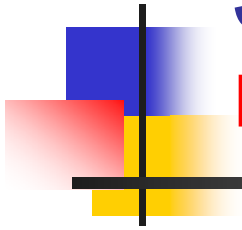
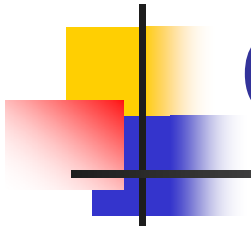


Java

Licence Professionnelle, CISI 2009-2010

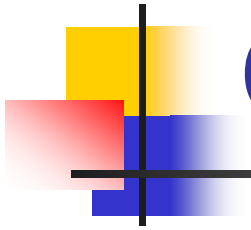
Cours 9 : Gestion des exceptions

Entree/Sortie



Gestion des exceptions

- Introduction : qu'est-ce qu'une exception ?
  - De nombreux langages de programmation de haut niveau possèdent un mécanisme permettant de gérer les erreurs qui peuvent intervenir lors de l'exécution d'un programme
  - Le mécanisme de gestion d'erreur le plus répandu est celui des exceptions
  - Le concept d'exception
    - Lorsqu'une fonction n'est pas définie pour certaines valeur de ses arguments on lance une exception
    - Par exemple
      - la fonction factorielle n'est pas définie pour les nombres négatifs :

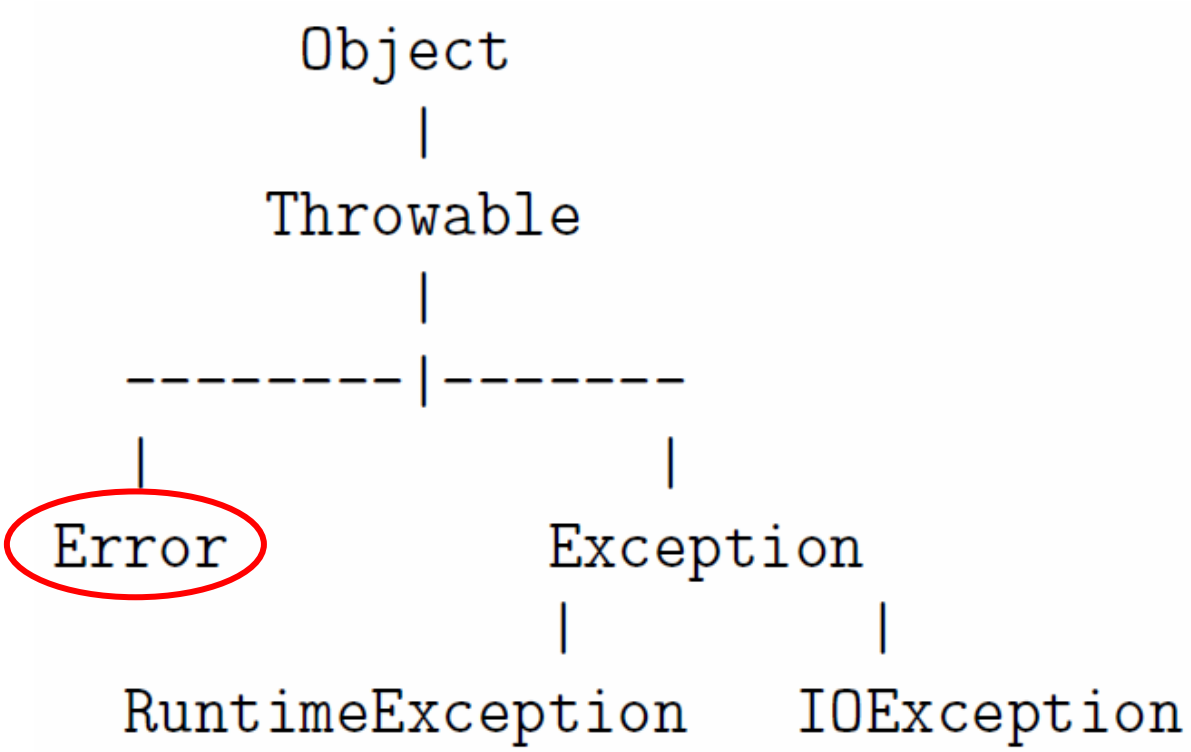


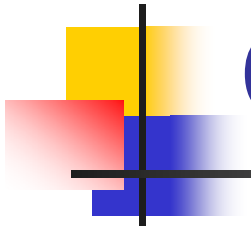
Gestion des exceptions

```
class Factorielle {
    static int factorielle(int n){
      int res = 1;
      if (n<0){
                throw new PasDefini();
      }
      for(int i = 1; i <= n; i++) {
                res = res * i;
      }
      return res;
    }
}
class PasDefini extends Error {}
```

