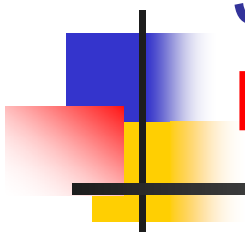


Java

License Profesionnelle CISI 2009-2010

Cours 13 : révisions

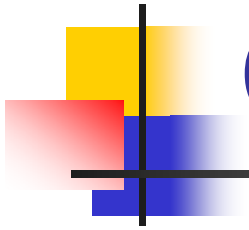
Classes et objets

- Exercice 1
  - On suppose que l'on dispose de la la classe A ainsi définie

```
class A
{void f (int n, float x) {…}
  public void g (byte b) {…}
  …
}
```

  - Soient ces déclarations
    - A a; int n; byte b; float x; double y;
    - Dire si les appels suivants sont corrects et sinon pourquoi ?

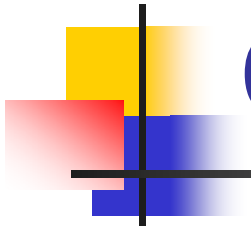
```
a.f (n, x) ;
a.f (b+3, x) ;
a.f (b, x) ;
a.f (n, y) ;
a.f (n, (float)y) ;
a.f (n, 2*x) ;
a.f (n+5, x+0.5) ;

a.g (b) ;
a.g (b+1) ;
a.g (b++) ;

a.g (3) ;
```



Classes et objets

Exercice 1 : solution

```
a.f (n, x) ;          // OK : appel normal
a.f (b+3, x) ;        // OK : b+3 est déjà de type int
a.f (b, x) ;          // OK : b de type byte sera converti en int
a.f (n, y) ;          // erreur : y de type double ne peut être converti en float
a.f (n, (float)y) ;   // OK
a.f (n, 2*x) ;        // OK : 2*x est de type float
a.f (n+5, x+0.5) ;    // erreur : 0.5 est de type double, donc x+0.5 est de
                      //  type double, lequel ne peut pas être converti en float
a.g (b) ;             // OK : appel normal
a.g (b+1) ;           // erreur : b1+1 de type int ne peut être converti en byte
a.g (b++) ;           // OK : b1++ est de type int
                      //  (mais peu conseillé : on a modifié la valeur de b1)
a.g (3) ;             // erreur : 3 de type int ne peut être convertie en byte
```



Classes et objets

- Exercice 2
  - Quelles erreurs figurent dans la définition de la classe suivante ?

```
class Surdef
{ public void f (int n)          {  }
  public int f (int p)           { return p ;}
  public void g (float x)        { }
  public void g (final double y) { }
  public void h (long n)         { }
  public int h (final long p)    { return 2 ; }
}
```

Classes et objets

- Exercice 2

```
class Surdef
{ public void f (int n) {  }
  public int f (int p) {return p ;}
  public void g (float x) { }
  public void g (final double y) { }
  public void h (long n) { }
  public int h (final long p) {
    return 2 ; }
}
```

- Erreurs
  - Les deux méthodes f ont des arguments de même type, d'où ambiguïté à la compilation
  - La sur-définition des méthodes g ne présente pas d'anomalie, leurs types étant de types différents
  - Les deux méthodes h ont des arguments de même type(long), le qualificatif final n'intervenant pas ici. La compilation signalera également une ambiguïté à ce niveau

Classes et objets

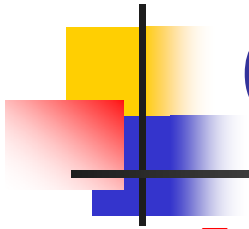
- Exercice 3

On dispose de la classe *Point* suivante permettant de manipuler des points d'un plan.

```
class Point
{ public Point (double x, double y)           { this.x = x ;  this.y = y ; }
  public void deplace (double dx, double dy) { x += dx ;     y += dy ;    }
  public void affiche ()
  { System.out.println ("coordonnees = " + x + " " + y ) ;
  }
  private double x, y ;
}
```

En ajoutant les fonctionnalités nécessaires à la classe *Point*, réaliser une classe *Segment* permettant de manipuler des segments d'un plan et disposant des méthodes suivantes :

```
segment (Point origine, Point extremite)
segment (double xOr, double yOr, double xExt, double yExt)
double longueur() ;
void deplaceOrigine (double dx, double dy)
void deplaceExtremite (double dx, double dy)
void affiche()
```



