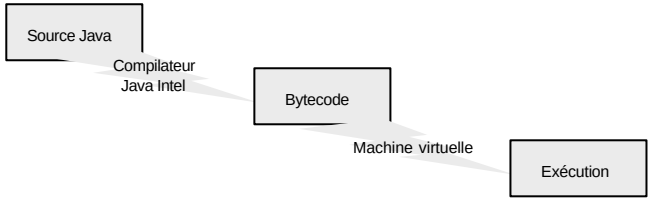
Cours de base Java

Java Le Langage

Java - Introduction

- Langage orienté objet multi-plateforme
  - Conçu par Sun Microsystems
    - Write Once Run Anywhere
  - Fonctionne en mode interprété et s'exécute sur toute machine disposant de l'interpréteur
    - disponible sur Windows, Mac, Unix, et certains mainframes
  - Langage orienté objet, inspiré de C++ par certains aspects
  - De nombreuses fonctionnalités Réseau et Internet
  - Gestion du multitâche
- Historique
  - Développement de OAK, système embarqué, en 1991
  - Développement de WebRunner, renommé en HotJava
    - navigateur écrit en java, interpréteur de java
  - Apparition du Java Development Kit (JDK) distribué par Javasoft, filiale de Sun
    - Mise à disposition des sources
- Pourquoi Java
  - Langage fortement typé
  - Langage orienté objet
    - Notions de classes et d'objets
    - Inspiré de C++
    - Gestion mémoire réalisée par un Garbage Collector
  - Langage compilé vers du pseudo code binaire
    - Code binaire portable, nommé "ByteCode", interprété au sein d'une machine virtuelle (VM)
    - Portabilité = disponibilité de la machine virtuelle Java



Java Introduction

- Pourquoi Java (suite)
  - Multitâche : le multithreading
    - thread = tâche spécifique et indépendante
    - Le multithreading est pris en charge par la machine virtuelle
  - Exécution dynamique
    - Téléchargement de code à la volée
  - Java versus C++
    - Absence de pointeurs
    - Pas de variables globales
    - Pas de types énumérés
    - Pas d'héritage multiple
    - Pas de surcharge des opérateurs
- Application et applet
  - Applications
    - Exécution en dehors d'un navigateur Web
    - Accès à l'ensemble des composants du système sans restriction
  - Applets
    - S'exécutent dans un navigateur Web (mode SandBox)
    - Intégré au HTML
    - Machine virtuelle intégrée au navigateur
    - Applications distribuées, téléchargeables depuis un serveur HTTP
    - Restrictions d'accès sur la machine locale. Exemple : ne peut dialoguer directement qu'avec la machine depuis laquelle il a été téléchargé
- Outils de développement
  - Visual Café
  - Jbuilder
  - Visual J++
  - PowerJ

Machine virtuelle et JDK

- Machine virtuelle Java
  - Architecture d'exécution complète
    - Jeu d'instructions précis
    - des registres
    - une pile
  - La sécurité est fondamentale
    - Le langage et le compilateur gèrent entièrement les pointeurs
    - Un programme de vérification du bytecode veillent à l'intégrité du code java
    - Le chargeur de classes (class loader) est chargé d'autoriser ou de refuser le chargement d'une classe.
    - Une classe est chargée d'effectuer la vérification des appels aux API
  - Disponibilité
    - Machines virtuelles JDK
    - Machines virtuelles intégrées aux navigateurs
    - Machines virtuelles des environnements de développement
- Java Development Kit

  Ensemble de composants permettant le développement, la mise au point et l'exécution des programmes Java.
  - Un ensemble d'outils;
  - Un jeu de classes et de services;
  - un ensemble de spécifications.

  Un développement java peut être entièrement réalisé avec le JDK, avec des outils en ligne de commande.

  Les fichiers sources ont l'extension .java

  Les fichiers compilés ont l'extension .class

Nom	Description
Java.exe	Machine virtuelle java
Javac.exe	Compilateur java
Appletviewer.exe	Machine virtuelle java pour l'exécution d'applets
Jar.exe	Permet la création d'archives java
Javadoc.exe	Générateur de documentation java
Javap.exe	Désassembleur de classes compilées
Jdb.exe	Débogueur en ligne de commande

Machine virtuelle et JDK (suite)

- Version du JDK
  - JDK 1.0, 1.1, 1.2 (java2), 1.3 (nouvelle plate-forme java2)
    - Les JDK sont disponibles sur Internet http://java.sun.com/products/JDK
  - JDK 1.02
    - première version réellement opérationnelle, API et classes élémentaires
    - première version de la bibliothèque AWT (Abstract Windowing Toolkit)
    - Utilisé seulement pour une compatibilité maximale.
  - JDK 1.1: 1.1.8
    - améliorations et extensions du langage, améliorations de AWT
    - Apparition des composants Java et JavaBeans, JDBC (Java Database Connectivity), RMI (Remote Method Invocation)
    - Nouveau modèle d'événements, amélioration de la sécurité (signature des applets)
    - Java côté serveur (Servlets), JNI (Java Native Interface)
  - JDK 1.2
    - Intégration des composants Java Swing, dans les JFC (Java Foundation Classes)
    - Amélioration des JavaBeans, intégration de l'API Java 2D, API d'accessibilité
    - Intégration de l'IDL en standard pour le support natif de CORBA, intégration des bibliothèques CORBA
    - Support du drag and drop.
- Bibliothèque

  AWT: composants d'interface homme machine portables, basé sur des classes d'implémentation spécifiques à chaque plate-forme. Un composant AWT correspond à un composant natif.

  SWING : nouvelle génération de composants; 100% Java, Look & Feel paramétrable, modèle VMC (Vue Modèle Contrôleur)
- Exercice 1: mon premier programme

  Fichier HelloWorld.java:

```
public class HelloWorld {
  public static void main(String[] args)
  {
    System.out.println("Hello World!");
  }
}
```

  Exemple construit sur une classe publique

  La méthode main est le point d'entrée du programme

  Le nom du fichier et le nom de la classe doivent concorder

  Compilation

```
javac HelloWorld.java
```

  Exécution

```
java HelloWorld
```

Jbuilder 3

- Des experts et assistants
- Basé sur le JDK 1.2, mais support des JDK passés
- Apports
  - support intégré du JDK 1.2
  - bibliothèques de composants
  - séparation DataExpress/JBCL
  - navigation dans les sources
  - environnement plus simple d'accès
  - intégration de swing
  - nouveaux experts et nouvelles versions d'experts
  - explorateur et moniteur SQL améliorés
  - intégration de VisiBroker 3.4
- IDE
  - Mode conception pour le développement
  - Mode exécution réservée aux phases de tests et de mise au point
- Caractéristiques des versions
  - Voir page 23, "Le programmeur Jbuilder3"

Le langage Java

- Eléments généraux du langage
  - Codage du texte
    - Java peut utiliser un codage Unicode ou ASCII
  - Commentaires

```
// ceci est un commentaire sur une seule ligne
/* ceci est un commentaire
qui peut s'étaler sur plusieurs lignes */
```

  - Commentaires javadoc
    - Commentaires javadoc introduits par /**

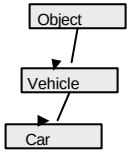
Tag	Description	Applicable à
@see	Nom de classe associé	Class, method ou variable
@author	Nom de l'auteur	Classe
@version	Numéro de version	Classe
@param	Nom de paramètre et description	Méthode
@return	Description de la valeur de retour	Méthode
@exception	Nom de l'exception et description	Méthode
@deprecated	Déclare un item obsolete	Classe, méthode ou variable

- Types
  - Langage typé: toute variable a un type, connu au moment de la compilation.
  - Java maintient aussi des informations consultables au moment de l'exécution.
  - Types primitifs : boolean, char, byte, short, int, long, float, double.
  - Les types primitifs ne sont pas des objets, mais disposent de classes Wrapper.
  - Déclaration et initialisation

```
int toto;
double b1, b2;
int foo = 12;
```

  - Les variables non initialisées sont automatiquement positionnées à "zero".
  - Références
    - Les objets sont toujours manipulés par référence.
    - Les références sont passées par valeur.

```
// deux références sur le même objet
Car myCar = new Car();
Car anotherCar = myCar;
// passage de l'objet par référence
myMethod(myCar);
```



Le langage Java

- Chaînes
  - Les chaînes sont des objets de classe String

```
System.out.println("Hello my string…");
String s = "I am a value";
String t = "Je dis \"I am a string\"";
String quote = "Ceci " + "est " + s;
```

- Instructions
  - Blocs de code

    Paranthensés par { }
  - Méthodes

    Blocs de code paramétrés
  - Variables

    Durée de vie = limites du bloc dans lequel elle est déclarée
  - Instructions

```
if ( condition ) instruction; [ else instruction; ]
if ( condition ) { instructions; }
while ( condition ) instruction;
do instruction; while (condition );
for (initialisation; condition; incrementation ) instruction;
switch (int expression) {
  case int expression: instruction; break;
  [ case in expression: instruction;
  …
  default: instruction; ]
}
```

    break : sortie du bloc en cours

    continue: passage à l'itération de boucle successive sans continuer le code
  - Instructions et affectations

    Une initialisation est une instruction

    Une affectation est une expression

Le langage Java : expressions

- Expression
  - Opérateurs

    ++ et -- incrémenter, décrémenter

    +, -, *, /: opérateurs unaires et binaires arithmétiques

    +: concaténation de chaînes

    <<, >>: décalages à gauche et à droite

    >>> décalage à droite sans extension de signe

    <, <=, >, >=: comparaison numérique

    ~: complément binaire, !: complément logique

    (type): conversion

    ==, !=: égalité et différence de valeur et de référence

    & : 'et' binaire et logique, | : 'ou' binaire et logique

    ^ : 'ou exclusif' binaire et booléen

    && et ||: 'et' conditionnel, 'ou' conditionnel

    ?: : opérateur conditionnel ternaire, =: affectation

    *=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, &=, ^=, |=: affectation avec opération

    instanceof : comparaison de types
  - Affectation

    Une affectation est une expression qui peut être utilisée comme une valeur dans une autre expression
  - null

    Peut être assignée à une référence pour indiquer "pas de référence"
  - Accès aux variables et appel de méthodes

    Accès aux variables dans un objet: '.'

```
int i; String s;
i = myObject.length; s = myObject.name;
int len = myObject.name.length();
int initialLen = myObject.name.substring(5, 10).length();
```

  - Création d'objet

```
Object o = new Object();
```

  - instanceof

    Déterminer le type d'un objet à l'exécution. Rend une valeur booléenne.

```
Boolean b;
String str = "foo";
b = ( str instanceof String ); b = ( str instanceof Object ); // true
b = ( str instanceof Date ); b = ( str instanceof Date[] ); // false
str = null; b = ( str instanceof String ); b = ( str instanceof Object
); // false
```

- Exercice 2: le tri par bulle

  Enoncé: développer un programme de tri par bulle d'un tableau d'entiers

Le langage Java : la gestion des exceptions

- Classes
  - Les exceptions sont des instances de java.lang.Exception et ses dérivés
  - Les classes dérivées peuvent contenir des informations spécifiques
- Gestion des exceptions

```
try {
  readFromFile("toto");
  ...
}
catch( FileNotFoundException e) {
  System.out.println("Fichier non trouvé");
  …
}
catch( Exception e) {
  System.out.println("Exception pendant la lecture du fichier: "+e);
  …
}
```

Un seul bloc catch est exécuté.

On remonte au bloc try qui s'applique le plus proche, en déroulant la pile.

On ne revient pas à l'emplacement de l'erreur après traitement

L'objet de classe Exception contient des informations contextuelles

```
try {   // instructions
}
catch( Exception e)
{ e.printStackTrack(System.err); }
```

- Checked et Unchecked

Les exceptions java sont divisées en "vérifiées" et "non vérifiées".

Les méthodes doivent déclarer les exceptions qui sont vérifiées.

```
Void readFile(String s) throws IOException, InterruptedException { … }
```

La méthode appelante peut traiter l'exception ou déclarer qu'elle peut elle-même en rejeter.

Les exceptions qui sont des dérivées de java.lang.RunTimeException ou java.lang.Error ne sont pas vérifiées. Les méthodes n'ont pas besoin de déclarer qu'elles peuvent déclencher ces exceptions.

- Déclencher des exceptions

```
throw new Exception(); throw new Exception("This is my error message");
throw new SecurityException("Accès au fichier untel interdit");
```

Toutes les exceptions ont un constructeur avec un paramètre String.

- La clause finally est exécutée après les autres clauses
- Pas de pénalités en termes de performances

Le langage Java : Tableaux

- Les tableaux
  - Java crée une classe spécifique lors de la déclaration d'une variable de type tableau.
  - L'opérateur [] permet l'accès aux éléments du tableau
  - Création d'une instance de tableau par l'opérateur new
  - Syntaxe

```
int [] arrayOfInts;
int arrayOfInts[];
String [] someStrings;
```

  - Création

```
arrayOfInts = new int[42];
double [] someNumbers = new double[20];
```

Initialisation à la valeur par défaut du type.

La chaîne contient des références aux objets et pas les objets eux-même.

```
Int [] prime = {1,2,3,4,5};
```

  - Utilisation

Le premier indice commence à 0.

La variable publique length donne la longueur du tableau

L'accès après le dernier élément déclenche une exception ArrayIndexOutOfBoundsException.

La méthode System.arraycopy permet de copier deux tableaux

```
System.arraycopy(source, srcstart, destination, dststart, longueur)
```

  - Tableaux anonymes

Créer un tableau temporaire qui n'est plus utilisé/référencé par la suite.

```
setPets(new Animal [] { pokey, squiggles, jasmine } );
```

- Tableaux multidimensionnels

```
Color [][][] rgbCube = new Color [256] [256] [256];
rgbCube [0] [0] [0] = Color.black;
rgbCube [255] [255] [0] = Color.yellow;
```

Les objets en Java

- Introduction
  - Penser les objets en termes d'interface et non d'implémentation.
  - Se placer le plus possible en termes abstraits.
  - Éviter les variables publiques.
  - Utiliser les relations de composition (has-a) et d'héritage (is-a) à bon escient.
  - Minimiser les relations entre objets et organiser les objets en packages (paquets)
- Classes
  - La base de toute application Java.
  - Une classe contient

    des méthodes

    des variables

    un code d'initialisation
  - Exemples

    Définition

```
class Pendule {
  float masse;
  float length = 1.0;
  int cycles;
  float position (float time) {
     …
  }
}
```

    Utilisation

```
Pendule p; p = new Pendule();
p.masse=5.0;
float pos = p.position(1.0);
```

  - Accéder aux membres

    Dans une classe, on accède aux méthodes et aux variables en utilisant directement leurs noms.

    Les autres classes accèdent aux membres à travers une référence.

    Le mot clé private limite l'accès d'un membre à sa classe.

```
Class Pendule {
  private int cycles;
  void resetEverything() {
    cycles = 0;  mass = 1.0;
    …
   float startingPosition = position(0.0);
   …
}
```


Les objets en Java

  - Membres statiques : associés à la classe.

    Variables de classe et méthodes de classe.