Gestion des exceptions

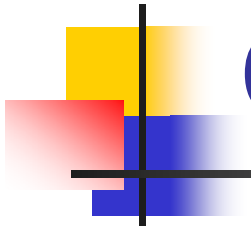
- Error : classe prédéfinie en Java pour traiter les exceptions





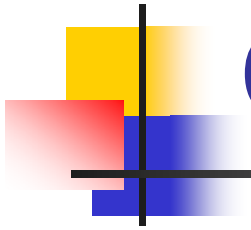
Gestion des exceptions

- Introduction : qu'est-ce qu'une exception ?
  - Une exception représente une erreur
    - Il est possible de lancer une exception pour signaler une erreur, comme lorsqu'un nombre négatif est passé en argument à la fonction factorielle
  - Jusqu'ici, lever une exception signifiait interrompre définitivement le programme avec un message d'erreur décrivant l'exception en question
  - Cependant, il est de nombreuses situations où le programmeur aimerait gérer les erreurs sans que le programme ne s'arrête définitivement
    - Il est alors important de pouvoir intervenir dans le cas où une exception a été levée



Gestion des exceptions

- Introduction : qu'est-ce qu'une exception ?
  - Les langages qui utilisent les exceptions possèdent toujours une construction syntaxique permettant de "rattraper" une exception, et d'exécuter un morceau de code avant de reprendre l'exécution normale du programme
  - En Java, les exceptions sont des objets représentant les erreurs qui peuvent survenir au cours de l'exécution d'un programme

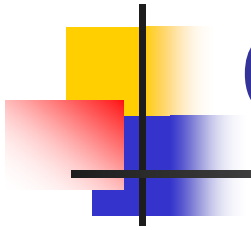


Gestion des exceptions

- Définir des exceptions
  - Afin de définir une nouvelle sorte d'exception, on crée une nouvelle classe en utilisant une déclaration de la forme suivante :

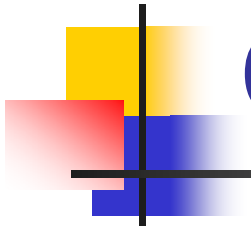
    class NouvelleException extends ExceptionDejaDefinie {}
  - NouvelleException est le nom de la classe d'exception que l'on désire définir en "étendant" ExceptionDejaDefinie qui est une classe d'exception déjà définie
  - Sachant que Error est prédéfinie en Java, la déclaration suivante définit bien une nouvelle classe d'exception PasDefini :

    class PasDefini extends Error {}



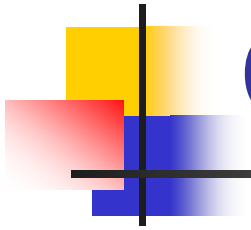
Gestion des exceptions

- Classes d'exception prédéfinies en Java
  - Trois catégories :
    1. Celles définies en étendant la classe Error :
       - représentent des erreurs critiques qui ne sont pas censée être gérées en temps normal
       - Par exemple, une exception de type OutOfMemoryError est lancée lorsqu'il n'y a plus de mémoire disponible dans le système
    2. Celles définies en étendant la classe Exception :
       - représentent les erreurs qui doivent normalement être gérées par le programme
       - Par exemple, une exception de type IOException est lancée en cas d'erreur lors de la lecture d'un fichier



Gestion des exceptions

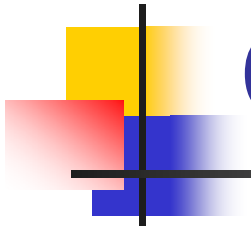
3. Celles définies en étendant la classe RuntimeException
   - représentent des erreurs pouvant éventuellement être gérée par le programme
   - L'exemple typique de ce genre d'exception est NullPointerException, qui est lancée si l'on tente d'accéder au contenu d'un tableau ou d'un objet qui vaut null



Gestion des exceptions

- Lancer une exception
  - Lorsque l'on veut lancer une exception, on utilise le mot clé throw suivi de l'exception à lancer
  - Cette exception doit être d'abord créée avec new NomException()
    - de la même manière que lorsque l'on crée un nouvel objet par un appel à l'un des constructeurs de sa classe
  - Ainsi lancer une exception de la classe PasDefini s'écrit :

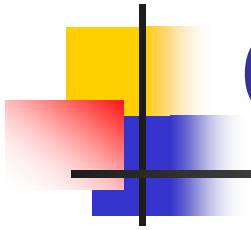
```
throw new PasDefini();
```



Gestion des exceptions

- Lancer une exception (suite)
  - Lorsqu'une exception est levée, l'exécution normale du programme s'arrête et on saute toutes les instructions jusqu'à ce que l'exception soit rattrapée ou jusqu'à ce que l'on sorte du programme
  - Exemple : Arret.java

```
public class Arret {
public static void main(String [] args) {
    System.out.println("Coucou 1"); // 1
    if (true) throw new Stop();
    System.out.println("Coucou 2"); // 2
    System.out.println("Coucou 3"); // 3
    System.out.println("Coucou 4"); // 4
}
}
class Stop extends RuntimeException {}
```



