2i002 - Cours 11 : Annales

Vincent Guigue - vincent.guigue@lip6.fr

1 2014
2 2015
3 2013
4 2012
5 2011
6 2013R

```java
System.out.println("coucou");
```

A quoi correspondent : `System`, `out` et `println` ? Les choix possibles de réponses sont : au nom d'une classe, au nom d'une variable d'instance, à une variable statique, à un appel à une méthode d'instance, à un appel à une méthode statique. Justifier brièvement.

Soit les classes suivantes `A` et `B` et le `main` associé :

```java
public class A{
  private int i;
  public A(int i){ this.i=i;}
  public A clone(){
    return new A(i);}
}
```

```java
public class B{
  private A a;
  public B(A a){ this.a=a; }
  public B clone(){
    return new B(a);}
}
```

```java
public static void main(String[] args){
  A a1 = new A(2);
  A a2 = a1.clone();
  A a3 = a1;
  if(a1 == a2) System.out.println("a1 == a2");
  if(a2 == a3) System.out.println("a2 == a3");
  if(a3 == a1) System.out.println("a3 == a1");
}
```

Combien y a-t-il d'instances de la classe `A` créées lors de l'exécution du `main` ?

Qu'est-ce qui s'affiche lors de l'exécution du programme ?

Nous ajoutons les instructions suivantes dans le `main` précédent.

```
1   if(a1.equals(a2)) System.out.println("a1.equals(a2)");
2   if(a2.equals(a3)) System.out.println("a2.equals(a3)");
3   if(a3.equals(a1)) System.out.println("a3.equals(a1)");
```

Le programme compile-t-il toujours ? Dans l'affirmative, qu'est-ce qui s'affiche lors de l'exécution ? (Justifier brièvement)
Donner le code de la méthode standard `boolean equals(Object o)` à ajouter dans la classe `A` pour tester l'égalité structurelle entre 2 instances

Etudions le code du programme suivant :

```
1 public static void main(String[] args){
2   A a1 = new A(2);
3   B b1 = new B(a1);
4   B b2 = b1.clone();
5 }
```

- Donner la représentation de la mémoire à la fin de l'exécution du code. Combien d'instances de `A` et `B` sont créées ?
- La question précédente met en évidence le mauvais fonctionnement de la méthode de clonage de la classe `B` : proposer une amélioration.
- Donner le code de la méthode standard `boolean equals(Object o)` à ajouter dans la classe `B` pour tester l'égalité structurelle entre 2 instances.

Encore une étude de cas :

```
1 public static void main(String[] args){
2   A[] tab = new A[5];
3   for(int i=0; i<=5; i++) System.out.println(tab[i]);
4 }
```

Le programme compile-t-il ? Dans l'affirmative, que se passe-t-il lors de l'exécution ?

```java
public class Mere {
  public Mere(){}
  public void maMethode(double d){
    System.out.println("arg : double ");
  }
}
public class Fille extends Mere{
  public Fille(){}
  public void maMethode(int i){
    System.out.println("arg : int = "
  }
}
```

```java
public static void main(Strin
  Mere m = new Mere();
  Fille f = new Fille();
  Mere mf = new Fille();

  m.maMethode(2);
  f.maMethode(2);
  mf.maMethode(2);

  m.maMethode(2.5);
  f.maMethode(2.5);
  mf.maMethode(2.5);
}
```

Quels sont les affichages observés dans la console suite à l'exécution de ce programme ?
NB : 5 réponses sont assez triviales mais 1 ligne est plus délicate. Expliquer en détail ce qui se passe avec cette ligne.

Nous ajoutons une implémentation de la méthode `public double maMethode(double d)` dans la classe `Mere` : le code compile-t-il toujours ?

```java
public class Pile {
  private static final
      int TAILLE = 10;
  private int[] contenu;
  private int niveau;

  public Pile(){
    contenu = new int[TAILLE];
    niveau  = 0;
  }
  public void push(int i){
    contenu[niveau] = i;
    niveau++;
  }
  public int pop(){
    int r = contenu[niveau];
    niveau --;    return r;
  }
}
```

```java
class PileException
       extends Exception{
       public PileException
              (String message) {
        super(message);
    }
}
```

Proposer des nouvelles versions pour les méthodes `push` et `pop` qui lèvent une `PileException` lorsque nous tentons :

- d'appeler `push` alors que la pile contient déjà 10 éléments
- d'appeler `pop` alors que la pile est vide (0 éléments dans `contenu`)

NB : les exceptions sont déléguées, elles ne sont pas traitées au niveau local.