    Mot clé: static

```
Class Pendule {
  private int cycles;
  static float gravAccel = 9.80;
  …
  public float getWeight() { return mass* gravAccel; }
}
...
Pendule.gravAccel = 8.76;
```

    Déclaration de constantes par le mot clé final

```
Class Pendule {
  private int cycles;
  static final float EARTH_G = 9.80;
  …
  public float getWeight() { return mass* EARTH_G; }
}
```

- Les méthodes
  - Une méthode doit toujours spécifier un type de retour
  - Une méthode a un nombre fixe d'arguments
  - Les variables locales sont temporaires et connues seulement dans le scope de la méthode.
  - Les variables locales cachent les données membres lorsqu'elles ont le même nom
  - La référence this fait référence à l'objet courant.
  - Les méthodes statiques sont des méthodes de classe
  - Les variables locales doivent être initialisées avant d'être utilisées
  - Pour modifier la référence elle-même dans une méthode, il faut passer l'objet dans un container
  - Il est possible de surcharger des méthodes

Les objets en Java

- Création d'objet
  - Le constructeur est une méthode qui porte le même nom que la classe
  - Le constructeur peut être surchargé

```
class Date {
  long time;
  …
  Date() { time = currentTime(); }
  Date(String date) { time = parseDate(date); }
  …}
```

  - Un constructeur ne peut pas être abstract, synchronized ou final.
  - Si un constructeur appelle un autre constructeur, ce doit être le premier appel dans le constructeur
  - Une bloc de code statique est exécuté une fois lors de l'initialisation de la classe.

```
Class ColorModel {
  static Hashtable colors = new Hashtable();
  // set up colors
  static {
   colors.put("Red", Color.red);
   colors.put("Green", Color.green);
   colors.put("Blue", Color.blue);
  .. }
  }
```

- Destruction d'objet
  - Garbage Collector
    - Java gère la destruction des objets par lui-même
    - Le mécanisme est le Garbage collecting ou Garbage collector
    - Chaque machine virtuelle java peut le gérer comme elle le souhaite
    - Le garbage collector est exécuté dans un thread à basse priorité
    - Il est possible de forcer le GC en exécutant System.gc()
  - Finalization (Destructeur)
    - Le méthode finalize() est appelée pour faire les actions pre-mortem.
    - La méthode est appelée par le Garbage Collector juste avant la destruction de l'objet

Exercice 3

- Définir une classe Personne, avec :
  - Variables privées Nom et Prénom
  - Méthodes: constructeur, display, accesseurs et modificateurs
- Définir une classe Etudiant dérivant de Personne
  - Variables privées: université et bourse(o/n)
  - Méthodes: constructeur, display, accesseurs et modificateurs
- Définir une classe Salarié dérivant de Personne
  - Variables privées: société et salaire
  - Méthodes: constructeur, display, accesseurs et modificateurs
- Ajouter un comptage d'instances en utilisant des variables statiques

Relations entre classes

- Dérivation et héritage

```
class Animal {
  float weight;
  …
  void eat() { … }
}
class Mammal extends Animal {
  int heartRate;
  void breathe() { … }
  ..
}
…
Cat simon = new Cat();
Animal creature = simon;
```

  - Une classe hérite de tous les membres de la classe même (ou superclasse) non marqués private.
  - La référence super permet de faire référence à la superclasse.
- Redéfinition des méthodes
  - SI la méthode de la classe fille a la même signature, alors elle redéfinit la méthode mère.
  - On ne peut pas redéfinir une méthode statique en une méthode non statique
  - Une méthode marquée final ne peut plus être redéfinie
  - Une méthode dérivée doit adhérer ou rafiner les exceptions gérées par la méthode mère.
- Conversion
  - Le casting en java fonctionne sur les références
  - Il permet permet de spécialiser une référence
- Constructeurs
  - super permet d'appeler explicitement un constructeur de la classe mère (superclasse)

```
class Doctor extends Person {
  Doctor(String name, String specialty) {
    super(name);
  }
}
```


Relations entre classes

- Méthodes et classes abstraites
  - Une classe abstraite ne peut pas être instanciée

    Il n'est pas possible de créer un objet dans cette classe
  - Si une classe est déclarée abstraite ou si l'une de ses méthodes est abstraite, la classe ne peut pas être instanciée

```
abstract class truc {
  abstract void machin();
}
```

    Les classes abstraites fournissent un cadre général, qui doit être "rempli" par les classes dérivées.
- Interfaces
  - Une interface est un prototype de classe sans implémentation
  - Elle spécifie un ensemble de points d'entrée qu'une classe doit implémenter, une interface.
  - Exemple

```
interface Driveable {
  boolean startEngine();
  void stopEngine();
  float accelerate(float acc);
  boolean turn(Direction dir);
}
```

    Il est possible d'utiliser le modifieur abstract mais ce n'est pas nécessaire.

    Les méthodes sont toujours public.
  - Le mot clé implements signale qu'une classe implémente une interface.

```
Class Car implements Driveable {
  …
  public boolean startEngine() {
    if (notTooCold) engineRunning = true;
  }
  ..
}
…
Car auto = new Car();
Lawnmower mower = new Lawnmower();
Driveable vehicle;

vehicle = auto;
vehicle.startEngine();
vehicle.stopEngine();
vehicle = mower;
vehicle.startEngine();
vehicle.stopEngine();
```

Relations entre classes

- Interfaces (suite)
  - Les interfaces sont utilisées pour implémenter des callbacks
  - Les interfaces peuvent contenir des constantes (static final)
  - Les mécanismes d'héritage peuvent être utilisés entre les interfaces

```
interface Foo extends Driveable, Edible {
  void good();
}
```

- Package et unité de compilation
  - Unité de compilation

    Le code source d'une classe java est une unité de compilation; Une unité de compilation ne contient qu'une classe publique. Elle doit porter le nom de cette classe publique.

    Une unité de compilation = un fichier avec une extension .java

    Une classe est déclarée appartenir à un package par le mot clé package.

```
package mytools.text;
class TextComponent {
…
}
```

  - Les noms de package sont conceptuellement hiérarchiques. C'est simplement une convention de nommage.
  - Visibilité

    Par défaut une classe n'est accessible qu'aux classes du même package. Pour être disponible elle doit être déclarée publique.
  - L'instruction import permet d'importer des classes ou des packages

```
import mytools.text.TextEitor;
import mytools.text.*;
```

  - Par défaut, une classe est définie dans le package "nameless".

Relations entre classes

- Gestion des accès de base

  Par défaut les variables et méthodes sont accessibles aux classes du même package.

  Public: les membres sont accessibles par tout le monde

  Private: les membres sont accessibles seulement de la classe elle-même

  Protected: les membres sont accessibles dans le même package et dans les classes dérivées. Les membres de la superclasse ne sont accessibles que dans l'objet dérivé.

  Il est possible de redéfinir une méthode private en public

  Il n'est pas possible de redéfinir une méthode public en private

  Les interfaces se comportent comme des classes

- Classes internes (inner classes)

```
class Animal {
  class Brain {
}
void faireQuelqueChose();
}
```

  Les objets Brain peuvent faire des accès aux membres de l'objet Animal dans lequel ils ont été déclarés, auquel ils appartiennent.

  Un objet Brainn'a pas de sens hors de l'animal.

  - Classes internes anonymes

```
new Thread( new Runnable() { public void run() { performSomething(); }
}).start();
```

Exercice 4

- Interfaces
  - Définir une Interface Item, avec la capacité de s'afficher et de se cloner
  - Définir une classe List manipulant des Item, avec les méthodes
    constructeur, ajout en tête, suppression en tête, comptage, affichage des éléments, destructeur
  - Créer des listes de personnes
- Classes abstraites et interfaces
  - Reprise exercice ci-dessus.
  - Création de la classe abstraite conteneur
  - List hérite de conteneur
  - Réaliser un comptage d'instance au niveau conteneur
  - Montrer en créant une nouvelle classe, que l'on peut ajouter d'autres types d'éléments dans une liste.

Utilitaires sur les Objets et les Classes

- La classe Object
  - java.lang.Object est la classe mère de toutes les classes
  - Méthodes communes:

    toString(): appelée pour représenter un objet sous forme de valeur textuelle

    equals(): détermine si deux objets sont équivalents. Est différent de ==, qui fonctionne au niveau référence. Attention l'arguement est de type Object

    hashCode(): par défaut rend un entier unique. S'il existe une notion de classes d'équivalence, il est intéressant de redéfinir cette classe

    clone(): autoriser le clonage et le pratiquer. Méthode protégée. Si l'objet n'est pas clonable, doit renvoyer CloneNotSupportedException. Pour être clonable, un objet doit implémenter l'interface java.lang.Cloneable.

```
class Sneakers extends Shoes {
  public boolean equals(Object arg) {
    if((arg != null) && (arg instanceof Sneakers)) {
      // comparer ici
      return true;
    }
    return false;
  }
}

import java.util.Hashtable;
public class Sheep implements Cloneable {
  Hashtable flock = new Hashtable();
  public Object clone() {
    try {
      return super.clone();
    }
    catch(CloneNotSupportedException e) {
      throw new Error("This should never Happen!");
    }
  }
}
Sheep one = new Sheep();
Sheep anotherOne = (Sheep)one.clone();
```

Attention, il faudrait faire une copie "en profondeur"

```
 public Object clone() {
    try {
      Sheep copy = (Sheep)super.clone();
      copy.flock = (Hashtable)flock.clone();
      return copy;
    }
    catch(CloneNotSupportedException e) {
      throw new Error("This should never Happen!");
    }
  }
```

Utilitaires sur les classes et les Objects

- La classe Object (suite)
  - Méthode clone()

    Il est possible de rendre la méthode publique si nécessaire

- La classe Class
  - La méthode getClass() de Object trend une référence à un objet Class
  - Les classes sont des instances de java.lang.Class.

    Il y a un objet Class pour chaque classe utilisée. Il permet la réflection.
  - Exemple

```
String myString = "Foo!";
Class c = myString.getClass(); // dynamique
Class c = String.class; // statique
System.out.println(c.getName()); // "java.lang.String"
```

    Produire une nouvelle instance d'un objet: newInstance(). Type retour: Object

```
try {
  String s2 = (String)c.newInstance();
}
catch(InstantiationException(e)) { …}
catch(IllegalAccessException(e)) { …}
```

    Recherche l'objet classe correspondant à une chaine

```
try {
  Class sneakersClass = Class.forName("Sneakers");
}
catch(ClassNotFoundException e) { }
```

  - Réflexion (197)

    Le package java.lang.reflect permet au langage "de s'examiner lui-même".

    Méthodes: getFields(), getMethods(), getConstructors(), etc…

Les conventions de nommage

- Nommage

  Les noms doivent être en anglais.

  Les noms doivent être "parlants". Éviter d'utiliser des abréviations. Utiliser la complétion automatique de l'éditeur de texte, pour cela, sous emacs, taper les premières lettres du nom puis M-/.

  Les noms de paquetage ne doivent contenir que des minuscules et chaque mot séparé par un ".".

  Les noms des classes et des interfaces doivent commencer par une majuscule et avoir la forme suivante s'ils contiennent plusieurs noms : MyClass.

  Les noms de classe héritant de la classe Exception doivent se terminer par Exception : MyException.

  Les noms des attributs static final doivent être en majuscule et avoir la forme suivante s'ils contiennent plusieurs noms : MY_CONSTANT.

  Les noms des autres attributs et des méthodes doivent commencer par une minuscule et avoir la forme suivante s'ils contiennent plusieurs noms : myAttribut, myMethod().

- Les recommandations

  Minimiser les * dans les imports.

  Écrire une méthode main pour chacune des classes testable.

  Le main de l'application doit être dans une classe séparée Main.

  Utiliser des interfaces aussi souvent que possible si vous pensez que votre implémentation est susceptible d'être remplacée par une autre ultérieurement.

  Évitez de déclarer des attributs comme public. Implantez plutôt pour un attribut 'attribut', deux méthodes : setAttribut() et getAttribute().

  Initialisez explicitement les attributs.

  Initialisez les objets qui peuvent retourner une Enumeration (Vector, HashTable..) avec autre chose que null.

  Minimisez les static non final.

  Éviter de masquer les attributs par héritage.

  Utilisez la forme Type [] var; pour déclarer les tableaux.

  Écrire des méthodes qui ne font qu'une chose et en utilisent d'autres.

  Déclarer toutes les méthodes public comme synchronized si elles peuvent être utilisée dans un contexte concurrent (threads).

  Déclarer les méthodes qui sont souvent utilisées comme final.

  Pensez à implanter la méthode clone et déclarez la classe comme implantant Cloneable.

  Si possible écrire un constructeur par défaut pour pouvoir utiliser (newInstance()).

  Utilisez equals() plutôt que ==.

  Ne déclarez les variables locales qu'à l'endroit où vous en avez réellement besoin.

  Affectez null à toutes les références qui ne sont plus utilisées.

  Éviter la réutilisation de variable.

  Évitez les affectations à l'intérieur de "()" (if, appel de méthodes,...).

Développement d'applets

Applets

- Programme exécuté à l'intérieur d'un autre.
  - étend java.applet.Applet.
  - La classe applet ne maîtrise pas totalement son exécution, controlée par le navigateur au moyen des méthodes suivantes :
    - public void init() appelée après le chargement de l'applet.
    - public void start() appelée chaque fois que l'on entre dans le document la contenant.
    - public void stop() appelée chaque fois que l'on sort du document la contenant.
    - public void destroy() appelée pour détruire les ressources allouées par l'applet.
    - La méthode stop() ne fait rien par défaut, l'applet pourra donc continuer à s'exécuter en arrière plan.
    - Entre un appel à start() et l'appel suivant à stop() la méthode isAlive retourne true.
  - Une applet est incluse dans un document HTML grâce au tag <APPLET>, un exemple d'inclusion est le suivant :

```
<APPLET CODE="Clock" NAME="Horloge" WIDTH=50 HEIGHT=50>
<PARAM NAME="Couleur" VALUE="bleu">
</APPLET>
```

  - Récupérer les paramètres passés dans la page HTML :
    - public String getParameter(String name)
  - Exécution sûre pour le navigateur l'incluant. Manipulations interdites:
    - accès au système de fichier local
    - lancement de tache au moyen de exec()
    - chargement de librairies ou définition de méthodes natives
    - accès au System.getProperty() donnant des informations sur l'utilisateur, la machine locale.
    - modification des propriétés système
    - accès à un autre groupe de thread.
    - changement de ClassLoader, SocketImplFactory, SecurityManager, ContentHandlerFactory, URLStreamHandlerFactory
    - ouvrir un connexion réseau vers une autre machine que celle dont elle provient
    - accepter des connexions.
  - Récupérer des informations sur le document HTML :
    - public URL getDocumentBase() retourne l'URL de base du document contenant l'applet
    - public URL getCodeBase() retourne l'URL de base de l'applet.

Applets

- Interagir avec le navigateur exécutant l'applet ou avec les autres applets de la page. Il faut récupérer un objet implémentant l'interface AppletContext.
  - public AppletContext getAppletContext()
  - Méthodes de AppletContext
    ```
    public abstract Applet getApplet(String name)
    public abstract Enumeration getApplets()
    ```
  - Le nom à passer en paramètre de la première méthode est le champ NAME donné à l'applet dans le tag HTML (Horloge dans l'exemple précédent).
  - Pour interagir avec le navigateur, il est possible d'appeler les méthodes de l'objet implémentant AppletContext :
    ```
    public abstract void showDocument(URL url)
    public abstract void showDocument(URL url,String target)
    ```
  - qui permettent de charger un nouveau document. La deuxième méthode permet de choisir une frame (_self , _parent, _top) particulière, une nouvelle fenetre (_blank) ou une fenetre de nom particulier.
  - La méthode public void showStatus(String msg) de la classe Applet permet également de visualiser une ligne d'état souvent en bas du navigateur.
- Récupérer simplement des images ou des sons :
  - public Image getImage(URL url)
  - public Image getImage(URL url, String name)
  - public AudioClip getAudioClip(URL url)
  - public AudioClip getAudioClip(URL url, String name)
- Manipuler les sons : méthodes de java.applet.AudioClip :
  - public abstract void play()
  - public abstract void loop()
  - public abstract void stop()
  - Des méthodes sont également disponibles pour jouer directement les sons :
  - public voidplay(URL url)
  - public voidplay(URL url, String name)


```
<HTML>
<HEAD>
</HEAD>
<BODY>
<script language="javascript">
for(i=0;
</script>
<script language="vbscript">
qdgksjqgqsd
</script>
<applet code="toto.class" codebase="http://ww.wanadoo.fr/mon-java">
</applet>
Ceci est un message
</BODY>
</HTML>
```

Threads

Threads

- Qu'est qu'un thread ?
  - Conceptuellement: un flux de calcul dans un programme
  - Plusieurs threads s'exécutent dans la même application
  - Problème: la synchronisation
- La classe Thread et l'interface Runnable
  - Un thread est représenté par une instance de java.lang.Thread.
    Actions possibles: démarrer le thread, le terminer, le suspendre, etc..
  - Il est possible d'étendre la classe Thread, mais ce n'est pas la solution la plus naturelle.
  - Les ordres sont transmis via l'interface java.lang.Runnable

```
public interface Runnable {
  abstract public void run();
}
```

Un thread commence en exécutant la méthode `run` d'un objet particulier

  - Créer et démarrer des threads

```
class Animation implements Runnable {   …
  public void run() {     …
    while(true) {
      // Do something
    }}}
Animation happy = new Animation("Mr Happy");
Thread myThread = new Thread(happy);
myThread.start();
…
```

start est appelée une fois dans la vie d'un thread
stop permet de tuer le thread

```
class Animation implements Runnable {
  Thread myThread;
  Animation (String name) {
    myThread = new Thread(this);
    myThread.start();
  }
…
```

utilisation d'une classe adapter

```
class Animation {
  public void startAnimating() {
    myThread = new Thread (new Runnable() {
      public void run() { drawFrames(); }
    });
    myThread.start();
  }
private void drawFrames() {
  // faire quelquechose
}
```

Les Threads

- Contrôler les threads
  - méthodes

    stop, suspend, resume, sleep , interrupt

```
try {
  Thread.sleep(1000);
  // Thread.currentThread().sleep(1000);
}
catch(InterruptedException e)
{ //do
}
```

    join: bloquer l'appelant jusqu'à complétion. Peut être paramétré par un nombre de millisecondes
- Applets
  - Les applets peuvent être startées et stoppées plusieurs fois.
  - L'applet est démarrée lors de l'affichage, et l'arrête quand l'utilisateur va dans une autre page ou scrolle l'applet hors de vue

```
public class UpdateApplet extends java.applet.Applet
  implements Runnable {
  private Thread updateThread;
  int updateInterval = 1000;

  public void run() {
    while(true) {
      try{
        Thread.sleep(updateInterval);
      } catch(InterruptedException) {}
      repaint();
    }
  }
  public void start() {
    if(updateThread == null) {
      updateThread = new Thread(this); updateThread.start();
  }
  public vois stop() {
    if(updateThread!=null) {
      updateThread.stop();
      updateThread=null;
    }
  }
}
public class Clock extends UpdateApplet {
  public void paint(java.awt.Graphics g) {
    g.drawString(new java.util.Date().toString(), 10, 25);
  }
}
```


Les Threads

- Considérations

  L'applet est tuée à chaque arrêt (dépendant des navigateurs).