Gestion des exceptions

- Lancer une exception (suite)
  - l'exécution de la commande java Stop produira l'affichage suivant :
    - > java Arret
      - Coucou 1
      - Exception in thread "main" Stop at Arret.main(Arret.java:5)
    - C.à.d que les instructions 2, 3 et 4 n'ont pas été exécutées
    - Le programme se termine en indiquant que l'exception Stop lancée dans la méthode main à la ligne 5 du fichier Arret.java n'a pas été rattrapée

Rattraper une exception : la construction try catch

- Le rattrapage d'une exception en Java se fait en utilisant la construction

```
try {
... // 1
} catch (UneException e) {
... // 2
}
.. // 3
```

- Le code 1 est normalement exécuté
- Si une exception est lancée lors de cette exécution, les instructions restantes dans le code 1 sont sautées
- Si la classe de l'exception est UneException, alors le code 2 est exécuté
- Dans le code 2, on peut faire référence à l'exception en utilisant le nom donné à celle-ci après sa classe (ici le nom est e)
- Si la classe de l'exception n'est pas UneException, le code 2 et le code 3 sont sautés

Gestion des exceptions

- Ainsi, le programme suivant : Arret2.java

```
public class Arret2 {
    public static void main(String [] args) {
        try {
            System.out.println ("Coucou 1"); // 1
            if (true) throw new Stop();
            System.out.println ("Coucou 2"); // 2
        } catch (Stop e) {
            System.out.println ("Coucou 3"); // 3
            }
            System.out.println ("Coucou 4"); // 4
    }
}
class Stop extends RuntimeException {}
```

```
> java Arret
    Coucou 1
    Coucou 3
    Coucou 4
```

Gestion des exceptions

- En revanche le programme suivant, dans lequel on lance l'exception Stop2 : Arret3.java

```
public class Arret3 {
    public static void main(String [] args) {
        try {
            System.out.println ("Coucou 1"); // 1
            if (true) throw new Stop2();
            System.out.println ("Coucou 2"); // 2
        } catch (Stop e) {
            System.out.println ("Coucou 3"); // 3
        }
        System.out.println ("Coucou 4"); // 4
    }
}
```

```
> java Arret
Coucou 1
Exception in thread "main" Stop2 at Arret.main(Arret.java:5)
```

Rattraper plusieurs exceptions

- Il est possible de rattraper plusieurs types d'exceptions en enchaînant les constructions catch : Arret4.java

```
public class Arret4 {
    public static void main(String [] args) {
        try {     System.out.println("Coucou 1"); //
1
                if (true) throw new Stop2();
                 System.out.println("Coucou 2"); //
2
        } catch (Stop e) {
                 System.out.println("Coucou 3"); //
3
        } catch (Stop2 e) {
```

```
> java Arret
Coucou 1
Coucou 3 bis
Coucou 4
```

Un autre exemple : Except2.java

```
class Point2
{ public Point2(int x, int y) throws ErrConst
  { if ( (x<0) || (y<0)) throw new ErrConst() ;
    this.x = x ; this.y = y ;
  }
  public void deplace (int dx, int dy) throws
        ErrDepl
  { if ( ((x+dx)<0) || ((y+dy)<0)) throw new
        ErrDepl() ;
    x += dx ; y += dy ;
  }
  public void affiche()
  { System.out.println ("coordonnees : " + x + " "
        + y) ;
  }
  private int x, y ;
}
class ErrConst extends Exception
{ }
class ErrDepl extends Exception
{ }
```

```
public class Except2
{ public static void main (String args[])
  { try
    { Point2 a = new Point2 (1, 4) ;
      a.affiche() ;
      a.deplace (-3, 5) ;
      a = new Point2 (-3, 5) ;
      a.affiche() ;
    }
    catch (ErrConst e)
    { System.out.println ("Erreur construction ") ;
      System.exit (-1) ;
    }
    catch (ErrDepl e)
    { System.out.println ("Erreur deplacement ") ;
      System.exit (-1) ;
    }
  }
}
```

