Contrôle Programmation Java

Correction

1ère Année Informatique

Question 1) Un fichier `salut.java` contient le texte suivant :

```
class Salut {
    static public void main(String args[]) {
        System.out.println("Salut");
    }
};
```

Lesquelles des lignes commandes suivantes sont incorrectes et pourquoi ?

1. `javac salut.java`
2. `java salut.java`
3. `javac Salut.class`
4. `java Salut.class`
5. `java Salut`

2. on n'exécute pas un source.

3. inversement, on ne compile pas un fichier .class

4. l'extension .class est inutile

Question 2) On compile le fichier `prog.java` qui contient le programme suivant :

```
class MaClasse {
    public void f() {
        System.out.println("Bonjour");
    }
     static public int g() {
         System.out.println("Salut");
     }
 };
```

a) Quels sont les fichiers générés ?

Un seul fichier : `Maclasse.class`

b) Dans un programme *main*, on écrit :

```
1. double x; int i ;
2. MaClasse m;
3. MaClasse.f();
4. m.f();
5. x = 25;
6. i = x + 1;
7. m.g();
```

Certaines de ces lignes contiennent des erreurs de compilation lesquelles ?

3. `f` est une méthode d'instance
4. `m` n'est pas encore initialisée
6. types incompatibles, perte de précision
7. `g` est une méthode de classe (`static`), ne s'applique pas à une instance

Question 3) Soit la classe `Point` du TP. On voudrait rajouter une méthode nommée `verticaux` qui teste si deux points sont alignés verticalement.

1. Proposer deux façons de définir l'entête de cette méthode.

```
public boolean verticaux (Point p) {…}
static public boolean verticaux (Point p, Point q)  {…}
```

2. Donner un exemple d'utilisation de chaque cas.

```
// p, q déjà déclarées et initialisées.
boolean b ;
b = p.verticaux (q) ;
b = Point.verticaux (p, q) ;
```

Question 4) Pourquoi dans une classe Java certains attributs sont `private`, `public` ou `protected` ?

En bref, `public` indique une caractéristique publique, utilisable partout. `private` indique une caractéristique utilisable uniquement dans sa classe. `protected` pour dire utilisable aussi dans les sous classes.

Qestion 5) Quel est l'intérêt de déclarer certaines méthodes `static` et d'autres pas ?

`static` sert à introduire des fonctions d'utilité générale ne dépendant pas des instances. Dites aussi méthodes de classe. Par exemple `sqrt()` de la classe `Math`, `valueOf()` de la classe `String` ou `parseInt()` de la classe `Integer`.

Question 6) Que fait le programme suivant :

```
class Blabla {
    public static void main(String[] args) {
        String a = "malabata";
        StringBuffer b = new StringBuffer();
        int i = 0;
       while (i+1 < a.length()){
           b.append (a.charAt(i+1));
           i += 2;
        }
        System.out.println(b);
      }
}
```

Ce programme imprime « `aaaa` ». Il parcourt la chaîne « `malabata` » par pas de deux et imprime le prochain caractère à chaque fois.

Pourquoi la variable `b` n'est-elle pas de même type `String` que la variable `a` ? Comment appelle-t-on la différence entre les deux ?

La variable `b` est de type `StringBuffer` parce qu'elle est modifiable dans le programme (méthode `append(..)`). On dit d'un objet qu'il est *mutable* quand on peut le modifier.
NB. Quand on concatène sur un objet `String`, c'est un nouveau string qui est créé.

Question 7) On a une chaine `s` égale à "123". Comment récupérer cette valeur 123 dans un entier `i` ?

```
i = Integer.parseInt("123" ) ;
```

Question 8) Quel est le résultat de la séquence d'instructions :

```
double x = 5;
double y = 3.14;
System.out.println(x + y);
System.out.println(x+" "+y);
```

`8.14` et `"5 3.14"`. En effet, `x+y` est évaluée en double (ensuite en string pour l'impression), alors que `x+" "+y` est évalué en `String`, `x` et `y` étant "String-ifiées" auparavant.

Question 9) La classe `Rectangle` du TP, est définie par deux points diagonalement opposés `hg` et `bd` de classe `Point`.

1. Donner la définition d'au moins deux constructeurs pour cette classe `Rectangle`, et donner aussi un exemple d'utilisation de chacun.

Corrigé quelque peu détaillé

```
public Rectangle(){
// rectangle par défaut. Choisir son initialisation
    hg = new Point();   //ou hg.setX(0); hg.setY(1);
    bd = new Point();   //ou bd.setX(2); bd.setY(0);
}

public Rectangle(Point h, Point b){
// initialisation des coins à partir de paramètres
// données
    hg = new Point(h);
    bd = new Point(b);
   //ou hg = new Point(h.getX(), h.getY());
   //ou bd = new Point(b.getX(), b.getY());
}

public Rectangle(Rectangle r){
// initialisation des coins de this à partir de ceux
// du rectangle r
hg = new Point(r.getHg());
bd = new Point(r.getBd());
}
```

NB. Seule deux constructeurs `Rectangle` sont demandés.
Sachant que : la classe `Point` est munie d'un constructeur-copie `Point(Point)` utilisé dans `new Point(h)` et `new Point(r.getHg())` par exemple.

2. Lequel serait un constructeur par copie dans ce cas ?

Le troisième `public Rectangle(Rectangle r).`

3. Est-ce une copie normale, superficielle ou en profondeur ? Justifier.

C'est une copie en <u>profondeur</u>, car on instancie de nouveaux points par `new`. Cela suppose que le constructeur par copie de Point l'est aussi (en profondeur). Ce qui est le cas ici, vue que la classe `Point` ne contient que des types primitifs.

Question 10) Soit les classes

```
class A  {...    public void f(){..}     ...}
class B extends A {...   public void f(){..}     ...}
class C extends A {...   public void f(){..}      ...}
```

1. Expliquer le mot `extends` des classes `B` et `C`

Le mot `extends` indique un héritage d'une autre autre classe (`A` ici).

2. Quel en est l'intérêt sur cet exemple, et en général dans le langage Java

L'héritage permet de réutiliser des caractéristiques déjà définies, celles de la classe `A` ici (pas seulement `f`).

Avec la réutilisation, l'héritage en Java permet un développement des classes de façon progressif et structuré. On peut rajouter de nouvelles caractéristiques une classe ou redéfinir celles déjà existantes. Cela permet aussi de raffiner des objets en les spécialisant. Ainsi, on peut voir des objets comme étant de même classe générale à un niveau, ou de classes spécifiques à un niveau plus fin. (cf. question suivante)

3. As-t-on le droit d'avoir la même fonction `f` dans les 3 classes `A`, `B` et `C` ? Si oui, pourquoi faire?

<u>Oui</u>, les classes B et C peuvent redéfinir f pour en donner <u>leur propre version</u>

4. Quel avantage cela procure t-il ? Considérer le cas :

```
A a[] = {new B(), new C(), new A()} ;
for (int i=0; ...)
       a[i].f() ;
```

L'avantage c'est le polymorphisme. Ici, des instances de B et de C, sont considérées comme des objets A (première ligne). Ce qui permet d'écrire `a[i].f()`, où la méthode appelée sera celle correspondant à l'instance en cours, A, B ou C etc.