1 2014
2 2015
3 2013
4 2012
5 2011
6 2013R

Retro-engineering

```
1  public static void main(String args[]){
2          Expression v1=new Valeur(4.);
3          Expression v2=new Valeur(1.);
4          Expression v3=new Valeur(7.);
5          Expression v4=new Valeur(5.);
6          Expression v5=new Valeur(3.);
7          Expression v6=v5;
8          Operation p1=new Plus(v1,v2);
9          Operation m2=new Moins(v3,v4);
10         Operation mult=new Multiplie(p1,v5);
11         Operation p2=new Plus(v6,mult);
12         Operation d=new Divise(p2,m2);
13         System.out.println(d+"="+d.getVal());
14 }
```

Donner la hierarchie des classes et les signatures des méthodes

1 2014
2 2015
3 2013
4 2012
5 2011
6 2013R

Question 1 (2 points) : Soit l'instruction : `Integer.parseInt(chaine);` qui transforme une chaîne de caractères en un entier. Dans cette instruction, (a) à quoi correspond `Integer` ? (b) à quoi correspond `parseInt(chaine)` ? Les choix possibles de réponses sont : au nom d'une classe, au nom d'une variable d'instance, à une variable statique, à un appel à une méthode d'instance, à un appel à une méthode statique. Expliquez précisément chacune de vos réponses aux questions (a) et (b).

Question 2 (2 points) : Soit la classe : `public class A { }` et l'instruction `A obj=new A();` L'objet référencé par `obj` : (a) possède-t-il une méthode `toString()` ? (b) a-t-il été construit avec un constructeur ? Si oui lequel. Expliquez précisément vos réponses aux questions (a) et (b).

Question 3 (3 points) : On considère les coordonnées GPS (latitude et longitude) des villes suivantes : Paris 48.86N 2.35E, Marseille 43.30N 5.37E, Lyon 45.76N 4.84E. Pour simplifier, on supposera que toutes les coordonnées considérées sont dans l'hémisphère Nord et à l'Est. Par exemple, cela signifie que pour la ville de Paris la latitude est 48.86 et la longitude 2.35.

Q3.1 : Dans une méthode `main`, donner la déclaration et l'initialisation d'un tableau `tabCoord` de `double` à deux dimensions permettant de stocker la valeur de latitude et de longitude de ces 3 villes.

Q3.2 : Soit la classe suivante :

```
1 public class CoordGPS {
2         private double latitude, longitude;
3         public CoordGPS(double la,double lo) { latitude=la; longitude=lo; }
4         public String toString() { return latitude+"N␣"+longitude+"E";  }  }
```

Dans une méthode `main`, donner la déclaration et l'initialisation d'un tableau `tabObj` de `CoordGPS` permettant de stocker la valeur de latitude et de longitude de ces 3 villes.

Q3.3 : Quel est le résultat produit par chacune des instructions suivantes :
(a) `System.out.println(tabCoord[10][0])` ? (b) `System.out.println(tabObj[1])` ?

Question 4 (3 points) : Soient les classes suivantes :

```
1 public class Recipient {
2         private static int nbRecipients=0;
3         private int numero;
4         private double volume;
5         public Recipient(double vol) {
6                 nbRecipients++;
7                 numero=nbRecipients;
8                 volume=vol;
9         }
10        public static int getNumero() { return numero; }
11 }
12 public class Verre extends Recipient {
13        private double hauteur;
14        public Verre(double volume) { hauteur=volume; }
15        public Verre(double hauteur) {   this.hauteur=hauteur; }
16 }
```

Dans ces classes, il y a 3 catégories très différentes d'erreurs détectées à la compilation (on ne prendra pas en compte les erreurs de sens qui ne sont pas détectées à la compilation). Expliquez précisément ces trois catégories d'erreurs et proposez une solution qui compile pour chaque erreur.

```
1  public class Animal {
2          public String toString() { return "Animal"; }    }
3  public class Oiseau extends Animal {
4          public String toString() { return "Oiseau"; }    }
5  public class Insecte extends Animal { }
6  public class Cigogne extends Oiseau { }
7  public class Corbeau extends Oiseau {
8          public String toString() { return "Corbeau"; }    }
9  public class Libellule extends Insecte { }
10 public class Sauterelle extends Insecte {
11         public String toString() { return "Sauterelle"; }    }
12 public class TestAnimal {
13         public static void main(String [] args) {
14                 Cigogne cig1=new Cigogne();System.out.println(cig1);
15                 Corbeau corb1=new Corbeau();System.out.println(corb1);
16                 Libellule l1=new Libellule();System.out.println(l1);
17                 Sauterelle s1=new Sauterelle();System.out.println(s1);
18                 Animal a1=new Cigogne();System.out.println(a1);
19                 Animal a2=new Corbeau();System.out.println(a2);
20                 Animal a3=new Libellule();System.out.println(a3);
21                 Animal a4=new Sauterelle();System.out.println(a4);
22
23                 Insecte i1=new Animal();
24                 Insecte i2=new Oiseau();
25                 Insecte i3=new Libellule();
26
27                 Insecte i4=(Insecte)a4;
28                 Insecte i5=(Insecte)a2;       }    }
```

Q5.1 : Qu'affiche chacune des instructions des lignes 14 à 17 ? Expliquez. Quelle différence avec l'affichage des lignes 18 à 21 ? Expliquez.

Q5.2 : Parmi les lignes 23 à 25, lesquelles sont correctes, lesquelles sont fausses. Expliquez.