- Il est également possible d'imbriquer les constructions try catch :

```
public class Arret5 {
    public static void main(String [] args) {
        try {
            try {
                System.out.println("Coucou 1"); // 1
                if (true) throw new Stop();
                System.out.println("Coucou 2"); // 2
            }
            catch (Stop e) {
                System.out.println("Coucou 3"); // 3
            }
            System.out.println("Coucou 4"); // 4
        }
        catch (Stop2 e) {
            System.out.println("Coucou 5"); // 5
        }
        System.out.println("Coucou 6"); // 6
    }

}
class Stop extends RuntimeException {}
class Stop2 extends RuntimeException {}
```

```
> java Arret
Coucou 1
Coucou 3
Coucou 4
Coucou 6
```

En remplaçant throw new Stop() par throw new Stop2(), on obtient :

```
> java Arret
Coucou 1
Coucou 5
Coucou 6
```

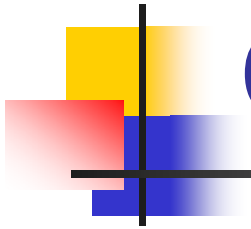
Exceptions et méthodes

- Exception non rattrapée dans le corps d'une méthode
  - Si une exception lancée lors de l'exécution d'une méthode n'est pas rattrapée, elle "continue son trajet" à partir de l'appel de la méthode
  - Même si la méthode est sensée renvoyer une valeur, elle ne le fait pas :

```
public class Arret6 {
    static int lance(int x) {
      if (x < 0) { throw new Stop();
                   }
      return x;
    }
    public static void main(String [] args) {
      int y = 0;
      try {System.out.println("Coucou 1");
           y = lance(-2);
       System.out.println("Coucou 2");
      } catch (Stop e) {
       System.out.println("Coucou 3");
      }
       System.out.println("y vaut "+y);
    }
}
class Stop extends RuntimeException {}
```

A l'exécution on obtient :

```
> java Arret
Coucou 1
Coucou 3
y vaut 0
```



Gestion des exceptions

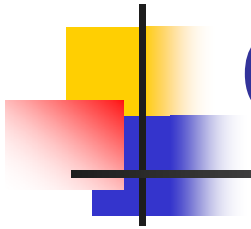
- Exceptions et méthodes
  - Déclaration throws
    - Le langage Java demande à ce que soient déclarées les exceptions qui peuvent être lancées dans une méthode sans être rattrapées dans cette même méthode
    - Cette déclaration est de la forme
      - throws Exception1, Exception2, ...
    - et se place entre les arguments de la méthode et l'accolade ouvrante marquant le début du corps de la méthode
    - Cette déclaration n'est pas obligatoire pour les exceptions de la catégorie Error ni pour celles de la catégorie RuntimeException

Gestion des exceptions

- Exceptions et méthodes
  - Exemple : Factorielle.java

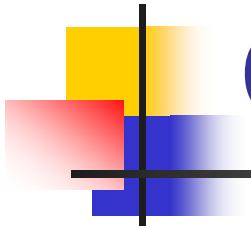
```
class Factorielle {
static int factorielle(int n)
    throws PasDefini { // (1)
int res = 1;
if (n<0){
throw new PasDefini(); // (2)
}
for(int i = 1; i <= n; i++) {
res = res * i;
}
return res;
}
```

```
public static void main (String []
    args) {
int x;
System.out.println("Entrez un
    nombre (petit):");
x = Lire.i();
try { // (3)
System.out.println(factorielle(x));
} catch (PasDefini e) { // (3 bis)
System.out.println("La factorielle de
    "
+x+" n'est pas d´efinie !");
}}}
class PasDefini extends Exception
    {} // (4)
```



Gestion des exceptions

- Exceptions et méthodes
  - Exemple résumé : factorielle
    - Dans ce programme, on définit une nouvelle classe d'exception PasDefini au point (4)
    - Cette exception est lancée par l'instruction throw au point (2) lorsque l'argument de la méthode est négatif
    - Dans ce cas l'exception n'est pas rattrapée dans le corps et comme elle n'est ni dans la catégorie Error ni dans la catégorie RuntimeException,
      - on la déclare comme pouvant être lancée par factorielle, en utilisant la déclaration throws au point (1)
    - Si l'exception PasDefini est lancée lors de l'appel à factorielle, elle est rattrapée au niveau de la construction try catch des points (3) (3 bis) et un message indiquant que la factorielle du nombre entré n'est pas définie est alors affiché



Gestion des exceptions

- Exceptions et méthodes
  - Exemple résumé : factorielle
    - Voici deux exécutions du programme avec des valeurs différentes pour x (l'exception est lancée puis rattrapée lors de la deuxième exécution) :

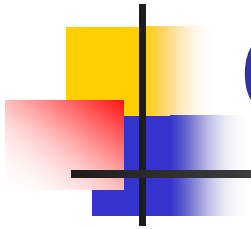
```
> java Factorielle
Entrez un nombre (petit):4
24
> java Factorielle
Entrez un nombre (petit):-3
La factorielle de -3 n'est pas définie !
```

