5. Solutions des exercices

Exercice 1.1

Donnez un exemple de fonctionnalité largement réutilisable.
Réponse : Parmi les fonctionnalités les plus répandues et pouvant être réutilisée dans les applications actuelles l'authentification des utilisateurs d'une application ou d'un site Web.

Exercice 1.2

Quels sont les principaux nouveautés du modèle en spiral par rapport aux autres modèles ?
Réponse : Le modèle spiral est un modèle complexe qui intègre pratiquement tous les autres modèles de processus logiciels (prototypage, séquentiel ...etc). Sa nouveauté réside dans le fait qu'il incorpore **l'analyse des risques** à chaque révolution de la spirale ce qui constitue un excellent contrôle durant le processus de développement contre les risques de natures divers (solution obsolète, logiciel développé déjà par un rival, besoin devenu non impertinent ...etc).

Exercice 1.3

Dans le processus de développement par composants, que faire si on trouve pas de composant qui assure une fonction donnée ?
Réponse : Dans le cas d'absence d'un composant adéquat pour un besoin donné on procède comme suit :

1. On essaye tout d'abord de modifier, dans la mesure du possible, le besoin afin qu'il coïncide avec un composant existant.
2. Dans le cas échéant, on créé le composant (par l'équipe de développement ou par sous-traitance).

L'idée est de favoriser toujours **la réutilisation** qui est au cœur du développement par composant.

Exercice 2.1

Donnez le code du plus petit programme en Java affichant le message « HelloWorld ».

```
1 // fichier Main.java
2 public class Main {
3     public static void main(String args[]) {
4         System.out.println("HelloWorld") ;
5     }
6 }
```

Même l'affichage d'un message nécessite une classe principale et une méthode principale main. Java est un langage purement orienté objet (on ne peut pas écrire du code en dehors des classes).

Exercice 2.2

Étant donné une classe java quelconque, on veut que cette classe soit directement interprétable (exécutable) par la JVM, que doit-on ajouter à cette classe ?

On doit ajouter à la classe la méthode principale : public static void main(String args[]) ....

On peut instancier des objets de la classe dans la méthode main et invoquer les différentes méthodes a titre de test. Dans une application Java constituée de plusieurs classes on peut ajouter à chaque classe une méthode main et on peut interpréter cette classe séparément des autres (pour test unitaire). Après les tests, on peut enlever les différentes méthodes main et ne laisser qu'une seule pour l'application.

Exercice 2.3

Donnez le code d'une classe Java « First ». Ajoutez un constructeur par défaut à cette classe qui affiche le message « Un Objet est créé » puis un destructeur qui affiche « Un Objet est détruit ». Ajoutez une méthode principale à cette classe (main) dans laquelle vous instanciez (créez) deux objets f1 et f2.

```
1  // fichier First.java
2  public class First {
3      public First() {
4          System.out.println("Un Objet est créé") ;
5      }
6      public void finalize(){
7          System.out.println("Un Objet est détruit") ;
8      }
9      public static void main(String args[]) {
10         First f1=new First() ;
11         First f2=new First() ;
12     }
13 }
```

Q1) Comment détruire l'objet f1 ? Pour détruire un objet Java, il faut le « déréférencer » i.e le nombre (compteur) de références à cet objet doit être 0. Dans notre cas (une seule référence f1) il suffit de faire : f1=null ;

Q2) Comment faire pour s'assurer qu'à chaque destruction d'un objet le message du destructeur s'affiche ?

Le ramasse-miettes gc (garbages-collector) de la JVM s'exécute en concurrence avec nos applications, rien n'assure qu'il procède au nettoyage nécessaire au moment voulu (lorsque on déréférence l'objet). Pour s'assurer que le gc exécute le destructeur de notre classe (et par conséquent l'affichage) il faut l'invoquer manuellement par : System.gc() ; // à exécuter après f1=null ;

