TD/TP de Java Numéro 1
Constructeurs, Types Primitifs, Classes, Objets, Variables et Méthodes d'Instance, Variables et Méthodes de Classe, Visibilité et Références d'Objet

Dr. KENMOGNE Edith Belise

Questions de Cours

1) Quelle est la différence entre la notion de classe et la notion d'objet ?
2) Qu'est ce qu'une instance d'une classe ?
3) Définir les expressions suivantes : (a) variable de classe, (b) variable d'instance, (c) méthode de classe, (d) méthode d'instance, (e) constructeur d'une classe, (f) accesseur en modification, (g) accesseur en consultation.
4) Donner la signification de la visibilité *public* quand elle est affectée à une méthode.
5) Qu'est ce qu'un *package* ?
6) Même question qu'en 3) avec *private* et *package*.
7) Donner la signification de la visibilité *public* quand elle est affectée à une variable d'instance ou de classe.
8) Quelle est la signification du mot clé *this sans parenthèse* dans un constructeur ?
9) Quelle est la signification du mot clé *this avec parenthèses* dans un constructeur ?
10) Quelle est la signification du mot clé *this* dans une méthode d'instance ?
11) Même question qu'en 5) avec *private* et *package*.
12) Quel est l'opérateur d'instanciation (de création d'un objet) en Java ?
13) Quel est l'opérateur d'accession à une variable d'instance et de classe ?
14) Quel est l'opération d'envoi de message à un objet ?
15) Il n'y a pas de type booléen en C. Cependant, y a un type booléen en Java. Quel est son nom ?
16) Même question que la question 14) pour les chaînes de caractères.
17) L'opérateur « == » est utilisé pour comparer les contenus de deux variables. Pourquoi permet il de comparer deux variables d'un type primitif et pas deux variable décalrées comme des objets d'une classe ?

Problème 1 : Analyse de Code, Correction de Code et Enrichissement de Code

Partie I : Analyse du Code de la Classe Point

1) On rappelle qu'en Java, une variable d'un type primitif contient une valeur et une variable déclarée comme objet contient une référence. Schématiser l'évolution des contenus des variables f, p1 et p2 pendant l'exécution des six premières instructions de la méthode *main*.

2) Quelle est l'intérêt de la méthode toString ? **On rappelle qu'en Java, lorsqu'on demande d'afficher un objet avec la méthode *println*, le message *toString()* est d'abord envoyé au dit objet et l'objet *String* retourné lors de l'exécution de la méthode *toString()* est affiché.** Qu'est ce qui est affiché par la septième et la huitième instruction de la méthode main de la classe Point ?

3) Pourquoi la méthode main de la classe Point a le droit d'accéder d'accéder aux membre de l'objet p1 en écriture (*à la neuvième instruction*) et en lecture (*à la dixième instruction*).

4) Pourquoi la méthode main de la classe *Point* a le droit d'accéder au constructeur à un argument de la classe Point (*à la onzième instruction*).

5) Donner une instruction équivalente pour la septième et huitième instruction en remplaçant *System.out.println* par *presenteToi()*.

6) Donner la signification de l'unique instruction (*this(a, a)*) du constructeur de Point à un argument.

7) **On rappelle qu'on n'a pas le droit d'envoyer un message à une reference nulle, ni a une variable qui ne pointe pas sur un objet.** En déduire pourquoi le bout de code suivant est incorrect « *Point p ; p.setX(16) ;* »

Partie II : Analyse et Correction du Code de la Classe Cercle

1) Expliquer pourquoi les quatre dernières lignes marquées en gras ne sont correctes.

2) Y aurait-il une erreur si la visibilité de des variables d'instance x et y de Point, du constructeur à un argument de Point et de la méthode *PresenteToi* de *Point* était *public* ?

