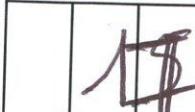


NOM POUPA
Prénom Adrien
Promo L3 C 2018
Date 25/03/16



POUPA Adrien
L3PRIME - 2015

17,50
Trés Bon travail

1/3

MATIÈRE Java 2

Questions de cours

95/10

- I) 1) ~~La fonction~~ La fonction `getContentPane` retourne un objet de type `Container`, qui peut être un `JPanel` par exemple, `Container` contenant plusieurs interfaces de containers. Il s'agit de l'espace de la fenêtre Swing où on pourra ajouter les objets voulus (`JButton`, `JTextArea` ...),
- 2) Grâce au modèle événementiel des classes Swing, le programmeur n'a pas à écrire de classe définissant un événement, il peut directement utiliser des composants déjà réalisés (ActionListener implémentant `actionPerformed` pour un clic par exemple). De même, il n'a pas d'écriture d'évenements à écrire.
- 3) Le container nécessaire au bon déroulement du modèle événementiel des classes Swing permet de changer le comportement d'un objet précis qu'il contient lorsqu'une action est déclenchée.
- 4) ActionListener doit implémenter la méthode :
`public void actionPerformed(ActionEvent e)`

~~faire Adaptez~~

5) On peut utiliser une classe Anonyme pour cela :
On pourra @override les méthodes désirées.) Moi

II) 6) La programmation concorrente consiste en la gestion de deux accès concurrents simultanés sur une même ressource. Il s'agit d'éviter les erreurs et la corruption de données.

7) On peut utiliser une classe anonyme ou une classe dédiée au Thread qui implemente l'interface Runnable en faisant @ override de public void run(). La classe anonyme est plus rapide à écrire mais ne peut pas être lancée directement via :

~~Thread t = new Thread(new MaClasseDeThread());~~
Cette dernière façon est la meilleure, on peut lancer autant de threads que voulu, appliquer un join().

8) La seconde méthode est en ~~*~~, pour la première, on peut faire dans le code de Swing par exemple :

```
public static void main (String [] args) {
```

```
    SwingUtilities.invokeLater (new Runnable () {
```

@override

```
    public void run() {
```

ju

- -

?); ?

g) La synchronisation d'une application concurrente permet à deux utilisateurs de travailler sur la même ressource sans avoir à se soucier de répercussions du travail de l'un sur l'autre (gestion de comptes en banque par exemple).

h) Le mot clé `synchronized`, utilisé en bloc :

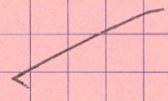
`Object lock = new Object();`

`public void m-Fonction() {`

`synchronized(lock) { } // On peut aussi faire synchronized(this)`

`// Thread safe, Ⓛ performant`

`}`



ou bien dans le prototype d'une fonction

`public void synchronized () { } // Fonction entière protégée, plus lent`

`-----`

`}`



assure la synchronisation.

`Volatile` à peu près le même effet mais ne s'applique que sur un attribut de la classe au particulier, sans protéger toute une fonction.

- 11) Une classe définissant un événement hérite de `java.util.EventObject`
- 1/15
1/2
- 12) ~~Serializable et l'interface nécessaire à la sérialisation des Java Beans. L'objet doit implementer un ID unique et être non mutable~~

B/ Les méthodes pour l'abonnement et le désabonnement d'un JavaBean:

~~public void addListener(EventListener e) { ... }~~
~~public void removeListener(EventListener e) { ... }~~

Elles servent à stocker ou enlever d'un container de type Vector les objets à notifier d'un changement de son état où elles sont implémentées.

Pour être plus précis, elles stockent un événement qui lui contiendra l'obj et à notifier.

- 14) Un JavaBean est :
- non mutable
 - Serializable
 - contient des accessseurs get... et set...
ou is... pour des booléens.
- Type de Propriétés
- à un constructeur par défaut (sans arguments)
 - à un ID unique
- non Propriété non Propriété

- 15) Ces méthodes se trouvent dans le cadre de propriétés liées. Elle permet de déclencher (fire) les événements attachés au changement d'une propriété.
- Par exemple, le changement de température d'un point chaud doit être notifié aux thermomètres posés sur lui.

