredient(1);
        ingFromMG.setQuantity(ingFromMG.getQuantity() + 10);
        ingFromMG2.setQuantity(ingFromMG2.getQuantity() + 10);
        ingFromNP.setQuantity(ingFromNP.getQuantity() + 10);
        System.out.println("POINT 3");
        System.out.println(mg);
        System.out.println(mg2);
        System.out.println(np);
        // POINT_3 draw memery scheme at this point
    }
}
```