

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



POUPA Adrien  
L3PRIME - 2015

Bon Travail

1/3

## MATIÈRE Java 2

Questions de cours 95/100

1) 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'événements 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) {  
}
```



## Thread Adapter

5) On peut utiliser une classe Anonyme pour cela :  
On pourra ~~3~~ @override les méthodes désirées. / Non

II) 6) La programmation concurrente 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 implémente 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 thread = 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 () {
```

```
        // ...  
    }  
});
```

```
};
```



9) 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).

10) 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
```

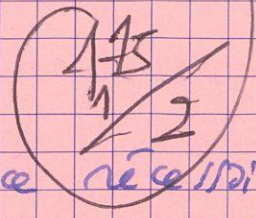
```
}
```

assure la synchronisation.

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



11) Une classe définissant un événement hérite de `java.util.EventObject`



12) `Serializable` est l'interface nécessaire à la sérialisation des Java Beans. L'objet doit implémenter 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 extraire d'un container de type `Vector` les objets à notifier d'un changement de la classe où elles sont implémentées.

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

14) Un JavaBean est : Type de Propriétés

- non mutable
- serializable
- contient des accesseurs `get...` et `set...`  
ou is... pour des booléens.

nomPropriété      nomPropriété

15) Cette méthode se trouve 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()); // dleirfrah
```

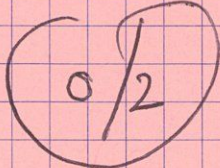
18) package mutable;

```
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 <sup>final</sup> class Cercle {  NON COMPLET

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);  
}

}



## Exercice 2

3/3

Implémentation du Design Pattern Singleton:

```
public class Connexion {  
    private String nom;  
    private static Object connexion = new Object("JDBC Database");  
  
    // Constructeur privée  
    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 JDBC?



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

### Exercice 3

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

```
test.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("Test 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 ajout des boutons au Panel de plus loin!



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

3/3

## MATIÈRE Java 2

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

Il faut importer :

```
import java.awt.*;  
import javax.swing.*; // JAPRÉCIS
```

Note : je ne suis pas absolument certain de la syntaxe `getSource` 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(...)
```

ju



et en changeant le contenu du print ln();

cette classe s'appelle avec :

```
public class Lancher {
```

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

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

```
@ override
```

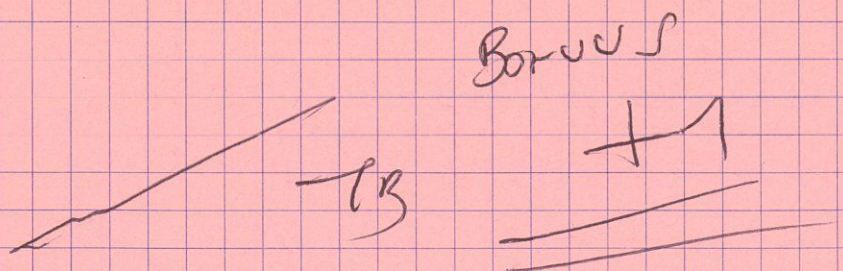
```
public void run () {
```

```
Buttons Listener b = new Buttons Listener;  
b.set Visible (true);
```

```
}
```

```
});
```

```
}
```



pour avoir une fenetre Swing appelee en Thread-safe.