  Le problème est que nous n'avons aucun contrôle sur les mouvements de l'utilisateur. Si l'applet est simplement suspendue lorsqu'elle est hors de vue, nous n'avons aucune garantie de la tuer un jour. Dans les cas simples, on garde l'exemple précédent. Dans les cas complexes, on utilise la méthode **destroy** appelée lorsque l'applet va être supprimée (du cache par exemple)

```
public void start() {
  if(updateThread == null) {
    updateThread = new Thread(this);
    updateThread.start();
  }
  else updateThread.resume();
public void stop()
  updateThread.suspend();
}
public void destroy() {
  if(updateThread!=null) {
    updateThread.stop();
    updateThread = null;
}
```

- Synchronisation
  - Utilisation de moniteurs cf (Hoare)
  - Chaque objet dispose d'un verrou (lock).
  - Chaque classe et chaque instance d'une classe peut être verrouillée pour séquentialiser l'accès aux ressources.
  - Le mot clé synchronized indique que le thread doit être seul à exécuter simultanément une action sur l'objet.

    Une seule méthode synchronisée est possible sur l'objet à un instant donné.

```
synchronized int sumRow() {
  return cellA1 + cellA2 + cellA3;
}
```

    Si deux méthodes différentes de la même classe ne peuvent pas s'exécuter simultanément, il suffit de les déclarer les deux synchronized: la ressource partagée est la classe.

    Autre possibilité : poser un verrou sur un objet particulier

```
synchronized (myObject) {
  // des choses qui doivent être protégées
}
```

  - wait() et notify()

    wait et notify sont des méthodes de Object, donc tout le monde en hérité

    wait abandonne le lock et s'endort

    notify rend le lock et réveille le wait

Les Threads

- Exemple

```
class Quelquechose {
synchronized void attente() {
  // je fais quelque chose
  // j'ai besoin d'attente un événement externe pour continuer
  wait();
  // je reprends là où j'ai laissé
}
synchronized void notifierMethod() {
  // faire quelque chose
  // on signale qu'on l'a fait
  notify();
  // faire des choses en plus
}
}
```

- Scheduling et priorités
  - La méthode de scheduling est spécifique à chaque implémentation.
  - Chaque thread a une priorité
  - Par défaut tous les threads ont la même priorité
  - Quand un thread commence, il continue jusqu'à:
    - un appel à sleep ou wait
    - une attente de verrouillage sur une instance
    - une attente en entrée/sortie, type read(),
    - un appel à yield()
    - un arrêt du thread par stop().
  - La plupart des machines virtuelles font du timeslicing
  - Pour tester le timeslicing

```
class PtitThread {
  public static void main(String args[]) {
    new MonThread("coucou").start();
    new MonThread("toto").start();
  }
}
class MonThread extends Thread {
  String message;
  MonThread(String msg) { message = msg; }
  public void run() { while(true) System.out.println(message); }
}
```

  - Priorité

    Méthode Thread.setPriority(x), où x est Thread.MIN_PRIORITY, NORM_PRIORITY, MAX_PRIORITY.

Threads

- Scheduling et priorités (suite)
  - Donner du temps

```
class MonThread extends Thread {
  …
  public void run() {
    while(true) {
      System.out.printmn(message);
      yield();
    }
  }
}
```

Permet de rendre du temps au système pour éviter qu'un thread s'accapare le temps machine

Les threas natifs sont différents

Il est possible de gérer des groupes de threads (cours avancé)

Utilitaires

Classes utilitaires de base

- Strings
  - les strings sont constantes. Pour changer une chaîne il faut en créer une nouvelle
    - length() rend la longueur d'une chaîne
    - concat() concatène deux chaînes
    - + est équivalent à concat

```
String machaine = new String({'L','a','l','a'});
```

    - String.valueOf utilise la méthode toString de Object pour convertir vers une String
    - Méthode Double.valueOf, Integer.valueOf pour convertir depuis des chaînes
    - charAt() rend le caractère à une position donnée
    - toCharArray() rend une chaîne de caractères
    - == compare les références
    - equals compare les chaines
    - equalsIgnoreCase compare en caseInsensitive
    - compareTo fait une comparaison lexicale
    - startsWith et endsWith comparent respectivement le début et la fin d'une chaîne
    - indexOf rend l'indice d'occurrence d'une sous-chaîne
    - trim() suppression des blancs
    - toUpperCase et toLowerCase
    - substring() extraction d'une sous-chaîne
    - replace(): remplacement de sous-chaînes, etc…
  - StringBuffer
    - java.lang.StringBuffer est un buffer qui peut croitre par opposition aux Strings.
    - La méthode append() permet d'ajouter des caractères
    - toSTring() pour se ramener à des chaînes
  - java.util.StringTokenizer
    - décodage d'un buffer, par défaut utilisation des délimiteurs 'blancs'
    - méthodes: hasMoreTokens, nextToken, countTokens

```
String text = "Now is the time for, all goods";
StringTokenizer st = new StringTokenizer(text);
while(st.hasMoreTokens()) {
  String word=st.nextToken();
  ...
}
```

  - java.lang.Math
    - Toutes les méthodes sont statiques et math ne peut pas être instanciée.
    - Méthodes: cos, abs, sin, atan, random, pow, etc…
  - Classes wrapper pour les types de base
    - void -> java.lang.Void, boolean -> java.lang.Boolean, etc…
    - Méthodes de conversion de type
    - Utilisation de conteneurs d'objets

Classes utilitaires de base

- Dates
  - java.util.Date et java.util.GregorianCalendar
  - java.text.DateFormat
- Vector et Hashtables
  - tableau dynamique qui peut grandir pour ajout de nouveaux items.

```
String one = "one";Integer i= new Integer(0);
String two = "two";
String three = "three";
Vector things = new Vector();
things.addElement(one);things.addElement(i);
things.addElement(three);
things.insertElementAt(two, 1);
String bibi = (String)things.firstElement();
```

Méthodes : elementAt(), firstElement(), lastElement(), indexOf(), removeElement(), size()

  - hashtable = dictionnaire

```
Hashtable dates = new Hashtable();
dates.put("christmas", new GregorianCalendar(1997, Calendar.DECEMBER,
25));
dates.put("independence", new GregorianCalendar(1997, Calendar.JULY,
4));
dates.put("groundhog", new GregorianCalendar(1997, Calendar.FEBRUARY,
2));
GregorianCalendar g = (GregorianCalendar )dates.get("christmas");
dates.remove("christmas");
```

Autres méthodes: containsKey(), keys(), elements()

- Properties

Principaux packages

- Principaux packages java

Package	Remarques
Java.applet	Applet qui permet de gérer l'ensemble du comportement des applets
Java.awt	Classes d'IHM de AWT
Java.beans	Classes et interfaces nécessaires à la mise en œuvre des composants Javabeans
Java.io	Entrées/sorties
Java.lang	Classes de base du langage java
Java.math	Classes mathématiques
Java.net	Implémentation des sockets et gestion des urls
Java.rmi	Communication entre deux machines virtuelles java
Java.security	Signatures numériques
Java.sql	Connection à des bases de données via JDBC
Java.text	Formatage de données
Java.util	Outils

Fichiers

Les Streams

- Représentent un canal de communication.
  - Utilisés pour la lecture ou l'écriture depuis un terminal, un fichier ou le réseau.
  - Les données sont écrites ou lues les unes à la suite des autres.
    Les données ne sont pas accédées de façon aléatoire en fonction d'une position absolue, comme c'est le cas pour un tableau, mais séquentiellement, dans l'ordre du flot et relativement à la position courante.
  - Propriétés d'un flot
    les données sont lues dans le flot dans l'ordre où elles ont été écrites ;
    les limites des données ne sont pas préservées par le flot, c'est-à-dire que si l'écriture d'un octet dans le flot est suivie de l'écriture de deux octets, rien n'oblige le lecteur à lire un octet puis deux. Il peut lire, par exemple, deux octets puis un ou toute autre combinaison ;
    un flot peut utiliser un tampon pour améliorer les performances ou encore remettre un ou plusieurs octets dans le flot pour faciliter l'analyse des données qu'il véhicule.
- Les classes et interfaces de manipulation de flot sont regroupées dans le paquetage java.io en deux groupes :
  - celles qui manipulent des octets
  - celles qui manipulent des caractères.
- Les caractères nécessitent un traitement particulier en Java car ils sont codés sur deux octets (UNICODE).
- Toutes les méthodes sur les flots peuvent renvoyer IOException

Flots d'octets

- Toutes les classes qui manipulent des flots d'octets héritent de InputStream ou OutputStream

  Les entrées-sortie standards sont des flots d'octets. Ils sont accessibles comme des membres statiques (in, out et err) de la classe java.lang.System.

```
InputStream stdin = System.in;
OutputStream stdout = System.out;
```

  - Les méthodes d'InputStream

    int read() retourne l'octet suivant dans le flot ou -1 si la fin du flot est atteinte ;

    int read(byte[] b) lit dans le flot au plus b.length octets qui sont placés dans le tableau b. Les octets sont stockés dans le tableau dans l'ordre de lecture, de la première case vers la dernière. Le nombre d'octets effectivement lu est retourné ou -1, si la fin du flot est atteinte ;

    int read(byte[] b, int offset, int len) est équivalente à la méthode précédente si ce n'est que les octets sont stockés à partir de la case d'indice offset et qu'au plus len octets sont lus.

  - Les méthodes d'OuputStream

    void write(int b) permet d'écrire l'octet de poids faible de b dans le flot ;

    void write(byte[] b) écrit dans le flot les b.length octets stockés dans le tableau. L'ordre d'écriture des octets dans le flot est celui du tableau ;

    void write(byte[] b, int offset, int len) a le même comportement que la méthode précédente, mais écrit len octets de b en partant de la case d'indice offset .

```
try{
   byte b;
   int val = System.in.read();
   if(val != -1)
      b= (byte)val;
   System.out.write(b);
}
catch(IOException e)
```

    La méthode void close() permet de libérer les ressources associées au flot.

- Les flots permettent un accès séquentiel aux données.

  Parfois, il est utile de modifier ce comportement. La méthode long skip(long n) permet de consommer les n premiers octets d'un flot. Le nombre d'octets effectivement consommé est retourné par cette méthode.

  Certains flots supportent un marquage en vue d'un retour arrière (boolean markSupported()).

  mark(int taillemax) et reset() permettent, respectivement, le marquage de la position courante dans le flot et un retour arrière vers la position préalablement marquée.

```
if(in.markSupported()){
  in.mark(100);
  // Diverses actions sur le flot
  in.reset(); // Retourne à la position marquée si le nombre
              // d'octets consommés est inférieur à 100
}
```


Les flots de caractères

- Les classes de flots de caractères dérivent des classes abstraites Reader et Writer
- Les méthodes de cette classe sont équivalentes à celles des classes InputStream et OuputStream ; seul le type des données lues est différent.
- Dans un flot de caractères, l'unité de lecture n'est plus l'octet mais le caractère. Ainsi, les méthodes read() assurent la lecture de deux octets insécables.
- Le passage d'un flot d'octets à un flot de caractères se fait relativement à un codage au moyen des classes OutputStreamWriter de InputStreamReader.

```
import java.io.*;
public class Converter {
  String fromEnc;
  String toEnc;
  public Converter(String fromEnc, String toEnc) {
    this.fromEnc = fromEnc;
    this.toEnc = toEnc;
  }
  public void convert(InputStream in, OutputStream out)
    throws IOException {
    InputStreamReader reader = new InputStreamReader(in, fromEnc);
    OutputStreamWriter writer = new OutputStreamWriter(out, toEnc);
    char[] tampon = new char[256];
    int nbLus;
    while((nbLus = reader.read(tampon))!= -1) {
      writer.write(tampon,0,nbLus);
    }
    writer.close();
    reader.close();
  }
  public static void main(String[] args) throws Exception{
    Converter conv = new Converter("8859_1","Cp037");
    conv.convert(System.in,System.out);
  }
}
```

Les flots concrets

- Classes qui permettent de créer des instances de flots liées à des ressources concrètes telles que des fichiers ou des tableaux.
- Quatre principaux types de flots :
  - flots construits sur des tableaux, qui sont ByteArrayInputStream, ByteArrayOutputStream, CharArrayReader et CharArrayWriter. Les flots sur des tableaux en écriture sont très utiles pour construire des tableaux dont la taille n'est pas connue au moment où les premières données à stocker sont obtenues ;

```
ByteArrayOutputStream out = new ByteArrayOutputStream();
int b;
while((b = System.in.read())!=-1) {
    out.write(b);             // La taille du tampon augmente
}
byte[] tab = out.toByteArray();
out.reset();                  // Les données sont vidées du tampon
```

  - les flots construits sur des chaînes de caractères, qui sont StringReader et StringWriter ;

```
String str = "ABC";
StringReader reader = new StringReader(str);
int c;
while((c = reader.read())!= -1) {
  System.out.println((char)c);
}                             // Affiche A, puis B et C
```

  - Les tubes, qui sont PipedInputStream, PipedOutputStream, PipedReader et PipedWriter. Un flot en lecture est connecté à un flot en écriture (ou l'inverse) en passant ce dernier en paramètre du constructeur du flot.

```
PipedWriter out = new PipedWriter();
PipedReader in = new PipedReader(out);
out.write('A');
System.out.println((char)in.read()); // Affiche A
```

  - les flots sur des fichiers.

Les fichiers

- La classe File encapsule les accès aux informations relatives au système de fichiers.

```
import java.io.*;

public class ListDir {
  public static void main(String[] args) throws Exception {
    for(int i=0;i<args.length;i++){
      listDirectory(args[i]);
    }
  }
  public static void listDirectory(String dirName)
                      throws FileNotFoundException {
    File f = new File(dirName);
    if (!f.exists() || !f.isDirectory()){
      throw new FileNotFoundException();
    }
    String[] ls =f.list();
    for(int i=0;i< ls.length;i++){
      StringBuffer s = new StringBuffer(20);
      File file = new File(ls[i]);
      s.append(file.isDirectory()?"d":"-");
      s.append(file.canRead()?"r":"-");
      s.append(file.canWrite()?"w\t":"-\t");
      s.append(file.length());
      s.append("\t");
      s.append(file.getName());
      System.out.println(s);
    }
  }
}
```

- Les classes FileReader, FileWriter, FileInputStream, FileOutputStream permettent de créer des flots concrets associés à des fichiers.

