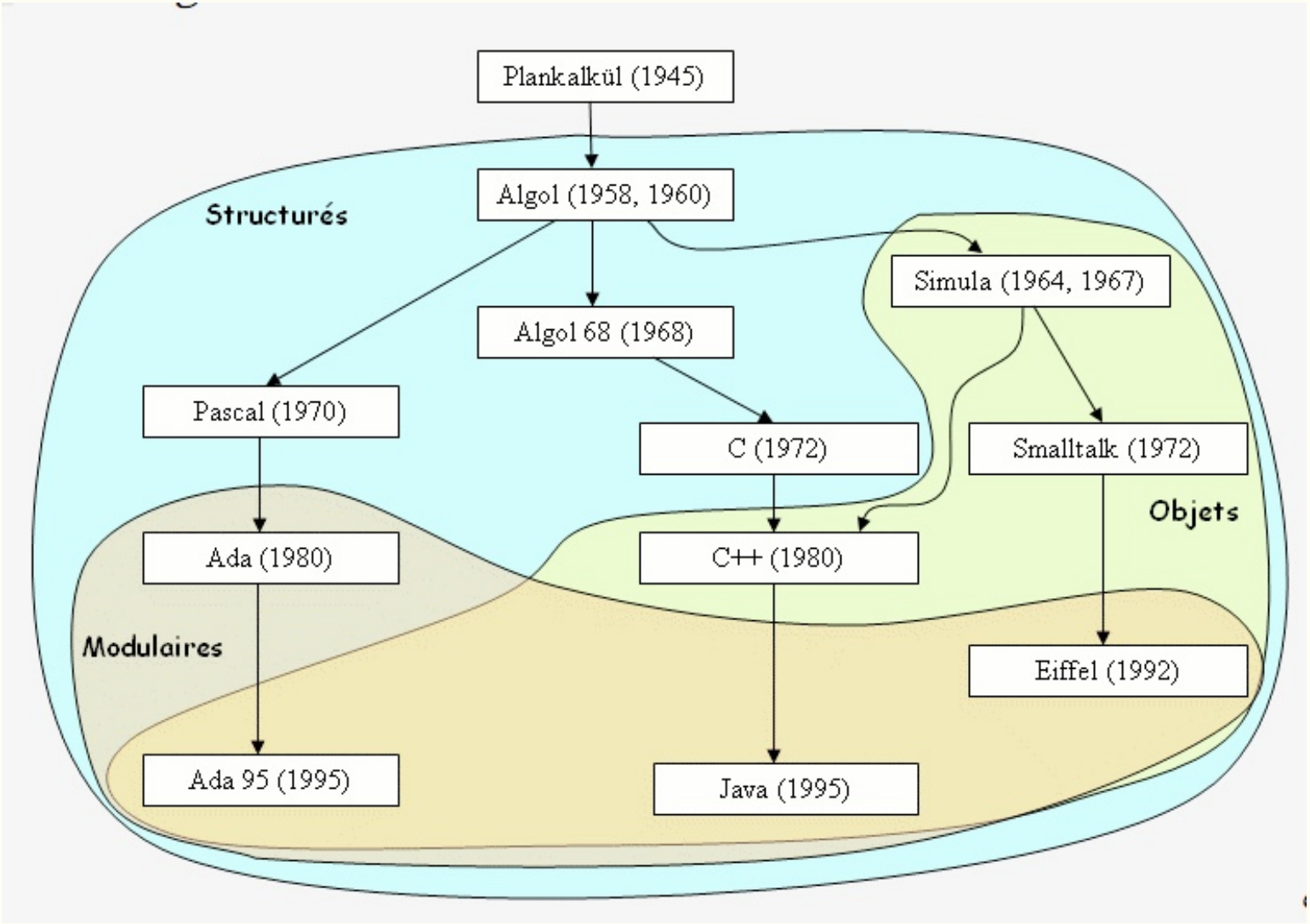
Partie II du cours

Introduction au langage Java

Introduction au langage JAVA



Java

- Langage de programmation moderne créé par Sun Microsystems en 1995
  - Basé sur C++ (années 70)
  - En 2010, Oracle (base de données) a racheté Sun
  - Site http://www.oracle.com/technetwork/java/index.html

- Caractéristiques
  - Basé sur C++ :
    - plus propre
    - pas de pointeurs (apparents) et pas d'arithmétique de pointeurs
    - plus de mots clés -> langage plus précis
    - exceptions : mieux que Segmentation fault

  - Java 7 & Java 8: proposent plus de 4000 classes ! Toutes les activités les plus courantes sont incluses. *Actuellement : Java 9 (6005 classes) ...*

Introduction au langage Java

Pour travailler en Java

- Environnement d'**exécution** : JRE (Java Runtime Environment)
  http://www.java.com/fr/
  - déjà inclus sur Mac et sous Windows

- Outil de **compilation** d'un programme java : JDK (Java Development Kit)
  http://www.oracle.com/technetwork/java/javase/downloads/index.html
  - compilateur, débugger, documentation et beaucoup d'autres choses

  - Version au 28/08/2017 : Java SE 8u144 includes important bug fixes.
  Oracle strongly recommends that all Java SE 8 use upgrade to this release.
  - environ 90Mo