Autre exemple

```
class Point3
{ public Point3(int x, int y) throws ErrConst
  { if ( (x<0) || (y<0)) throw new ErrConst(x, y) ;
    this.x = x ; this.y = y ;
  }
  public void affiche()
  { System.out.println ("coordonnees : " + x + " " + y) ;
  }
  private int x, y ;
}
class ErrConst extends Exception
{ ErrConst (int abs, int ord)
  { this.abs = abs ; this.ord = ord ;
  }
  public int abs, ord ;
}
```

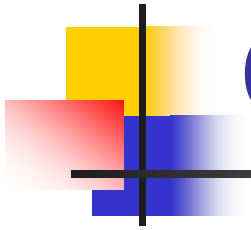
Autre exemple

```
public class Exinfo1
{ public static void main (String args[])
  { try
    { Point3 a = new Point3 (1, 4) ;
      a.affiche() ;
      a = new Point3 (-3, 5) ;
      a.affiche() ;
    }
    catch (ErrConst e)
    { System.out.println ("Erreur construction Point") ;
      System.out.println ("  coordonnees souhaitees : " + e.abs + " "
       + e.ord) ;
      System.exit (-1) ;
    }
  }
}
```



Gestion des exceptions

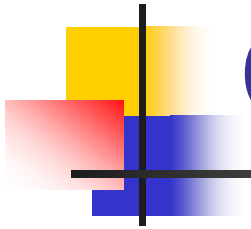
- Le boc **finally**
  - Nous avons vu que le déclenchemnt d'une exception provoque un branchement inconditionnel au gestionnaire, à quelque niveau qu'il se trouve
  - L'exécution se poursuit avec les instructions suivant ce gestionnaire
  - Cependant, Java permet d'introduire, à la suite d'un bloc try, un bloc particulier d'instructions qui seront toujours exécutées :
    - Soit après la fin naturelle du bloc try, si aucune exception n'a été déclenchée
    - Soit après le gestionnaire d'exception
  - Ce bloc est introduit par le mot clé finally



Gestion des exceptions

- Exemple :

```
public class PropagationException {
public static void main(String[] args) {
 try{ Object chaine="bonjour";
     Integer i=(Integer)chaine;//levée d'une ClassCastException
     System.out.println("fin du programme");
     }
     finally{ System.out.println("on passe par le bloc finally");
}
}
}
```

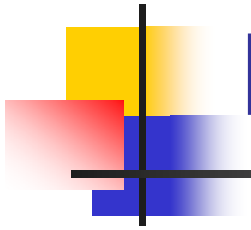


Gestion des exceptions

- Annexe : quelques exceptions prédéfinies
  - NullPointerException :
    - accès à un champ ou appel de méthode non statique sur un objetvalant null
    - Utilisation de length ou accès à un case d'un tableau valant null
  - ArrayIndexOutOfBoundsException :
    - accès à une case inexistante dans un tableau.
  - ArrayIndexOutOfBoundsException :
    - accès au ième caractère d'une chaîne de caractères de taille inférieure à i
  - ArrayIndexOutOfBoundsException :
    - création d'un tableau de taille négative
  - NumberFormatException :
    - erreur lors de la conversion d'un chaîne de caractères en nombre
  - La classe Terminal utilise également l'exception TerminalException pour signaler des erreurs

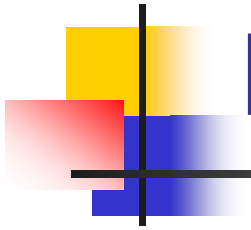
Exemple : ExcStd1.java

```
public class ExcStd1
{ public static void main (String args[])
  { try
    { int t[] ;
      System.out.print ("taille voulue : ") ;
      int n = Clavier.lireInt() ;
      t = new int[n] ;
      System.out.print ("indice : ") ;
      int i = Clavier.lireInt() ;
      t[i] = 12 ;
      System.out.println ("*** fin normale") ;
    }
    catch (NegativeArraySizeException e)
    { System.out.println ("Exception taille tableau negative : " +
      e.getMessage() ) ;
    }
    catch (ArrayIndexOutOfBoundsException e)
    { System.out.println ("Exception indice tableau : " + e.getMessage() ) ;
    }
  }
}
```



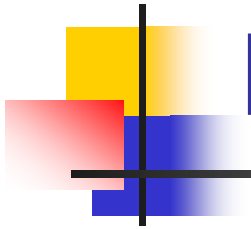
Entrée/Sortie

- Introduction
  - En Java, les échanges de données entre un programme et l'extérieur (autre programme, fichier, réseau) sont effectués par un « flot ou flux (streams en anglais) » de données
  - Un flot permet de transporter séquentiellement des données
  - Les données sont transportées une par une (ou groupe par groupe), de la première à la dernière donnée



