Université Mohammed V - Rabat
Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes
ENSIAS

Examen M2.5.2 – Programmation Objet – 12/06/2017

Année Universitaire : 2016 - 2017
Filière :
Professeur : A. EL FAKER

Durée : 1 heure
Documents **non électroniques** autorisés (cours et TPs)

Q1. Considérons l'interface I et les classes **A** et **B**, indiquez les instructions correctes:

A ✓ `B x = new B() ;`
B `I x = new I() ;`
C ✓ `I x = new B() ;`
D `A x = new A() ;`
E ✓ `A x = new B() ;`
F Aucune des instructions n'est correcte

```
interface I {
  public void m();
}
```

```
abstract class A {
  abstract void h() ;
  public void g() {...}
}
```

```
class B extends A implements I {
   public void m() {...}
   public void m(int x) {...}
   public void h() {...}
}
```

Q2. Avec les mêmes classes/interface, quelles sont les propositions correctes:

A ✓ Dans B, g() est héritée
B Dans B, m() est hérité
C ✓ Dans B, m() est surchargée
D ✓ m() doit obligatoirement être définie dans B
E ✓ h() doit obligatoirement être définie dans B
F Aucune des propositions n'est correcte

Q3. On vous demande de réécrire la classe `JourDeTravail` suivante :

```
public class JourDeTravail
{
   public int jour; //entre 1 et 5
}
```

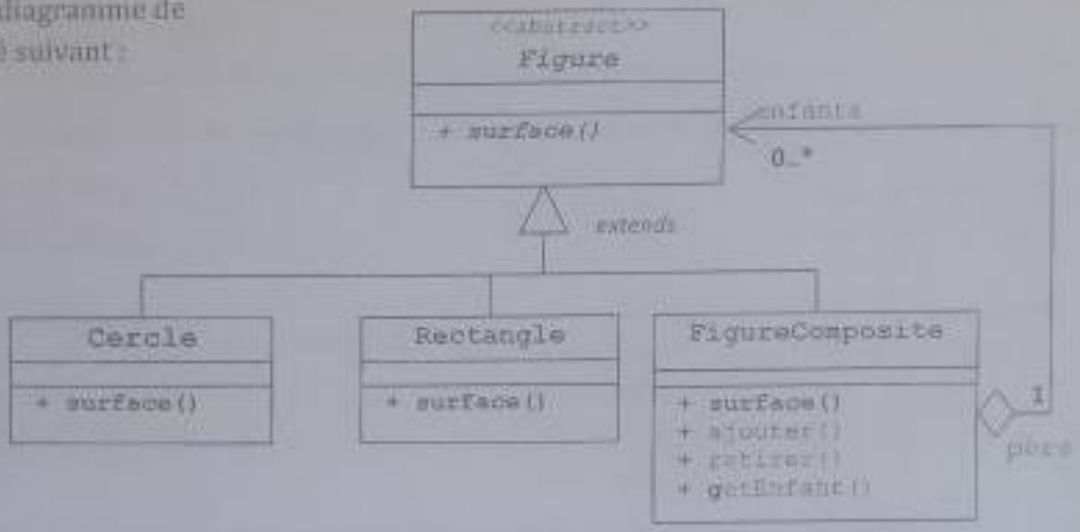
a) La classe doit respecter le principe **d'encapsulation**

b) Elle fournit des méthodes d'instance qui permettront à la variable d'instance d'être **interrogée** et **modifiée** via un objet de la classe

c) Une de vos méthodes (ci-dessus) doit incorporer une routine de vérification des erreurs pour **empêcher de fixer un jour non valide**. Utiliser pour cela la gestion des **exceptions**.

On peut manipuler un groupe d'objets de la même façon que s'il s'agissait d'un seul objet. On utilise pour cela ce qu'on appelle un **objet composite**. Une `Figure` géométrique, par exemple, peut être un `Cercle`, un `Rectangle` ... ou bien une **composition** de différentes figures (`FigureComposite`).

Considérons le diagramme de classes simplifié suivant :



Q4.

a) Ecrire en Java la classe `Figure` : c'est l'abstraction pour toutes les figures géométriques.

b) Les deux lignes suivantes sont-elles **correctes** ou **incorrectes** ? **Justifier.**

```
Figure[] t = new Figure[10] ;
t[3] = new Cercle() ;
```

Q5. Ecrire en Java la classe `FigureComposite`. Cette classe stocke des composants « enfants » et permet d'y accéder. Vous ajouterez des attributs si nécessaire, ceux-ci sont déclarés en respectant le **principe d'encapsulation**.

Vous définissez notamment:

- Un **constructeur sans paramètre** où les attributs sont correctement instanciés
- La méthode `double surface()` en utilisant les « enfants »
- La méthode `void ajouter(Figure f)` pour ajouter une figure à l'objet composite
- La méthode `void retirer(Figure f)` pour retirer une figure de l'objet composite
- La méthode `Figure getEnfant(int i)` pour récupérer la i^{e} figure de l'objet composite

- Q1: Voir exam_2015_2016
- Q2: Voir exam_2015_2016

```
package exam_2016_2017;

public class JourDeTravail {
  private int jour;

  public int getJour() {
    return jour;
  }

  public void setJour(int jour) {
    if (jour < 1 || jour > 5) {
      throw new IllegalArgumentException("Argument jour doit etre
compris entre 1 et 5.");
    }
    this.jour = jour;
  }
}
```

```
package exam_2016_2017;

public abstract class Figure {
  public abstract double surface();

  /*
   * On peut creer un tableau de type Figure: Figure[] figures = new
Figure[10];
   * Mais on doit instancier chacun des elements du tableau par une
classe non abstraite
   * qui hérite de Figure.
   */
}
```

```java
package exam_2016_2017;

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

public class FigureComposite extends Figure {
  private List<Figure> enfants;

  public FigureComposite() {
    this.enfants = new ArrayList<>();
  }

  public List<Figure> getEnfants() {
    return enfants;
  }

  public void ajouter(Figure figure) {
    enfants.add(figure);
  }

  public void retirer(Figure figure) {
    enfants.remove(figure);
  }

  public Figure getEnfant(int i) {
    return enfants.get(i);
  }

  @Override
  public double surface() {
    double surface = 0;
    for (Figure figure : enfants) {
      surface += figure.surface();
    }
    return surface;
  }
}
```