Q5.3 : Les lignes 27 et 28 provoquent-elles une erreur à la compilation ? à l'exécution ? Expliquez.

Q5.4 : Dans la classe `Animal`, on rajoute une méthode abstraite. Expliquez les conséquences pour la classe `Animal` et pour ses classes filles. Cela ajoute-t-il des erreurs dans la méthode `main` ?

```java
public class Ampoule {
        private int puissance;
        public Ampoule() { puissance=(int)(Math.random()*91)+10; }
        public String toString() { return "Ampoule␣"+puissance+"W";}
}
public class Lampe {
        private Ampoule amp1, amp2;
        public Lampe() { amp1=new Ampoule(); amp2=new Ampoule(); }
        public String toString() { return "Lampe␣avec␣"+amp1+"␣et␣"+amp2; }
}
public class TestLampe {
        public static void main(String [] args) {
                Lampe lp1=new Lampe();
        }
}
```

On souhaite créer une nouvelle lampe qui soit la copie conforme de la lampe `lp1` dans la méthode `main`. Donnez précisément toutes les méthodes et instructions qu'il faut ajouter, y compris les instructions nécessaires dans la méthode `main`.

Question 7 (3 points) : Chaîne de caractères et flux

`String` est une classe immutable, c'est-à-dire qu'une variable de type `String` ne peut pas être modifiée. Lorsque l'on pense modifier un objet `String`, en vérité, on crée un nouvel objet `String`, et on perd l'ancien. Par exemple, soit `String s="Bonjour" ;` alors `s=s+" Paul" ;` est équivalent à `s=new String(s+" Paul") ;`. On veut écrire un programme en java qui crée une unique chaîne de caractères à partir des arguments qui sont passés en ligne de commande. Par exemple, pour les lignes de commande suivantes :

```
java TestChaine Bonjour à tous le monde.
java TestChaine Comment ça va ?
```

les chaînes à créer sont respectivement "Bonjour à tous le monde." et "Comment ça va ?". On rappelle que les arguments passés en ligne de commande sont contenus dans le tableau de chaînes de caractères `args` de la méthode `main`, chaque case du tableau est un argument. Par exemple, pour la première ligne de commande, `args[0]="Bonjour"`, `args[1]="à"`, `args[2]="tous"`, `args[3]="le"` et `args[4]="monde."`. Soit la classe `Testchaine` contenant la méthode `main` suivante :

```
1 public class TestChaine {
2     public static void main(String [] args) {
3         System.out.println("La␣chaine␣saisie␣est␣=␣"+concatener(args));
4     }
5 }
```

Q7.1 : Dans la classe `Testchaine`, écrire la méthode statique `String concatener1(String [] args)` qui retourne une chaîne de caractères qui est la concaténation de toutes les chaînes contenues dans `args` séparées par un espace.

Q7.2 : La classe `StringWriter` permet de gérer un flux de caractères. Elle contient les méthodes :

`StringWriter()`	Constructeur qui crée un nouvel objet `StringWriter`
`void write(String s)`	Écrit la chaîne de caractères `s` à la suite dans le flux
`String toString()`	Retourne les caractères du flux sous forme d'un objet `String`

Écrire la méthode statique `String concatener2(String [] args)` qui réalise la même chose qu'à la question précédente, mais en utilisant la classe `StringWriter`.

Q7.3 : Quelle est la méthode qui crée le moins d'objets ? Expliquez.

1 2014
2 2015
3 2013
4 2012
5 2011
6 2013R

Attention :

- Pour chaque question ci-dessous, cocher la ou les cases correspondantes **aux affirmations qui sont vraies.** Les cases cochées doivent être clairement identifiables.
- Pour chaque question, **plusieurs** réponses peuvent être correctes, il y a toujours **au moins 1 réponse correcte**.
- Chaque erreur (mauvaise réponse cochée ou oubli de cocher une bonne réponse) entraîne la perte de 0.5 points.

Question 1 : Dans l'instruction : `System.out.println("Bonjour");`

a) `System` correspond à :

☐ une classe ☐ un appel de méthode ☐ une variable d'instance ☐ une variable statique

b) `out` correspond à :

☐ une classe ☐ un appel de méthode ☐ une variable d'instance ☐ une variable statique

c) `println("Bonjour")` correspond à :

☐ une classe ☐ un appel de méthode ☐ une variable d'instance ☐ une variable statique

Question 2 : Une constante doit être :

☐ `const` ☐ `abstract` ☐ `final` ☐ `static`

☐ ne peut être modifiée après initialisation ☐ peut être modifiée après initialisation

Question 3 : La méthode `main` doit être :

☐ `public` ☐ `private` ☐ `abstract` ☐ `final` ☐ `static`

☐ retourner `void` ☐ retourner `int` ☐ ne prendre aucun paramètre

Question 4 : Un constructeur doit être :

- ☐ `abstract` ☐ `final` ☐ `static` ☐ porter le nom de la classe
- ☐ retourner `void` ☐ ne pas avoir de type de retour ☐ ne prendre aucun paramètre

Question 5 : Dans une classe `Point`, je souhaite déclarer une variable `cptPoints` dont le but est de compter le nombre d'instances de la classe `Point` créées, je déclare cette variable :

- ☐ `abstract` ☐ `final` ☐ `static`
- ☐ j'initialise la variable lors de sa déclaration ☐ j'initialise la variable dans le constructeur

Question 6 : Soit la classe `Point` définie ainsi : `class Point { private int x; }`. Quel est (sont) le(s) constructeur(s) possible(s) pour cette classe ?

