IFT1020 - Session Été, Intra

Mohamed Lokbani

IFT1020 – INTRA - Solutionnaire

Inscrivez tout de suite votre nom et code permanent.

Nom: _____| Prénom(s): _____|

Signature: _____| Code perm: _____|

Date : 14 juin 2005

Durée: 2 heures (de 17h30 à 19h30) Local: 1360 du Pavillon A.-AISENSTADT.

Directives:

- Toute documentation permise.
- Calculatrice **non** permise.

- Répondre directement sur le questionnaire.
- Les réponses **doivent être brèves, précises et clairement présentées.**

1._____/15 (1.1, 1.2, 1.3, 1.4)

2._____/10 (2.1)

3._____/10 (3.1)

4._____/20 (4.1, 4.2)

5._____/25 (5.1, 5.2)

6._____/20 (6.1, 6.2, 6.3, 6.4)

Total:_____/100

Question 1 (15 points)

1.1 Que signifient les termes "une variable d'instance" et "une méthode d'instance"?

Une variable d'instance et une méthode d'instance ne sont pas des variables et des méthodes static de la classe. Elles n'appartiennent pas donc à la classe mais plutôt à l'objet, instance de cette classe. Ces variables et méthodes traduisent le comportement de l'objet

1.2 Expliquer rapidement, quels sont les problèmes posés suite à l'utilisation de l'héritage multiple? Comment a-t-on résolu ces problèmes en Java?

Les problèmes posés par l'héritage multiple sont des problèmes d'ambiguïté. Deux cas d'ambiguïté sont possibles : A et B deux classes qui dérivent de la classe C qui définit la méthode affiche. La classe D qui dérive au même temps de A et B, ne sait pas de quelle méthode affiche il s'agit. Celle qui est héritée par A ou bien celle qui est héritée par B. Un autre cas d'ambiguïté est comme suit : Une classe Z qui drive au même temps de X et Y, sachant qu'on a défini la méthode affiche dans X et Y. Laquelle affiche Z va-t-elle héritée ? Celle de X ou celle de Y ? Ces problèmes d'ambiguïté ont été solutionnés dans Java, en interdisant d'abord l'héritage multiple d'implémentation (au niveau des classes), et en autorisant l'héritage multiple au niveau des interfaces. On remonte ainsi toutes les choses communes au niveau de l'interface, par la suite aux classes de décider oui ou non d'implémenter ces interfaces. C'est une question de design résolue à l'aide des interfaces.

1.3 Une applet peut définir la méthode suivante:

```
public boolean keyDown(Event evt, int key);
```

Quelle est l'utilité d'une telle méthode? À quoi sert l'argument « key »? Pourquoi cette méthode est-elle déclarée « public »?

Cette méthode sera appelée par le système qu'un usager appuie sur une touche du clavier. L'argument « key » fait référence à la touche du clavier appuyée par l'usager. La méthode est public car elle est appelée par le système en dehors de la classe où elle a été définie.

1.4 Un des gros problèmes dans le développement de logiciels est la réutilisation d'un travail déjà réalisé. Expliquer brièvement comment la programmation par objets peut aider à solutionner ce problème de la réutilisation.

Une classe est considérée comme un moule. Ce moule est une composante logicielle indépendante qui peut-être copiée en entier d'un projet de programmation vers un autre. C'est donc un exemple de réutilisation indépendant de la notion de P.O.O. En P.O.O, un programmeur peut bâtir sur son ancien travail, ses extensions ou modifications, en s'aidant pour cela des sous-classes (héritage) et cela à partir des classes déjà existante. Ainsi donc, le programmeur n'aura qu'à introduire les changements nécessaires pour adapter les anciennes choses à la problématique posée par le nouveau projet.

Question 2 (10 points) **Tout en justifiant votre réponse**, que va afficher en sortie le programme suivant qui compile et s'exécute correctement?

```
class A{}
class B extends A {}
class C extends B {}
class D extends C {}
public class MaClasse {
    public static void main(String args []) {

        B b = new C();
        A a = b;
        if (a instanceof A) System.out.println("A");
        if (a instanceof B) System.out.println("B");
        if (a instanceof C) System.out.println("C");
        if (a instanceof D) System.out.println("D");
        if (a instanceof Object) System.out.println("O");
    }
}
```

Chaque instance de C est une instance aussi de A et B, car les classes A et B sont des superclasses à la classe C. Par ailleurs, la classe Objet est par défaut au dessus de toutes les classes. De ce fait, le programme affiche ce qui suit :

A
B
C
O

Question 3 (10 points) **Tout en justifiant votre réponse**, quels sont les constructeurs qui doivent exister dans la classe Base pour que le fragment de code suivant puisse compiler correctement ?

```
public class Test extends Base {
   public Test(int j) {
   }
   public Test(int j, int k) {
     super(j, k);
   }
}
```

Le constructeur Test(int) fera appel au constructeur par défaut de la classe Base i.e. Base(). Le constructeur Test(int,int) fera appel au constructeur Base(int,int) à cause de l'appel de super(j,k).