  Les classes FileReader et FileWriter (classes de commodité) utilisent le codage de la plate-forme pour écrire les caractères dans le fichier.

- Constructeur prend en argument le nom du fichier ou un objet File encapsulant la référence du fichier. En lecture, le flot est systématiquement placé en début de fichier. En écriture, le flot est placé en début du fichier préalablement tronqué, ou placé en fin de fichier

```
FileInputStream fistream = new FileInputStream(new File("toto"));
FileOutputStream fostream = new FileOutputStream("toto",true); //
Ajout à la fin du fichier
```

Les flots

- Les fichiers
  - La classe RandomAccessFile permet à la fois la lecture et l'écriture dans le même flot.
    La lecture ou l'écriture s'effectue à la position courante dans le fichier (récupérable par la méthode getFilePointer()).
  - La position courante est ensuite augmentée de la taille de la donnée lue ou écrite.
    Ce flot permet également un déplacement aléatoire dans le fichier grâce à la méthode void seek(long pos).
    ```
    RandomAccessFile raf = new RandomAccessFile("toto","rw"),
    ```
- Les filtres
  - Pour faciliter leur utilisation, il est possible positionner un filtre sur un flot. Un filtre est un flot particulier qui enveloppe un autre flot. Les données lues dans un filtre proviennent du flot enveloppé.
    Les filtres peuvent hériter des classes abstraites FilterInputStream, FilterOutputStream, FilterReader ou FilterWriter.
    Le flot enveloppé est stocké dans le champ in (resp. out) pour les flots en lecture (resp. en écriture).

- DataInputStream et DataOutputStream
  - Lecture et écriture binaire des types primitifs (int, long, etc.), ainsi que des chaînes de caractères, indépendamment de la plate-forme.

```
ByteArrayOutputStream out = new ByteArrayOutputStream();
DataOutputStream dataOut = new DataOutputStream(out);
int i = 10;
double d = 3.14;
dataOut.writeInt(i);
dataOut.writeDouble(d);
dataOut.flush();
byte [] tab = out.toByteArray();
// Taille de tab = taille int (4) + taille double (8)
System.out.println(tab.length);    // Affiche 12
```


Les flots

- LineNumberReader
  La classe LineNumberReader lit un flot de caractères ligne par ligne grâce à la méthode readLine(). Ce filtre permet également de connaître le numéro de la ligne courante.

```
InputStreamReader systemReader = new InputStreamReader(System.in);
LineNumberReader reader = new LineNumberReader(systemReader);
String line;
while((line = reader.readLine())!=null) {
  // Affiche les lignes lues sur l'entrée standard
  // et les affiche précédées de leur numéro
  System.out.println(reader.getLineNumber() + ":\t" + line);
}
```

- ObjectInputStream et ObjectOutputStream
  - Ces interfaces généralisent les précédentes en permettant de lire aussi des objets.
  - Les objets écrits ou lus doivent implémenter l'interface java.io.Serializable ou java.io.Externalizable.

```
ObjectInputStream p = new ObjectInputStream(new
FileInputStream("tmp"));
int i = p.readInt();
String today = (String)p.readObject();
Date date = (Date)p.readObject();
p.close();
```

- PrintReader ou PrintStream
  - Permet d'écrire simplement toute variable sous forme chaîne de caractères.
    Dans le cas des objets la méthode toString (qui peut être masquée) est appelée.
    Contrairement aux autres flux il n'y a pas d'exception levée mais une méthode checkError().
  - PrintStream fait les conversions de caractères par défaut en ISO8859-1.

```
PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out,
8859_1), true);
out.println(1);
out.println("un");
out.println(out);
```

Les tampons

- Pour améliorer les performances des entrées-sorties il est préférable d'utiliser des tampons de lecture et d'écriture.
- Un tel tampon peut permettre le marquage et le retour arrière pour certains flots qui ne le supportent pas par défaut.

```
BufferInputStream bu = new BufferInputStream(System.in,256);
bu.mark(10);
```

- Lecture de données sous forme de chaîne de caractères
  - Le filtre StreamTokenizer permet de lire de façon simple des valeurs numériques entrées sous forme de chaîne de caractères.

```
import java.io.*;
import java.util.*;
public class Polonaise{
  public static void main(String[] arg) throws IOException {
    Stack stack = new Stack();
     Double value;
    StreamTokenizer in =
     new StreamTokenizer(new InputStreamReader(System.in));
    in.wordChars('+','+');
    in.wordChars('-','-');
    in.wordChars('/','/');
    in.wordChars('*','*');
    in.eolIsSignificant(false);
    do {
      in.nextToken();
      if(in.ttype == StreamTokenizer.TT_NUMBER){
        stack.push(new Double(in.nval));
      }
      if(in.ttype == StreamTokenizer.TT_WORD) {
        if(in.sval.length() == 1) {
          switch(in.sval.charAt(0)) {
          case '+':
            try{
              value = new Double(((Double)stack.pop()).doubleValue()
                          + ((Double)stack.pop()).doubleValue());
              stack.push(value);
              System.out.println(value);
          }
          catch(EmptyStackException e)
          {
               System.out.println("Empty stack!");
          }
          break;
          ....
          }}}}
    while(in.ttype != StreamTokenizer.TT_EOF);
  }}
```


La sérialisation

La sérialisation

- Le mécanisme de sérialisation permet d'enregistrer ou de récupérer des objets dans un flux: persistance des objets ou envoi d'objets sur le réseau.

  La sérialisation d'un objet est effectuée lors de l'appel de la méthode writeObject() sur un objet implémentant l'interface ObjectOutput (par exemple un objet de la classe ObjectOutputStream).

  La désérialisation d'un objet est effectuée lors de l'appel de la méthode readObject() sur un objet implantant l'interface ObjectInput (par exemple un objet de la classe ObjectInputStream).

  Lorsqu'un objet est sauvé, tous les objet qui peuvent être atteints depuis cet objet sont également sauvés. En particulier si l'on sauve le premier élément d'une liste tous les éléments sont sauvés.

  Un objet n'est sauvé qu'une fois grâce à un mécanisme de cache. Il est possible de sauver une liste circulaire.

  Pour qu'un objet puisse être sauvé ou récupéré sans déclencher une exception NotSerializableException il doit implémenter l'interface Serializable ou l'interface Externalizable.

  L'interface Serializable ne contient pas de méthode. Tout objet implantant l'interface Serializable peut être enregistré ou récupéré même si une version différente de sa classe (mais compatible) est présente. Le comportement par défaut est de sauvegarder dans le flux tous les champs qui ne sont pas static, ni transient. Des informations sur la classe (nom, version), le type et le nom des champs sont également sauvegardées afin de permettre la récupération de l'objet.

Sérialisation (exemple)

```
import java.io.*;
public class Point implements Serializable {
  int x;
  int y;
  public Point(int x, int y){
    this.x = x;
    this.y = y;
  }
  public String toString() {
    return "(" + x + "," + y + ")";
  }
  public static void main(String [] args)
    throws Exception {
    Point a = new Point(1,2);
    File f = File.createTempFile("tempFile","test");
    f.deleteOnExit(); // Effacer le fichier à la fin du programme
    ObjectOutputStream out =
      new ObjectOutputStream(
          new FileOutputStream(f));
    out.writeObject(a); // Écriture de l'objet
    out.close();
    ObjectInputStream in = new ObjectInputStream(
      new FileInputStream(f));
    Point aPrime = (Point) in.readObject(); // Lecture de l'objet
    in.close();
    System.out.println("a = " + a);
    System.out.println("aPrime = " + aPrime);
    System.out.println("Equals = " + a.equals(aPrime));
  }
}
```

```
Produit :

% java Point
a = (1,2)
aPrime = (1,2)
Equals = false
```

```
Vector v = new Vector();
Point a=new Point(10,10);
v.addElement(new Point(10,20));
v.addElement(a);
v.addElement(new String("kjsdfhjhsd"));
v.addElement(a);
out.writeObject(v);
Vector v2=(Vector)in.readObject();
if(v2.elementAt(1)==v2.elementAt(3))
 // j'ai gagné
```

Sérialisation (suite)

- L'objet récupéré est une copie de l'objet sauvé.
- Si l'objet implante les méthodes :

  private void writeObject(ObjectInputStream s) throws IOException

  private void readObject(ObjectOutputStream s) throws IOException

  celles-ci sont appelées pour sauvegarder ou lire les champs propres de la classe (pas ceux de la classe héritée) plutôt que les méthodes par défaut. Ceci peut servir pour sauvegarder des informations complémentaires ou sauvegarder des champs non sérialisables.

```
Les méthodes suivantes sont équivalentes à la sauvegarde par défaut :
private void writeObject(ObjectInputStream s) throws IOException{
s.defaultWriteObject();
}
private void readObject(ObjectOutputStream s) throws IOException {
s.defaultReadObject();
}
```

- L'interface Externalizable laisse toute liberté (et problèmes) à l'utilisateur. Utiliser les méthodes :

```
public void writeExternal(ObjectOutput s) thows IOException
public void readExternal(ObjectInput s) thows IOException
```

  Toutes les sauvegardes et lecture, en particulier celles des super-classes, sont laissées à la charge de l'utilisateur. De plus toute sous-classe peut masquer ces méthodes ce qui peut poser des problèmes de sécurité.

- Une certain nombre de classes ne sont pas sérialisables :

  Thread

  InputStream et OutputStream

  Peer

  JDBC

La sauvegarde des objets

- Chaque fois qu'un objet est sauvé dans un flux, un objet handle, unique pour ce flux, est également sauvé.

  Ce handle est attaché à l'objet dans une table de hashage. Chaque fois que l'on demande de sauver à nouveau l'objet, seul le handle est sauvé dans le flux. Ceci permet de casser les circularités.

- Les objets de la classe Class sont sauvegardés et restaurés de façon particulière.
  - L'objet Class n'est pas sauvé ou récupéré directement dans le flux. Un objet de la classe ObjectStreamClass qui lui correspond prend sa place. Cette classe contient les méthodes suivante :

    public static ObjectStreamClass lookup(Class cl) retourne l'objet ObjectStreamClass qui correspond à l'objet Class passé en paramètre si la classe implémente java.io.Serializable ou java.io.Externalizable.

    public String getName() retourne le nom de la classe correspondante.

    public long getSerialVersionUID() retourne le numéro de version de l'objet classe. Deux classes sont compatibles si elles ont même numéro de version.

    public Class forClass() retourne l'objet Class qui correspond à l'objet courant.

    public String toString() affiche une description de l'objet.

  - Lors de l'écriture dans le flux d'un tel objet sont écrits dans le flux :

    son nom ;

    le serialVersionUID ;

    le nombre de champs sauvables ;

    la liste des noms, types et modifieur des champs ;

    l'objet ObjectStreamClass de la super-classe.

    Il est possible d'ajouter des informations dans le flux. Par exemple une URL où récupérer le .class correspondant, comme dans le cas des RMI, ou bien le .class lui-même. Pour cela il faut masquer dans la classe ObjectOutputStream la méthode : protected void annotateClass(Class cl) throws IOException

La sérialisation (exemple)

```
package test.serial;
import java.io.*;
import util.*;

public class MyObjectOutputStream extends ObjectOutputStream{
  String baseUrl;

  public MyObjectOutputStream(OutputStream out, String baseUrl) throws
IOException{
    super(out);
    this.baseUrl = baseUrl;
  }

  protected void annotateClass(Class cl) throws IOException{
    this.writeObject(baseUrl);
  }

  public static void main(String[] args) throws IOException {
    MyObjectOutputStream out = new MyObjectOutputStream(new
FileOutputStream(args[0]), "http://massena.univ-
mlv.fr/~roussel/CLASSES");
    out.writeObject(new Cons(new Integer(1),new Nil()));
    out.close();
  }
}
```

Une fois compilé on lance :

```
prompt> java test.serial.ObjectOutputStream fichier
puis si l'on lance
prompt> strings fichier
on obtient
util.Cons
elemt
Ljava/lang/Object;L
tailt
Lutil/List;t
+http://massena.univ-mlv.fr/~roussel/CLASSESxpsr
java.lang.Integer
valueq
java.lang.Number
util.Nil?
```

Lors de la lecture dans le flux des informations concernant la classe, l'objet ObjectStreamClass est reconstruit et la méthode :
protected Class resolveClass(ObjectStreamClass v) throws IOException, ClassNotFoundException
est appelée. Celle-ci retourne un objet Class dont serialVersionUID est comparé à celui trouvé dans le flux. Par défaut cette méthode appelle le ClassLoader standard. Il est possible masquer cette méthode pour par exemple utiliser les informations utilisées sauvées dans le flux au moyen de annotateClass() pour récupérer la classe.

Exemple utilisant la classe util.NetworkClassLoader:

```
package test.serial;
import java.io.*;
import java.util.*;
import util.*;

public class MyObjectInputStream extends ObjectInputStream{
  Hashtable hashtable = new Hashtable();

  public MyObjectInputStream(InputStream in) throws IOException{
    super(in);
  }

  protected Class resolveClass(ObjectStreamClass v) throws
IOException, ClassNotFoundException  {
    String baseUrl = (String) this.readObject();
    ClassLoader loader = (ClassLoader) hashtable.get(baseUrl);
    if(loader == null) {
      loader = new NetworkClassLoader(baseUrl);
      hashtable.put(baseUrl,loader);
    }
    if (loader == null)
      throw new ClassNotFoundException();
    return loader.loadClass(v.getName());
  }

  public static void main(String[] args) {
    try{
      MyObjectInputStream in = new MyObjectInputStream(new
FileInputStream(args[0]));
      List l = (List) in.readObject();
      System.out.println(l);
      in.close();
    }
    catch(Exception e){
      e.printStackTrace();
    }
  }
}
```

Une fois compilé on lance :

```
prompt> java test.serial.ObjectInputStream fichier
et on obtient :
Loading http://massena.univ-mlv.fr/~roussel/CLASSES/util/Cons.class...
Loading http://massena.univ-mlv.fr/~roussel/CLASSES/util/Nil.class...
Cons(1, Nil())
```

La sécurité

- La sécurité au niveau des classes chargées est la même que celle qui existe par défaut. En effet, le code des classes n'est pas par défaut stocké dans le flux mais récupéré par un mécanisme externe que peut contrôler l'utilisateur.
- Tous les champs sont a priori sauvés dans le flux, en particulier les champs private. Par défaut, une classe ne peut pas être sauvée, il faut qu'elle implémente une des interfaces Serializable ou Externalizable, les objets de classes vraiment sensibles ne peuvent donc pas être sauvés (sauf demande explicite du programmeur). Si le programmeur veut quand même sauver un tel objet il doit marquer transient les champs qu'il ne vaut pas voir apparaître pas dans le flux.

Le contrôle de version

- Java propose un mécanisme permettant de "résoudre" le problème de la lecture d'un objet d'une classe qui a évolué depuis que l'objet a été sauvegardé.