- ☐ `Point(int p){this=p;}` ☐ `Point(int x) {this.x=x;}`
- ☐ `Point(int y){x=y;}` ☐ `this(int a) {this.x=a;}`

Question 7 : L'instruction suivante : `this();`

- ☐ correspond à un appel au premier constructeur créé dans la classe
- ☐ peut-être utilisée dans toute méthode publique
- ☐ provoque une erreur si la classe ne possède pas de constructeur sans paramètre
- ☐ peut-être utilisée dans la méthode `main` pour créer un nouvel objet
- ☐ doit être placée avant toute autre instruction dans le constructeur

Question 8 : Soient : `int [] tab=new int[5];System.out.println(tab[5]);`

- ☐ 0 est affiché ☐ `null` est affiché ☐ la valeur de `tab[5].toString()` est affiché
- ☐ une exception est levée à l'exécution ☐ une erreur est détectée à la compilation

Question 9 : Soient : `Integer [] tab=new Integer[8];System.out.println(tab[5]);`

- ☐ 0 est affiché ☐ `null` est affiché ☐ la valeur de `tab[5].toString()` est affiché
- ☐ une exception est levée à l'exécution ☐ une erreur est détectée à la compilation

Question 10 : On considère une classe `A`, et une classe `B` qui hérite de `A`. Les 4 instructions suivantes ne provoquent pas d'erreur : `A a1=new A();A a2=new B();B b1=(B)a2; A a3=a1;`

a) Combien y a-t-il d'objets créés dans ces 4 instructions ?

☐ 1 ☐ 2 ☐ 3 ☐ 4

b) Combien y a-t-il de références (*handles*) créées dans ces 4 instructions ?

☐ 1 ☐ 2 ☐ 3 ☐ 4

Question 11 : On considère une classe `Fleur` déclarée abstraite, et une classe `Tulipe` qui hérite de `Fleur`. Chacune de ces 2 classes contient un constructeur sans paramètre. Cocher les instructions correctes :

☐ `Fleur f1=new Fleur();` ☐ `Fleur f2=new Tulipe();`

☐ `Tulipe t1=new Tulipe();` ☐ `Tulipe t2=new Fleur();`

```
class Animal {
      public void f() { }
      public String toString() {return "Animal";}    }
class Poisson extends Animal {
      public void g() { }
      public String toString() {return "Poisson";}   }
class Cheval extends Animal { }
class Zoo { }
```

et les déclarations suivantes :

```
Animal a1=new Animal(); Poisson p1=new Poisson();
Cheval c1=new Cheval(); Zoo z1=new Zoo();
```

```
class Animal {
      public void f() { }
      public String toString() {return "Animal";}    }
class Poisson extends Animal {
      public void g() { }
      public String toString() {return "Poisson";}   }
class Cheval extends Animal { }
class Zoo { }
```

Question 12 : Cocher les instructions qui ne provoquent pas d'erreur à la compilation :

☐ `a1.f();` ☐ `p1.f();` ☐ `a1.g();` ☐ `p1.g();`

```
class Animal {
      public void f() { }
      public String toString() {return "Animal";}    }
class Poisson extends Animal {
      public void g() { }
      public String toString() {return "Poisson";}   }
class Cheval extends Animal { }
class Zoo { }
```

Question 13 : Que retournent les instructions suivantes ?

a) L'instruction `a1.toString()` retourne :

☐ "Animal" ☐ "Poisson" ☐ "Cheval" ☐ "Zoo" ☐ "Animal@1bc4459"

b) L'instruction `p1.toString()` retourne :

☐ "Animal" ☐ "Poisson" ☐ "Cheval" ☐ "Zoo" ☐ "Poisson@1bc4459"

c) L'instruction `c1.toString()` retourne :

☐ "Animal" ☐ "Poisson" ☐ "Cheval" ☐ "Zoo" ☐ "Cheval@1bc4459"

d) L'instruction `z1.toString()` retourne :

☐ "Animal" ☐ "Poisson" ☐ "Cheval" ☐ "Zoo" ☐ "Zoo@1bc4459"

```
class Animal {
      public void f() { }
      public String toString() {return "Animal";}    }
class Poisson extends Animal {
      public void g() { }
      public String toString() {return "Poisson";}   }
class Cheval extends Animal { }
class Zoo { }
```

Question 14 : Cocher les instructions qui ne provoquent pas d'erreur à la compilation :