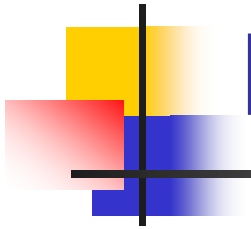
Entrée/Sortie

- Flux de sortie
  - Précédemment, quand on écrivait :
    - System.out.println
  - En fait out est ce que l'on nomme un « flux » de sortie
  - En Java, la notion de flux est générale, puisqu'un flux de sortie désigne n'importe quel canal susceptible de recevoir de l'information sous forme d'une suite d'octets
  - Il peut s'agir d'un canal de sortie (ex. out) mais il peut également s'agir d'un fichier ou encore d'une connexion distante…



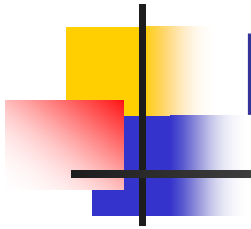
Entrée/Sortie

- Flux d'entrée
  - De manière équivalente, il existe des flux d'entrée, c.à.d des canaux délivrant de l'information sous forme d'une suite d'octets
  - Là encore, il peut s'agir d'un périphérique de saisie (clavier), d'un fichier, d'une connexion…



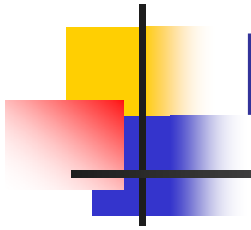
Entrée/Sortie

- Flux binaires et flux texte
  - Java distingue les flux binaires des flux texte
    - Dans le premier cas, l'information est transmise sans modification de la mémoire au flux ou du flux à la mémoire
    - Dans le second cas, en revanche, l'information subit une transformation, nommée formatage, de manière à ce que le flux transmette une suite de caractères



Entrée/Sortie

- Création séquentielle d'un fichier binaire
  - Exemple
    - Écrire un programme qui enregistre dans un fichier binaire différents nombres entiers fournis par l'utilisateur au clavier
    - La classe abstraite outputStream sert de base à toutes les classes relatives à des flux binaires de sortie
    - La classe FileOutputStream, dérivée de OutputStream, permet de manipuler un flux binaire associé à un fichier en écriture

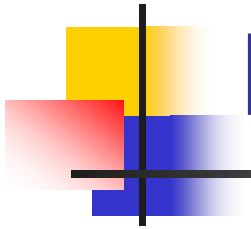


Entrée/Sortie

- L’un des constructeurs s’utilise comme suit

```
FileOutputStream f = new
    FileOutputStream("entier.dat");
```

- Cette instruction associe l’objet f à un fichier de nom entiers.dat
- Si ce fichier n’existe pas, il est alors créé
- On a affaire à une opération d’ouverture de fichier en écriture

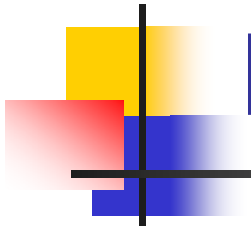


Entrée/Sortie

- La classe DataOutputStream
  - Cette classe comporte des méthodes plus évoluées
  - Elle a un constructeur qui reçoit en argument un objet de type FileOutputStream
    - DataOutputStream sortie = new DataOutputStream (f)
  - Crée un objet de sortie qui, par l'intermédiaire de l'objet f, se trouve associé au fichier entiers.data
  - Cette classe dispose de méthodes permettant d'envoyer sur un flux une valeur d'un type primitif quelconque
    - writeInt(pour int), writeFloat(pour Float)…

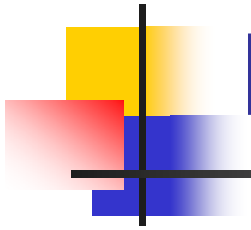
Exemple : Crsfic1.java

```
import java.io.* ;
public class Crsfic1
{ public static void main (String args[]) throws IOException
  {
    String nomfich ;
    int n ;
    System.out.print ("donnez le nom du fichier a creer : ") ;
    nomfich = Lire.S() ;
    DataOutputStream sortie = new DataOutputStream
                          ( new FileOutputStream (nomfich)) ;
    do { System.out.print ("donnez un entier : ") ;
         n = Lire.i() ;
         if (n != 0)
          { sortie. writeInt (n)  ;
          }
       }
    while (n != 0) ;
    sortie.close () ;
    System.out.println ("*** fin creation fichier ***");
  }
}
```