  Par défaut une exception ClassNotFoundException est levée si les deux classes sont différentes.
  Cas où la nouvelle classe pourrait être utilisée à la place de la précédente:
  Les changements ne concerne que des méthodes, des champs static ou transient.
  Des champs ont été ajoutés. Ils peuvent être initialisés par défaut ou par la méthode readObject().
  Des classes ont été ajoutées à la hiérarchie ou un implement Serializable a été ajouté . Les champs associés peuvent être initialisés par défaut ou par readObject().
  Des classes ont été retirées à la hiérarchie ou un implement Serializablea été retiré. Les champs associés dans le flux peuvent être lus et détruits s'ils sont de type primitif et créés si ce sont des objets.
  Une méthode readObject() a été ajoutée qui commence par appeler defaultReadObject() et qui gère les exception pour les données supplémentaires.
  La méthode readObject() n'existe plus. Il est possible d'essayer de lire l'objet avec la méthode defaultReadObject() en détruisant les champs non-attentdus.
  Dans ces cas là, il est possible de spécifier qu'une classe est compatible avec la précédente. Par cela il suffit de récupérer le serialVersionUID de la classe précédente est d'ajouter un champ : static final long serialVersionUID = SUID de la classe précédente; dans la nouvelle classe.
  Si dans une classe un tel champ n'est pas spécifié dans une classe le serialVersionUID est calculé par un hashage (Secure Hash Algortihtm -1 du National Institute of Standards and Technology) sur le nom de la classe, ses modifieurs , ces interfaces, les noms et modifieurs des champs et méthodes.
  La valeur de ce hashage donne une clef de 64bit qui est récupérable par la commande serialver -show

  - Les objets de la classe String sont sauvés sont la forme UTF-8.
  - Les tableaux sont sauvés sous la forme : ObjectStreamClass correspondant, suivi de la taille du tableau et des éléments.
  - Pour les autres objets le mécanisme de sauvegarde est le suivant si l'objet est sérialisable :

    sauvegarde de l'objet ObjectStreamClass de la classe la plus précise de l'objet implémentant Serializable ;
    Pour cette classe et ses super-classes les champs sont sauvés dans un ordre particulier par la méthode defaultWriteObject ou par la méthode writeObject si elle a été définie (la méthode readObject devra les lire dans le même ordre). Si les champs sont d'un type primitif la méthode write* correspondante est appelée.
    Si la classe implémente Externalizable , seule la méthode writeExternal est appelée.
    Dans chacun de ces cas si l'objet n'est pas sauvable (n'implémente pas Serializable , ni Externalizable) un exception NotSerializableException est retournée.

Abstract Window Toolkit

Création d'interfaces avec AWT

- Vue d'ensemble
  - La partie graphique est un ensemble de composants imbriqués
  - Cette imbrication créée une hiérarchie
- Principaux composants
  - Conteneurs (containers)
    Composants pouvant contenir d'autres composants. La forme la plus courante est le panneau (panel).
  - Les canevas (canvases)
    Surface de dessin.
  - Les composants d'interface.
    Boutons, listes, simples menus popup, cases à cocher, etc…
  - Composant de construction de fenêtres
    Décor: fenêtre, cadfe, barres de menu, boîtes de dialogue.
  - La racine des composants est Component
- Exemple: ajout d'un composant à une applet

```
public void init() {
  Button b = new Button("OK");
  add(b);
}
```

  - La position du composant à l'écran dépend du gestionnaire de fenêtre.
- Etiquettes
  - Label: chaîne de texte non modifiable
  - Création
    Label(), Label(String), Label(String, int). Int est une constante d'alignement (Label.LEFT, Label.RIGHT, Label.CENTER).
  - Police de caractères
    La méthode setFont est appelée sur l'étiquette ou sur le conteneur père pour changer la police pour tous les composants du conteneur.
  - Exemple

```
import java.awt.*;
public class LabelTest extends java.applet.Applet {
  public void init() {
    setFont(new Font("Helvetica", Font.BOLD, 14));
    setLayout(new GridLayout(3,1));
 add(new Label("aligned left", Label.LEFT));
 add(new Label("aligned center", Label.CENTER));
```

Abstract Window Toolkit (java.awt)

- **Les composants**
  L'AWT est une API qui permet de construire des interfaces graphiques (*gui*) indépendamment de la plate-forme.
  Une interface graphique est composée de composants (Component) simples et standards qui sont créés au moyen d'une de ces classes :
  - Button un bouton
    ```
    button = new Button("Bouton");
    ```
  - Canvas une zone de dessin
    ```
    canvas = new Canvas();
    ```
  - Checkbox une case à cocher
    ```
    checkbox = new Checkbox("un");
    ```
  - des CheckBox peuvent être regroupées dans un CheckboxGroup
    ```
    group = new CheckboxGroup();
    checkbox.setCheckboxGroup(group);
    ```
  - Choice un menu de sélection
    ```
    choice = new Choice();
    choice.addItem("un");
    choice.addItem("deux");
    ```
  - Label un texte
    ```
    label =new Label("texte");
    ```
  - List une liste
    ```
    list = new List();
    list.add("un");
    list.add("deux");
    ```
  - Scrollbar un ascenseur
    ```
    s = new Scrollbar(Scrollbar.VERTICAL,ini,pas,min,max);
    s.getValue();
    ```
  - TextArea une zone de saisie de texte
    ```
    texte = new TextArea("Hello",5,40,SCROLLBARS_BOTH);
    texte.append(" World");
    texte.getText();
    ```
  - TextField une ligne de saisie de texte
    ```
    field = new TextField("texte",taille);
    ```


Les conteneurs

- Les composants simples sont inclus dans un conteneur (Container) qui est construit au moyen d'une des classes suivantes. Les composants et certains conteneurs sont inclus dans les conteneurs au moyen de la méthode add() (parfois surchargée).
  - Frame une fenêtre
    ```
    frame = new Frame("Titre");
    frame.add(bouton,"North");
    frame.pack();
    frame.show();
    ```
  - Dialog une fenêtre de dialogue (FileDialog permet de sélectionner un fichier)
    ```
    modal = new Dialog(frame,"Titre",true);
    modal.add("North",new Label("Cliquez !");
    modal.add("Center",new Button("Ok"));
    ```
  - Panel une zone de fenêtre
    ```
    panel = new Panel();
    panel.add(new Button("Oui"));
    panel.add(new Button("Non"));
    ```
  - ScrollPane une zone avec ascenseur d'une fenêtre
    ```
    pane = new ScrollPane();
    pane.add(new Canvas());
    ```
- Les barres de menus
  - Seule une Frame peut contenir une barre de menu (Menubar). Celle-ci contient des Menu (ajoutés avec la méthode add()) eux-mêmes constiués de MenuItem, CheckboxMenuItem ou de Menu. Elle est ajoutée à la fenêtre au moyen de la méthode setMenuBar().
  - Tous les composants peuvent contenir un PopUpMenu ajouté au moyen de la méthode add(). Celui-ci est un Menu et peut contenir des MenuItem, des CheckboxMenu et des Menu.
  - Des raccourcis clavier (MenuShortcut) peuvent être associés aux menus.
    ```
    menu = new Menu("Fichier");
    menu.add(new MenuItem("Ouvrir",new MenuShortcut(KeyEvent.VK_F8)));
    menuBar = new MenuBar();
    menuBar.add(menu);
    frame.setMenuBar(menuBar);
    popup = new PopupMenu(); popup.add(menu);
    ```

Le positionnement des composants

- Généralités
  - Le positionnement de chacun des composants d'une fenêtre est effectué lors de l'initialisation ou de la modification de la taille de la fenêtre grâce à un layout qui se charge suivant son type de disposer les composants.
  - Il existe des layouts par défaut qui sont associés à chaque conteneur.
  - Il est possible de le changer au moyen de la méthode setLayout().
  - Dans une fenêtre plusieurs layout sont utilisés en imbriquant des Panel. Pour forcer le recalcul du placement il faut appeler la méthode validate() sur le conteneur.
- Il existe plusieurs type de layout :

  BorderLayout arrange les composants en fonction d'une position relative (North, South, East, West ou Center)

```
panel.setLayout(new BorderLayout());
panel.add(new Button("Ok"),"South"); //1.1
panel.add("North", new Button("Ok")); //1.0
```

  CardLayout arrange les composants en onglets

```
panel.setLayout(new CardLayout());
```

  FlowLayout arrange les composants de gauche à droite

```
frame.setLayout(new FlowLayout(FlowLayout.LEFT));
```

  GridLayout arrange les composants dans une grille de taille fixe de gauche à droite et de haut en bas

```
frame.setLayout(new GridLayout(2,3));
```

  GridBagLayout arrange les composants relativement à une grille (à éviter)

  null les composants sont alors positionnés de façon absolue grâce à setLocation().

- La taille des composants
  - Calcul

    La taille des composants et des conteneurs est calculée automatiquement (i.e. taille du bouton relative au label de celui-ci).

    Pour ceux, comme les Canvas qui ont à priori une taille nulle laméthode setSize() est utilisée ou la méthode getPreferredSize() est masquée.

    La méthode pack() appliquée à une fenêtre force tous ses composants et elle-même à prendre leur taille "idéale".
  - L'affichage

    Une fenêtre et ses composants devient visible (et/ou passe au premier plan) grâce à la méthode show().

    Un composant ou un conteneur peut être rendu visible ou non au moyen de la méthode setVisible() .

La gestion des événements

- La version 1.1 de la gestion des événements est basée sur la délégation. Dérivés de AWTEvent.
  - les événements bas-niveau ComponentEvent
    - FocusEvent
    - InputEvent
    - KeyEvent
    - MouseEvent
    - ContainerEvent
    - WindowEvent
  - Les événements sémantiques
    - ActionEvent
    - AdjustementEvent
    - ItemEvent
    - TextEvent
  - L'événement contient son type getID() et toutes les informations relatives à ce type dans la classe correspondante.
  - Les événements sont envoyés aux composants au moyen de la méthode dispatchEvent().
  - L'événement n'est pas géré au niveau du composant mais délégué à un autre objet qui s'est préalablement déclaré au moyen des méthodes

    add<EvenType>Listener() ou set<EventType>Listener() comme intéressé par l'événement. Celui-ci doit implanter <EventType>Listener ou hériter de <EventType>Adapter.
  - Les différents <EventType> sont :
    - Component dans Component
    - Focus dans Component
    - Key dans Component
    - Mouse dans Component
    - MouseMotion dans Component
    - Container dans Container
    - Window dans Dialog et Frame
    - Action dans MenuItem, Button, TextField et List
    - Item dans Choice, Checkbox, CheckBoxMenuItem et List
    - Text dans TextField et TextArea
    - Adjustement dans Scrollbar

La gestion des événements (suite)

- Si il existe plusieurs listeners l'ordre d'envoi n'est pas normé. Tous les événements qui peuvent être modifiés (disposant d'une méthode set*) sont clonés pour chaque listener. L'envoi des événements est synchrone.

```
class Listener implements ActionListener {
 public void actionPerformed(ActionEvent e){...}
}
class GUI extends Frame {
 Button b = Button("Action");
 b.addActionListener(new Listener());
 pack();
 show();
}
```

- Gains de performance car le composant peut ignorer tous les événements sur lesquels aucun listener n'est enregistré.
- Il est encore possible de gérer les événements par héritage. Pour cela un composant doit être dérivé et masquer, soit une méthode appelée pour un certain événement process<EventType>Event, soit la méthode de sélection globale processEvent().
- La méthode qui masque doit appeler la méthode de la super-classe afin d'assurer le bon traitement des événements.
- Si aucun listener n'est enregistré l'événement est ignoré il faut donc accepter les événements du type désiré au moyen de la méthode enableEvents().

```
class GUI extends Frame {
 public GUI(String title) {
 super(title);
 enableEvents(AWTEvent.FOCUS_EVENT_MASK);
 }
 protected void processFocusEvent(FocusEvent e) {
 …
 super.processFocusEvent(e);
 }
}
```

- Le composant est rendu actif ou inactif au moyen de la méthode setEnabled().

Dessiner

- Il est possible de dessiner dans tous les composants, mais celui qui est défini pour cela est Canvas.
- Trois méthodes:
  - repaint() demande d'afficher de nouveau le contenu dès que possible (plusieurs demandes peuvent être regroupées)
  - update() efface le contenu du composant définit le rectangle de dessin et appelle paint()
  - paint() définit quoi afficher
  - Les méthodes paint() et update() prennent en paramètre le contexte graphique de la classe Graphics. C'est à lui que sont associés :
    - la zone de dessin draw*() ou fill*()
    - la fonte setFont() et getFont()
    - le rectangle de mise à jour setClip() et getClip() (getClipRect() et clipRect() en 1.0)
    - la couleur setColor() et getColor()
    - l'opération (XOR ou paint) setXORMode() et setPaintMode()
    - la couleur du XOR

Dessiner (exemple)

```
package test;
import java.awt.*;
import java.awt.event.*;
/**
 * @author Gilles ROUSSEL
 * @version 0.1
 */
public class MyCanvas extends Canvas
{
  static final int WIDTH = 1000;
  static final int HEIGHT = 1000;
  Image draw;

  public MyCanvas() {
    setSize(WIDTH, HEIGHT);
    setBackground(Color.white);
    setForeground(Color.black);
  }
  public void update(Graphics g) {
    paint(g);
  }
  public void paint(Graphics g) {
    if(draw == null) {
      draw = createImage(WIDTH,HEIGHT);
      Listener listener = new Listener(draw.getGraphics());
      addMouseMotionListener(listener);
      addMouseListener(listener);
    }
    g.drawImage(draw,0,0,null);
  }

  class Listener extends MouseMotionAdapter
                 implements MouseListener {
    int prevX;
    int prevY;
    Graphics graphics;
    Listener(Graphics graphics) {
      this.graphics = graphics;
    }
```


Dessiner (exemple, suite)

```
    public void mouseDragged(MouseEvent e) {
      graphics.drawLine(prevX,prevY,e.getX(),e.getY());
      prevX = e.getX();
      prevY = e.getY();
      repaint();
    }
    public void mousePressed(MouseEvent e) {
      prevX = e.getX();
      prevY = e.getY();
    }
    public void mouseClicked(MouseEvent e){}
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
  }
}

package test;
import java.awt.*;
import java.awt.event.*;
/**
 * @author Gilles ROUSSEL
 * @version 0.1
 */
public class MyFrame extends Frame
{
  ScrollPane scrollPane = new ScrollPane();
  MyCanvas canvas = new MyCanvas();
  Graphics graphics;

  public MyFrame() {
    scrollPane.setSize(200,200);
    scrollPane.add(canvas);
    add(scrollPane);
    pack();
    show();
  }
  public static void main(String[] args) {
    new MyFrame();
  }
}
```

Classes "réseau"

Le package java.net: adresses internet

- Les adresses Internet : java.net.InetAddress
  - Cette classe permet de manipuler les adresses Internet.
  - Il n'existe pas de constructeur. Pour obtenir une instance de cette classe il est nécessaire d'utiliser des méthodes de classe.
  - Ces méthodes de classe sont les suivantes :

    public static InetAddress getLocalHost() throws UnknownHostExeception

    Retourne un objet contenant l'adresse Internet locale.

    public static synchronized InetAddress getByName(String host_name) throws UnknownHostExeception

    Retourne un objet contenant l'adressse Internet de la machine dont le nom est passé en paramètre.

    public static synchronized InetAddress[] getAllByName(String host_name) throws UnknownHostExeception

    Retourne un tableau d'objets contenant l'ensemble des adressses Internet de machines qui répondent au nom passé en paramètre.
  - Méthodes :

    public String getHostName()

    Retourne le nom de la machine dont l'adresse est stockée dans l'objet.

    public byte[] getAddress()

    Retourne l'adresse internet stockée dans l'objet sous forme d'un tableau de 4 octets dans l'ordre réseau.

    public String toString()

    Retourne une chaîne de caractères qui liste le nom de la machine et son adresse.
  - Exemple :

```
InetAddress adresse = InetAdress.getLocalHost ();
System.out.println(adresse);
```

Le package java.net: les datagrammes udp

- Pour utiliser le protocole de transport UDP, deux classes DatagramPacket et DatagramSocket
- La classe DatagramPacket
  - Cette classe permet de créer des objets qui contiendront les données envoyées ou reçues ainsi que l'adresse de destination ou en provenance du datagramme.
  - Constructeurs, un pour les paquets à recevoir l'autre pour les paquets à envoyer.

    public DatagramPacket(byte buffer[], int taille)

    Construit un objet pour recevoir un datagramme. Le paramètre buffer correspond à la zone où doit être stocké le datagramme reçu et le paramètre taille correspond à la taille maximale des datagrammes à recevoir.

    public DatagramPacket(byte buffer[], int taille, InetAddress adresse, int port)

    Construit un objet pour envoyer un datagramme. Le paramètre buffer correspond à la zone où est stocké le datagramme à envoyer, le paramètre taille correspond à la taille du datagramme à envoyer, adresse correspond à l'adresse de la machine à qui envoyer le datagramme et port sur quel port UDP.
  - Méthodes :

    public synchronized InetAddress getAddress ()

    Retourne l'adresse stockée dans le paquet.

    public synchronized int getPort()

    Retourne le port stocké dans le paquet.

    public synchronized byte[] getData ()

    Retourne les données stockées dans le paquet.

    public synchronized int getLength ()

    Retourne la taille des données stockées dans le paquet.

    public synchronized void setAddress(InetAddress iaddr)

    Modifie ou affecte l'adresse de destination.

    public synchronized void setPort(int iport)

    Modifie ou affecte le port de destination.

    public synchronized void setData(byte ibuf[])

    Modifie ou affecte la référence de la zone contenant les données.

    public synchronized void setLength(int ilength)

    Modifie ou affecte la taille de la zone contenant les données.