- ☐ `Animal a2=p1;`
- ☐ `Animal a3=(Animal)p1;`
- ☐ `Poisson p2=a1;`
- ☐ `Poisson p3=(Poisson)a1;`

```
1     class PasMultipleException extends Exception {
2           public PasMultipleException(int nombre, int diviseur) {
3                 super(nombre+" n'est pas un multiple de "+diviseur);
4           }
5     }
6     class TestE {
7           public static int divise(int nombre,int diviseur) ① {
8                 int resultat;
9                 resultat=nombre/diviseur;
10                if (resultat*diviseur!=nombre)
11                      ②
12                return resultat;
13          }
14          public static void main(String [] args) {
15                      System.out.println("res="+divise(10,3));
16          }
17    }
```

Question 15 : Dans la méthode `divise`, si la variable `diviseur` vaut 0 alors l'exception `ArithmeticException` générée par la division par zéro doit être capturée et la méthode `divise` doit retourner 0. Pour cela :

☐ je rajoute entre la ligne 8 et la ligne 9 les instructions :

```
if ( diviseur == 0 ) {
      throw new ArithmeticException();
      return 0;
}
```

☐ je rajoute entre la ligne 8 et la ligne 9 les instructions :

```
if ( diviseur == 0 ) {
      throws ArithmeticException ;
      return 0;
}
```

☐ je remplace la ligne 9 par les instructions :

```
try {
      resultat=nombre/diviseur;
} catch (ArithmeticException ae) {
      return 0 ;
}
```

☐ je rajoute entre la ligne 8 et la ligne 9 les instructions :

```
try {
      if (diviseur==0) return 0;
      else resultat=nombre/diviseur;
} catch (ArithmeticException ae){
      System.out.println(ae.getMessage());
}
```

☐ je rajoute throws ArithmeticException à la ligne 7 à la place de ①

Question 16 : Dans la méthode `divise`, si `nombre` n'est pas un multiple de `diviseur` (c'est-à-dire si le test à la ligne 10 est vrai) alors une exception `PasMultipleException` est levée. Cette exception est ensuite capturée dans la méthode `main`. Pour cela :

a) à la place de ②, je rajoute à la ligne 11:

- ☐ `throws PasMultipleException`
- ☐ `throw new PasMultipleException();`
- ☐ `throws new PasMultipleException("pas multiple");`
- ☐ `throw new PasMultipleException("pas multiple");`
- ☐ `throw new PasMultipleException(nombre,diviseur);`
- ☐ je ne rajoute rien à la ligne 11

b) à la place de ①, je rajoute à la la ligne 7:

- ☐ `throws PasMultipleException`
- ☐ `throw new PasMultipleException();`
- ☐ `throws new PasMultipleException("pas multiple");`
- ☐ `throw new PasMultipleException("pas multiple");`
- ☐ `throw new PasMultipleException(nombre,diviseur);`
- ☐ je ne rajoute rien à la ligne 7

c) dans la méthode `main`, je peux remplacer la ligne 15 par :

☐
```
try {
      System.out.println("res="+divise(10,3));
} catch(PasMultipleException pme) {
      System.out.println(pme.getMessage());
}
```

☐
```
try {
      System.out.println("res="+divise(10,3));
} catch(Exception e) {
      System.out.println(e.getMessage());
}
```

☐ je ne dois pas changer la méthode `main`

Question 17 : Un point est défini par sa coordonnée `x`. Un segment est défini par deux points. Soient les définitions de classes suivantes :

```
class Point {
        private int x ;
        public Point(int x){ this.x=x; }
        public void setX(int x) { this.x=x; }
        public int getX() { return x; }
}
class Segment {
        private Point a,b ;
        public Segment(Point a, Point b) { this.a=a; this.b=b; }
        public void changerAX(int x) { a.setX(x); }
        public int obtenirAX() { return a.getX(); }
}
```

Soient également les déclarations suivantes écrites dans une méthode `main` :

```
Point a1=new Point(1); Point a2=new Point(2);
Segment s1=new Segment(a1,a2);
```

On souhaite créer un segment `s2` qui ait les mêmes valeurs de coordonnées que le segment `s1`. Puis on modifie la coordonnée du premier point de `s1` par la valeur 4. Pour réaliser cela, on considère les instructions :

```
Segment s2=s1;
s1.changerAX(4);
```

Ces 2 instructions ont pour but de réaliser ce qui est demandé, mais il y a une erreur que l'on va corriger dans les questions b), c) et d).

a) Quel est l'affichage obtenu par l'instruction : `System.out.println(s2.obtenirAX());` ?