Or vu que définir le constructeur Base(int,int) dans la classe Base a comme conséquence de masquer la présence du constructeur par défaut Base() de la classe Base. De ce fait, il faudra définir le constructeur sans argument Base() de manière explicite.

En conclusion, nous avons besoin de définir de manière explicite deux constructeurs dans la classe Base : Base() et Base(int,int).

Question 4 (20 points)

4.1 Un Compteur est un objet défini par les propriétés suivantes:

- il possède une valeur entière, positive ou nulle, nulle à sa création; qui ne peut varier que par pas de 1 (incrémentation).
- Il possède aussi, deux méthodes incremente et getValeur, pour respectivement incrémenter la valeur entière et afficher cette dernière en sortie.

Écrire la définition de la classe Compteur (y compris la définition des méthodes).

(Une solution parmi d'autres ….)

```
class Compteur {
      private int valeur ;
      public void incremente() {valeur++ ;}
      public int getValeur() {return valeur ;}
}
```

4.2 Écrire la méthode Question, sans arguments et sans valeur de retour, qui permet de réaliser ce qui suit :

- Lancer aléatoirement 100 fois une pièce de monnaie.
- En supposant que nous avons une chance sur deux d'avoir une des deux faces possibles (pile ou face avec donc 50/50 de chance) ; pour chaque tirage on met à jour les compteurs pour le coté pile et le coté face de la pièce, on utilise pour cela la classe Compteur de **6.1**.
- Après les 100 tirages, on affiche en sortie les résultats obtenus.
- Vous pouvez supposer que nous avons déjà importé dans le programme tous les paquetages nécessaires. Que la méthode Question se trouve dans une classe qui fait partie du même paquetage que la classe Compte (Compte est donc accessible) etc.

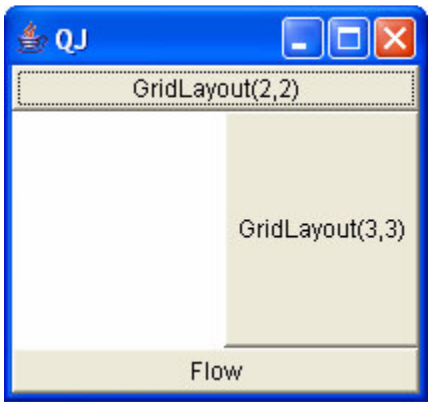
(Une solution parmi d'autres ….)

```
void Question(){
      Compteur pile = new Compteur() ;
      Compteur face = new Compteur() ;
      for (int =0 ; i<100 ; i++)
            if (Math.random()<0.5)
                  pile.incremente();
            else
                  face.incremente();

      System.out.println("Nbre de pile: " + pile.getValeur());
      System.out.println("Nbre de pile: " + pile.getValeur());
}
```

Question 5 (20 points) Soit le programme suivant qui compile et s'exécute correctement.

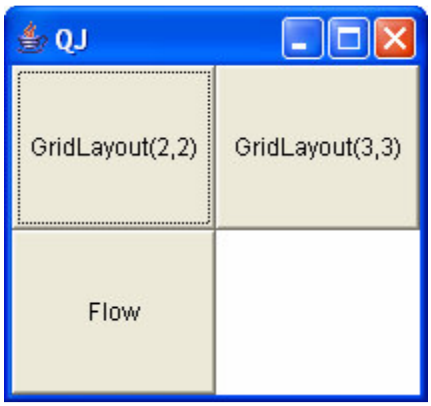
```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Question extends Applet implements ActionListener{
    Button btnGrid = new Button("GridLayout(2,2)");
    Button btnGrid2 = new Button("GridLayout(3,3)");
    Button btnFlow = new Button("Flow");
    public  void init(){
        setLayout(new BorderLayout());
        unemethode();
    }
    public void unemethode(){
        btnGrid.addActionListener(this);
        btnGrid2.addActionListener(this);
        btnFlow.addActionListener(this);

        add(btnGrid,"North");
        add(btnGrid2,"East");
        add(btnFlow,"South");
    }
    public void actionPerformed(ActionEvent evt) {
        String arg =evt.getActionCommand();
        if(arg.equals("GridLayout(2,2)")){
            setLayout(new GridLayout(2,2));
            validate(); // permet de forcer l'apparition des changements
        }
        if(arg.equals("GridLayout(3,3)")){
            System.out.println(arg);
            setLayout(new GridLayout(3,3));
            validate();
        }
        if(arg.equals("Flow")){
            System.out.println(arg);
            setLayout(new FlowLayout());
            validate();
        }
    }
}
```

