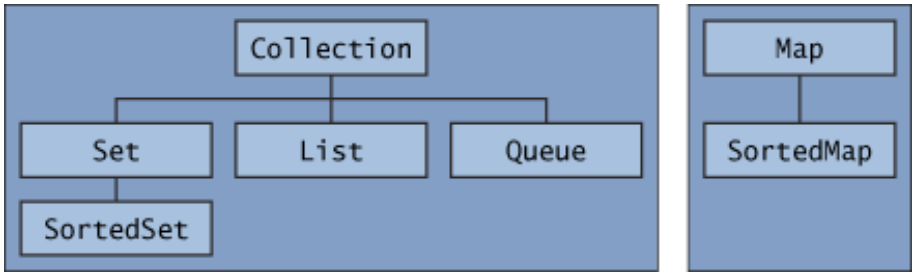
Cours 9

Collections

Collections

- types de données
  - interfaces
  - implémentations
  - algorithmes
- Interfaces:



Collections: les interfaces

Les collections sont des interfaces génériques

- Collection<E>: add, remove size toArray…
  - Set<E>: éléments sans duplication
    - SortedSet<E>: ensembles ordonnés
  - List<E>: des listes éléments non ordonnés et avec duplication
  - Queue<E>: files avec tête: peek, poll (défiler), offer (enfiler)
- Map<K,V>:association clés valeurs
- SortedMap<K,V> avec clés triées

Certaines méthodes sont **optionnelles** (si elles ne sont pas implémentées UnsupportedOperationException).

En plus:

- Iterator<E>: interface qui retourne successivement les éléments next (), hasNext(), remove()
- ListIterator<E>: itérateur pour des List, set(E) previous, add(E)

Collection

```
public interface Collection<E> extends Iterable<E> {
    // operations de base
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);          //optionnel
    boolean remove(Object element); //optionnel
    Iterator<E> iterator();

    // operations des collections
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optionnel
    boolean removeAll(Collection<?> c);        //optionnel
    boolean retainAll(Collection<?> c);        //optionnel
    void clear();                              //optionnel

    // Array
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Collection

- Les collections sont *génériques*
- Parcours:
  - On peut parcourir les éléments par « for »:

```
for (Object o : collection)
    System.out.println(o);
```

  - Ou avec un Iterator:

```
static void filter(Collection<?> c) {
    for (Iterator<?> it = c.iterator();
   it.hasNext();)
        if (!cond(it.next()))
            it.remove();
}
```

Collection

- On peut convertir une collection en tableau
  - En tableaux de Object
  - En tableaux d'objet du type paramètre de la collection
- Il existe aussi une **classe** Collections qui contient des méthodes statiques utiles

Set

- Interface pour contenir des objets différents
  - Opérations ensemblistes
  - SortedSet pour des ensembles ordonnés
- Implémentations:
  - HashSet par hachage (preformances)
  - TreeSet arbre rouge-noir
  - LinkedHashSet ordonnés par ordre d'insertion

Set

```
public interface Set<E> extends Collection<E> {
    // Basic operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);          //optional
    boolean remove(Object element); //optional
    Iterator<E> iterator();

    // Bulk operations
    boolean containsAll(Collection<?> c); // sous-ensemble
    boolean addAll(Collection<? extends E> c); //optionnel- union
    boolean removeAll(Collection<?> c);        //optionnel- différence
    boolean retainAll(Collection<?> c);        //optionnel- intersection
    void clear();                              //optionnel

    // Array Operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Exemple:

```
public static void chercheDoublons(String ... st){
       Set<String> s = new HashSet<String>();
       for (String a : st)
           if (!s.add(a))
               System.out.println("Doublon: " + a);

       System.out.println("il y a "+s.size() + " mots différents: " + s);
}
public static void chercheDoublonsbis(String st[]){
       Set<String> s=new HashSet<String>();
       Set<String> sdup=new HashSet<String>();
       for(String a :st)
           if (!s.add(a))
               sdup.add(a);
       s.removeAll(sdup);
       System.out.println("Mots uniques:    " + s);
       System.out.println("Mots dupliqués: " + sdup);
 }
```

Lists

- En plus de Collection:
  - Accès par position de l 'élément
  - Recherche qui retourne la position de l'élément
  - Sous-liste entre deux positions
- Implémentations:
  - ArrayList
  - LinkedList

List

```
public interface List<E> extends Collection<E> {
    // Positional access
    E get(int index);
    E set(int index, E element);    //optional
    boolean add(E element);         //optional
    void add(int index, E element); //optional
    E remove(int index);            //optional
    boolean addAll(int index,
        Collection<? extends E> c); //optional

    // Search
    int indexOf(Object o);
    int lastIndexOf(Object o);

    // Iteration
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    // Range-view
    List<E> subList(int from, int to);
}
```

Itérateur pour listes

```
public interface ListIterator<E> extends Iterator<E> {
    boolean hasNext();
    E next();
    boolean hasPrevious();
    E previous();
    int nextIndex();
    int previousIndex();
    void remove(); //optional
    void set(E e); //optional
    void add(E e); //optional
}
```

Exemple

```
public static <E> void swap(List<E> a, int i, int j) {
        E tmp = a.get(i);
        a.set(i, a.get(j));
        a.set(j, tmp);
}
public static void melange(List<?> list, Random rnd) {
        for (int i = list.size(); i > 1; i--)
            swap(list, i - 1, rnd.nextInt(i));
}
```

Suite…

```
public static <E> List<E> uneMain(List<E> deck, int n) {
       int deckSize = deck.size();
       List<E> handView = deck.subList(deckSize - n, deckSize);
       List<E> hand = new ArrayList<E>(handView);
       handView.clear();
       return hand;
   }
public static void distribuer(int nMains, int nCartes) {
    String[] couleurs = new String[]{"pique","coeur","carreau","trèfle"};
       String[] rank = new String[]
       {"as","2","3","4","5","6","7","8", "9","10","valet","dame","roi"};
       List<String> deck = new ArrayList<String>();
       for (int i = 0; i < couleurs.length; i++)
           for (int j = 0; j < rank.length; j++)
               deck.add(rank[j] + " de " + couleurs[i]);
       melange(deck,new Random());
       for (int i=0; i < nMains; i++)
           System.out.println(uneMain(deck,nCartes));
   }
```