☐ 1 ☐ 2 ☐ 3 ☐ 4

b) Pour corriger l'erreur détectée à la question a), dans la classe `Point`, je rajoute le constructeur :

☐ `Point(Point p) {x=p.x;}`

☐ `Point(Point p) {p.x=x;}`

☐ `Point(Point p) {x=new Integer(p.x);}`

☐ `Point(Point p) {this(p.x);}`

c) Pour corriger l'erreur détectée à la question a), dans la classe `Segment`, je rajoute le constructeur :

☐ `Segment(Segment s) {a.setX(s.a);b.setX(s.b);}`

☐ `Segment(Segment s) {super(s.a,s.b);}`

☐ `Segment(Segment s) {a=new Point(s.a);b=new Point(s.b);}`

☐ `Segment(Segment s) {this(s.a,s.b);}`

d) Pour corriger l'erreur détectée à la question a), je remplace :

☐ l'instruction : `s1.changerAX(4);` par : `s2.changerAX(4);`

☐ l'instruction : `s1.changerAX(4);` par : `a1.setX(4);`

☐ l'instruction : `Segment s2=s1;` par : `Segment s2=new Segment(s1);`

☐ l'instruction : `Segment s2=s1;` par : `Segment s2=new Segment(s1.a,s1.b);`

1 2014
2 2015
3 2013
4 2012
5 2011
6 2013R

```
1  class Transport {
2          private int cptTransports=0;
3          private int numero;
4          protected String compagnie;
5          public Transport(String compagnie) {
6                  compagnie=compagnie;
7                  cptTransports++;
8                  numero=cptTransports;
9          }
10         public abstract void seDeplacer();
11 }
12 class Bus extends Transport {
13         public Bus() {
14                 compagnie="Inconnue";
15                 numero=5;
16         }
17         public Bus() {
18                 super(compagnie);
19         }
20         public Bus(String compagnie) {
21                 this(compagnie);
22         }
23 }
```

Ces 2 classes contiennent 9 erreurs, dont 7 sont détectées à la compilation. Pour chaque erreur, donnez le numéro de la ligne et expliquez, puis si possible, donnez une correction pour l'erreur. Remarque : plusieurs erreurs peuvent se trouver sur la même ligne.

Question 2 (6 points) : Soit le programme suivant qui ne contient **pas** d'erreurs aux lignes 1 à 9 :

```
1  abstract class Personne { }
2  class Etudiant extends Personne {
3          private static int cptEtudiants=0;
4          private String niveau;
5          public void afficherNiveau() { System.out.println(niveau); }
6          public static void afficherNbEtudiants() {
7             System.out.println("Il y a "+cptEtudiants+" etudiants.");
8          }
9          public static void main(String [] args) {
10                 Etudiant e1=new Etudiant();
11                 Personne p1=new Personne();
12                 Personne p2=e1;
13                 Etudiant e2=p2;
14                 niveau="L2";
15                 cptEtudiants++;
16                 e1.afficherNiveau();
17                 afficherNiveau();
18                 p2.afficherNiveau();
19                 e1.afficherNbEtudiants();
20                 afficherNbEtudiants();
21         }
22 }
```

Question 2a) Parmi les lignes 10 à 13, lesquelles sont fausses ? Expliquez, et corrigez si possible.

Question 2 (6 points) : Soit le programme suivant qui ne contient **pas** d'erreurs aux lignes 1 à 9 :

```
1  abstract class Personne { }
2  class Etudiant extends Personne {
3          private static int cptEtudiants=0;
4          private String niveau;
5          public void afficherNiveau() { System.out.println(niveau); }
6          public static void afficherNbEtudiants() {
7             System.out.println("Il y a "+cptEtudiants+" etudiants.");
8          }
9          public static void main(String [] args) {
10                 Etudiant e1=new Etudiant();
11                 Personne p1=new Personne();
12                 Personne p2=e1;
13                 Etudiant e2=p2;
14                 niveau="L2";
15                 cptEtudiants++;
16                 e1.afficherNiveau();
17                 afficherNiveau();
18                 p2.afficherNiveau();
19                 e1.afficherNbEtudiants();
20                 afficherNbEtudiants();
21         }
22 }
```

Question 2b) La ligne 14 est-t-elle fausse ? Si oui, expliquez et proposez une solution, sinon expliquez pourquoi elle est juste.

Question 2 (6 points) : Soit le programme suivant qui ne contient **pas** d'erreurs aux lignes 1 à 9 :

```
1  abstract class Personne { }
2  class Etudiant extends Personne {
3          private static int cptEtudiants=0;
4          private String niveau;
5          public void afficherNiveau() { System.out.println(niveau); }
6          public static void afficherNbEtudiants() {
7             System.out.println("Il y a "+cptEtudiants+" etudiants.");
8          }
9          public static void main(String [] args) {
10                 Etudiant e1=new Etudiant();
11                 Personne p1=new Personne();
12                 Personne p2=e1;
13                 Etudiant e2=p2;
14                 niveau="L2";
15                 cptEtudiants++;
16                 e1.afficherNiveau();
17                 afficherNiveau();
18                 p2.afficherNiveau();
19                 e1.afficherNbEtudiants();
20                 afficherNbEtudiants();
21         }
22 }
```

Question 2c) Même question que la question 2b) pour la ligne 15.