3) Corriger les lignes incorrectes.
4) Expliquer pourquoi la première ligne en gras est correcte.
5) Schématiser l'évolution des contenus des variables c, p1 et c2 pendant l'exécution de *main*.
6) Donner ce qui est affiché par l'avant dernière instruction de la méthode principale main.

```
public class Point{
  private float x ;
  private float y ;
  public void Point(float a, float b) {
    this.x=a ;
    this.y=b ;
  }

  private Point(float a) {
    this(a, a) ;
  }

  public float getX() {return this.x ; }
  public float getY() {return this.y ; }
  public void setX(float a) {this.x=a ; }
  public void setY(float b) {this.y=b ; }
  public String toString(){
     return "(" +  this.x  + ","   +  this.y  +  ")" ;
  }
  private void presenteToi() {
    System.out.println(this) ;
  }
}
public static void main(String[ ] args) {
  float f=14 ;
  Point p1= new Point(13, 12) ;
  Point p2 ;
  p2=p1 ;
  p2.setX(18) ;
  p.setY(22) ;
  System.out.println(p1) ;
  System.out.println(p2) ;
  p1.x=25 ;
  System.out.println(p1.y) ;
  Point p3=new Point(120) ;
}
```

```
public class Cercle{
  public static final float PI=3,14 ; //Constante de Classe
  private Point centre ;
  private float rayon ;
  void Cercle(float x, float y, float ray) {
    this.centre=new Point(x, y); //Correct
    this.rayon=ray ;
  }
  void Cercle(float x, float ray) {
    this.centre=new Point(x); //Erreur
    this.rayon=ray ;
  }
  void Cercle(Point center, float ray) {
    this.centre=center ;
    this.rayon=ray ;
  }
  float getRayon() {return this.rayon ; }
  Point getCentre() {return this.center ; }
  void setRayon(float rayon) {this.rayon=rayon ; }
  void setCentre(Point centre) {this.centre=centre ; }
  void setCentre(float a, float b) {
     this.centre.x=a ;  //Erreur
     this.centre.y=b ;  //Erreur
   }
  public String toString(){
     return "[" + (this.centrer).toString() + "," +this.rayon  + "]" ;
  }
  public static void main(String[ ] args) {
     Cercle c= new Cercle(0, 0, 14) ;
     Point p1= new Point(15, 17) ;
     p1.setX(17) ;
     c.setCentre(p1) ;
     System.out.println(c) ;
     Cercle c2=new Cercle(p1, 25) ;
     System.out.println(c) ;
     p1.presenteToi() ; //Erreur
  }
```

Partie III : Enrichissement du Code de la Classe Point

Ecrire et tester les méthodes décrites ci-dessous.

1) La méthode d'instance *public float distance(Point p)* invite l'objet recepteur du message à calculer et à retourner la distance qui le sépare de l'objet porté par le message.

2) La méthode de classe *public float distance(Point p, Point)* invite la classe Point receptrice du message à calculer et à retourner la distance qui sépare les deux objets portés par le message.

3) La méthode d'instance *public Point milieu(Point p)* invite l'objet recepteur du message à calculer et à retourner le milieu entre lui et l'objet porté par le message.

4) La méthode de classe *public Point milieu(Point p1, Point p2)* invite la classe Point receptrice du message à calculer et à retourner le milieu entre les deux objets portés par le message.

5) La méthode d'instance *boolean equals(Point p)* invite l'objet recepteur du message à calculer et à retourner *true* s'il est égal à l'objet Point porté par le message et *false* sinon.
6) La méthode d'instance *public Object clone( )* invite l'objet recepteur du message à construire et à retourner une copie de lui même (C'est le clonage). Il s'agit de faire une dublication en profondeur. **On rappelle que Object est la classe ancêtre de toutes les classes Java.**

Partie IV : Enrichissement du Code de la Classe Cercle
Ecrire et tester les méthodes décrites ci-dessous.
1) La méthode d'instance *public float perimetre( )* invite l'objet recepteur du message à calculer et à retourner son périmètre.
2) La méthode de classe *public float perimetre(Cercle c)* invite la classe Cercle receptrice du message à calculer et à retourner le périmètre de l'objet porté par le message.
3) La méthode d'instance *public float surface( )* invite l'objet recepteur du message à calculer et à retourner sa surface.
4) La méthode de classe *public float surface(Cercle c)* invite la classe Cercle receptrice du message à calculer et à retourner la surface de l'objet porté par le message.
5) La méthode d'instance *public boolean surLeCercle(Point p)* invite ll'objet Cercle recepteur du message à retourner true si l'objet Point porté par le message est sur le cercle et false sinon. Le Point p est sur le cercle si la distance qui le sépare du centre du cercle est égale au rayon du cercle.
6) La méthode d'instance *public boolean dansLeCercle(Point p)* invite ll'objet Cercle recepteur du message à retourner true si l'objet Point porté par le message est dans le cercle et false sinon. Le Point p est dans le cercle si la distance qui le sépare du centre du cercle est inférieure ou égale au rayon du cercle.
7) La méthode d'instance *public boolean equals(Cercle c)* invite l'objet recepteur du message à calculer et à retourner *true* s'il est égal à l'objet porté par le message et *false* sinon.
8) La méthode d'instance *protected Object clone( )* invite l'objet recepteur du message à construire et à retourner une copie de lui même (C'est le clonage). Il s'agit de faire une dublication en profondeur. **On rappelle que Object est la classe ancêtre de toutes les classes Java.**