Map

- Map associe des clés à des valeurs
  - Association injective: à une clé correspond exactement une valeur.
  - Trois implémentations, comme pour set
    - HashMap,
    - TreeMap,
    - LinkedHashMap
  - Remplace Hash

Map

```
public interface Map<K,V> {
    // Basic operations
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();
    // Bulk operations
    void putAll(Map<? extends K, ? extends V> m);
    void clear();
    // Collection Views
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();
    // Interface for entrySet elements
    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```

Exemples

```
public static void mapFreq(String ... t) {
        Map<String, Integer> m = new HashMap<String,
                                                Integer>();
        for (String a : t) {
            Integer freq = m.get(a);
            m.put(a, (freq == null) ? 1 : freq + 1);
        }
        System.out.println("Il y a: " + m.size() +
            " mots différents:\n"+m);
    }
// ordre arbitraire
```

Exemples

```
public static void mapFreq(String ... t) {
        Map<String, Integer> m = new TreeMap<String,
                                              Integer>();
        for (String a : t) {
            Integer freq = m.get(a);
            m.put(a, (freq == null) ? 1 : freq + 1);
        }
        System.out.println("Il y a: " + m.size() +
            " mots différents:\n"+m);
    }
// ordre arbitraire
```

Exemples

```
public static void mapFreq(String ... t) {
        Map<String, Integer> m = new LinkedHashMap<String,
                                                Integer>();
        for (String a : t) {
            Integer freq = m.get(a);
            m.put(a, (freq == null) ? 1 : freq + 1);
        }
        System.out.println("Il y a: " + m.size() +
            " mots différents:\n"+m);
    }
// ordre arbitraire
```

Queue

- Pour représenter une file (en principe FIFO):
  - Insertion: offer -add
  - Extraction: poll - remove
  - Pour voir: peek –element
  - (retourne une valeur - exception
- PriorityQueue implémentation pour une file à priorité

Interface Queue

```
public interface Queue<E> extends
   Collection<E> {
     E element();
     boolean offer(E e);
     E peek();
     E poll();
     E remove();
}
```

Exemple

```
public static void compteur(int n)
            throws InterruptedException {
        Queue<Integer> queue = new
                            LinkedList<Integer>();
        for (int i = n; i >= 0; i--)
            queue.add(i);
        while (!queue.isEmpty()) {
            System.out.println(queue.remove());
            Thread.sleep(1000);
        }
    }
```

Exemple

```
static <E> List<E> heapSort(Collection<E> c) {
    Queue<E> queue = new PriorityQueue<E>(c);
    List<E> result = new ArrayList<E>();
    while (!queue.isEmpty())
        result.add(queue.remove());
    return result;
}
```

Des implémentations

- HashSet<E>: implémentation deSet comme table de hachage. Recherche/ ajout suppression en temps constant
- TreeSet<E>: SortedSet comme arbre binaire équilibré O(log(n))
- ArrayList<E>: liste implémentée par des tableaux à taille variable accès en O(1) ajout et suppression enO(n-i) (i position considérée)
- LinkedList<E>: liste doublement chaînée implémente List et Queue accès en O(i)
- HashMap<K,V>: implémentation de Map pas table de hachage ajout suppression et recherche en O(1)
- TreeMap<K,V>: implémentation de SortedMap à partir d'arbres équilibrésajout, suppression et recherche en O(log(n))
- WeakHashMap<K,V>: implémentation de Map par table de hachage
- PriorityQueue<E>: tas à priorité.

Comparaisons

- Interface Comparable<T> contient la méthode
  - public int compareTo(T e)
  - "ordre naturel"
- Interface Comparator<T> contient la méthode
  - public int compare(T o1, T o2)

Quelques autres packages

- System méthodes static pour le système:
  - entrée-sorties standard
  - manipulation des propriétés systèmes
  - utilitaires "Runtime" exit(), gc() …

Runtime, Process

- Runtime permet de créer des processus pour exécuter des commande: exec
- Process retourné par un exec méthodes
  - destroy()
  - exitValue()
  - getInputStream()
  - getOutputStream()
  - getErrorStream()

Exemple

- exécuter une commande (du système local)
  - associer l'entrée de la commande sur System.in
  - associer la sortie sur System.out.

Exemple

```
class plugTogether extends Thread {
    InputStream from;
    OutputStream to;
    plugTogether(OutputStream to, InputStream from ) {
        this.from = from; this.to = to;
    }
    public void run() {
        byte b;
        try {
            while ((b= (byte) from.read()) !=  -1) to.write(b);
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            try {
                to.close();
                from.close();
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
}
```

Exemple suite