Question 2 (6 points) : Soit le programme suivant qui ne contient **pas** d'erreurs aux lignes 1 à 9 :

```
1  abstract class Personne { }
2  class Etudiant extends Personne {
3          private static int cptEtudiants=0;
4          private String niveau;
5          public void afficherNiveau() { System.out.println(niveau); }
6          public static void afficherNbEtudiants() {
7             System.out.println("Il y a "+cptEtudiants+" etudiants.");
8          }
9          public static void main(String [] args) {
10                 Etudiant e1=new Etudiant();
11                 Personne p1=new Personne();
12                 Personne p2=e1;
13                 Etudiant e2=p2;
14                 niveau="L2";
15                 cptEtudiants++;
16                 e1.afficherNiveau();
17                 afficherNiveau();
18                 p2.afficherNiveau();
19                 e1.afficherNbEtudiants();
20                 afficherNbEtudiants();
21         }
22 }
```

Question 2d) Quelles lignes sont fausses parmi les lignes 16 à 20 ? Expliquez.

1 2014
2 2015
3 2013
4 2012
5 2011
6 2013R

```java
public class A {
    public A() { System.out.println("constructeur sans argument"); }
    public A(int x){ System.out.println("x="+x+" est un entier");}
    public A(double y){ System.out.println("y="+y+" est un double");}
    public A(String s){ System.out.println("s="+s+" est une chaine");}
}
public class B extends A {
    public B(int x) { super(x); }
    public B(char x) { super(x); }
    public B(float x) { super(x); }
    public B(int x,int y) { super(x+y); }
    public B(char x,char y) { super(x+""+y); }
    public B(B b) { System.out.println("constructeur par copie"); }
}
public class TestConstructeur {
    public static void main(String [] args) {
        B b1=new B(5);
        B b2=new B(3.5f); // rappel : 3.5f est de type float
        B b3=new B('Z');
        B b4=new B('Z','Z');
        B b5=new B(new B(5));
    }
}
```

Pour chaque instruction des lignes 17 à 21, donnez l'affichage obtenu et expliquez. Rappel : le code ASCII de 'Z' est 90.

```java
public abstract class Fourniture {
    public String toString() { return "Fourniture";}
}
public class Agrafeuse extends Fourniture {
    public String toString() { return "Agrafeuse";}
}
public class Gomme extends Fourniture {}
public class Crayon extends Fourniture {}
public class CrayonAPapier extends Crayon {}
public class Portemine extends Crayon {
    public String toString() { return "Portemine";}
}
```

On suppose que l'on est dans une méthode `main`.

Q2.1 : Expliquez pourquoi il est possible de déclarer un tableau contenant (dans l'ordre) : u
agrafeuse, une gomme, un crayon, un crayon à papier et un portemine. Puis donnez cette déclaratio

```java
public abstract class Fourniture {
    public String toString() { return "Fourniture";}
}
public class Agrafeuse extends Fourniture {
    public String toString() { return "Agrafeuse";}
}
public class Gomme extends Fourniture {}
public class Crayon extends Fourniture {}
public class CrayonAPapier extends Crayon {}
public class Portemine extends Crayon {
    public String toString() { return "Portemine";}
}
```

Q2.2 : Donnez les instructions nécessaires pour afficher dans l'ordre le résultat de la méthode `toString()` de chaque objet dans le tableau sans appeler directement la méthode `toString()`. Expliquez.

```java
public abstract class Fourniture {
    public String toString() { return "Fourniture";}
}
public class Agrafeuse extends Fourniture {
    public String toString() { return "Agrafeuse";}
}
public class Gomme extends Fourniture {}
public class Crayon extends Fourniture {}
public class CrayonAPapier extends Crayon {}
public class Portemine extends Crayon {
    public String toString() { return "Portemine";}
}
```

Q2.3 : D'après l'ordre des objets dans le tableau, quel est précisément l'affichage obtenu ? Expliq

Question 3 (3 points) : Soient les classes suivantes :

```
1 public abstract class Electromenager {
2     public abstract void avancer();
3 }
4 public class Aspirateur extends Electromenager {
5     public void aspirer(){System.out.println("J'aspire");}
6     public void avancer(){System.out.println("J'avance␣en␣roulant");}
7 }
8 public class AspirateurASac extends Aspirateur {
9     public void aspirer(){System.out.println("J'aspire␣avec␣un␣sac")
10    public void changerSac(){System.out.println("Le␣sac␣est␣change")
11 }
```

Q3.1 : Parmi les instructions suivantes, quelles instructions ne compilent pas ? Expliquez cha
erreur.

```
1 AspirateurASac aas1=new AspirateurASac();
2 Aspirateur a1=new Aspirateur();
3 Electromenager e1=new Electromenager();
4
5 Electromenager e2=a1;
6 Aspirateur a2=aas1;
7 AspirateurASac aas2=aas1;
8
9 e2.aspirer();
10 a2.aspirer();
```

```
11 aas2.aspirer();
12
13 e2.avancer();
14 a2.avancer();
15 aas2.avancer();
16
17 e2.changerSac();
18 a2.changerSac();
19 aas2.changerSac();
```

Q3.2 : Qu'affiche l'instruction : `a2.aspirer()` ? Expliquez.