Le package java.net: les datagrammes udp

- DatagramSocket : créer des sockets UDP qui permettent d'envoyer et de recevoir des datagrammes UDP.
  - Avant toute communication en mode UDP il est nécessaire de créer une socket aussi bien du coté client que du coté serveur. Pour cela Java propose trois constructeurs :

    public DatagramSocket () throws SocketException : Crée un objet de type socket et l'attache à un port disponible de la machine locale. Ce constructeur doit être utilisé dans les clients pour lesquels le port d'attachement n'a pas besoin d'être connu.

    public DatagramSocket (int port) throws SocketException: Crée un objet de type socket et l'attache au port UDP local passé en paramètre. En particulier, ce constructeur doit être utilisé dans les serveurs pour lesquels le port d'attachement a besoin d'être fixé préalablement afin qu'il soit connu des clients.

    public DatagramSocket(int port, InetAddress laddr) throws SocketException: Crée un objet de type socket, l'attache au port UDP local passé en paramètre et à une adresse spécifique de la machine locale. Ce constructeur n'est utile que si la machine locale dispose de plusieurs adresse Internet.
  - Une fois la socket (objet de type DatagramSocket) créée et attachée à un port particulier de la machine locale il est possible d'envoyer et de recevoir des datagrammes, via cette socket, au moyen des méthodes suivantes :

    public void send(DatagramPacket data) throws IOException

    Permet d'envoyer les données contenues dans la variable data vers la machine et le port dont les valeurs ont été préalablement spécifiées dans la variable data.

    public synchronized void receive(DatagramPacket data) throws IOException

    Permet de recevoir un datagramme qui sera stocké dans data. Après appel, data contient les données reçues, leur taille, l'adresse de l'envoyeur ainsi que son port d'attachement. Cette méthode est bloquante tant qu'il n'y a rien à recevoir. Si le message est trop long pour être stocké, celui-ci est tronqué, et le reste est perdu. Il n'est donc pas possible de recevoir des messages dont on ne connait pas préalablement la taille.

    Il est possible de spécifier un délai d'attente maximal en reception. Pour cela, il faut positionner une variable de timeout sur la socket au moyen de la méthode :

    public synchronized void setSoTimeout(int timeout) throws SocketException

    Une valeur de 0 (zéro) correspond à ne pas avoir de timeout.
  - Autres méthodes

    public voidclose ()

    Ferme la socket et libère les ressources qui lui sont associées. La socket ne pourra plus etre utilisée ni pour envoyer, ni pour recevoir des datagrammes.

    public int getLocalPort()

    Retourne le port d'attachement de la socket.

    public synchronized int getSoTimeout() throws SocketException

    Retourne la valeur courante du timeout associé à la socket.

Exemple

Deux classes :

une classe ServeurEcho qui attend une chaîne de caractères et la retourne
une classe ClientEcho qui envoye une chaîne de caractères, attend que le serveur la lui retourne et l'affiche.

```
import java.io.*;
import java.net.*;
class ServeurEcho
{
    final static int port = 8532;
    final static int taille = 1024;
    final static byte buffer[] = new byte[taille];
    public static void main(String[] args]) throws Exception
     {
       DatagramSocket socket = new DatagramSocket(port);
       while(true)
       {
         DatagramPacket data =
         new DatagramPacket(buffer,buffer.length);
         socket.receive(data);
         System.out.println(data.getAddress());
         socket.send(data);
       }
     }
}


class ClientEcho
{
  final static int taille = 1024;
  final static byte buffer[] = new byte[taille];
  public static void main(String[] args) throws Exception
  {
    InetAddress serveur = InetAddress.getByName(args[1]);
    int length = args[2].length();
    byte buffer[] = new byte[length];
    args[2].getBytes(0,length,buffer,0);
    DatagramPacket data =
    new DatagramPacket(buffer,length,serveur,ServeurEcho.port);
    DatagramSocket socket = new DatagramSocket();

    socket.send(data);
    socket.receive(data);
    System.out.write(buffer);
  }
}
```


Le package java.net: tcp

- L'API utilisée pour accéder au protocole TCP se décompose en deux classes, une utilisée par les clients et l'autre par les serveurs.
- La classe Socket
  - La classe Socket est utilisée par les clients TCP.
  - Pour créer un objet de la classe Socket, il est nécessaire d'utiliser un des constructeurs suivant :

    public Socket (String machine, int port) throws UnknownHostException, IOException

    public Socket (InetAddress adresse, int port) throws UnknownHostException, IOException

    public Socket(String machine, int port, InetAddress local, int portLocal) throws IOException

    public Socket(InetAddress adresse, int port, InetAddress local, int portLocal) throws IOException
  - La création de cet objet entraîne la création d'un point de connexion (la socket) et la connexion vers une autre socket (le serveur).
  - L'adresse du serveur est composée de l'adresse d'une machine (sous forme d'un nom ou d'un objet de la classe InetAddress) et d'un numéro de port.
  - L'adresse locale et le port local peuvent être spécifiés.
  - Par défaut, l'appel au constructeur est bloquant tant que la connexion TCP n'est pas établie.
  - Une fois la connexion établie, il est possible de récupérer le flux d'entrée et le flux de sortie de la connexion TCP vers le serveur au moyen des méthodes :

    public InputStream getInputStream () throws IOException
    public OutputStream getOutputStream () throws IOException
  - Il est alors possible d'échanger des données avec le serveur au moyen de toutes les primitives de lecture et d'écriture des différentes classes du package java.io.

    Par défaut les primitives de lecture, tel que read(), sont bloquantes tant que rien n'est lisible sur le flux.
    La primitive suivante permet de définir un temps maximal d'attente :

    public synchronized void setSoTimeout(int timeout) throws SocketException

Le package java.net: tcp

- Une fois la connexion terminée il est important de fermer le flux d'entrée et le flux de sortie, mais aussi la socket au moyen de la méthode :
  - public synchronized void close () throws IOException
  - Par défaut cette primitive n'est pas bloquante mais la socket reste ouverte tant qu'il reste des paquets à envoyer (en paticulier le datagramme FIN).
- Définir un temps maximum d'attente avant de fermer la socket.
  - public void setSoLinger(boolean on, int val) throws SocketException
- Par défaut, les implémentations du protocole TCP essaient de remplir au maximum chaque paquet (Nagle algorithm) afin de diminuer le traffic réseau.
  - Pour éviter ce comportement, ce qui est souvent souhaitable dans le cas d'application interactives une primitive est disponible.
  - public void setTcpNoDelay(boolean on) throws SocketException
- Autres méthodes :
  - public InetAddress getInetAddress ()
  - public int getPort ()
  - public InetAddress getLocalAddress ()
  - public int getLocalPort ()
  - int getLocalPort ()
  - public int getSoLinger () throws SocketException
  - public synchronized int getSoTimeout() throws SocketException
  - public boolean getTcpNoDelay() throws SocketException
- Remarque : une applet ne peut (pour des questions de sécurité) se connecter qu'à la machine depuis laquelle elle a été chargée.

Le package java.net: tcp

- Exemple :

```
package test;

import java.io.*;
import java.net.*;

public class SmtpClient {
  public static void main(String[] args) throws Exception {
    sendmail("aaa","roussel@univ-mlv.fr");
  }
  static void sendmail(String message, String to) throws Exception  {
    Socket s = new Socket(InetAddress.getByName("pixel.univ-
mlv.fr"),25);
    PrintStream output = new PrintStream(s.getOutputStream());
    output.println("HELO qqcvd.univ-mlv.fr\r");
    output.println("MAIL FROM: \r");
    output.println("RCPT TO:<" + to + ">\r");
    output.println("DATA\r");
    output.println(message);
    output.println("\r\n.\r");
  }
}
```

Le package java.net: tcp

- La classe ServerSocket
  - Cette classe est utilisée (comme son nom l'indique) pour créer une socket du coté serveur.
  - La classe ServerSocket permet de créer un point de communication, sur un port particulier, en attente de connexions en provenance de clients. Contrairement à la classe Socket elle n'ouvre pas de connexion.
    Constructeurs, où port correspond au port d'attente et count au nombre maximum de connexions en attente, non encore acceptées.

    public ServerSocket(int port) throws IOException

    public ServerSocket(int port, int count) throws IOException

    public ServerSocket(int port,int count,
    InetAddress locale) throws IOException
  - Une fois la socket créée, on attend les connexions de clients avec la méthode bloquante :

    public Socket accept() throws IOException
  - Il est possible de spécifier un temps maximal d'attente. Pour cela il faut appeler la méthode suivante avant l'appel à accept() :

    public synchronized void setSoTimeout(int timeout) throws SocketException

    Cette méthode retourne un nouvel objet de la classe Socket qui est connecté avec le client qui a demandé la connexion. Il est alors possible de récupérer le flôt d'entrée et de sortie comme pour la socket du client.
  - Exemple :

```
public class DaytimeServeur {
  public static void main(String[] args) throws Exception {
    ServerSocket s = new ServerSocket(0);
    System.out.println(s.getLocalPort());
    while(true) {
      Socket serviceSocket =  s.accept();
      PrintStream output =
      new PrintStream(serviceSocket.getOutputStream());
      output.println(new Date());
      serviceSocket.close();
}}}
```

  - Autres méthodes :

    public void close() throws IOException

    public int getLocalPort()

    public InetAddress getInetAddress()

    public synchronized int getSoTimeout() throws IOException

Le package java.net: la classe url

- Interface de haut niveau pour accéder aux informations du Web en utilisant des URLs.
  - Pour créer un objet de la classe URL nous disposons de quatre constructeurs :

    public URL(String protocol, String machine, int port, String fichier) throws MalformedURLException

    Crée une URL absolue à partir du protocole, de la machine, du port et du fichier.

    public URL(String protocol, String machine, String file) throws MalformedURLException

    Crée une URL absolue à partir du protocole, de la machine et du fichier, en utilisant le port par défaut du protocole.

    public URL(String url) throws MalformedURLException

    Crée une URL absolue à partir d'une chaîne de caractères brute.

    public URL(URL context, String url) throws MalformedURLException

    Crée une URL absolue à partir d'une chaîne de caractères brute et d'une URL contexte.
  - Exemple

    Les trois URLs suivantes représentent la meme ressource.

```
new URL("http://massena.univ-mlv.fr/index.html");
new URL("http","massena.univ-mlv.fr",80,"index.html");
new URL("http://massena.univ-mlv.fr/","index.html");
```

  - Comparaison de deux URLs en incluant ou non l'ancre

    public boolean sameFile(URL other)

    public boolean equals(Object other)
  - Manipulation de composants de l'URL

    public String getProtocol(): retourne le protocole ;

    public String getHost() : retourne le nom de la machine ;

    public int getPort() : retourne le numéro de port ;

    public String getFile() : retourne la partie fichier ;

    public String getRef() : retourne la référence ou ancre dans le fichier ("#ref") ;

    public String toExternalForm() : retourne la fome canonique de l'URL.
  - Une fois l'objet URL construit, pour accéder aux informations contenues dans l'URL, il faut construire un objet URLConnection en utilisant la méthode :

    public URLConnection openConnection() throws IOException

    Cette méthode crée (si elle n'existe pas déjà) un objet permettant de créer une connexion vers la ressource référée par l'URL. Cette méthode invoque le bon gestionnaire de protocole (http par exemple).

Le package java.net: la classe url

- Exemple :

  URLConnection connexion = url1.openConnection() ;

  - Il est possible, si l'on ne désire pas d'informations sur l'objet contenu, de le récupérer directement avec la méthode :

    public final Object getContent () throws IOException

  - qui est un racourci pour :

    openConnection().getContent()

- Exemple :

```
public class Url {
public static void main(String[] args){
    try{
      URL url = new URL("http://massena.univ-mlv.fr/~roussel");
      String Object texte = (String) url.getContent();
      System.out.println(texte);
    }
    catch(MalformedURLException e){
      System.out.println(e);
    }
    catch(IOException e){
     System.out.println(e);
    }
  }
}
```

- La Classe URLConnection
  - Cette classe est une classe abstraite qui propose une interface simple pour manipuler les entêtes et le contenu d'une connexion via une URL, commune à tous les gestionnaire de protocole. Un objet de cette classe est construit par l'appel à la méthode openConnection() sur un objet de la classe URL.
  - La première chose à faire consiste à spécifier les entêtes de requêtes :

    public void setRequestProperty(String key, String value)

    D'autres méthodes raccourcis sont disponibles :

    public long getIfModifiedSince() ...
  - Il faut ensuite ouvrir la connexion et parser les entêtes de réponse au moyen de la méthode :

    public abstract void connect() throws IOException

Le package java.net: la classe url

- La Classe URLConnection (suite)
  - Récupérer chacune des entêtes par une des méthodes suivantes :

    public String getHeaderField(String name) : retourne l'entête dont le nom est passé en paramètre si elle est présente et null sinon.

    public int getHeaderFieldInt(String name, int default)

    public long getHeaderFieldDate(String name,long default)

    public String getHeaderFieldKey(int n)

    public String getHeaderField(int n)
  - Les méthodes suivantes sont des racourcis pour les entêtes les plus courantes :
  - public int getContentLength ()
  - public String getContentType ()
  - public long getDate ()
  - Pour récupérer le contenu du document il suffit d'utiliser la méthode :

    public Object getContent() throws IOException
  - D'autres intéraction de plus bas niveau sont également possible en récupérant les fulx d'entrée et de sortie sur la connexion.

    public InputStream getInputStream() throws IOException

    public OutputStream getOutputStream() throws IOException
- Exemple :

```
int len = connexion.getContentLength();
InputStream input = connexion.getInputStream();
for(;len != 0; len--)
   System.out((char)input.read());
....
```

Remote Method Invocation

Les Remote Method Invocation

- Les RMIs permettent d'invoquer de façon simple des méthodes d'objets distribués sur un réseau Internet.
- Les RMIs propose un ensemble d'outils et de classes prédéfinies qui rendent l'implantation d'appels de méthodes et d'objets distants aussi simple que leur implantation dans un contexte local.
- Les RMIs utilise le mécanisme standard de sérialisation de JAVA.
- Les RMIs sont basées sur la notion de Stub/Skeleton.

  Du coté client, le Stub qui implémente la même interface que l'objet distant est chargé de transformer l'appel de méthode en une suite d'octets à envoyer sur le réseau (Marshaling) et de recontruire le résultat reçu sous le même format (Unmarshaling). Le format d'un appel de méthode contient l'identificateur de l'objet distant, l'indentificateur de la méthode et les paramètres sérialisés. La valeur de retour contient le type (valeur ou exception) et la valeur sérialisée.

  Du coté serveur le Skeleton se charge reconstruire les paramètres, de trouver l'objet appelé et d'appeler la méthode. Il se charge ensuite de retourner le résultat.
- Pour écrire un objet distant :

  Ecrire l'interface de l'objet distant.

  Ecrire l'implémentation de cette interface.

  Générer les Stub/Skeleton correspondant.

  Exporter l'objet implémentant l'interface, l'objet distant attend alors les requêtes via le Skeleton.

  Appeler une méthode de l'objet via le Stub.
- Exemple d'un serveur de Frame

  L'objet distant est un objet capable d'afficher une Frame passée en paramètre de sa méthode showFrame() .