```
public class Main {
    public static Process userProg(String cmd)
    throws IOException {
      Process proc = Runtime.getRuntime().exec(cmd);
      Thread thread1 = new plugTogether(proc.getOutputStream(), System.in);
      Thread thread2 = new plugTogether(System.out, proc.getInputStream());
      Thread thread3 = new plugTogether(System.err, proc.getErrorStream());
      thread1.start(); thread2.start(); thread3.start();
      try {proc.waitFor();} catch (InterruptedException e) {}
        return proc;
    }
    public static void main(String args[])
    throws IOException {
       String cmd = args[0];
       System.out.println("Execution de: "+cmd);
       Process proc = userProg(cmd);
    }
}
```

Java Swing

Principes de base

- Des composants graphiques
  (exemple: JFrame, JButton ...)
  - Hiérarchie de classes
- Des événements et les actions à effectuer
  (exemple presser un bouton)
- (Et d'autres choses...)

Principes

- Définir les composants (instance de classes)
- Les placer à la main (layout Manager) dans un JPanel ou un content pane ou en utilisant des outils comme eclipse ou netbeans
- Définir les actions associées aux événements (Listener) et les associer aux composants graphiques

Principes

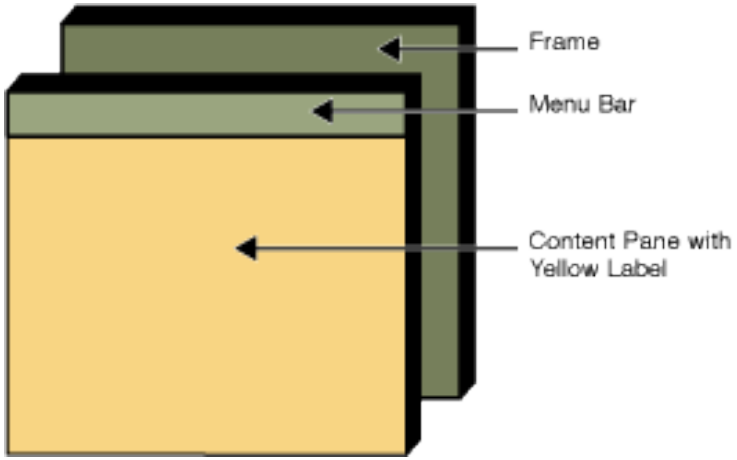
- Dans une interface graphique, le programme réagit aux interactions avec l'utilisateur
- Les interactions génèrent des événements
- Le programme est dirigé par les événements (event-driven)

Afficher…

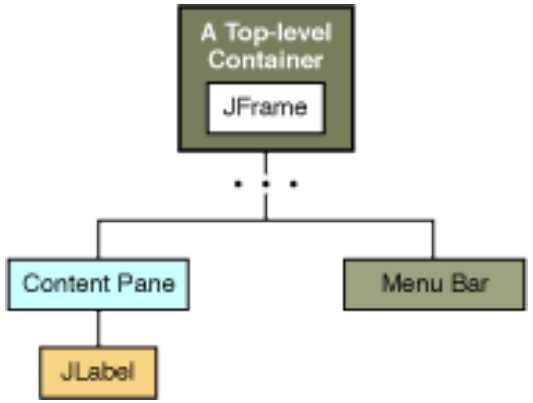
- Pour pouvoir être affiché, il faut que le composant soit dans un top-level conteneur:

  (JFrame, JDialog et JApplet)
- Hiérarchie des composants: arbre racine top-level

Exemple



- Correspond à la hiérarchie



Le code

```
import java.awt.*;
import javax.swing.*;

public class TopLevel {
    /**
     * Affiche une fenêtre JFrame top level
     * avec une barre de menu JMenuBar verte
     * et un JLabel jaune
     */
    private static void afficherMaFenetre() {
            //créer la Jframe
            //créer la JMenuBar
           //créer le Jlabel
           // mettre le JMenuBar et le Jlable dans la Jframe
           //afficher la Jframe
    }
}
```

Le code

```
//Creer la JFrame
  JFrame frame = new JFrame("TopLevelDemo");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  //Créer la JMenuBar
  JMenuBar greenMenuBar = new JMenuBar();
  greenMenuBar.setOpaque(true);
  greenMenuBar.setBackground(new Color(0, 200, 0));
  greenMenuBar.setPreferredSize(new Dimension(200, 20));
  //Créer le JLabel
  JLabel yellowLabel = new JLabel();
  yellowLabel.setOpaque(true);
  yellowLabel.setBackground(new Color(250, 250, 0));
  yellowLabel.setPreferredSize(new Dimension(200, 180));
  //mettre la JmenuBar et position le JLabel
  frame.setJMenuBar(greenMenuBar);
  frame.getContentPane().add(yellowLabel, BorderLayout.CENTER);
  //afficher...
  frame.pack();
  frame.setVisible(true);
```

Et le main