Q3) On veut garder la trace du nombre d'objets de la classe « First » existant à chaque instant en utilisant un compteur entier - Modifiez le code de la classe First en ajoutant une méthode qui affiche à chaque fois la valeur du compteur d'objet

```
1  // fichier First.java
2  public class First {
```

```
3       static int nbObject ;
4       public First() {
5           System.out.println("Un Objet est créé") ;
6           nbObject ++ ;
7       }
8       public finalize(){
9           System.out.println("Un Objet est détruit") ;
10          nbObject - - ;
11      }
12      Public static printnbObject(){
13          System.out.println("Nombre d'objets actifs :"+nbObject) ;
14      }
15      public static void main(String args[]) {
16          First f1=new First() ; First.printnbObject() ;
17          First f2=new First() ; First.printnbObject() ;
18          f1=null ; First.printnbObject() ;
19          System.gc() : First.printnbObject() ;
20      }
21  }
```

Q4 (à documenter par l'étudiant) Parmi les fonctionnalités (rôles) des constructeurs d'une classe l'initialisation des attributs d'instance (objets). Existe-t-il une méthode pour initialiser les attributs de classe ?

Réponse : Oui, Comme pour les attributs d'instances, on peut inclure un Constructeur de classe dont le rôle est l'initialisation des attributs de classe. Syntaxe : static..... // ceci est un bloc d'instructions constituant le constructeur static de la // classe

Exercice 2.4

Q1) Donner le code Java de la classe Cercle avec :
Attributs : double rayon ;
Méthodes :

- Cercle() ;
- Cercle(double nR) ;
- double getSurface() ;
- double getPerimetre() ;
- void setRaon(double nR) ;

Solution :

```
1   public class Cercle {
2       double rayon;
3       public Cercle() {
4           rayon=1;
5       }
6       public Cercle(double rayon) {
7           // this représente une référence de l'objet en cours,
        obligatoire de l'utiliser dans le cas de chevauchement entre le nom
        de l'attribut et le nom du paramètre formel
8           this.rayon = rayon;
9       }
```

```
10      public double getSurface(){
11          return rayon*rayon*Math.PI;
12      }
13      public double getPerimetre() {
14          return 2*rayon*Math.PI;
15      }
16  }
```

a) Ecrire une Méthode principale main dans Cercle qui utilise la classe Cercle pour créer 3 objets (cercle1,cercle2 et cercle3) ayants respectivement les rayons 1, 25 et 125 et permettre d'afficher le périmètres et la surface de chacun des trois objets Cercle. Ensuite, le programme change le rayon du second avec la valeur 100 et affiche son rayon et sa surface.

```
1  public static void main(String args[]){
2      Cercle cercle1=new Cercle();
3      System.out.println("La surface du cercle ayant le rayon="+cercle1.
       rayon+" est:"+              cercle1.getSurface()+ "et son périmètre
        est:"+cercle1.getPerimetre());
4      Cercle cercle2=new Cercle(25);
5      System.out.println("La surface du cercle ayant le rayon="+cercle2.
       rayon+" est:"+              cercle2.getSurface()+ "et son périmètre
        est:"+cercle2.getPerimetre());
6      Cercle cercle3=new Cercle(125);
7      System.out.println("La surface du cercle ayant le rayon="+cercle2.
       rayon+" est:"+              cercle2.getSurface()+ "et son périmètre
        est:"+cercle2.getPerimetre());
8  }
```

b) Pour détruire les objets de la classe Cercle, un programmeur a ajouter la méthode suivante dans la classe Cercle :

```
1  public static void detruire(Cercle c){
2  c=null ;
3  }
```

Et dans la méthode main il a ajouté l'instruction :

```
1  Cercle.detruire(cercle1) ; // pour détruire le premier objet
```

La méthode ajoutée va-t-elle détruire effectivement l'objet cercle1 ? justifiez votre réponse.

Réponse : Non la méthode ne détruira pas l'objet cercle1. En java, le seul mode de passage des paramètres est le passage par valeur, la référence passée à la méthode detruire(...) sera dupliquée, la méthode affecte la valeur null à la copie dupliquée de la référence. Après l'exécution de la méthode détruire l'objet cercle1 restera toujours en vie (non null).