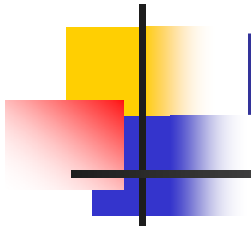
Fichier binaire

- Remarque
  - Notez l’instruction
    - sortie.close();
  - Elle ferme le flux



Fichier binaire

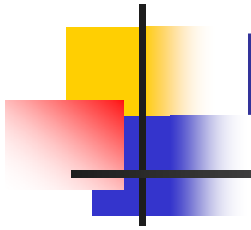
- Lecture séquentielle
  - Par analogie à ce qu'on a vu, la classe FileInputStream, dérivée de InputStream, peut être utilisée pour la lecture d'un fichier binaire
  - L'un de ses constructeurs est :
    - FileInputStream f = new FileInputStream("entier.dat");
  - Il permet d'ouvrir entier.dat en lecture
    - Comme précédemment, on utilisera DataInputStream qui possède des méthodes plus évoluées



Fichier binaire

- Exemple : Lecsfic1.java
  - Voici un programme qui relit un fichier binaire d'entiers du type créé précédemment

```
import javax.swing.* ;  // pour showInputDialog
import java.io.* ;
public class Lecsfic1
{ public static void main (String args[]) throws IOException
  { String nomfich ;
    int n = 0 ;

    System.out.print ("donnez le nom du fichier a lister : ") ;
    nomfich = Lire.S() ;
    DataInputStream entree = new DataInputStream
                        ( new FileInputStream (nomfich)) ;
    System.out.println ("valeurs lues dans le fichier " + nomfich + " :") ;
    boolean eof = false ;  // sera mis a true par exception EOFile
    while (!eof)
    { try
      {  n = entree.readInt ()  ;
      }
      catch (EOFException e)
      { eof = true ;
      }
      if (!eof) System.out.println (n) ;
    }
    entree.close () ;
    System.out.println ("*** fin liste fichier ***");
  }
}
```



Fichier binaire

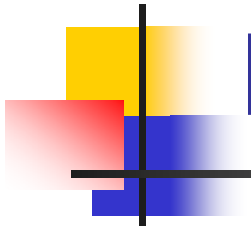
- Accès direct
  - Possible grâce à la clase RandomAccessFile qui contient les méthodes readInt, readFloat, writeInt, writeFloat…
  - Utilisation

```
RandomAccessFile entree = new RandomAccessFile
  ("entier.dat", "r");
```

    - « r » pour l'ouverture en lecture, mais on trouve « rw » pour la lecture-écriture
    - Par ailleurs, RandomAccessFile dispose d'une fonction utile, seek, pour agir sur le pointeur de fichier, une sorte de rang du prochain octet à lire.

Exemple d’accès à un fichier existant

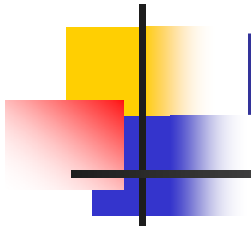
```
import java.io.* ;
public class Accdir
{ public static void main (String args[]) throws IOException
  {
    String nomfich ;
    int n, num ;
    RandomAccessFile entree ;
    System.out.print ("donnez le nom du fichier a consulter : ") ;
    nomfich = Lire.S() ;
    entree = new RandomAccessFile (nomfich, "r") ;

    do
     { System.out.print ("Numero de l'entier recherche : ") ;
       num = Lire.i() ;
       if (num == 0) break ;
       entree.seek (4*(num-1)) ;
       n = entree.readInt() ;
       System.out.println ("  valeur = " + n) ;
    }
    while (num != 0) ;

    entree.close () ;
    System.out.println ("*** fin consultation fichier ***");
  }
}
```



Fichier Texte

- La classe Writer-FileWriter
  - Elle sert de base à toutes les classes relatives à un flux texte de sortie
  - La clase FileWriter, dérivée de Writer, permet de manipuler un fichier texte

    FileWriter f = new FileWriter("carres.txt");
    - Cette opération associe f à un fichier texte de nom carres.txt.
    - S'il n'existe pas, il est créé (vide)
    - S'il existe, son ancien contenu est détruit
  - C'est donc une opération d'ouverture en écriture
  - Les méthodes de la classe FileWriter permettent d'écrire des caractères, des tableaux de caractères, des chaînes…



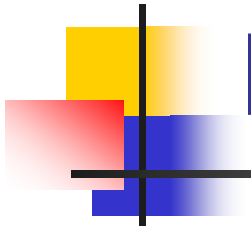
Fichier Texte

- La classe PrintWriter
  - Pour écrire dans le fichier texte, on se sert de la classe PrintWriter

    PrintWriter sortie = new PrintWriter(f);
  - Crée un objet sortie associé au fichier f
  - La classe PrintWriter contient les méthodes :
    - print et println