```
public class TopLevel {//afficherMaFenetre()
   public static void main(String[] args) {
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            afficherMaFenetre();
         }
      });
   }
}
```

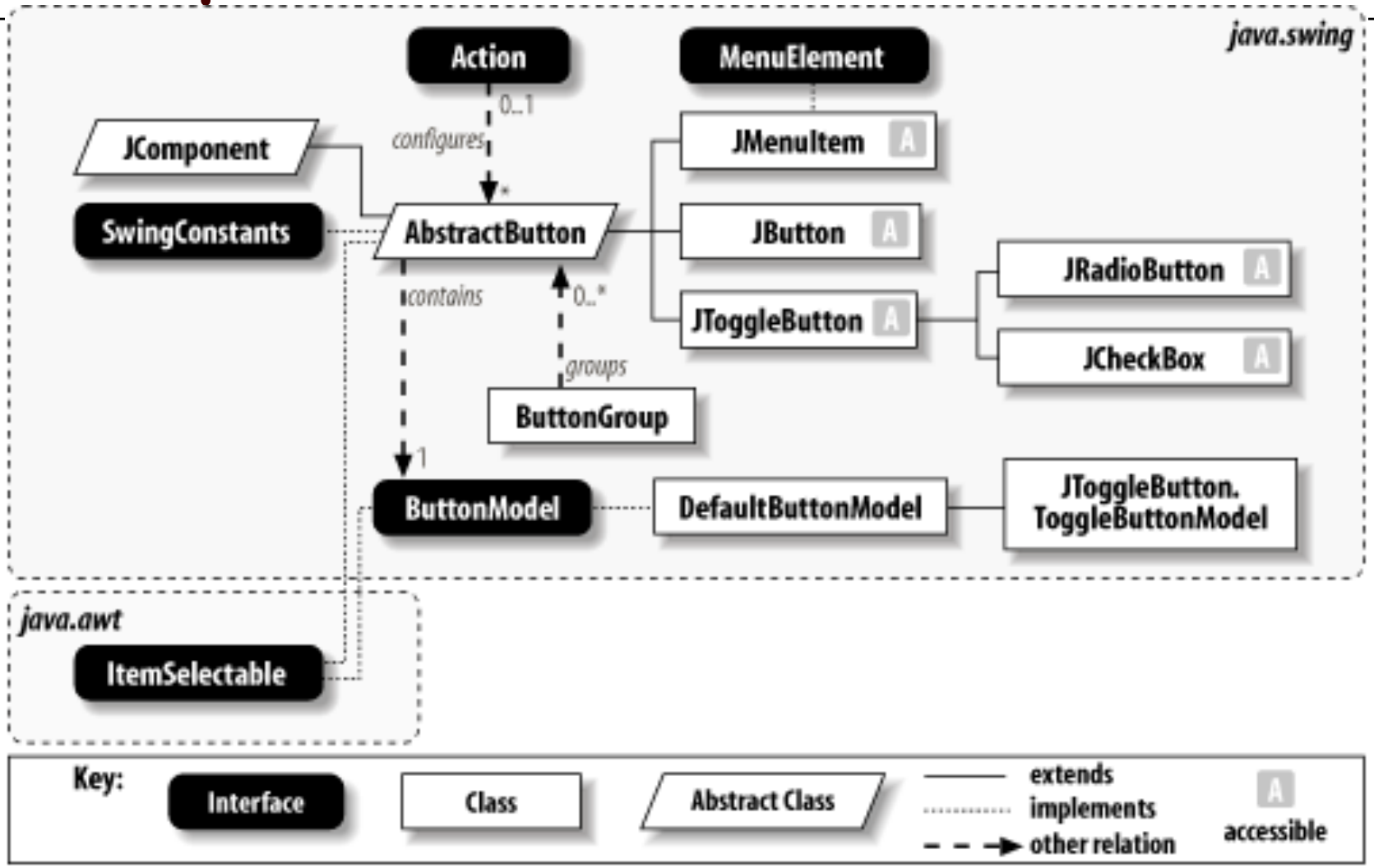
Evénements: principes

- Dans un système d'interface graphique:
  - Quand l'utilisateur presse un bouton, un "événement" est posté et va dans une boucle d'événements
  - Les événements dans la boucle d'événements sont transmis aux applications qui se sont enregistrées pour écouter.

Evénements

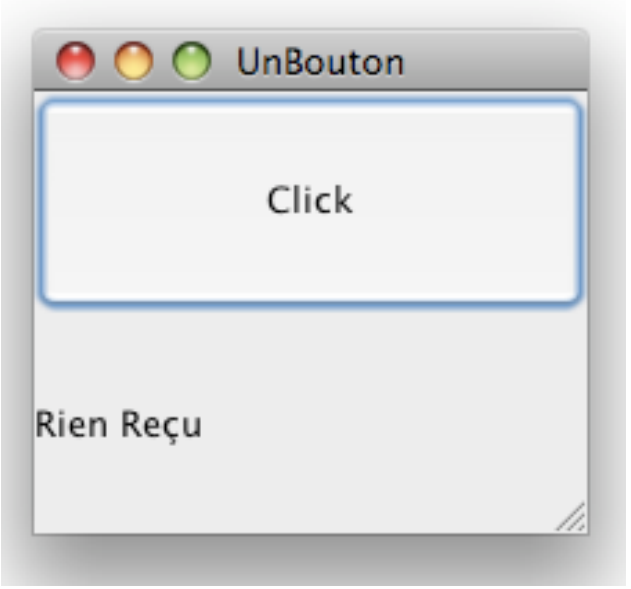
- Chaque composant génère des événements:
  - Presser un JButton génère un ActionEvent (système d'interface graphique)
    - Cet ActionEvent contient des infos (quel bouton, position de la souris, modificateurs…)
  - Un event listener (implémente ActionListener)
    - définit une méthode actionPerformed
    - S'enregistre auprès du bouton addActionListener
  - Quand le bouton est "clické",l'actionPerformed sera exécuté (avec l'ActionEvent comme paramètre)

Exemples Buttons



Un exemple

- Un bouton qui réagit



Le code:

- Un JButton
- Un JLabel
- Implementer ActionListener
  - actionPerfomed définit ce qui se passe quand le bouton est cliqué
- Placer le bouton et le label

Code:

```
import java.awt.*;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.Jcomponent;
import java.awt.Toolkit;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;

public class UnBouton extends Jpanel implements ActionListener {
    JButton bouton;
    String contenu="Rien Reçu";
    JLabel label=new JLabel(contenu);
    int cmp=0;
    public UnBouton() { //…}
    public void actionPerformed(ActionEvent e) {//…}
    private static void maFenetre(){//…}
    public static void main(String[] args) {//…}
}
```

Code

```
public UnBouton() {
        super(new BorderLayout());
        bouton = new JButton("Click");
        bouton.setPreferredSize(new Dimension(200, 80));
        add(bouton, BorderLayout.NORTH);
        label = new JLabel(contenu);
                label.setPreferredSize(new Dimension(200, 80));
        add(label,BorderLayout.SOUTH);
        bouton.addActionListener(this);
   }
public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
        label.setText("clické "+ (++cmp)+ " fois");
  }
```

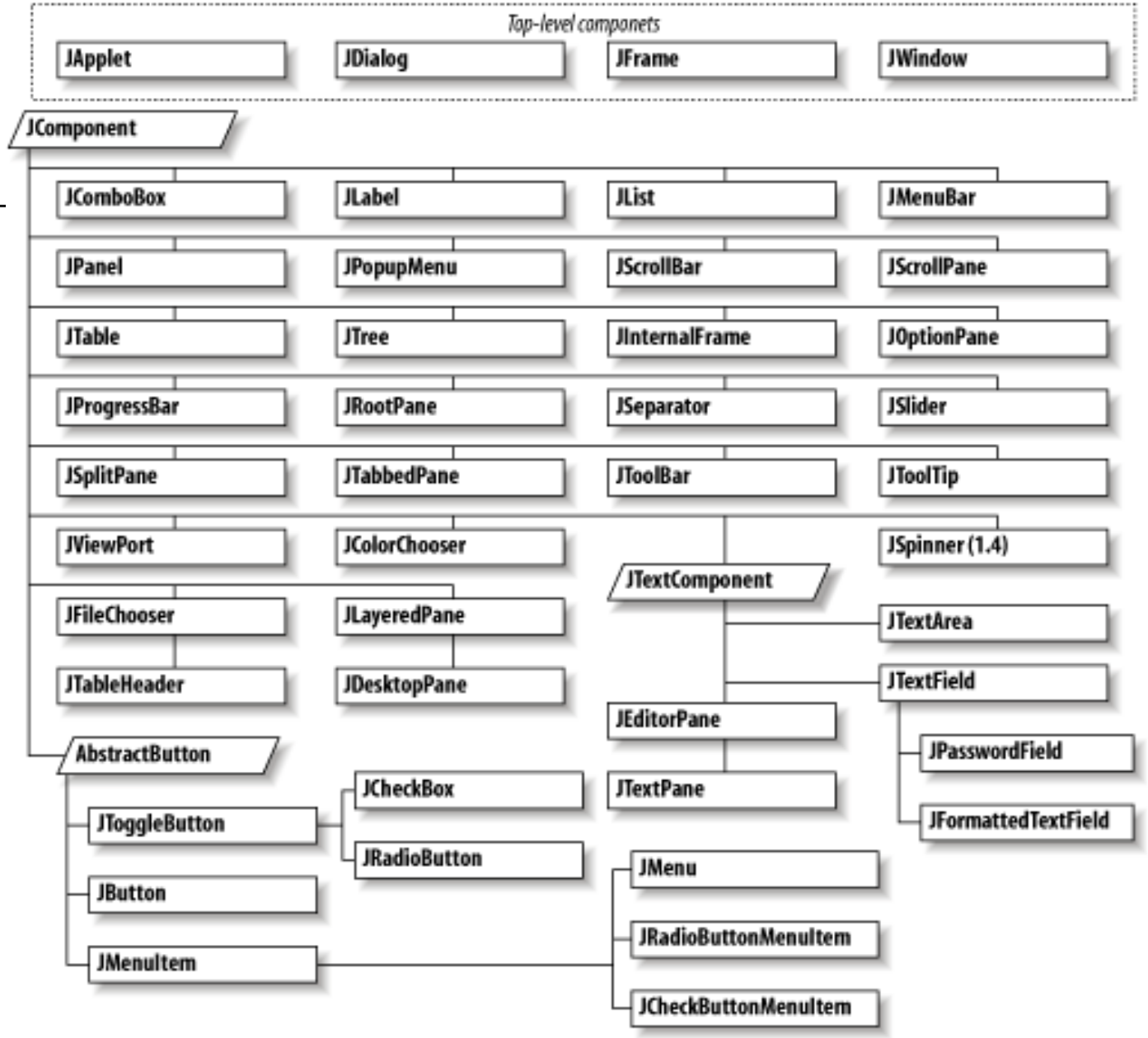
Code