5.1 **Tout justifiant votre réponse**, dessiner l'applet de départ



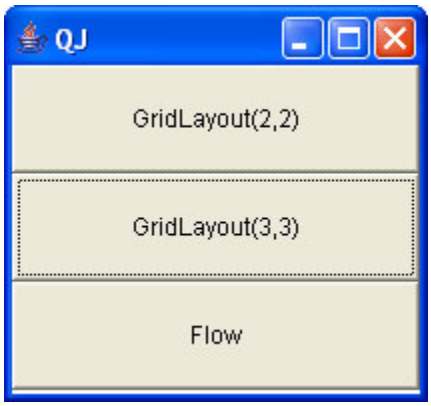
Le layout par défaut est le BorderLayout. Par défaut les boutons nord et sud vont occuper toute la surface. Ainsi donc btnGrid va occuper toute la surface Nord, le bouton btnFlow va occuper toute la surface Sud. Finalement le bouton btnGrid2 va être à l'Est entre les deux zones Nord et Sud.

5.2 <u>Tout justifiant votre réponse</u>, dessiner l'applet quand on clique sur le bouton btnGrid



Quand on clique sur btnGrid, on provoque une action qui consiste à modifier le gestionnaire de Layouts. Le gestionnaire de Layouts est maintenant un GridLayout de dimension 2x2. Donc nous avons affaire à une grille de 2 lignes et 2 colonnes. Chaque bouton va s'étaler dans la grille et les éléments seront disposés de gauche droite, ligne par ligne. Ainsi donc le bouton btnGrid va être disposé en premier dans la grille à la position (1,1). Il est suivi du bouton btnGrid2 qui sera disposé à la position (1,2). Comme il n'y a plus de place, le manager passe à la ligne suivante pour disposer le bouton btnFlow à la position (2,1). La clase (2.2) reste quand à elle vide.

5.3 Tout justifiant votre réponse, dessiner l'applet quand on clique sur le bouton btnGrid2



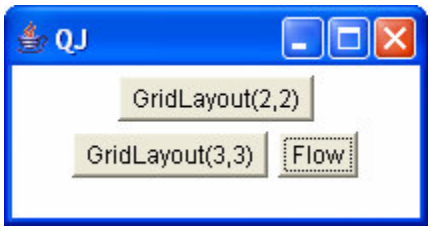
Même principe que dans la question 5.2 sauf qu'une autre condition s'ajoute dans la réalisation des GridLayout. Il faut que toutes les lignes soient occupées pour refléter correctement la surface occupée. On pourrait s'attendre à disposer les éléments aux positions (1,1), (1,2) et (1,3). Mais à cause de la contrainte et vu que nous avons que 3 éléments, ils vont se retrouver de facto aux positions (1,1), (2,1) et (3,1).

(je n'ai pas trop pénalisé pour ceux/celles qui ont raté ce détail associé à ce genre de Layout).

5.4 Tout justifiant votre réponse, dessiner l'applet quand on clique sur le bouton btnFlow

Si la fenêtre est assez large 

sinon



Ou bien 

Les éléments seront disposés de gauche à droite, vu qu'en cliquant sur le bouton Flow, c'est le FlowLayout qui sera utilisé. Dans un tel Layout les éléments seront disposés de gauche à droite en fonction de la largeur de l'interface graphique.