Exemple : Crftxt1.java

```
import java.io.* ;
public class Crftxt1
{ public static void main (String args[]) throws IOException
  {
    String nomfich ;
    int n ;
    System.out.print ("Donnez le nom du fichier a creer : ") ;
    nomfich = Lire.S() ;
    PrintWriter sortie = new PrintWriter (new FileWriter (nomfich)) ;

    do
     { System.out.print ("donnez un entier : ") ;
       n = Lire.i() ;
       if (n != 0)
        { sortie.println (n + " a pour carre " + n*n)  ;
        }
     }
    while (n != 0) ;
    sortie.close () ;
    System.out.println ("*** fin creation fichier ***");
  }
}
```



Fichier texte

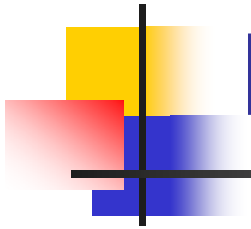
- Accès aux lignes
  - On utilise la classe FileReader en la couplant avec la classe BufferedReader qui dispose d'une méthode readLine
  - Avec FileReader seule, on ne peut lire que des caractères
  - On crée un objet entree de la façon suivante

```
BufferedReader entree = new
  BufferedReader (new
  FileReader("carres.txt"));
```

  - La méthode readLine de la classe BufferedReader fournit une référence à une chaîne correspondant à une ligne du fichier

Exemple : Lecftxt1.java

```
import java.io.* ;
public class Lecftxt1
{ public static void main (String args[]) throws IOException
  {
    String nomfich ;
    String ligne ;
    int n ;
    System.out.print ("Donnez le nom du fichier a lister : ") ;
    nomfich = Clavier.lireString() ;
    BufferedReader entree = new BufferedReader (new FileReader
     (nomfich)) ;

    do
      { ligne = entree.readLine() ;
        if (ligne != null) System.out.println (ligne) ;
      }
    while (ligne != null) ;
    entree.close () ;
    System.out.println ("*** fin liste fichier ***");
  }
}
```



Fichier Texte

- La classe StringTokenizer
  - Permet de découper une chaîne en différents tokens (chaînes), en se fondant sur des caractères de séparation qu'on choisit librement
  - Exemple
    - Soit un fichier reels.txt composé de réels séparés par des blancs
    - On voudrait lire le fichier ligne par ligne et récupérer ces réels par ligne
    - On va construire à partir de la ligne lue (ici ligneLue) un objet de type StringTokenizer
    - StringTokenizer tok = new StringTokenizer (ligneLue, " ");
    - On utilise ensuite :
      - counToken pour compter le nombre de tokens
      - nexToken qui fournit le token suivant

Exemple : Lectxt3.java

```
import java.io.* ;
import java.util.* ;    // pour StringTokenizer
public class Lectxt3
{ public static void main (String args[]) throws IOException
  { String nomfich ;
    double x, som = 0. ;
    System.out.print ("donnez le nom du fichier a lister : ") ;
    nomfich = Clavier.lireString() ;
    BufferedReader entree = new BufferedReader (new FileReader (nomfich)) ;

    System.out.println ("Flottants contenus dans le fichier " + nomfich + " :") ;

    while (true)
     { String ligneLue = entree.readLine() ;
          if (ligneLue == null) break ;
       StringTokenizer tok = new StringTokenizer (ligneLue, " ") ;  // espace separateur
       int nv = tok.countTokens() ;
       for (int i=0 ; i<nv ; i++)
       { x = Double.parseDouble (tok.nextToken()) ;
         som += x ;
         System.out.println (x + " ") ;
       }
      }
    entree.close () ;
    System.out.println ("somme des nombres = " + som) ;
    System.out.println ("*** fin liste fichier ***");
  }
}
```

Autre exemple : Lectxt2.java

```
import java.io.* ;
import java.util.* ;    // pour StringTokenizer
public class Lectxt2
{ public static void main (String args[]) throws IOException
  { String nomfich ;
    int nombre, carre ;
    System.out.print ("donnez le nom du fichier a lister : ") ;
    nomfich = Clavier.lireString() ;
    BufferedReader entree = new BufferedReader (new FileReader (nomfich)) ;

    System.out.println ("Nombres et carres contenus dans ce fichier") ;
    while (true)
     { String ligneLue = entree.readLine() ;
           if (ligneLue == null) break ;
       StringTokenizer tok = new StringTokenizer (ligneLue, " ") ;
       nombre = Integer.parseInt (tok.nextToken()) ;
       for (int i=0 ; i<3 ; i++)  tok.nextToken() ;  // pour sauter : a pour carre
       carre = Integer.parseInt (tok.nextToken()) ;
       System.out.println (nombre + " " + carre) ;
     }
    entree.close () ;
    System.out.println ("*** fin liste fichier ***");
 }
}
```