Exercice 2.5

Dans un fichier Poeme.java définissez la classe Fleur de sorte que le programme principal suivant :

```
1  public class Poeme {
2      public static void main(String[] args) {
3          Fleur f1 = new Fleur("Violette", "bleu");
4          Fleur f2 = new Fleur(f1);
5          System.out.print("dans un cristal ");
6          f2.eclore();
7          System.out.print("ne laissant plus ");
8          System.out.println(f1);
9          System.out.println(f2);
10     }
11 }
```

affiche le texte suivant :
Violette fraichement cueillie
Fragile corolle taillée dans un cristal veiné de bleu
ne laissant plus qu'un simple souffle
qu'un simple souffle

Solution : En analysant le texte à afficher :

- La classe Fleur doit comporter deux attributs ; nom et couleur
- Le constructeur de la classe Fleur doit afficher l'attribut nom suivi de "fraichement cueillie" dans une première ligne et "Fragile corolle taillée" dans une deuxième ligne.
- La classe Fleur doit comporter une méthode eclore() qui affiche «le texte « veiné de » suivi de l'attribut couleur.
- La conversion d'un objet de la classe Fleur en une chaine de caractère doit donner la chaine "qu'un simple souffle" pour que print(f1) et print(f2) puisse donner l'affichage désiré. La conversion se fait par une méthode toString héritée de la classe Object à redéfinir dans Fleur
- La classe doit inclure un constructeur par copie (on lui passant un objet comme paramètre il crée une copie identique à ce dernier (appelé clone).

Selon les spécifications précédentes le code de la classe Fleur est le suivant :

```
1  public class Fleur {
2      String nom;
3      String color ;
4      public Fleur(String nom,String color) {
5          this.nom=nom ;
6          this.color=color ;
7          System.out.println(nom+" fraichement cueillie") ;
8          System.out.print("Fragile corolle taillée ") ;
9      }
10     public Fleur(Fleur clone){
11         this.nom=clone.nom ;
12         this.color=clone.color ;
```

```
13      }
14      public void eclore(){
15          System.out.println("veiné de "+this.color) ;
16      }
17      @override // annotation de la redéfinition des méthodes
18      public String toString() {
19          return "qu'un simple souffle";
20      }
21  }
```

Exercice 2.6

Une Voiture est constitué de :

— un Chassis (caractérisé par un numéro de série)
— une Carosserie (caractérisée par son type : sport, berline, citadine)
— un Habitacle (caractérisé par le nombre de portes 02 ou 04)
— un Moteur (caractérisé par sa capacité en litre,puissance en nombre de chevaux et type d'énergie (Essence,Gas-oil ou GPL-C))
— une Boite de vitesse (caractérisé par type : manuelle/Automatique et le nombre de rapport 5,6 ..).
— 04 Roues
— 04 ou 02 Portes selon le type de carosserie (AV-G,AV-D,AR-G,AR-D) et
— 04 Optiques

Q1) Proposez le code Java correspondant aux différentes classes nécessaires à la modélisation de l'énoncé de cet exercice.

```
1  public Class Voiture {
2      Chassis ch
3      Carosserie cr ;
4      Habitacle hb ;
5      Moteur mt ;
6      Boite bt ;
7      Roue rag,rad,rrg,rrd ;
8      int nbPorte ;
9      Optique opag,opad,oprg,oprd ;
10 }
```

```
1  public class Chassis {
2      private int nSerie ;
3      public Chassis(ns) {nserie=ns ;}
4  }
```

```
1  public class Carosserie {
2      private String crtype ;
3      public Carosserie(String t) {crtype=t ;}
4  }
```

```
Public class Habitacle {
    int nbPorte ;
    public Habitacle(in nbPorte) {
    if ((nbPorte==2) || (nbPorte==4))
        this.nbPorte=nbPorte ; else nbPorte=4 ;
    }
}
```

```
public class Moteur {
    float capacite ;
    int nbChevaux ;
    String energie ;
    public Moteur (float c,int ch,String en) {
        capacite=c ;nbChevaux=ch ;energie=en ;
    }
}
```

```
public class Boite {
    char typeBoite ;
    int nbRapports ;
    public Boite(char tb,int nbr) {
        typeBoite='m' ;  // par défaut
        if(tb='a')typeBoite='m' ;
        if (nbr>=5) {nbRapports=nbr ;}
    }
}
```

```
Public class Optique {
// ...
}
```

Q2) Dans le langage Java On peut déclarer des classes à l'intérieur d'une classe (les inner-classes) mais cette pratique est très rares et déconseillée, et il est toujours suggérer de déclarer séparément les classes. A votre avis pourquoi ?
Réponse : les inner-classes présentent les inconvénients suivants :

1. Difficulté de réutilisation de telles classes
2. Conception ambigüe due à l'imbrication du code de ces classes dans d'autres classes

Exercice 2.7

Dans l'exercice 4 (classe Cercle) Comment peut-on éviter des valeurs négatives ou nulles pour l'attribut Rayon ?