```
private static void maFenetre() {
      JFrame frame = new JFrame("UnBouton");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JComponent newContentPane = new UnBouton();
      newContentPane.setOpaque(true);
      frame.setContentPane(newContentPane);
      frame.pack();
      frame.setVisible(true);
  }
  public static void main(String[] args) {
      //Formule magique
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
          public void run() {
              maFenetre();
          }
      });
  }
```

Variante

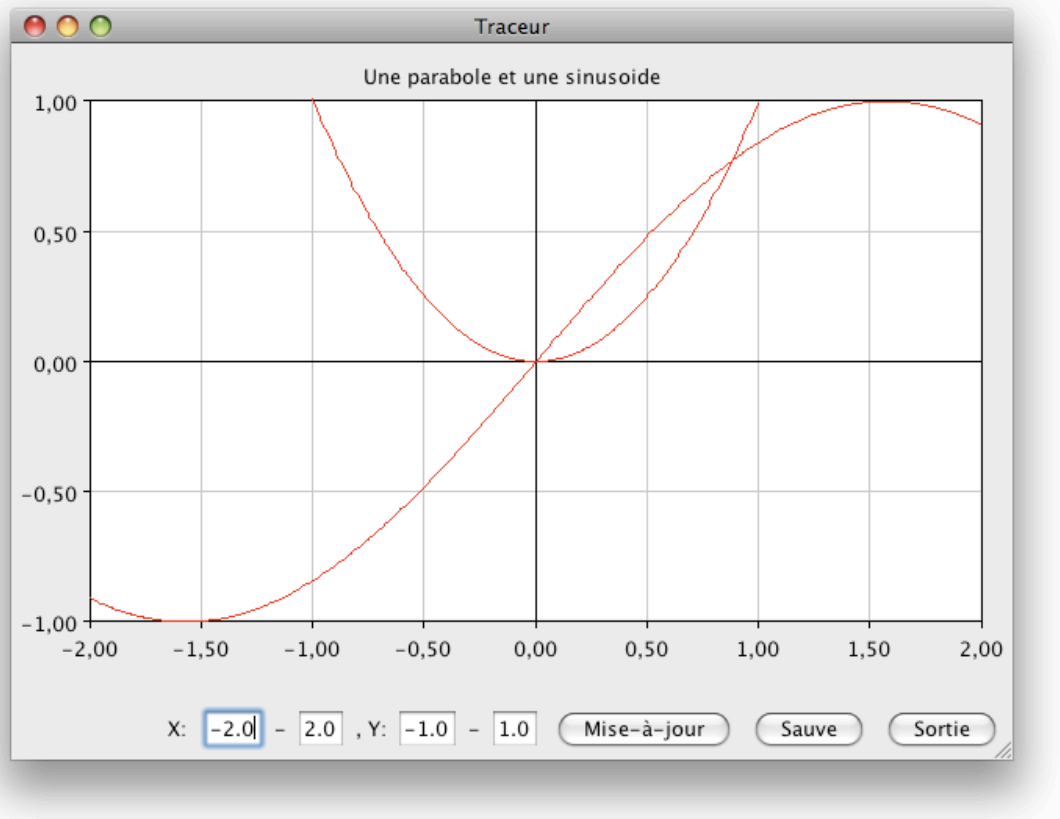
```
public class UnBoutonBis extends JPanel {
//…
 bouton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Toolkit.getDefaultToolkit().beep();
                label.setText("clické " + (++cmp) + " fois");
            } });
        }
//…
}
```

Hiérarchie des classes…


Top-level componets
JApplet
JDialog
JFrame
JWindow
JComponent
JComboBox
JLabel
JList
JMenuBar
JPanel
JPopupMenu
JScrollBar
JScrollPane
JTable
JTree
JInternalFrame
JOptionPane
JProgressBar
JRootPane
JSeparator
JSlider
JSplitPane
JTabbedPane
JToolBar
JToolTip
JViewPort
JColorChooser
JSpinner (1.4)
JFileChooser
JLayeredPane
JTableHeader
JDesktopPane
AbstractButton
JToggleButton
JCheckBox
JRadioButton
JButton
JMenuItem
JTextComponent
JTextArea
JTextField
JEditorPane
JTextPane
JPasswordField
JFormattedTextField
JMenu
JRadioButtonMenuItem
JCheckButtonMenuItem

Un exemple

- Un traceur de fonctions
  - Une interface graphique swing



Organisation

- GrapheSwing contient unGraphePanel extension de Jpanel
  - GraphePanel méthode paintComponent qui affiche le graphe de la fonction
    - Graphe est la classe contenant le gaphe et définissant une méthode draw pour l'affichage
    - Cette méthode appelle tracer de la classe abstraite Traceur
      - FonctionTraceur étend Traceur

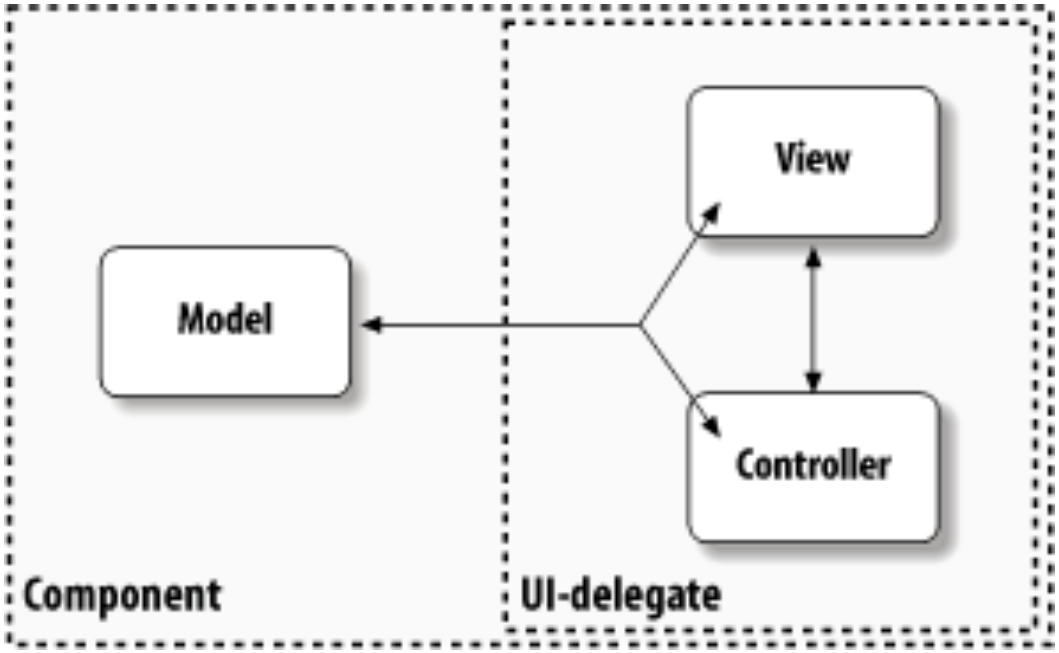
Le main

