IFT1020 - Session Été, Intra

Mohamed Lokbani

IFT1020 - INTRA

Inscrivez tout de suite votre nom et code permanent.

Nom: _____| Prénom(s): _____|

Signature: _____| Code perm: _____|

Date : 17 juin 2002

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

Directives:

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

- Répondre directement sur le questionnaire.
- Les réponses **doivent être clairement présentées.**

1._____/15 (1.1 ; 1.2 ; 1.3 ; 1.4 ; 1.5)

2._____/15 (2.1 ; 2.2)

3._____/15 (3.1 ; 3.2 ; 3.3)

4._____/30 ($4.1_{1, 2, 3, 4}$; $4.2_{1, 2, 3, 4}$; $4.3_{1, 2, 3, 4}$)

5._____/25 (5.1)

Total:_____/100

Question 1 (15 points)

Indiquez quelles affirmations sont vraies et lesquelles sont fausses et dites pourquoi. Pas de justification équivaut à une mauvaise réponse.

1.1 Pour être compilable, toute classe doit disposer explicitement d'au moins un constructeur.

VRAI ☐
FAUX ☐

Pourquoi? (Courte explication)

1.2 La première instruction dans un constructeur d'une classe, autre que la classe **`Object`**, est toujours un appel explicite ou implicite à un autre constructeur.

VRAI ☐
FAUX ☐

Pourquoi? (Courte explication)

1.3 Si aucun constructeur n'est écrit dans la classe **`MaClasse`** alors il est impossible d'étendre **`MaClasse`**.

VRAI ☐
FAUX ☐

Pourquoi? (Courte explication)

1.4 Si l'on écrit un constructeur avec argument dans une classe alors il faut nécessairement écrire également un constructeur sans argument dans cette classe.

VRAI ☐
FAUX ☐

Pourquoi? (Courte explication)

1.5 L'instruction `this()` est un appel au constructeur sans argument de la classe courante.

VRAI ☐
FAUX ☐

Pourquoi? (Courte explication)

Question 2 (15 points)

Soit le programme suivant, qui compile et s'exécute correctement :

```
class Parent  {
   int x  ;
   Parent(int k) {x=k;}
   int ajoute(int a) {      return x+a;  }
   public void moi() {      System.out.println(" x = "+ x); }
}
class Enfant1 extends Parent {
    int y ;
    Enfant1 (int k, int l) {
        super(k); y=l;
    }
    int ajoute(int a) { return x+2*a;}
}
class Enfant2 extends Enfant1 {
    int z ;
    Enfant2 (int k, int l, int m) {
        super(k, l); z= m;
    }
    int ajoute(int a) {     return x+3*a;}
    public void moi() {
        super.moi();
        System.out.println(" z = "+ z);
    }
}
class Essai{
    public static void main (String args[]) {
       int a =2;
       Parent p = new Parent(3);
       p.moi();
       System.out.println(" ajoute("+ a +") = "+ p.ajoute(a) );
       Enfant1 e1 = new Enfant1(3, 4);
       e1.moi();
       System.out.println(" ajoute("+ a +") = "+ e1.ajoute(a) );
       Enfant2 e2 = new Enfant2(3, 4, 5);
       e2.moi();
       System.out.println(" ajoute("+ a +") = "+ e2.ajoute(a) );
    }
}
```

2.1 Quels sont les attributs dont disposent les classes Enfant1 et Enfant2 ? (Courte explication)

2.2 Ecrivez le résultat de l'exécution de la classe Essai. (Courte explication)

Question 3 (15 points) La classe suivante définit une fenêtre qui propose une interface graphique permettant de tirer des nombres au hasard, dans l'intervalle [0...1000].

Commentez les lignes de code soulignées.

```
import java.awt.*;

public class Generateur extends Frame {
  static final int VMAX = 1000;
  Button boutonAutre;
  Button boutonQuitter;
  TextField nombreTire;

  public Generateur (){
   super("Generateur aleatoire");
   nombreTire = new TextField();
   nombreTire.setEditable(false);
   nombreTire.setText((new Long(tirerNombreHasard())).toString());
   boutonAutre = new Button("Autre nombre");
   boutonQuitter = new Button("Quitter");
   add(nombreTire,"North");
   add(boutonAutre,"Center");
   add(boutonQuitter,"South");
}
  public long tirerNombreHasard() {
   return Math.round(Math.random()*VMAX);
  }
  public static void main(String[] args) {
   Generateur ga = new Generateur();
   ga.setVisible(true);
  }
}
```