Réponse : en encapsulant l'attribut rayon par :

```
1 private double rayon ;
2 public void setRayon(double rayon){
3     if (rayon>0) this.rayon=rayon ;
4 }
5 public double getRayon(){
6     return rayon ;
7 }
```

Exercice 2.8

Un objet Robot qui a deux vitesses (gauche et droite). Il peut avancer en avant (ou reculer en arrière) quand ses deux vitesses sont égales et positives (négatif respectivement). Comme il peut tourner vers la gauche (ou la droite) si les 2 vitesses sont différentes dans le sens de la vitesse la plus grande. Il peut aussi freiner si les deux vitesses tendent vers le 0. Essayez de répondre aux questions suivantes :
Q1) Comment peut-on empêcher le blocage des roues ? Empêcher que vitG=vitD=0 pour ne pas avoir un freinage brutal ;
Q2) Comment empêcher l'inversion brutale du sens de rotation des moteurs ?
Q3) comment éviter de casser la mécanique (il marche en avant et on lui demande d'aller directement en arrière son freinage en changeant ses deux vitesses en valeurs négatives) ?
Q4) Comment éviter un virage trop serré qui remettrait en cause la stabilité de la trajectoire et éviter un renversement.
Donnez le code java de la classe Robot en implémentant les solutions proposées comme réponses aux questions ci-dessus.

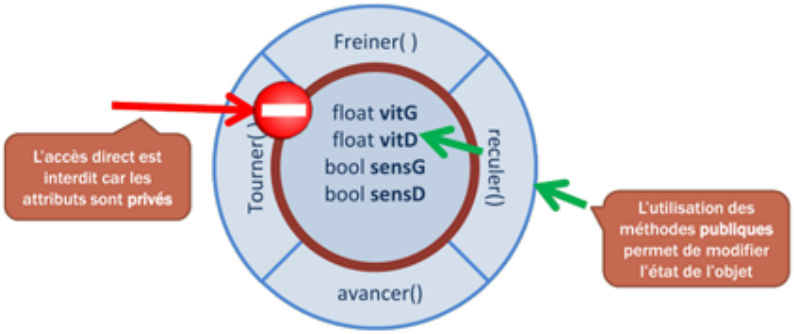

FIGURE 5.1 – Schéma du robot (données et méthodes)à programmer

Réponse : La solution consiste à empêcher la modification directe de l'état du robot, il faut utiliser plutôt les méthodes (avancer, tourner, ...) dont le développeur respectera les conditions imposées au mouvement du robot.
Les règles d'accès à l'état d'un objet sont indispensables pour éviter les mauvaises utilisations et sont implémentées grâce au mécanisme d'encapsulation.