  La première chose à faire est de définir l'interface de l'objet :

```
package test.rmi;
import java.rmi.*;
import java.awt.*;
public interface FrameDisplay extends Remote {
  public void showFrame(Frame frame) throws RemoteException;
}
```

RMI (suite)

Il faut ensuite en écrire l'implémentation :

```
package test.rmi;
import java.rmi.*;
import java.awt.*;
public class FrameDisplayImpl implements FrameDisplay {
  public FrameDisplayImpl() throws RemoteException {}
  public void showFrame(Frame frame) throws RemoteException {
    frame.show();
  }}
```

Il faut maintenant compiler cette classe et générer les Stub/Skeleton correspondant au moyen de la commande :

```
prompt> javac test/rmi/FrameDisplayImpl
prompt> rmic test.rmi.FrameDisplayImpl
```

Pour que l'objet soit accessible il faut maintenant le créer, l'exporter et l'enregistrer sous un nom pour qu'il soit possible de récupérer une référence sur celui-ci.

```
package test.rmi;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
public class ExportFrameDisplay {
  public static void main(String[] args) throws Exception{
    FrameDisplay frameDisplay = new FrameDisplayImpl();
    UnicastRemoteObject.exportObject(frameDisplay);
    Naming.rebind("//localhost:10003/FrameDisplay",frameDisplay);
    System.out.println("FrameDisplay registered");
  }}
```

Une fois compilée, son exécution entraîne l'attente d'appels à condition d'avoir préalablement installé le service de nommage.

```
prompt> javac test/rmi/ExportFrameDisplay
prompt> remiregistry 10003 &
prompt> java test.rmi.ExportFrameDisplay
```

- Le client. Celui-ci récupère un référence sur l'implémentation distant via le nom qui lui a été attribué précédemment, puis appel sur cette référence la méthode distante, comme si elle était locale.

RMI (suite)

```
package test.rmi;
import java.rmi.*;
import java.awt.*;
public class ClientFrame {
  public static void main(String[] args) throws Exception {
    Frame frame = new Frame (){
      public void paint(Graphics graphics) {
        graphics.setColor(Color.black);
        graphics.drawString("Hello World!",65,60);
      }    };
    frame.setSize(200,100);
    FrameDisplay display = (FrameDisplay)
Naming.lookup("rmi://localhost:10003/FrameDisplay");
   display.showFrame(frame);
  }}
```

Une fois compilé le lancement du client entraîne l'affichage de la fenêtre sur la machine où a été lancé le serveur.

- Le serveur
  - Actuellement la seule classe de serveur qu'il est possible d'instancier est la classe java.rmi.server.UnicastRemoteServer. Celle-ci hérite de la classe java.rmi.server.RemoteServer qui hérite elle-même de la classe java.rmi.server.RemoteObject.
    - La classe RemoteObject implémente le comportement de la classe Object pour les objets distants :
    - la méthode hashcode() pour être sûr que deux références à un même objet distant ait le même code de hashage.
    - la méthode equals() pour la même raison.
    - la méthode toString() pour avoir un affichage particulier.
  - La classe RemoteServer doit être la classe de base de toutes les futures (?) classes de serveurs. Celle-ci ne possède que trois méthodes :
    - public static String getClientHost() throws ServerNotActiveException qui permet de connaître l'identité du client pendant un appel de méthode.
    - public static void setLog(OutputStream out) qui permet définir un flux vers lequel sont journalisés tous les appels de méthode distante.
    - public static PrintStream getLog() qui récupère le flux de journalisation. Par défaut aucun flux n'est utilisé.
    - Un UnicastRemoteServer est un serveur basé sur TCP dont les références ne sont valide que pendant la durée du processus. Il implémente une méthode clone() permettant de créer un serveur distinct du précédent.
  - Du coté du client le serveur est accessible via la classe RemoteStub qui hérite de RemoteObject.

RMI (suite)

- Exporter un Objet

  Pour pouvoir être exporté un objet doit implémenter la classe Remote. Il y a alors deux possibilités pour être exporter :

  hériter de la classe UnicastRemoteObject . L'appel au constructeur exportera automatiquement l'objet courant.

  appeler la méthode exportObject()de la classe UnicastRemoteObject sur l'objet à exporter. Cette méthode retourne l'objet RemoteStub associé.

  Lorsqu'un objet est exporté un ensemble de threads est créé pour attendre les appels de méthodes.

- Le service de nommage
  - Le service de nommage permet de récupérer le Stub d'un objet distant à partir d'un nom. La classe Naming permet d'accéder de façon simple au service de nommage d'une machine en utilisant un format de type URL :

    rmi://machine:port/objet

    Avant de pouvoir enregistrer un objet il faut lancer un serveur de nommage, en général au moyen de

    rmiregistry port &

    Il est alors possible d'enregistrer un objet local au moyen des méthodes :

    public static void bind(String name, Remote obj) throws AlreadyBoundException, MalformedURLException, UnknownHostException, RemoteException

    public static void rebind(String name,Remote obj) throws RemoteException, MalformedURLException, UnknownHostException
  - Il est possible de récupérer le Stub d'un objet distant à partir de son URL au moyen de la méthode :

    public static Remote lookup(String name) throws NotBoundException, MalformedURLException, UnknownHostException, RemoteException
  - Les méthodes suivantes permettent de désenregistrer l'objet du service de nommage et de lister l'ensemble des objets enregistrés :

    public static void unbind(String name) throws RemoteException, NotBoundException, MalformedURLException, UnknownHostException

    public static String[] list(String name) throws RemoteException, MalformedURLException, UnknownHostException
  - Cette classe ne permet pas un nommage hiérarchique.

    Si l'on désire un service de nommage plus évolué il est possible de l'écrire en utilisant la classe java.rmi.registry.LocateRegistry . Celle-ci possède des méthodes de manipulation d'objet implémentant l'interface java.rmi.registry.Registry.

    public static Registry createRegistry(int port) throws RemoteException qui crée un nouveau service de nommage sur le port spécifié (ce service n'existe que pendant le temps de vie du serveur).

    public static Registry getRegistry(int port) throws RemoteException

    public static Registry getRegistry(String host) throws RemoteException, UnknownHostException

RMI (suite)

- L'interface Registry dispose des même méthodes que la classe Naming mais elles ne sont pas static :

  bind(), rebind() , lookup(), unbind() et list().
- A la place de la classe Naming on pourrait utiliser la classe suivante :

```
package test.rmi;
import java.rmi.registry.*;
import java.rmi.*;
public class Naming {
  public static Remote lookup(String host, int port, String
remoteName)
    throws RemoteException, UnknownHostException, NotBoundException,
AccessException {
    Registry registry = LocateRegistry.getRegistry(host,port);
    return registry.lookup(remoteName);
  }
  public static void bind(String host, int port, String name,Remote
obj)
    throws RemoteException, UnknownHostException,
AlreadyBoundException, AccessException {
    Registry registry = LocateRegistry.getRegistry(host,port);
    registry.bind(name, obj);
  }
}
```

- Le passage de paramètres et valeur de retour

  Le passage de paramètres pour les méthodes distantes n'a pas la même sémantique que le passage de paramètres pour les méthodes locales.

  Les variables de type primitif sont passées par valeurs.

  Les objets qui n'implémentent pas l'interface Remote sont passés par recopie à condition qu'ils implémentent l'interface Serializable ou Externalizable. Sinon une exception est levée.

  Les objets qui implémentent l'interface Remote sont remplacés par l'objet Stub correspondant lors du passage de paramètres.

  Le retour de valeur a la même sémantique.

RMI (suite)

- Les exceptions
  - La levée d'exception a la même sémantique que le passage de paramètres.
  - Toutes les signatures de méthodes distantes et des constructeurs d'objets distantes doivent contenir l'exception java.rmi.RemoteException . Celle-ci permet de retourner une erreur en cas de problème dû à la non localité de l'appel (par exemple un problème lors de la connexion à l'objet distant).
- Le chargement dynamique des classes
  - Dans l'exemple précédent nous n'avons pas précisé quelles classes étaient présentes sur le client et quelles classes étaient présentes sur le serveur. Si toutes les classes nécessaires sont présentes localement la classe locale est chargée par défaut.
  - Les RMIs proposent un mécanisme permettant de charger dynamiquement les classes depuis le réseau si elles ne sont pas présentes localement.
  - Si le client est une applet toutes les classes sont chargées depuis son codebase.
  - Si le client est une application toutes les classes qui apparaissent explicitement dans le code du client sont chargées avec le chargeur par défaut. Sinon, un objet de la classe RMIClassLoader est utilisé pour charger les classes (par exemple la classe Stub). Le schéma de recherche d'une classe est le suivant :

    Recherche dans le CLASSPATH

    Pour les paramètres ou les valeurs de retour recherche à l'URL encodé avec la classe de l'objet

    Pour le Stub ou Skeleton recherche de la classe à l'URL spécifiée dans la propriété système java.rmi.server.codebase.

RMI (suite)

  - L'URL encodée avec la classe de l'objet lors du passage de paramètre est :

    l'URL depuis laquelle a été chargée l'objet si elle existe.

    l'URL spécifiée dans la propriété système java.rmi.server.codebase sinon.
- Le schéma de chargement est le même pour le serveur.
  - Si la propriété système java.rmi.server.useCodebaseOnly est true, une classe non locale peut uniquement être chargée que depuis l'URL spécifiée dans codebase.
  - Il est possible de forcer le chargement d'une classe par le réseau en appelant explicitement le chargement d'une classe avant son utilisation. Pour cela on peut utiliser la méthode loadClass() de la classe RMIClassLoader.

```
Class c =
RMIClassLoader.loadClass("http://massena/CLASSES","Complex");
```

    Il est ainsi possible de bootstrapper complètement un client en supposant qu'il implémente un interface connue localement (ici Runnable).

```
Class c = RMIClassLoader.loadClass("http://massena/CLASSES","Client");
(Runnable) client = c.newInstance();
client.run();
```

  - Seul les URLs de type HTTP et FTP sont supportées par l'implantation actuelle.
  - Il n'est pas possible de charger depuis le réseau une classe tant qu'aucun gestionnaire de sécurité n'a été mis en place au moyen de System.setSecurityManager(). Les classes chargées par le réseau sont soumises à ce gestionnaire de sécurité.
  - Les RMIs propose un gestionnaire de sécurité RMISecurityManager qui par défaut interdit tout ce qu'un SecurityManager peut interdire, sauf la définition de classe. Pour permettre plus de chose il faut écrire son propre gestionnaire de sécurité héritant de RMISecurityManager .

RMI (suite)

Exemple d'un serveur de calcul de complexe ou toutes les classes sont chargées dynamiquement par le client.

Le client :

```
package test.rmi;
import java.rmi.*;
import java.util.*;
public class ComplexCompute {
  public static void main(String [] args) throws Exception {
    Properties p =System.getProperties();
    p.put("java.rmi.server.codebase","http://massena.univ-
mlv.fr/~roussel/CLASSES");
    System.setProperties(p);
    System.setSecurityManager(new RMISecurityManager());
    ComplexOps obj = (ComplexOps) Naming.lookup("//"+ args[0]+
":10003/ComplexOps");
    Complex c1 = obj.newComplex(1,1);
    Complex c2 = obj.newComplex(2,2);
    Complex c3 = obj.addComplex(c1,c2);
    Complex c4 = obj.multComplex(c1,c2);
    System.out.println(c1 + " + " + c2 + " = " + c3);
    System.out.println(c1 + " * " + c2 + " = " + c4);
  }
}
```

L'interface de l'objet distant et la classe l'implémentant (le serveur).

```
package test.rmi;
import java.rmi.*;
public interface ComplexOps extends Remote{
  public Complex newComplex(double re, double im) throws
java.rmi.RemoteException;
  public Complex addComplex(Complex c1, Complex c2) throws
java.rmi.RemoteException;
  public Complex multComplex(Complex c1, Complex c2) throws
java.rmi.RemoteException;
}
package test.rmi;
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.net.*;
public class ComplexOpsImpl extends UnicastRemoteObject implements
ComplexOps {
  public ComplexOpsImpl() throws java.rmi.RemoteException {
    super();
  }
  public Complex newComplex(double re, double im) throws
java.rmi.RemoteException{
    return new ComplexImpl(re,im);
  }
```


RMI (suite)

```
  public Complex addComplex(Complex c1, Complex c2) throws
java.rmi.RemoteException {
    return new
ComplexImpl(c1.getRe()+c2.getRe(),c1.getIm()+c2.getIm());
  }
  public Complex multComplex(Complex c1, Complex c2) throws
java.rmi.RemoteException {
    return new ComplexImpl(c1.getRe()*c2.getRe()-
c1.getIm()*c2.getIm(),c1.getRe()*c2.getIm()+c1.getIm()*c2.getRe());
  }
  public static void main(String[] args) throws Exception {
    Properties p =System.getProperties();
    p.put("java.rmi.server.codebase","http://massena.univ-
mlv.fr/~roussel/CLASSES");
    System.setProperties(p);
    System.setSecurityManager(new RMISecurityManager());
    String bindUrl = "//" + InetAddress.getLocalHost().getHostName() +
":10003/ComplexOps";
    Naming.rebind(bindUrl,new ComplexOpsImpl());
    System.out.println("ComplexOps Registered.");
  }
}
```

L'interface et l'implémentation de l'objet complexe :

```
package test.rmi;
public interface Complex {
  public double getRe();
  public void setRe(double re);
  public double getIm();
  public void setIm(double im);
  public String toString();
}
package test.rmi;
import java.io.*;
public class ComplexImpl implements Complex, Serializable {
  private double re;
  private double im;

  public ComplexImpl(double re){
    this.re=re;
  }
  public ComplexImpl(double re, double im){
    this.re=re;
    this.im=im;
  }
  public double getRe() { return re; }
  public void setRe(double re) { this.re = re; }
  public double getIm() { return im; }
  public void setIm(double im) { this.im = im; }
  public String toString(){
    return re + "+ i " +im;
  }
}
```

Le Distributed Garbage Collector

Java propose un mécanisme de ramasse-miettes distribué. Plus exactement, il propose un mécanisme permettant à un objet exporté de savoir s'il existe de références distantes à cet objet. Pour être informé un objet implémentant la classe RemoteObject doit également implémenter l'interface java.rmi.server.Unreferenced qui ne contient que la méthode public void unreferenced().

Chaque fois qu'il n'existe plus de référence distante à l'objet cette méthode est appelée. Cela ne veut pas dire qu'il n'existe plus de référence locale et donc l'objet n'est pas a priori détruit (il existe au moins un référence dans le serveur qui exporte l'objet). Après un appel à cette méthode, une application distante peut récupérer une référence sur cet objet, la méthode unreferenced()sera alors appelée au moins deux fois.

L'algorithme utilisé par Java suppose un rôle actif des clients qui chaque fois qu'ils reçoivent une référence à un objet distant informe le DGC associé à cet objet. Les clients doivent ensuite l'informer régulièrement qu'il possède toujours une référence sur l'objet. Dès que le client ne dispose plus de référence il en informe également le DGC qui peut, s'il n'y a plus de référence autre part, appeler la méthode unreferenced().

Cette méthode est particulièrement utile si l'on désire utiliser des serveur temporaire comme cela est le cas avec le mécanisme de rappel ou callback.

- Le mécanisme de rappel
  - Le mécanisme de rappel permet de rendre un appel de méthode distante asynchrone.
  - Le client peut s'il le désire effectuer des traitement pendant que le serveur attend les résultat de la méthode.
  - Ce mécanisme est particulièrement profitable dans le cas d'appel en cascade
  - Sans mécanisme de rappel tous les serveurs sont bloqués pendant l'appel de méthode. Avec le mécanisme de rappel il ne sont bloqués que pendant le temps de redirection de la requête. Une autre stratégie plus coûteuse n'utilisant pas le mécanisme de rappel qui est parfois utilisée est la méthode itérative.