Classes et objets

- Exercice 3 : solution
  - Pour l'instant, la classe Point n'est dotée ni de méthodes d'accès aux champs x et y, ni de méthodes d'altération de leurs valeurs
  - Si l'on prévoit de représenter un segment par deux objets de type Point, il faudra manifestement pouvoir connaître et modifier leurs coordonnées pour pouvoir déplacer l'origine ou l'extrémité du segment
  - Pour se faire, on pourra par exemple ajouter à la classe Point, les quatre méthodes suivantes :

```
Public double getX ()
{return x;}
Public double getY()
{return x;}
Public void  setX(double x)
{this.x=x;}
Public void setY(double y)
{this.y=y;}
```

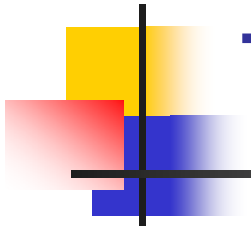
```
class Point
{ public Point (double x, double y)  { this.x = x ;  this.y = y ; }
  public void deplace (double dx, double dy) { x += dx ;     y += dy ;    }
  public double getX () { return x ; }
  public double getY () { return y ; }
  public void setX (double x) { this.x = x ; }
  public void setY (double y) { this.y = y ; }
  public void affiche ()
  { System.out.println ("coordonnees = " + x + " " + y ) ; }
  private double x, y ;
}
class Segment
{ public Segment (Point or, Point ext)
  { this.or = or ; this.ext = ext ;
  }
  public Segment (double xOr, double yOr, double xExt, double yExt)
  { or =  new Point (xOr, yOr) ;
    ext = new Point (xExt, yExt) ;
  }
  public double longueur()
  { double xOr =  or.getX(),  yOr =  or.getY() ;
    double xExt = ext.getX(), yExt = ext.getY() ;
    return Math.sqrt ( (xExt-xOr)*(xExt-xOr) + (yExt-yOr)*(yExt-yOr) ) ;
  }
  public void deplaceOrigine (double dx, double dy)
  { or.setX (or.getX() + dx) ;
    or.setY (or.getY() + dy) ;
  }
```

```
public void affiche ()
  { System.out.print ("Origine -   ") ; or.affiche() ;
    System.out.print ("Extremite - ") ; ext.affiche() ;
  }
  private Point or, ext ;
}
public class TstSeg
{ public static void main (String args[])
  { Point a = new Point(1, 3) ;
    Point b = new Point(4, 8) ;
    a.affiche() ; b.affiche() ;
    Segment s1 = new Segment (a, b) ;
    s1.affiche() ;
    s1.deplaceOrigine (2, 5) ;
    s1.affiche() ;
    Segment s2 = new Segment (3, 4, 5, 6) ;
    s2.affiche() ;
    System.out.println ("longeur = " + s2.longueur()) ;
    s2.deplaceExtremite (-2, -2) ;
    s2.affiche() ;
  }
}
```

Tableaux

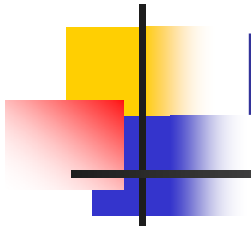
Quels résultats fournit le programme suivant ?

```
public class Tab2Ind1
{ public static void main (String args[])
  { int [] [] t = new int [3][] ;
    for (int i=0 ; i<3 ; i++)
     { t[i] = new int [i+1] ;
       for (int j=0 ; j<t[i].length ; j++)
         t[i][j] = i+j ;
     }

    for (int i=0 ; i<3 ; i++)
    { System.out.print ("tableau numero " + i + " = ") ;
      for (int j=0 ; j<t[i].length ; j++)
        System.out.print (t[i][j] + " ") ;
      System.out.println () ;
    }
  }
}
```



Tableaux

- Solution
  - In [ ][ ] t = new int[3] [ ];
    - crée un tableau de trois références à des tableaux d'entiers et place sa référence dans t
    - Pour l'instant, les références aux tableaux d'entiers sont initialisées à la valeur null
  - Pour chaque valeur de i,
    - t[i] = new int [i+1];
    - crée un tableau d'entiers de taille i+1 et en place la référence dans t[i]
  - L'instruction
    - t[i] [j] = i+j;
    - place des valeurs dans chacun des i+1 éléments de ce tableau
  - Les résultats sont
    - Tableau numero 0 = 0
    - Tableau numero 1 = 1 2
    - Tableau numero 2 = 2 3 4