Code java de la solution :

```
import java.io.*;     //pour la fontion sleep()
public class Robot {
//attributs :
    private double vitG, vitD;
    private boolean sensG, sensD;
    private float vitMax=100;
// méthodes :
    public Robot(){
        vitG = 0.1; vitD = 0.1;    //vitesse tende vers le 0
        sensG = true; sensD = true;
    }

    public void avancer(double v){
        if ((vitD<0)&& (vitG<0) )   //s'il marche en arrière ne peut pas avancer en avant
            System.out.println("vous devez freiner avant d'avancer en avant");
        else    {
            //vérifier si la vitesse d'avancement n'est pas négative ou supérieur à la vitesse max
            if(v>0 && v <vitMax ) {
                vitG = v; vitD = v;
                sensG = true; sensD= true
                System.out.println("j'avance");
            }else{
                System.out.println("je reste en état avancé");
            }
        }
    }
    public void tourner(boolean sensG, boolean sensD){
        if(sensG == sensD)
            System.out.println("vous devez spécifier le sens de direction pour tourner");
        else {
                this.sensD = sensD ;
                this.sensG = sensG ;
                if(sensG == true&&sensD == false) { // tourner vers la gauche
                    vitD = vitD - 1;  vitG = vitG + 1;   //vitesse gauche > v droite
                } else  {     // tourner vers la droite
                    vitD = vitD + 1;  vitG = vitG - 1;    //vitesse droite > v gauche
                }
        }
    }

    public void reculer (double v){
        //si les deux vitesses ne tendent pas vers le 0 on ne peut pas reculer
        if ((vitD != 0.1)&& (vitG != 0.1) )
            System.out.println("vous devez freiner avant de reculer en
```

```
    arrière");
45          else {
46                  // s'il est en état  arrière ou état freiner
47                  if(vitD<0 &&vitG< 0)||( vitD==0.1 &&vitG==0.1){
48                      if(v>0 && v <vitMax ) {
49                              //les deux vitesses doivent être négatives
50                              vitG = v* (-1); vitD = v*(-1);
51                              sensD = false; sensG = false;   // sensG et
     sensD doivent être false
52                              System.out.println("je recule");
53                      }else{
54                            System.out.println("je reste en état reculer"
    );
55                      }
56                  }
57          }
58      }
59      public void Freiner(){
60          if(vitG< 0)&&(vitD< 0) {    //dans le cas où il marche en arriè
    re et veut freiner
61              vitG = vitG*(-1); vitD = vitD*(-1);
62          }
63          //diminuer les vitesses progressivement (en 3 étapes) jusqu'à 0
64          double kG = vitG/3; vitG = vitG-kG;
65          double kD = vitD/3; vitD = vitD-kD;
66          // attendre 15 secondes en utilisant Thread.sleep(ms) (chiffre
    en millisecondes)
67          try{
68              Thread.sleep(15000);
69          }catch (Exception ex){ System.out.print(ex);}
70          vitG = vitG - kG; vitD = vitD-kD;
71          // attendre pendant une deuxième durée de 15 secondes
72          try{
73              Thread.sleep(15000);    // suspendre l'exécution 15s une
    deuxième fois
74          } catch (Exception ex){System.out.print(ex);}
75          vitG =  0.1 ; vitD =  0.1;    //tendent vers le 0
76          System.out.println("j'ai freiné");
77      }
78      publics static void main(String[] args) {
79          Robot R1=new Robot();
80          R1.avancer(20);
81          R1.reculer(10);
82          R1.Freiner();
83          R1.reculer(10);
84          R1.avancer(15);
85      }
86  }
```

Exercice 2.9

Une image est modélisée par son nombre de lignes (nl), de colonnes (nc) et une matrice (matc) contenant des entiers représentant les valeurs d'intensité de la couleur à chaque

case (i,j) de la matrice. Donnez le code de la classe Image en proposant un constructeur avec deux paramètres (lignes, colonnes).

```
public class Image {
    int nl,nc ;
    int [][] matc ;
    public Image(int nl,int nc) {
        this.nl=nl ;
        this.nc=nc ;
        matc=new int[nl][nc] ;
    }
}
```

Sachant que le nombre de lignes et de colonnes d'un objet image ne changent pas, proposez une encapsulation adéquate à ces attributs.

```
public class Image {
    private int nl,nc ;
    private int [][] matc ;
    public Image(int nl,int nc) {
        this.nl=nl ;
        this.nc=nc ;
        matc=new int[nl][nc] ;
    }
    public int getNl() {return nl ;}
    public int getNc() {return nc ;}
}
```

Un accesseur en modification (setter) pour l'attribut matc est défini par

```
    public void setMatc(int [][] m) { matc=m ; }
```

Discutez les risques liés à ce type d'accesseur et proposez un autre accesseur ne présentant pas les mêmes risques. **Réponse :** Les risques liés à ce type d'accesseur réside dans le fait que l'utilisateur remplace directement la matrice contenant les données de l'image par une autre et par conséquent il peut mettre des données erronées dans cette matrice, à titre d'exemple il peut mettre une matrice de dimension différente (nombre de lignes et de colonnes ne correspondent pas à nl et nc). Une solution consiste à donner l'accès à l'utilisateur de modifier juste une case de la matrice avec la nouvelle méthode :

```
public void setMat(int i,int j, int val) {
  if( (i*j !=0) && (i<nl) && (j<nc)) matc[i][j]=val ;
}
```

R c'est cette méthode qui est préconisée dans le standard JavaBeans pour l'encapsulation des attributs indexés (tableaux ...). L'accesseur Setter modifie chaque élément indexé séparément.