```
public static void main(String[] args) {  new GrapheSwing(unGraphe());}

  public static Graphe unGraphe() {
     PlotSettings p = new PlotSettings(-2, 2, -1, 1);
     p.setPlotColor(Color.RED);
     p.setGridSpacingX(0.5);
     p.setGridSpacingY(0.5);
     p.setTitle("Une parabole et une sinusoide");
     Graphe graphe = new Graphe(p);
     graphe.functions.add(new Parabole());
     graphe.functions.add(new FonctionTraceur() {
        public double getY(double x) {
           return Math.sin(x);
        }
        public String getName() {
           return "Sin(x)";
        }
     });
     return graphe;
  }
```

Composants

- Modèle Vue Contrôleur



Préliminaires…

- Lightweight et heavyweight composants
  - Dépendent ou non du système d'interface graphique
    - Lightweight écrit en Java et dessinés dans un heavyweight composant- indépendant de la plateforme
    - Les heavyweight composants s'adressent directement à l'interface graphique du système
  - (certaines caractéristiques dépendent du look and feel).

Look and feel

- Look and feel:

  Possibilité de choisir l'apparence de l'interface graphique.

  UIManager gère l'apparence de l'interface

```
public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName());
    } catch (Exception e) { }

    new SwingApplication(); //Create and show the GUI.
}
```

Multithreading

- Attention au « modèle, contrôleur, vue » en cas de multithreading:
  - Tous les événements de dessin de l'interface graphiques sont dans une unique file d'event-dispatching dans une seule thread.
  - La mise à jour du modèle doit se faire tout de suite après l'événement de visualisation dans cette thread.

Plus précisément

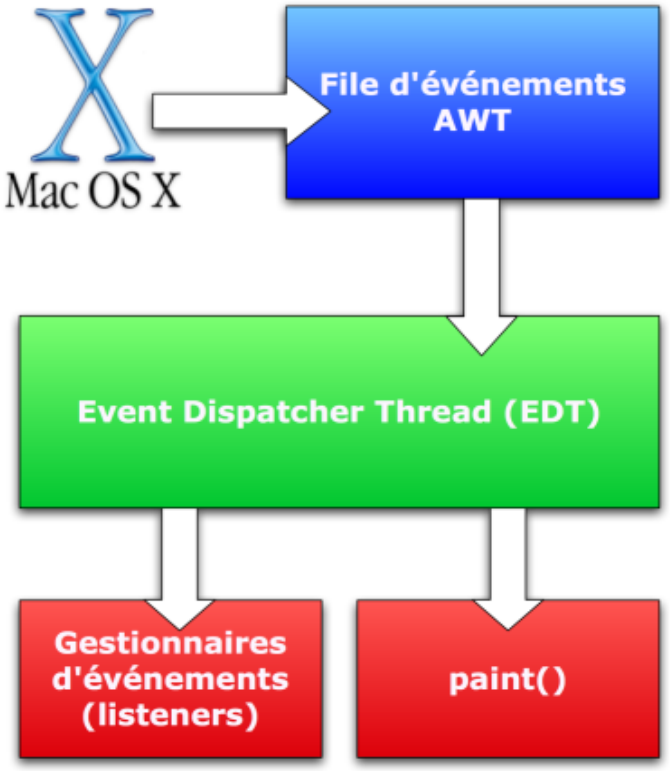
- Swing prend en charge la gestion des composants qui sont dessinés en code Java (lightweight)
- Les composants AWT sont eux liés aux composants natifs (heavyweight)
- Swing dessine le composants dans un canevas AWT et utilise le traitement des événements de AWT

Suite

Les threads

- Main application thread
- Toolkit thread
- Event dispatcher thread

- Toutes Les opérations d'affichage ont lieu dans une seule thread l'EDT



Principes

- Une tâche longue ne doit pas être exécutée dans l'EDT
- Un composant Swing doit s'exécuter dans l'EDT

Exemple