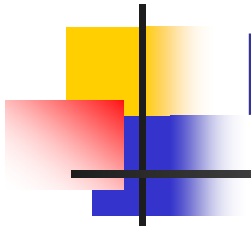


Héritage

- Exercice 1
  - On dispose de la classe suivante

```
Class Point
{   public void initialise (int x, int y) {this.x=x; this.y=y;}
    public void deplace(int dx, int dy){x +=dx; y +=dy;}
    public int getX(){return x;}
    public int getY(){return y;}
    private x,y;
}
```

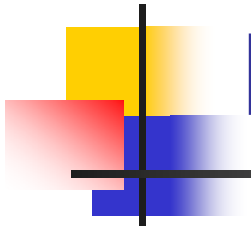
  - Réaliser une classe PointA, dérivée de Point disposant d'une méthode affiche affichant (en fenêtre console) les coordonnées d'un point
  - Écrire un programme utilisant les deux classes Point et PointA
  - Que se passerait-il si la classe Point ne disposait pas des méthodes getX et getY ?



Héritage

- Exercice 1 : solution
  - Il suffit de définir une classe dérivée en utilisant le mot clé extends. La méthode affiche comme toute méthode d'une classe dérivée a accès à tous les membres publics de la classe de base, donc en particulier à getX et getY

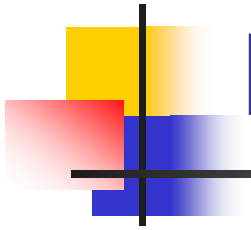
```
Class PointA extends Point
{void affiche()
{System.out.println("Coordonnees" ; + getX() + "  "  +
   getY());
}
}
```



Héritage

- On peut alors créer des objets de type PointA et leur appliquer aussi bien les méthodes publiques de PointA que celles de Point comme ce programme accompagné d'un exemple d'exécution

```
public class TsPointA
{   public static void main (String args[])
    {
            Point p = new Point();
            p.initialise(2,5);
            System.out.println("Coordonnees" ; + p.getX() + "  "  +
            p.getY());
            Point pa = new PointA();
            pa.initialise(1,8); //on utilise la méthode initialise de
            Point
            pa.affiche(); // et la méthode affiche de PointA
    }
}
```

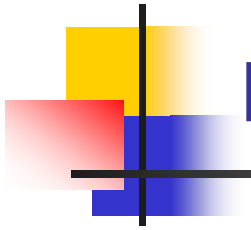


Héritage

- Notez bien qu'un appel tel que p.affiche() conduirait à une erreur de compilation puisque la classe de p (Point) ne possède pas de méthode affiche
- Si la classe Point n'avait pas disposé des méthodes d'accès getX et getY, il n'aurait pas été possible d'accéder à ses champs privés x et y depuis la classe PointA
- Il n'aurait donc pas été possible de la doter de la méthode affiche
- L'héritage ne permet pas de contourner le principe d'encapsulation
- Remarque
  - Comme nos classes ne disposent pas de constructeur, il est possible de créer des objets sans les initialiser
  - Dans ce cas, leurs champs auront simplement une valeur « nulle », c'est-à-dire ici la valeur entière 0

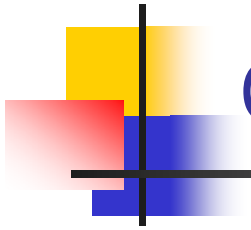
Quels résultats fournit ce programme ?

```
class A
{
  public A (int nn)
  { System.out.println ("Entree Constr A - n=" + n + " p=" + p) ;
    n = nn ;
    System.out.println ("Sortie Constr A - n=" + n + " p=" + p) ;
  }
  public int n ;   // ici, exceptionnellement, pas d'encapsulation
  public int p=10 ;
}
class B extends A
{ public B (int n, int pp)
  { super (n) ;
    System.out.println ("Entree Constr B - n=" + n + " p=" + p + " q=" + q) ;
    p = pp ;
    q = 2*n ;
    System.out.println ("Sortie Constr B - n=" + n + " p=" + p + " q=" + q) ;
  }
  public int q=25 ;
}
public class TstInit
{ public static void main (String args[])
  { A a = new A(5) ;
    B b = new B(5, 3) ;
  }
}
```