Exercice 2.10

Donnez le code Java de la classe TV avec :

Attributs :

- on :boolean
- chaine : int (>0 et <9999)
- volume int (>=0 et <=10)

Méthodes :

- void allumer() // mettre on à true
- void eteindre() // mettre on à false
- void setChaine(int ch) // change de chaine,
- void chaineUp() //aller à la chaine suivante,
- void chaineDown() // aller à la chaine précédente
- void volumeUp() // augmente le volume +1
- void volumeDown() //diminue le volume -1
- void reCall () // affiche la liste des 4 dernière chaines regardées

les différentes opérations ne doivent opérer que si la TV est allumée.

```
1  public class TV {
2      int chaine=1;
3      int volume=1;
4      boolean on=false;
5      int ch1=1,ch2=1,ch3=1,ch4=1 // pour la liste reCall;
6      // Les différentes initialisations étant faites, on n'a pas besoin
       d'un constructeur, le
7      // Compilateur insère automatiquement un constructeur vide
8      public void allumer() {if(!on) on=true;}
9      public void eteindre() {if(on) on=false;}
10     public void setChaine(int ch) {if(on && (chaine>=1) && (chaine
       <9999) && (ch!=chaine))
11         chaine=ch;
12         ch1=ch2;ch2=ch3;ch3=ch4;ch4=chaine;
13     }
14     public void chaineUp() {
15         if(on)
16             if(chaine<9999)
17                 chaine++;
18             else
19                 chaine=1;
20         ch1=ch2;ch2=ch3;ch3=ch4;ch4=chaine;
21     }
22     public void chaineDown() {
23         if(on)
24             if(chaine>1)
25                 chaine--;
26             else
27                 chaine=9999;
28          ch1=ch2;ch2=ch3;ch3=ch4;ch4=chaine;
29     }
30     public void volumeUp() {
31         if(on)
```

```
32              if(volume<10) volume++;
33      }
34      public void volumeDown() {
35          if(on)
36              if(volume>0)
37                  volume--;
38      }
39      public void reCall() {
40          System.out.println(ch4+","+ch3+","+ch2+","+ch1);
41      }
42  }
```

Exercice 3.1

Soit la classe Java Suivante :

```
1  public class Pixel {
2      int x ;
3      int y ;
4  }
```

1. Proposez un constructeur paramétré à la classe précédente
2. Encapsulez les attributs x et y de cette classe
3. A partir de la classe précédente, on veut construire la classe "ColoriedPixel" des pixels coloriés. Proposez une implémentation de cette classe avec son constructeur.

R La couleur est un tableau de 3 valeurs entières (dans le plan RGB).

Solution :

1

```
1  public class Pixel {
2  int x ;
3  int y ;
4  public Pixel (int x,int y) {
5      this.x=x ;
6      this.y=y ;
7    }
8  }
```

```
1  public class Pixel {
2  private int x ;
3  private int y ;
4  public Pixel (int x,int y) {
5      this.x=x ;
6      this.y=y ;
7    }
```

```
8  public void setX (int x) {this.x=x ;}
9  public int getX() {return this.x ;}
10 public void setY (int y) {this.y=y ;}
11 public int getY() {return this.y ;}
12 }
```

3

```
1  public class ColoriedPixel extends Pixel {
2  int color[]= {0,0,0} ;
3  public ColoriedPixel (int x,int y){
4    super(x,y) ; // appel au super-constructeur obligatoire cause :
       ce dernier est avec paramètres
5    for(i=0 ;i<color.length ;i++)
6       color[i]=0 ;
7  }
8  }
```

4. — Première variante :

```
1  public class ColoriedPixel extends Pixel {
2  int color[] ;
3  public ColoriedPixel (int x,int y){
4    super(x,y) ; // appel au super-constructeur obligatoire
       cause : ce dernier est avec paramètres
5    for(i=0 ;i<color.length ;i++)
6       color[i]=0 ;
7  }
8  public void setColor(int color[]) {this.color=color ;}
9  public int [] getColor() {return this.color}
10 }
```