```
public void actionPerformed(ActionEvent e){
   try {
      Thread.sleep(4000);
} catch (InterruptedException e) {    }  }
```

Provoque une interruption de l'affichage pendant 4 secondes

Une solution

```
public void actionPerformed(ActionEvent e){
  try{
    SwingUtilities.invokeLater(newRunnable(
          { public void run() {
               //opération longue
               }
           });
   } catch (InterruptedException ie) {}
     catch (InvocationTargetException ite) {}
   }
}
```

Le main

- Normalement la création d'une fenêtre ne devrait avoir lieu que dans l'EDT:

```
public static void main(String[] args) {
      //Formule magique
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
                              public void run() {maFenetre(); }
      });
  }
```

invokeLater crée une nouvelle thread qui poste la thread crée dans l'EDT

Attendre le résultat:

```
try {
        SwingUtilities.invokeAndWait(new Runnable() {
            public void run() {
                show();
            }
        });
    } catch (InterruptedException ie) {
    } catch (InvocationTargetException ite) {
    }
```

Chapitre IX

Entrées-sorties

Principes généraux

- entrées sorties
  - streams dans **java.io**
  - channels dans **java.nio** ("n" pour non-blocking)

- streams: séquences de données ordonnées avec une source (stream d'entrée) ou une destination (stream de sortie)
- channels et buffer. Buffer contient les données et le channel correspond à des connections

Streams

- Deux grandes sortes de Stream
  - character Streams contiennet des caractères UTF-16
  - byte Streams contiennent des octets
- Character versus byte
  - imput ou output stream pour les byte
  - reader, writer pour les character
  - deux hiérarchies qui se correspondent

Hiérarchie

- java.io.**InputStream** (implements java.io.Closeable)
  - java.io.**ByteArrayInputStream**
  - java.io.**FileInputStream**
  - java.io.**FilterInputStream**
    - java.io.**BufferedInputStream**
    - java.io.**DataInputStream** (implements java.io.DataInput)
    - java.io.**LineNumberInputStream**
    - java.io.**PushbackInputStream**
  - java.io.**ObjectInputStream** (implements java.io.ObjectInput, java.io.ObjectStreamConstants)
  - java.io.**PipedInputStream**
  - java.io.**SequenceInputStream**
  - java.io.**StringBufferInputStream**

Hiérarchie

- java.io.**OutputStream** (implements java.io.Closeable, java.io.Flushable)
  - java.io.**ByteArrayOutputStream**
  - java.io.**FileOutputStream**
  - java.io.**FilterOutputStream**
    - java.io.**BufferedOutputStream**
    - java.io.**DataOutputStream** (implements java.io.DataOutput)
    - java.io.**PrintStream** (implements java.lang.Appendable, java.io.Closeable)
  - java.io.**ObjectOutputStream** (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)
  - java.io.**PipedOutputStream**

Hiérachie…

- java.io.**Reader** (implements java.io.Closeable, java.lang.Readable)
  - java.io.**BufferedReader**
    - java.io.**LineNumberReader**
  - java.io.**CharArrayReader**
  - java.io.**FilterReader**
    - java.io.**PushbackReader**
  - java.io.**InputStreamReader**
    - java.io.**FileReader**
  - java.io.**PipedReader**
  - java.io.**StringReader**

Hiérarchie

- java.io.**Writer** (implements java.lang.Appendable, java.io.Closeable, java.io.Flushable)
  - java.io.**BufferedWriter**
  - java.io.**CharArrayWriter**
  - java.io.**FilterWriter**
  - java.io.**OutputStreamWriter**
    - java.io.**FileWriter**
  - java.io.**PipedWriter**
  - java.io.**PrintWriter**
  - java.io.**StringWriter**

Streams d'octets

- Class InputStream
  (toutes ces méthodes peuvent lancer IOEXception)
  - abstract int **read**() lit un octet
  - int **read** (byte[] b) lit et écrit dans b, (le nombre d'octets lus dépend de b et est retourné
  - int **read**(byte[] b, int off, int len)
  - long **skip**(long n)
  - int **available**() n d'octets pouvant être lus
  - void **mark**(int readlimit) et void **reset**() pose d'une marque et retour à la marque
  - void**close**()

Stream d'octets

- OutputStream

  (toutes ces méthodes peuvent lancer IOEXception)

  - abstract void **write**(int b) écrit un octet
  - void **write**(byte[] b)
  - void **write**(byte[] b, int off, int len)
  - void**close**()
  - void**flush**()

Exemples

```
public static int compteIS(String st)
        throws IOException{
    InputStream in;
    int n=0;
    if (st==null)in=System.in;
    else
        in= new FileInputStream(st);
    for(; in.read() != -1;n++);
    return n;
 }
```

Exemples

```
public static void trOS(String f, char from, char to){
    int b;
    OutputStream out=null;
    try{
      if(f==null) out=System.out;
       else
        out=new FileOutputStream(f);
        while((b= System.in.read())!=-1)
            out.write(b==from? to:b);
        out.flush();
        out.close();
    }catch(IOException ex){
            ex.printStackTrace();
    }
}
```

Reader-Writer

- Reader et Writer sont les deux classes abstraites pour les streams de caractères:
  - le read de InputStream retourne un byte comme octet de poids faible d'un int alors que le read de Reader retourne un char à partir de 2 octets de poids faible d'un int

Reader

- essentiellement les mêmes méthodes que pour InputStream:
  - int **read**()
  - int **read**(char[] cbuf)
  - abstract int **read**(char[] cbuf, int off, int len)
  - int **read**(CharBuffer target)
  - boolean **ready**() si prêt à lire
  - void**reset**()
  - void**mark**(int readAheadLimit)
  - boolean**markSupported**()
  - long**skip**(long n)

Writer

- similaire à OutputStream mais avec des caractères au lieu d'octets.
  - void **write**(char[] cbuf)
  - abstract void **write**(char[] cbuf, int off, int len)
  - void **write**(int c)
  - void **write**(int c)
  - void **write**(String str)
  - void **write**(String str, int off, int len)
  - Writer **append**(char c)
  - Writer **append**(CharSequence csq)
  - Writer **append**(CharSequence csq, int start, int end)
  - abstract void **close**()
  - abstract void **flush**()

Exemples