Dérivations successives et redéfinition

- Solution
  - Je suis un A
  - Je suis un A
  - Je suis un C
  - Je suis un D
  - Je suis un A
  - Je suis un C

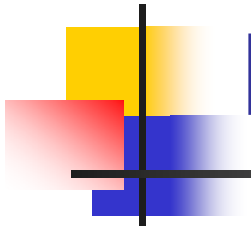


Classe String et chaînes de caractères

- Exercice
  - sur les arguments de la ligne de commandes
    - Écrire un programme qui récupère deux entiers sur la ligne de commande et qui en affiche la somme en fenêtre console
    - On vérifiera que les arguments fournis sont formés uniquement de chiffres (sans aucun signe); dans le cas contraire, le programme s'interrompra

- Solution

```
public class ArgLC2
{ public static void main (String args[])
  {
    int nbArgs = args.length ;
    if (nbArgs != 2) { System.out.println ("nombre arguments incorrect") ;
                       System.exit(-1) ;
                     }

    // on verifie que les caracteres de args[0] et args[1]
    //    sont bien des chiffres
    for (int i=0 ; i<2 ; i++)
      comp :  for (int j=0 ; j<args[i].length() ; j++)
        { for (int k=0 ; k<=9 ; k++)
            if (args[i].substring(j,j+1).equals(String.valueOf(k))) break comp ;
          System.out.println ("arguments pas tous numeriques") ;
          System.exit(-1) ;
        }

    int n1 = Integer.parseInt (args[0]) ;
    int n2 = Integer.parseInt (args[1]) ;
    System.out.println (n1 + " + " + n2 + " = " + (n1+n2)) ;
  }
}
```

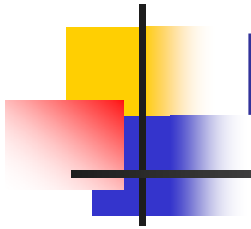


Les fichiers

- Exercice
  - Il s'agit de compter le nombre d'occurrences d'un mot donné dans un fichier donné
  - Le nom du fichier et le mot seront indiqués sur la ligne de commande

- Solution

```
import java.io.*;
import java.util.*;

class NbOcc
{
  public static void main (String[] argv) throws IOException
  {
    int nombre=0;
    String ligne;
    StringTokenizer st;
    String mot=new String(argv[1]);
    BufferedReader entree =new BufferedReader
  (new FileReader(argv[0]));

    while((ligne=entree.readLine())!=null)
      {
st=new StringTokenizer(ligne,".;() =[]");
while(st.hasMoreTokens())
   if (mot.equals(st.nextToken())) nombre++;
     }
    System.out.println("Le mot "+mot+" figure "+nombre+" fois");
  }
}
```



Les exceptions

- Exercice
  - Que produit le programme suivant lorsqu'on lui fournit en donnée
    - La valeur 0
    - La valeur 1
    - La valeur 2

```
class Except extends Exception
{ public Except (int n) { this.n = n ; }
  public int n ;
}
public class Chemin
{ public static void main (String
      args[])
  { int n ;
    System.out.print ("donnez un
      entier : ") ;
    n = Clavier.lireInt() ;
    try
    { System.out.println ("debut
      premier bloc try") ;
       if (n!=0) throw new Except (n) ;
       System.out.println ("fin premier
      bloc try") ;
    }
catch (Except e)
    { System.out.println ("catch 1 - n = " + e.n) ;
    }
    System.out.println ("suite du programme") ;
    try
    { System.out.println ("debut second bloc try") ;
      if (n!=1) throw new Except (n) ;
      System.out.println ("fin second bloc try") ;
    }
    catch (Except e)
    { System.out.println ("catch 2 - n = " + e.n) ;
      System.exit(-1) ;
    }
    System.out.println ("fin programme") ;
  }
}
```

```
donnez un entier : 0
debut premier bloc try
fin premier bloc try
suite du programme
debut second bloc try
catch 2 - n = 0

donnez un entier : 1
debut premier bloc try
catch 1 - n = 1
suite du programme
debut second bloc try
fin second bloc try
fin programme


donnez un entier : 2
debut premier bloc try
catch 1 - n = 2
suite du programme
debut second bloc try
catch 2 - n = 2
```