Question 6 (25 points) Indiquer si les println provoquent une erreur à la compilation. S'il y a erreur, vous devez la commenter. Dans le cas contraire, tout en expliquant votre réponse, vous devez donner l'affichage en sortie. À noter, que si un appel à println provoque une erreur de compile, cette dernière ne va pas planter le reste du programme. Après avoir expliqué l'origine de l'erreur, vous pouvez la considérer comme en commentaire pour le reste du programme. Ceci va vous permettre de répondre aux println restants.

```
interface I1{
    int m1();
}
interface I2 extends I1 {
    int m2();
}
class B implements I1 {
    int i=7;
    public int m1(){
        return i;
    }
}
class C extends B implements I2 {
    int i = -7;
    public int m2() {
        return i;
    }
    public int m1() {
        return i;
    }
}
public class Class1 {
    public static void main(String [] a) {

[illegible]
        System.out.println(x.i); // -1-
```

[X] correcte [ ] incorrecte

Pourquoi (affichage en sortie)

7

i de C ou de B ? Comme les variables suivent une ligature statique. Donc ça sera i de B et non pas de C.

```
        System.out.println(x.m1()); // -2-
```

[X] correcte [ ] incorrecte

Pourquoi (affichage en sortie)

-7

La méthode m1 étant présente dans B et C donc pas de problème à la compilation. Les méthodes suivent une ligature dynamique donc à l'exécution on va choisir m1 de C. Cette méthode fait appel à la variable i de C qui vaut -7, d'où l'affichage en sortie de -7.

```
        System.out.println(x.m2()); // -3-
```

[ ] correcte [X] incorrecte

Pourquoi (affichage en sortie)

La méthode m2 est présente que dans la classe C mais pas dans la classe B. Condition obligatoire pour permettre l'appel à la méthode m2. Erreur de compilation.

```
I1 y = (I1) x;
System.out.println(y.i); // -4-
```

[ ] correcte [X] incorrecte

Pourquoi (affichage en sortie)

L'interface I1 ne contient pas le membre (variable) i. Condition obligatoire pour permettre l'appel à la variable i. Erreur de compilation.

```
System.out.println(y.m1()); // -5-
```

[X] correcte [ ] incorrecte

Pourquoi (affichage en sortie)

-7, Ligature dynamique. La méthode m1 est déclarée dans I1 et dans C. Comme X fait référence à C, on fait appel à la méthode m1 de C (car différent dans le cas des variables cas -1-)

```
System.out.println(y.m2()); // -6-
```

[ ] correcte [X] incorrecte

Pourquoi (affichage en sortie)

L'interface I1 ne contient pas la méthode m2. Condition obligatoire pour permettre l'appel à la méthode m2. Erreur de compilation.

```
C z = (C) x;
System.out.println(z.i); // -7-
```

[X] correcte [ ] incorrecte

Pourquoi (affichage en sortie)

-7

L'objet z est du type C. Avec un cast on change le typage de x (qui était B) vers C. Mais comme x est une référence à une instance de C, donc pas de problème, la variable i est accessible. Comme les variables suivent une ligature statique, c'est donc la variable i de C, d'où -7 en sortie.

```
System.out.println(z.m1()); // -8-
```

[X] correcte [ ] incorrecte

Pourquoi (affichage en sortie)

-7

La méthode m1 est accessible suite au cast. Nous avons affaire ici à une ligature dynamique, donc appel à m1 de C d'où l'affichage de -7 en sortie.

```
        System.out.println(z.m2()); // -9-
```

[X] correcte [ ] incorrecte

Pourquoi (affichage en sortie)

-7

La méthode m2 est accessible suite au cast. Nous avons affaire ici à une ligature dynamique, donc appel à m2 de C d'où l'affichage de -7 en sortie.

```
        I2 v = (I2) x;

        System.out.println(v.i); // -10-
```

[ ] correcte [X] incorrecte

Pourquoi (affichage en sortie)

L'interface I2 ne contient pas le membre (variable) i. Condition obligatoire pour permettre l'appel à la variable i. Erreur de compilation.

```
        System.out.println(v.m1()); // -11-
```

[X] correcte [ ] incorrecte

Pourquoi (affichage en sortie)

-7, Ligature dynamique. La méthode m1 est définie dans I2 et dans C. Comme X fait référence à C, on fait appel à la méthode m2 de C d'où l'affichage en sortie -7.

```
        System.out.println(v.m2()); // -12-
```

[X] correcte [ ] incorrecte

Pourquoi (affichage en sortie)

-7, Ligature dynamique. La méthode m2 est définie dans I2 et dans C. Comme X fait référence à C, on fait appel à la méthode m2 de C d'où l'affichage en sortie -7.

```
    }
}
```