RMI (suite)

- Exemple de serveur avec mécanisme de rappel :

Le client :

```
package test.rmi;
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
public class CallbackClient{
  public static void main(String[] args) throws Exception {
    Properties p =System.getProperties();
    p.put("java.rmi.server.codebase","http://massena.univ-
mlv.fr/~roussel/CLASSES");
    System.setProperties(p);
    System.setSecurityManager(new RMISecurityManager());
    CallbackServer obj = (CallbackServer) Naming.lookup("//" +
args[0]+ ":10003/CallbackServer");
    CallbackReturn tmpServer1 = new CallbackReturnImpl(1);
    UnicastRemoteObject.exportObject(tmpServer1);
    obj.call("Ca marche !",tmpServer1);
    tmpServer1=null;
  }
}
```

Le serveur temporaire :

```
package test.rmi;
import java.rmi.*;
public interface CallbackReturn extends Remote {
  public void returnString(String text) throws RemoteException;
}
package test.rmi;
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.net.*;
public class CallbackReturnImpl implements CallbackReturn,
Unreferenced {
  private int i;
  public CallbackReturnImpl(int i) throws RemoteException {
    super();
    this.i = i;

}
```

RMI (suite)

```
    public void unreferenced(){
      System.out.println("Unreferenced !" + i);
      Thread current = Thread.currentThread();
      ThreadGroup group = current.getThreadGroup();
      group.stop();
    }
    public void returnString(String text) throws RemoteException {
      Thread current = Thread.currentThread();
      System.out.println(text);
    }
  }
```

L'objet distant :

```
package test.rmi;
import java.rmi.*;
public interface CallbackServer extends Remote {
  public void call(String text, CallbackReturn tmpServer) throws
RemoteException;
}
package test.rmi;
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.net.*;
public class CallbackServerImpl extends UnicastRemoteObject implements
CallbackServer {
  public CallbackServerImpl() throws RemoteException {
    super();
  }
  public void call(String text, CallbackReturn tmpServer)
   throws RemoteException {
    tmpServer.returnString(text);
    tmpServer=null;
    System.gc();
  }
  public static void main(String[] args) throws Exception {
    Properties p =System.getProperties();
    p.put("java.rmi.server.codebase","http://massena.univ-
mlv.fr/~roussel/CLASSES");
    System.setProperties(p);
    System.setSecurityManager(new RMISecurityManager());
    String bindUrl = "//" + InetAddress.getLocalHost().getHostName() +
":10003/CallbackServer";
    Naming.rebind(bindUrl,new CallbackServerImpl());
    System.out.println("CallbackServer Registered.");
  }
}
```


RMI (suite)

- Les threads et les RMIs
  - Il n'est pas garanti que deux appels à une méthode distante seront exécutés dans un même thread, ni qu'ils seront exécutés dans un thread différent. Seuls deux appels provenant de deux JVM différentes sont toujours exécutés dans deux threads distincts.
  - Il faut donc penser à synchroniser les méthodes distantes si cela est nécessaire.
- Les RMIs à travers un firewall
  - Pour permettre l'utilisation des RMIs derrière un firewall , Java peut encapsuler les appels de méthode dans des requêtes HTTP. Par défaut si un objet distant ne peut pas être atteint, les RMIs encapsulent la requête dans une méthode HTTP POST, tente un connexion vers le port 80 et attend la réponse qui doit contenir le résultat. Si le serveur n'est pas un proxy, il est possible d'utiliser un CGI-script java-rmi pour se connecter à l'objet distant.
  - Les appels effectués avec l'aide d'un serveur HTTP sont beaucoup plus lent.

JDBC

JDBC

- Installation
  - Installer le driver JDBC sur la machine
  - Installer une passerelle JDBC-ODBC
- Charger les drivers

  Class.forName("jdbc.DriverXYZ");
- Créer la connexion à la base de données

  URL dans le cas de la passerelle: jdbc:odbc:mabase

  Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");
- SQL
  - Création d'un table en SQL

    CREATE TABLE COFFEES (COF_NAME VARCHAR(32), SUP_ID INTEGER, PRICE FLOAT, SALES INTEGER, TOTAL INTEGER)
  - Sélection

    SELECT First_Name, Last_Name FROM Employees WHERE Last_Name LIKE "Washington" select First_Name, Last_Name from Employees where Last_Name like "Washington"
- Statement
  - méthode executeUpdate pour le DDL et les mises à jour de tables

```
Statement stmt = con.createStatement();
stmt.executeUpdate("CREATE TABLE COFFEES " + "(COF_NAME VARCHAR(32),
SUP_ID INTEGER, PRICE FLOAT, " + "SALES INTEGER, TOTAL INTEGER)");
Statement stmt = con.createStatement();
stmt.executeUpdate(
    "INSERT INTO COFFEES " +
    "VALUES ('Colombian', 101, 7.99, 0, 0)");
```

    Code retour de executeUpdate: nombre de lignes mises à jour

  - méthode executeQuery pour les requêtes

```
ResultSet rs = stmt.executeQuery( "SELECT COF_NAME, PRICE FROM
COFFEES");
while (rs.next()) {
   String s = rs.getString("COF_NAME");
   float n = rs.getFloat("PRICE");
   System.out.println(s + " " + n);
}
```

    Autre possibilité: utiliser le numéro de colonne au lieu du nom de colonne.

JDBC

- Méthodes utilitaires

  getByte, getShort getInt getLong getFloat getDouble getBigDecimal getBoolean getString getBytes getDate getTime getTimestamp getAsciiStream getUnicodeStream getBinaryStream getObject

- Mise à jour de tables

```
String updateString = "UPDATE COFFEES " +
                              "SET TOTAL = TOTAL + 75 " +
                              "WHERE COF_NAME LIKE 'Colombian'";
stmt.executeUpdate(updateString);
String query = "SELECT COF_NAME, TOTAL FROM COFFEES " +
               "WHERE COF_NAME LIKE 'Colombian'";
ResultSet rs = stmt.executeQuery(query);
while (rs.next()) {
  String s = rs.getString(1);
  int n = rs.getInt(2);
  System.out.println(n + " pounds of " + s + " sold to date.");
}
```

- Utilisation de PreparedStatement
  - L'instruction est compilé une fois pour toutes
  - Intéressant lors de l'exécution d'instructions répétitives
  - Exemple

    PreparedStatement updateSales = con.prepareStatement("UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");

    updateSales.setInt(1, 75);

    updateSales.setString(2, "Colombian");

  - Syntaxe

    setXXX(index, value)

  - Exemple

```
updateSales.setInt(1, 100);
updateSales.setString(2, "French_Roast");
updateSales.executeUpdate();
// changes SALES column of French Roast row to 100
updateSales.setString(2, "Espresso");
updateSales.executeUpdate();
// changes SALES column of Espresso row to 100 (the first
// parameter stayed 100, and the second parameter was reset
// to "Espresso")
```


JDBC

  - Autre exemple

```
PreparedStatement updateSales;
String updateString = "update COFFEES " + "set SALES = ? where
COF_NAME like ?";
updateSales = con.prepareStatement(updateString);
int [] salesForWeek = {175, 150, 60, 155, 90};
String [] coffees = {"Colombian", "French_Roast", "Espresso",
"Colombian_Decaf", "French_Roast_Decaf"};
int len = coffees.length;
for(int i = 0; i < len; i++) {
  updateSales.setInt(1, salesForWeek[i]);
  updateSales.setString(2, coffees[i]);
  updateSales.executeUpdate();
}
```

- Transactions
  - Par defaut, le mode est auto-commit
  - Désactiver: con.setAutoCommit(false);
  - Finir la transaction: méthode commit()
  - Revenir en arrière: rollback()
  - Exemple

```
con.setAutoCommit(false);
PreparedStatement updateSales = con.prepareStatement(
    "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");
updateSales.setInt(1, 50);
updateSales.setString(2, "Colombian");
updateSales.executeUpdate();
PreparedStatement updateTotal = con.prepareStatement(
    "UPDATE COFFEES SET TOTAL = TOTAL + ? WHERE COF_NAME LIKE ?");
updateTotal.setInt(1, 50);
updateTotal.setString(2, "Colombian");
updateTotal.executeUpdate();
con.commit();
con.setAutoCommit(true);
```

  - Les systèmes de BD qui utilisent les transactions déterminent un niveau de verrouillage (lock). Cet niveau peut être consulté et changé par getTransactionIsolation et setTransactionIsolation

Exemple

```
import java.sql.*;
public class TransactionPairs {
   public static void main(String args[]) {
   String url = "jdbc:mySubprotocol:myDataSource";
   Connection con = null;
   Statement stmt;
   PreparedStatement updateSales;
   PreparedStatement updateTotal;
   String updateString = "update COFFEES " +
     "set SALES = ? where COF_NAME like ?";
   String updateStatement = "update COFFEES " +
     "set TOTAL = TOTAL + ? where COF_NAME like ?";
   String query = "select COF_NAME, SALES, TOTAL from COFFEES";
   try {
     Class.forName("myDriver.ClassName");
   } catch(java.lang.ClassNotFoundException e) {
     System.err.print("ClassNotFoundException: ");
     System.err.println(e.getMessage());
   }
   try {
     con = DriverManager.getConnection(url,
       "myLogin", "myPassword");
     updateSales = con.prepareStatement(updateString);
     updateTotal = con.prepareStatement(updateStatement);
     int [] salesForWeek = {175, 150, 60, 155, 90};
     String [] coffees = {"Colombian", "French_Roast",
           "Espresso", "Colombian_Decaf",
           "French_Roast_Decaf"};
     int len = coffees.length;
     con.setAutoCommit(false);
     for (int i = 0; i < len; i++) {
           updateSales.setInt(1, salesForWeek[i]);
           updateSales.setString(2, coffees[i]);
           updateSales.executeUpdate();
           updateTotal.setInt(1, salesForWeek[i]);
           updateTotal.setString(2, coffees[i]);
           updateTotal.executeUpdate();
           con.commit();
     }
```


Exemple (suite)

```
     con.setAutoCommit(true);
     updateSales.close();
     updateTotal.close();
     stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery(query);
     while (rs.next()) {
           String c = rs.getString("COF_NAME");
           int s = rs.getInt("SALES");
           int t = rs.getInt("TOTAL");
           System.out.println(c + "     " +  s + "    " + t);
     }
     stmt.close();
     con.close();
   } catch(SQLException ex) {
     System.err.println("SQLException: " + ex.getMessage());
     if (con != null) {
     try {
           System.err.print("Transaction is being ");
           System.err.println("rolled back");
           con.rollback();
     } catch(SQLException excep) {
     System.err.print("SQLException: ");
     System.err.println(excep.getMessage());
   }}}}}
```

JDBC

- Procédures stockées
  - CallableStatement est dérivée de PreparedStatement

```
CallableStatement cs = con.prepareCall("{call SHOW_SUPPLIERS}");
ResultSet rs = cs.executeQuery();
```

- Exceptions
  - Traiter les exceptions au chargement du driver et à l'exécution de code.

```
try
{
// Code that could generate an exception goes here.
// If an exception is generated, the catch block below
// will print out information about it.
} catch(SQLException ex)
{
   System.err.println("SQLException: " + ex.getMessage());
}
try  {
   Class.forName("myDriverClassName");
} catch(java.lang.ClassNotFoundException e) {
   System.err.print("ClassNotFoundException: ");
   System.err.println(e.getMessage());
}
```

  - Récupérer l'état de SQL

```
try {
          // Code that could generate an exception goes here.
          // If an exception is generated, the catch block below
          // will print out information about it.
} catch(SQLException ex) {
          System.out.println("\n--- SQLException caught ---\n");
          while (ex != null) {
                    System.out.println("Message:   "
                              + ex.getMessage ());
                    System.out.println("SQLState:  "
                              + ex.getSQLState ());
                    System.out.println("ErrorCode: "
                              + ex.getErrorCode ());
                    ex = ex.getNextException();
                    System.out.println("");
          }
}
```


JDBC: Warnings

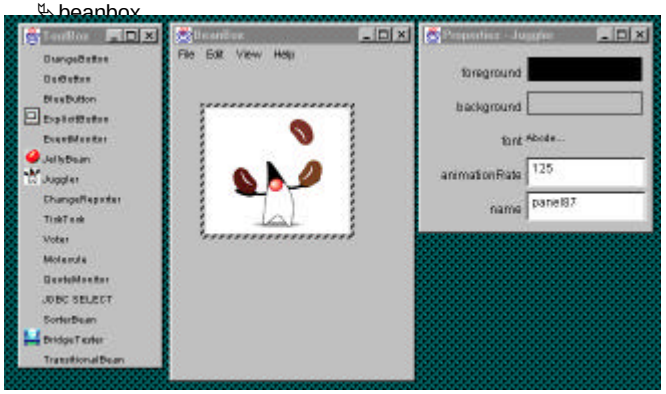
- Traiter les warnings
  - Permet de traîter des erreurs mineurs. Exemple: DataTruncation est une classe dérivée de SQLWarning
  - Exemple

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select COF_NAME from COFFEES");
while (rs.next()) {
          String coffeeName = rs.getString("COF_NAME");
          System.out.println("Coffees available at the Coffee Break:
");
          System.out.println("    " + coffeeName);
          SQLWarning warning = stmt.getWarnings();
          if (warning != null) {
                    System.out.println("\n---Warning---\n");
                    while (warning != null) {
                      System.out.println("Message: "
                               + warning.getMessage());
                      System.out.println("SQLState: "
                               + warning.getSQLState());
                      System.out.print("Vendor error code: ");
                      System.out.println(warning.getErrorCode());
                      System.out.println("");
                      warning = warning.getNextWarning();
                    }
          }
          SQLWarning warn = rs.getWarnings();
          if (warn != null) {
                    System.out.println("\n---Warning---\n");
                    while (warn != null) {
                      System.out.println("Message: "
                                        + warn.getMessage());
                      System.out.println("SQLState: "
                                        + warn.getSQLState());
                      System.out.print("Vendor error code: ");
                      System.out.println(warn.getErrorCode());
                      System.out.println("");
                      warn = warn.getNextWarning();
                    }
          }
}
```

Java Beans

Java Beans

- Ecriture de composants
  - "Self-contained"
  - exposent leurs propriétés, événements, etc car ils adhèrent à un schéma particulier (design pattern).
- Concepts
  - les outils visuels peuvent découvrir l'interface
  - les propriétés des javabeans peuvent être modifiés au moment de la conception d'interface
  - les propriétés peuvent être changées au moment de la conception
  - utilisation d'événements pour communiquer
  - persistence
- Le BDK permet de le développement de Beans
  - beanbox



  - Répertorie les beans utilisables à partir du répertoire beans
  - Permet de définir les interactions entre beans
  - Possibilité de changement des propriétés et de sérialisation

JavaBean

- Code simplifié:

```
import java.awt.*;
import java.io.Serializable;

public class SimpleBean extends Canvas
                        implements Serializable
{
    //Constructor sets inherited properties
    public SimpleBean(){
        setSize(60,40);
        setBackground(Color.red);
    }
}
```

Compilation: javac SimpleBean.java

Création du fichier manifest: manifest.tmp

```
Manifest-Version: 1.0
Name: SimpleBean.class
Java-Bean: True
```

Création du fichier jar

```
jar cfm SimpleBean.jar manifest.tmp SimpleBean.class
```

Introspection

```
java sun.beanbox.BeanBoxFrame > beanreport.txt
```

- Propriétés
  - getXXX et setXXX

public Color getColor() { ... }

public void setColor(Color c) { ... }

```
import java.awt.*;
import java.io.Serializable;

public class SimpleBean extends Canvas
                        implements Serializable
{
   private Color color = Color.green;
   //Constructor sets inherited properties
   public SimpleBean(){
        setSize(60,40);
        setBackground(Color.red);
   }
   public void setColor(Color newColor){
        color = newColor; repaint(); }
   public Color getColor(){ return color; }
   public void paint(Graphics g) {
           g.setColor(color);
           g.fillRect(20, 5, 20, 30);
    }
}
```


JavaBeans

- Propriétés liées
  - Un objet souhaite être tenu au courant des changements de propriété d'un autre objet.
  - Evenement déclenché après la modification

```
private PropertyChangeSupport changes =
           new PropertyChangeSupport(this);
public void addPropertyChangeListener(
               PropertyChangeListener l)
{
    changes.addPropertyChangeListener(l);
}
public void removePropertyChangeListener(
               PropertyChangeListener l)
{
    changes.removePropertyChangeListener(l);
}
public void setLabel(String newLabel) {
    String oldLabel = label;
    label = newLabel;
    sizeToFit();
    changes.firePropertyChange("label", oldLabel, newLabel);
}
```

  - Implémentation du listener

```
OurButton button = new OurButton();
...
PropertyChangeAdapter adapter = new PropertyChangeAdapter();
...
button.addPropertyChangeListener(adapter);
...
class PropertyChangeAdapter implements PropertyChangeListener
{
    public void propertyChange(PropertyChangeEvent e)
    {
        reporter.reportChange(e);
    }
}
```

- Propriétés contraintes
- Propriétés indexées
- Evénements déclenchables