3.1 `super("Generateur aleatoire");`

3.2 `nombreTire.setText((new Long(tirerNombreHasard())).toString());`

3.3 `add(nombreTire,"North");`

Question 4 (30 points) Soit le fragment de code suivant :

```
class Personne {
    public void parler() { System.out.println("hum!"); }
}
class Adulte extends Personne{
    public void parler() { System.out.println("Bonjour tout le monde!"); }
}
class AdulteDistingue extends Adulte {
    public void parler() { System.out.println("Mes chers amis, bonjour!"); }
}
class Jeune extends Personne {
}
class Ado extends Jeune {
    public void parler() { System.out.println("salut les mecs!"); }
}
class Enfant extends Jeune {
}
class Bebe extends Enfant {
    public void parler() { System.out.println("J'chui pas un bebe!"); }
}
class BebeCadum extends Bebe {
    public void parler() { System.out.println("Agheu, agheu!"); }
}
```

4.1 Ecrivez le résultat de l'exécution des fragments de code suivants :

4.1.1

```
Personne P1 = new Personne();
P1.parler();
```

Réponse

4.1.2

```
Adulte P2 = new AdulteDistingue();
P2.parler();
```

Réponse

4.1.3

```
Jeune P3 = new Ado();
P3.parler();
```

Réponse

4.1.4

```
BebeCadum P4 = new BebeCadum();
P4.parler();
```

Réponse

4.2 Soit le fragment de code suivant :

```
Personne P1 = new Personne();
Personne P2 = new Bebe();
Jeune P3 = new Ado();
BebeCadum P4 = new BebeCadum();
```

On effectue les affectations suivantes qui ne sont pas exécutées en séquence! Pour chaque affectation, dites si elle fonctionne sans erreur, si elle provoque une erreur à la compilation ou si elle provoque une erreur à l'exécution. Si il y a une erreur expliquez (rapidement) pourquoi.

4.2.1 `P1=P2;`

Sans Erreur ☐
Erreur Compilation ☐
Erreur Exécution ☐

Pourquoi? (Courte explication)

4.2.2 `P3=P1;`

Sans Erreur ☐
Erreur Compilation ☐
Erreur Exécution ☐

Pourquoi? (Courte explication)

4.2.3 `P4=P2;`

Sans Erreur ☐
Erreur Compilation ☐
Erreur Exécution ☐

Pourquoi? (Courte explication)

4.2.4 `P4 = (BebeCadum) P2;`

Sans Erreur ☐
Erreur Compilation ☐
Erreur Exécution ☐

Pourquoi? (Courte explication)

Q 4.3 Si on crée un tableau du type `Jeune` de la manière suivante:

```
Jeune[] e = new Jeune[4];
```

Et pour le fragment de code suivant:

```
Personne P1 = new Bebe();
Jeune P2 = new Ado();
Enfant P3 = new BebeCadum();
BebeCadum P4 = new BebeCadum();
```

Que faut-il faire pour que les affectations suivantes soient toutes valides (à la compilation et à l'exécution).

4.3.1 `e[0] = P1;`

Valide ☐
Non valide ☐

Si valide pourquoi (courte explication)? Sinon que faut-il faire?

4.3.2 `e[1] = P2;`

Valide ☐
Non valide ☐

Si valide pourquoi (courte explication)? Sinon que faut-il faire?

4.3.3 `e[2] = P3;`

Valide ☐
Non valide ☐

Si valide pourquoi (courte explication)? Sinon que faut-il faire?

4.3.4 `e[3] = P4;`

Valide ☐
Non valide ☐

Si valide pourquoi (courte explication)? Sinon que faut-il faire?

Question 5 (25 points) On définit une interface `Multipliable` comme suit:

```
interface Multipliable{
        public Object produit(Object a); //produit de l'objet courant par a
}
```

Un nombre complexe est caractérisé par sa partie réelle et sa partie imaginaire. Soit les deux nombres complexes z_1 et z_2 où $z_1=x_1+iy_1$ et $z_2=x_2+iy_2$. L'opération **`produit`** de deux nombres complexes z_1z_2 est définie comme suit : $z_1z_2=(x_1x_2-y_1y_2)+i(x_1y_2+y_1x_2)$.

Écrivez la classe `Complexe` qui décrit des nombres complexes avec,

- Les attributs,
- Au moins un constructeur permettant l'appel suivant :

```
Complexe z1=new Complexe(x1,y1);
```

Une méthode **`toString`** permettant l'appel suivant (libre à vous de définir le message affiché en sortie) :

```
System.out.println(z1);
```

- La méthode **`produit`** telle que décrite précédemment.
- Une surcharge de la méthode **`produit`** de façon que l'on puisse faire l'appel suivant :

```
Complexe z3=Complexe.produit(z1, z2); // z1 et z2 deux nombres complexes.
```