```
public static int compteR(String st)
        throws IOException{
   Reader in;
   int n=0;
   if (st==null)
    in=new InputStreamReader(System.in);
   else
    in= new FileReader(st);
   for(; in.read() != -1;n++);
   return n;
}
```

Exemples

```
public static void trWr(String f, char from, char to){
    int b;
    Writer out=null;
    try {
        if(f==null)
            out= new OutputStreamWriter(System.out);
        else
            out=new FileWriter(f);
        while((b= System.in.read())!=-1)
                out.write(b==from? to:b);
        out.flush();
        out.close();
   }catch(IOException e){System.out.println(e);}
}
```

Remarques

- les streams standard System.in et System.out sont des streams d'octets
- InputStreamReader permet de passer de InputStream à Reader
- System.in et System.out sont des PrintStream (obsolète à remplacer par la version caractère PrintWriter)

InputStreamReader et OutputStreamWriter

- conversion entre caractères et octets, la conversion est donnée par un **charset**
- Constructeurs:
  - **InputStreamReader**(InputStream in)
  - **InputStreamReader**(InputStream in, Charset cs)
  - **InputStreamReader**(InputStream in, String charsetName)
  - **OutputStreamWriter**(OutputStream out)
  - **OutputStreamWriter**(OutputStream out, Charset cs)
  - **OutputStreamWriter**(OutputStream out, CharsetEncoder enc)
  - **OutputStreamWriter**(OutputStream out, String charsetName)

Autres classes

- FileInputStream
- FileOutputStream
- FileReader
- FileWriter
- Ces classes permettent d'associer une stream à un fichier donné par son nom (String) par un objet File ou un objet FileDescriptor (un mode append peut être défini pour les ecritures)

Quelques autres classes

- Les filtres
  - FilterInputStream
  - FilterOutputStream
  - FilterReader
  - FilterWriter
- Buffered
  - BufferedInputStream
  - BufferedOutputStream
  - BufferedReader
  - BufferedWriter
- Tubes
  - PipedInputStream
  - PipedOutputStream
  - PipedReader
  - PipedWriter

Exemple d'un filtre

```
class conversionMaj extends FilterReader{
    public conversionMaj(Reader in){
        super(in);
    }
    public int read() throws IOException{
        int c=super.read();
        return(c==-1?c:Character.toUpperCase((char)c));
    }
    public int read(char[] buf, int offset, int count)throws
    IOException{
        int nread=super.read(buf,offset,count);
        int last=offset+nread;
        for(int i=offset; i<last;i++)
            buf[i] = Character.toUpperCase(buf[i]);
        return nread;
    }
}
```

Exemple suite:

```
StringReader source=new StringReader("ma chaîne");
FilterReader filtre=new conversionMaj(source);
int c;
try{
  while((c=filtre.read())!= -1)
      System.out.print((char)c);
  }catch(IOException e){
  e.printStackTrace(System.err);
 }
```

Pipe (tube)

- Un tube associe une entrée et un sortie: on lit à une extrémité et on écrit à l'autre.

Exemple pipe

```
class PipeExemple extends Thread{
    private Writer out;
    private Reader in;
    public PipeExemple(Writer out,Reader in){
        this.out=out;
        this.in=in;
    }
    public void run(){
        int c;
        try{
            try{
                while ((c=in.read())!=-1){
                    out.write(Character.toUpperCase((char)c));
                }
            }finally{out.close();}
        }catch(IOException e){e.printStackTrace(System.err);}
    }
}
```

Suite

```
PipedWriter out1=new PipedWriter();
PipedReader in1=new PipedReader(out1);
PipedWriter out2=new PipedWriter();
PipedReader in2=new PipedReader(out2);
PipeExemple pipe=new PipeExemple(out1, in2);
pipe.start();
try{
    for(char c='a';c<='z';c++) {
        out2.write(c);
        System.out.print((char)(in1.read()));
    }
}finally {
    out2.close();
}
```

ByteArray, String…

- java.io.**InputStream** (implements java.io.Closeable)
  - java.io.**ByteArrayInputStream**
  - java.io.**StringBufferInputStream**
- java.io.**OutputStream** (implements java.io.Closeable, java.io.Flushable)
  - java.io.**ByteArrayOutputStream**
- java.io.**Reader** (implements java.io.Closeable, java.lang.Readable)
  - java.io.**CharArrayReader**
  - java.io.**StringReader**
- java.io.**Writer** (implements java.lang.Appendable, java.io.Closeable, java.io.Flushable)
  - java.io.**CharArrayWriter**
  - java.io.**StringWriter**

Print Streams

- java.io.**PrintStream** (implements java.lang.Appendable, java.io.Closeable)
- java.io.**PrintWriter**
  - méthode println()

Streams pour les données

- DataInput: interface pour lire des bytes d'une stream binaire et les transformer en données java de type primitif
- DataOutput: interface pour convertir des valeurs de type primitif en stream binaire.

Sérialisation

- sauvegarder des objets java en flot d'octets:
  - objet vers byte: sérialisation
  - byte vers objet: désérialisation
- java.io.**ObjectInputStream** (implements java.io.ObjectInput, java.io.ObjectStreamConstants)
- java.io.**ObjectInputStream** (implements java.io.ObjectInput, java.io.ObjectStreamConstants)

Sauver des objets

- la serialisation-désérialisation doit permettre de sauvegarder et restituer des objets.
- sauver et restituer des objets n'est pas si simple. Pourquoi?
- interface serializable
- (alternative XML)

Manipuler des fichiers…

- java.io.**File** (implements java.lang.Comparable<T>, java.io.Serializable)

nio

- entrée sorties de haute performance avec contrôle sur les buffers.