Exercice 1 : Les Rectangles
Un rectangle est défini par sa son point supérieur gauche, sa largeur et sa longueur. Il a trois variables d'instance : le point supérieur gauche, la largeur et la longueur. Nous travaillons ici en dimension 2 (2D). On utilisera la classe Point du problème 1. Ecrire et tester les méthodes décrites ci-dessous.
1) Un constructeur sans argument et un autre qui prend quatre arguments : l'abcisse du point supérieur gauche, l'ordonné du point supérieur gauche, la largeur et la longueur.
2) Les accesseurs en consultation et en modification.
3) Une méthode d'instance *public float perimetre()* qui invite le rectangle recepteur du message à calculer et à retourner son périmètre.
4) Une méthode d'instance qui invite le rectangle recepteur du message à calculer et à retourner sa surface.
5) Une méthode d'instance *public boolean equals(Rectangle r)* qui invite le rectangle recepteur du message à dire s'il est égal ou non au rectangle *r* porté par le message *equals*. Deux rectangles sont égaux s'ils ont le même point supérieur gauche, la même largeur et la même longueur.
6) Une méthode d'instance *public String toString()* qui invite l'objet (de la classe Rectangle) recepteur du message à retourner une chaîne de caractères qui le décrit. Par exemple *Rectangle[(4,5), 5000, 10000]*.
7) La méthode d'instance *protected Object clone( )* invite l'objet recepteur du message à construire et à retourner une copie de lui même. Il s'agit de faire une dublication en profondeur.
8) Une méthode de classe *public static float perimetre(Rectangle r)* qui invite la classe *Rectangle* receptrice du message à calculer et à retourner le périmètre du rectangle porté par le message..
9) Une méthode de classe qui invite la classe *Rectangle* receptrice du message à calculer et à retourner la surface du rectangle porté par le message.
10) Une méthode d'instance *public boolean estDans(Point p)* qui invite la classe *Rectangle* recepteur du message à retourner *true* si le point porté par le message *est* dans le rectangle et false sinon.

11) Une méthode d'instance *public boolean estInclus(Rectangle)* qui invite la classe *Rectangle* recepteur du message à retourner *true* si le rectangle *r* porté par le message est inclus dans le rectangle recepteur du message et *false* sinon.

Exercice 3 : Les Elèves

Un élève est décrit par un nom, une note en mathématique, une note en physique, une note en français et une note en anglais. Le nom d'un élève est une chaîne de caractères en majuscule. Ecrire et tester les méthodes décrites ci-dessous.

1) Un constructeur à un argument (qui prend en entrée le nom) et un autre à cinq arguments (qui prend en entrée le nom et les notes des quatre matières). On utilisera la méthode d'instance *String toUpperCase()* de la clase String pour convertir le nom en majuscule.

2) Les accesseurs en modifications par matières.

3) Une méthode d'instance *public String toString()* qui invite l'objet Elève recepteur du message à retourner une chaîne de caractères qui le décrit. Par exemple *Eleve[math 15, phy 18, fra 14, ang 16]*.

4) Une méthode d'instance *public float moyenne()* qui invite l'objet Elève recepteur du message à calculer et retourner retourne sa moyenne.

5) On voudrait modifier la classe élève de manière à ce que chaque élève possède **un matricule qu'on ne peut pas changer et que deux élèves possèdent des matricules distincts.** Pour cela on introduit une variable de classe appelée *nbreCreation* initialisée à *0* qui contiendra à chaque instant le nombre d'élève déjà crées. Dans chaque constucteur, la valeur de *nbreCreation* est affectée comme matricule de l'objet Eleve en cours de création puis incrémentée de 1.

6) Un accesseur en consultation sur le matricule. Pourquoi, il ne faudrait pas prévoir un accesseur en modification sur le matricule ?

7) La méthode d'instance *public boolean equals(Eleve e)* invite l'objet recepteur du message à calculer et à retourner *true* s'il est égal à l'objet porté par le message et *false* sinon. Deux élèves sont égaux s'ils ont le même matricule.

8) La méthode d'instance *protected Object clone( )* invite l'objet recepteur du message à construire et à retourner une copie de lui même (C'est le clonage).

Exercice 4: Les Segments

Nous travaillons ici en dimension 2 (2D). Un segment est décrit par deux points. On utilisera la classe Point. Ecrire et tester les méthodes décrites ci-dessous.

1) Un constructeur qui construit un segment à partir de deux points.

2) Les accesseurs en modifications et en consultation.

3) Une méthode d'instance *public String toString()* qui invite l'objet Segment recepteur du message à retourner une chaîne de caractères qui le décrit. Par exemple Segment*[(4,5), (10,15)]*.

4) La méthode d'instance *public boolean equals(Segment s)* invite l'objet recepteur du message à calculer et à retourner *true* s'il est égal à l'objet porté par le message et *false* sinon.

5) Une méthode d'instance appelée *longueur* qui calcule la longueur d'un segment.

6) La méthode d'instance *protected Object clone( )* invite l'objet recepteur du message à construire et à retourner une copie de lui même (C'est le clonage).

7) Une méthode d'instance *public float distance()* qui retourne la distance du segment recepteur du message.

8) Une méthode d'instance *public boolean appartient(Point p)* qui retourne *true* si le point porté par le message appartient au segment recepteur du message et *false* sinon. Un point *p* appartient à un segment si la somme des deux distances qui le sépare aux deux extrémités du segment est égale à la longueur du segment.

9) Une méthode d'instance *boolean estInclus(Segment s)* qui dit si le segment porté par le message est inclus dans le segment recepteur du message. Le segment *s* est inclus dans le segment *this* si les deux extrémités de *s* appartiennent à *this*.