NOM PouPA
Prénom Adrien
Promo L'3 C 2018
Date 25/03/16

2/3
(15/15)

MATIÈRE Java 2

- IV) 16) Un objet non mutable est :
- plus sûr (pas de corruption de données, Thread safe)
 - plus rapide si on ne crée pas de nouvelles références à chaque getter
 - plus facile à manipuler
- 17) La classe suivante est mutable : dans une fonction main, on fait

```
Cat lazy = new Cat(new StringBuilder("Garfield"));  
System.out.println(lazy.getName()); // Garfield.  
lazy.getName().reverse();
```

System.out.println(lazy.getName()); // Oldlefrahi

18) package mutable;

```
public final class Cat {
```

```
    private final StringBuilder name;
```

```
    public Cat(StringBuilder name) {
```

```
        this.name = new StringBuilder(name);  
    }
```

```
    public StringBuilder getName() {
```

```
        return new StringBuilder(name);  
    }
```

}

On aurait pu utiliser String, non mutable,
au lieu de String Builder qui est mutable.

Exercice 1

public ~~final~~ class Cercle { 0/2 non compris

private int x;

private int y;

private int rayon;

public Cercle (int x, int y, int rayon) {

this.x = new Integer(x);

this.y = new Integer(y);

this.rayon = new Integer(rayon);

}

public void translate (int dx, int dy) {

x = new Integer(x + dx);

y = new Integer(y + dy);

}

}

JW

Exercice 2

(3/3)

Implémentation du Design Pattern Singleton :

```
public class Connexion {  
    private String nom;  
    private static Object connexion = new Object("IDBC Database");  
  
    // Constructeur privé  
    private Connexion(String nom) {  
        this.nom = nom;  
    }  
  
    // Méthode retournant le singleton désiré  
    public Object getInstance() {  
        return connexion;  
    }  
}
```

On a donc un constructeur privé \Rightarrow la classe n'est pas instanciable.

L'objet de connexion est statique, instancié dans les attributs \Rightarrow il sera créé au chargement de la classe et pas à l'appel de getInstance.

Toutefois, je n'ai pas compris dans le sujet l'utilité de la String de nom ? Ni si l'objet Object ici, devrait être un objet de type IDBC ?

Utilisant JPA pour mon projet, je ne m'étais
pas penché sur la question.

~~✓~~

Exercice 3

~~1 2 3 4 5~~

~~IMPORT~~

Au niveau des "...", on ajoute le code suivant :

Tst. addActionListener (new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

System.out.println("Tst clic");

}

});

for (JButton b : buttons) {

b.addActionListener (new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String text = ((JButton)e.getSource()).getText();

System.out.println("*" + text + "*");

}

});

}

et ajouter des Boutons au Panel de panneau

NOM Poupa
Prénom Adrien
Promo L13C 2018
Date 25/03/16

3/3

MATIÈRE Java 2

```
this.setLayout(new FlowLayout()); // ajoute en x
this.add(test); // ajoute bouton test
for (JButton b : buttons) {} // ajoute Un Deux Trois
this.add(b);
} now pack(); // affichage effectif
return 1 panel
```

Il faut importer :

```
import java.awt.*;
import javax.swing.*; } Imports
```

Note : je ne suis pas absolument certain de la syntaxe get Source pour récupérer l'origine du bouton cliqué puis du getText pour récupérer son nom. Dans le pire des cas, on peut dupliquer 3 fois le code écrit pour le bouton de test en faisant buttons[0].add(...)
buttons[1].add(...)
buttons[2].add(...)

et en changeant le contenu du `println()`.

Cette classe s'appelle `avec`.

```
public class Launcher {
```

```
    public static void main (String [] args) {
```

```
        SwingUtilities.invokeLater (new Runnable () {
```

```
            @Override
```

```
            public void run () {
```

```
                BoutonsListeners b = new BoutonsListeners();
```

```
                b.setVisible (true);
```

```
}
```

```
});
```

```
{
```

-13

Bonus

+1

~~=~~

Pour avoir une fenêtre Swing appelée en `Thread-safe`,