Question 4 (3 points) : Soient les classes suivantes :

```
1 public class Meuble {}
2 public class Bureau extends Meuble {}
3 public class BureauMultimedia extends Bureau {}
```

Q4.1 : Parmi les instructions suivantes, quelles instructions ne compilent pas ? Expliquez chaque erreur.

```
1 Meuble m1=new Bureau();
2 Meuble m2=new BureauMultimedia();
3
4 Bureau b1=new Meuble();
5 Bureau b2=new BureauMultimedia();
6
7 BureauMultimedia bm1=new Meuble();
8 BureauMultimedia bm2=new Bureau();
```

Q4.2 : Soient les instructions suivantes :

```
1 Meuble m3=new Meuble();
2 Bureau b3=new Bureau();
3 BureauMultimedia bm3=new BureauMultimedia();
4 Meuble m4=b3;
5 Meuble m5=bm3;
6
7 Bureau b4=m4;
8 Bureau b5=(Bureau)m3;
9 Bureau b6=(Bureau)m4;
10 Bureau b7=(Bureau)m5;
11 Bureau b8=(BureauMultimedia)m4;
12 Bureau b9=(BureauMultimedia)m5;
```

Les lignes 1 à 5 sont correctes. Parmi les lignes 7 à 12, quelles instructions ne compilent pas ? quelles instructions provoquent une erreur à l'exécution ? Expliquez précisément chaque erreur.

Question 5 (3 points) : Soient les classes suivantes :

```
1 public class DepassementException extends Exception {
2     public DepassementException() { super("Depassement"); }
3 }
4 public class Balance {
5     private int MAX=150;
6     public void peser (int poids) throws DepassementException {
7         if (poids >= MAX) {
8             throw new DepassementException();
9         }
10        System.out.println("Poids : "+poids+" kg");
11    }
12 }
```

Q5.1 : Le programme suivant compile-t-il ? Si non, donner précisément la ligne de la première erreu
et expliquez. Si oui, provoque-il une erreur à l'exécution ? Expliquez.

```
1 public class TestException {
2     public static void main(String [] args) {
3         Balance b1=new Balance();
4         b1.peser(50);
5         b1.peser(170);
6     }
7 }
```

Q5.2 : En ajoutant seulement des instructions dans la méthode `main`, proposez une solution q
compile et s'exécute. Puis donnez l'affichage obtenu par l'exécution de la méthode `main` corrigée,
expliquez.

Question 6 (3 points) : La classe `System` du package `java.lang` possède une variable statique
type `PrintStream` appelée `out` qui permet d'écrire sur la sortie standard. On rappelle qu'il ex
qu'une seule sortie standard. Soit la classe suivante qui a pour but de fournir des méthodes p
afficher une chaîne de caractères sur la sortie standard.

```
1 import java.io.PrintStream;
2 public class S {
3     private static PrintStream ps=System.out;
4
5     public static void afficher1(String s) {
6         ps.println(s);
7     }
8     public void afficher2(String s) {
9         ps.println(s);
10    }
11 }
```

Q6.1 : Expliquez l'instruction à la ligne 3. Pourquoi la variable `ps` est-elle déclarée `static` ?

Q6.2 : Donnez l'instruction qui permet d'afficher la chaîne de caractères `"Bonjour !"` en utilisan méthode `afficher1`.

Q6.3 : Donnez l'instruction qui permet d'afficher la chaîne de caractères `"Bonjour !"` en utilisan
méthode `afficher2`.

Q6.4 : La classe `PrintStream` possède une méthode `format` dont la signature est :

```
public PrintStream format(String format, Object... args)
```

qui prend en paramètre un format et des variables et qui affiche la chaîne formattée construite. nombre de variables en argument dépend du nombre de variables à afficher.
Par exemple : format(`"%5s=%6.2f\n"`,`"PI"`,3.14159) demande d'afficher `"PI"` comme une chaîne (% sur 5 caractères, et le nombre `3.14159` au format décimal (%f) sur 6 caractères, dont exactement chiffres après la virgule. La chaîne affichée sera donc : `"␣␣␣PI=␣␣3,14"`. Remarque, il y a 3 espac insérés avant la lettre 'P' pour afficher la chaîne `"PI"` sur 5 caractères et deux espaces insérés ent '=' et '3' pour afficher le nombre sur 6 caractères.
Dans la classe `S`, ajoutez une méthode statique `afficherTab(double [] tab)` qui affiche le table de `double` passé en paramètre sur 4 colonnes. Chaque nombre est affiché sur 8 caractères avec chiffre après la virgule. Aide : la fonction `println()` sans paramètre affiche un retour à la ligne. Vo un exemple d'affichage pour un tableau de 10 nombres réels quelconques.

```
 2.9     52.3     23.8      9.7
73.9      4.6     80.5     42.1
71.5     23.2
```