Inconvénient : lorsqu'il s'agit d'un attribut indexé (contenant plusieurs éléments tels que les tableaux, listes ...etc) la manipulation de attribut en entier peut affecter le principe d'encapsulation comme dans l'exemple suivant :
Soit le code qui utilise la classe ColoriedPixel précédente :

```
1  ColoriedPixel cp=new ColoriesPixel(1,2) ;
2  int tc[]=cp.getColor() ;
3  tc[0]=100;// accès direct à l'attribut contraire au principe
     d'encapsulation
```

— Deuxième variante (solution qui respecte mieux le principe d'encapsulation) :

```
1  public class ColoriedPixel extends Pixel {
2  int color[] ;
3  public ColoriedPixel (int x,int y){
4    super(x,y) ; // appel au super-constructeur obligatoire
       cause : ce dernier est avec paramètres
5    for(i=0 ;i<color.length ;i++)
```

```
6         color[i]=0 ;
7   }
8   public void setColor(int index,int value) {if(index<this.
        color.length) this.color[index]=value ;}
9   public int getColor(int index) {if(index<this.color.length)
        return this.color[index] ;}
10  }
```

Exercice 3.2

— Donnez le code Java d'une classe nommé « Segment » contenant deux attributs encapsulés de type de la classe précédente ColoriedPixel (exercice 3.1).
— On veut construire maintenant la classe des rectangles de 2 manières :
  — La première par héritage de la classe Segment
  — La deuxième par composition (la classe comporte deux attributs objets de la classe Segment).

  Donnez le code des deux variantes.
— Quelle est la meilleure implémentation à votre avis.

1

```
1  public class Segment {
2      private ColoriedPixel px1 ;
3      private ColoriedPixel px2 ;
4      public Segment (int x1,int y1,int x2,int y2) {
5          px1=new ColoriedPixel(x1,y1) ;
6          px2=new ColoriedPixel(x2,y2) ;
7      }
8      public Segment (ColoriedPixel px1,ColoriedPixel px2) {
9          this.px1=px1 ;
10         this.px2=px2 ;
11     }
12     public void setPx1 (int x,int y,int  []color) {
13         px1.setX(x) ;px1.setY(y) ;
14         if (color.length==3)
15             for (int i=0 ;i<color.length ;i++)
16                 px1.setColor(i,color[i]) ;
17     }
18     public ColoriedPixel getPx1() {return this.px1 ;}
19     public void setPx2 (int x,int y,int color[]) {
20         px2.setX(x) ;px2.setY(y) ;
21         if (color.length==3)
22             for (int i=0 ;i<color.length ;i++)
23                 px2.setColor(i,color[i]) ;
24
25     }
26     public void ColoriedPixel getPx2() {return this.px2 ;}
27 }
```

2/
— 1ère Variante (Héritage) ;

```
1  public class Rectangle extends Segment {
2  // px1 et px2 sont hérités de la classe Segment
3  private ColoriedPixel px3 ;
4  private ColoriedPixel px4 ;
5  public Rectangle (ColoriedPixel px1,ColoriedPixel px2,
       ColoriedPixel px3,ColoriedPixel px4) {
6        super (px1,px2) // appel au superconstructeur pour
       initialiser les attributs px1 et px2
7        this.px3=px3 ;
8        this.px4=px4 ;
9  }
10 // à compléter par les accesseurs de px3 et px 4 de la même
       manière que dans Segment pour px1 // et px2
11 }
```

— 2eme variante (agrégation) :

```
1  public class Rectangle {
2  private Segment sg1 ;
3  private Segment sg2 ;
4  public Rectangle (ColoriedPixel px1,ColoriedPixel px2,
       ColoriedPixel px3,ColoriedPixel px4) {
5        sg1=new Segment (px1,px2) ;
6        sg2=new Segment (px3,px4) ;
7  }
8
9  // compléter par les accesseurs nécessaires pour modifier les
        différentes valeurs de sg1 et sg2
10 }
```

Exercice 3.3

Quels sont les avantages et les inconvénients des deux types d'invocation des méthodes statique et dynamique ?

Réponse :

	Avantages	**Inconvénients**
Liaison statique	exécution rapide, liaison faite au moment de l'édition de lien	- ne permet pas d'invoquer la méthode du type courant de l'objet
Liaison dynamique	permet l'invocation du type courant	exécution lente