- Un outil **pour taper du code**
  - **éditeur de texte (e.g. jEdit)**
  - Eclipse (http://www.eclipse.org),
    ~180Mo, installé dans les salles de TP
  - NetBeans (v8.0) : peut être téléchargé en même temps que le JDK

Downloads, documentation et a https://www.java.com/fr/ ide en ligne

- https://www.java.com/fr/
- http://www.oracle.com/technetwork/java/javase/downloads/index.html
- http://docs.oracle.com/javase/8/
- http://docs.oracle.com/javase/8/docs/api/index.html

Un programme en Java

```
class ProgrammePrincipal {
  public static void main ( String arg[ ] ) {
   Complexe z1 = new Complexe(1, 2);
   Complexe z2 = new Complexe(2, 2);
   z1 = z1.ajouter(z2);
   System.out.println ( z1.module() ) ; // affiche 5
   z1.affiche();//affiche 3+4i si la méthode d'affichage est bien
 faite
  }
}
```

Exemple

- En JAVA, le compilateur vérifie que l'opération (e.g. **`z1.affiche()`**) peut être effectuée (typage fort)
(pour qu'il n'y ait pas d'erreur à l'exécution).

Autre exemple

Exemple

```
// Classe CompteBancaire -
/* Classe représentant un compte bancaire et les méthodes qui lui sont associées.*/

  public class CompteBancaire {
    private String nom;        // le nom du client
    private String adresse;    // son adresse
    private int numéro;        // numéro du compte
    private int solde;         // solde du compte

    // Les méthodes
   public CompteBancaire(…) { // à compléter
   }
    public void créditer(int montant) {
        solde += montant;
    }
    public void débiter(int montant) {
        solde -= montant;
    }
    public void afficherEtat() {
        System.out.println("Compte numéro " + numéro + " ouvert au nom de " + nom);
        System.out.println("Adresse du titulaire : " + adresse);
        System.out.println("Le solde actuel du compte est de " + solde + " euros.");
   }
}
```

- *Rmq. private vs public — String vs int —*

La méthode "main"

```
class ProgrammePrincipal {

  public static void main(String[] a){
    . . .
  }

}
```

- ***static*** : parce que une seule instance de cette classe
- ***void*** : parce qu'elle ne retourne pas de résultat
- ***String[] a*** : parce que l'on peut passer des paramètres *(dans un tableau de chaînes de caractères)*
- l'exécutable est un processus qui porte le nom de la classe: `ProgrammePrincipal`

Compilation et exécution en Java

- Une classe "`ProgrammePrincipal`" dans un fichier "`ProgrammePrincipal.java`" :

```
>>> javac ProgrammePrincipal.java
```

produit un fichier ProgrammePrincipal.class (si compilation ok)

```
>>> java ProgrammePrincipal
```

exécute le "`main`" de la classe `ProgrammePrincipal`

Compilation et exécution en Java (suite)

Et la compilation séparée ?

◁ Supposons une classe `SousProgramme` utilisée par ma classe `ProgrammePrincipal`...

◁ qui crée un *nouvel* objet `m` de type `SousProgramme`

◁ et appelle la méthode `afficheMessage()` de l'objet *m*

```
>>> javac SousProgramme.java
>>> javac ProgrammePrincipal.java
>>> java ProgrammePrincipal
message disponible
>>>
```

◁ SousProgramme.java

```
import java.io.*;
public class SousProgramme {
    public void afficheMessage() {
        System.out.println("message disponible");
    }
}
```

◁ ProgrammePrincipal.java

```
public class ProgrammePrinciapl {
public static void main(String[] argv) {
        SousProgramme m = new SousProgramme();
        m.afficheMessage();
    }
}
```

Compilation et exécution en Java (suite)

Et la compilation séparée ?

◁ Supposons une classe `SousProgramme` utilisée par ma classe `ProgrammePrincipal`...

◁ qui crée un *nouvel* objet `m` de type `SousProgramme`

◁ et appelle la méthode `afficheMessage()` de l'objet *m*

```
>>> javac SousProgramme.java
>>> javac ProgrammePrincipal.java
>>> java ProgrammePrincipal
message disponible
>>>
```

◁ On n'a fait aucun "include": Java se débrouille pour trouver la classe `SousProgramme`

◁ Pas de makefile non plus :

◁ pas d'exécutable, donc pas d'édition des liens

◁ mieux encore : Java recompile les fichiers **.class** utilisés qui ne sont pas à jour à la compilation d'un fichier **.java**

◁ S'il n'y a pas d'include, à quoi servent les **import** ?

◁ à utiliser des ensembles de *composants* déjà compilés (packages)

◁ SousProgramme.java

```
import java.io.*;
public class SousProgramme {
    public void afficheMessage() {
        System.out.println("message disponible");
    }
}
```

◁ ProgrammePrincipal.java

```
public class ProgrammePrinciapl {
public static void main(String[] argv) {
        SousProgramme m = new SousProgramme();
        m.afficheMessage();
    }
}
```

Types de base et structures algorithmiques de base

- Les Types de base en Java

`boolean` identique au bool du C++

`char` presque identique au C++ (16 bits)

`int, float, double` identique C++
`byte, long, short` identique C++

◁ Tous les types de base ont leur équivalent Classe d'objets

```
Ex. Integer entier = new Integer(5); int
a = entier.intValue();
```

int	Integer
double	Double
float	Float
char	Characte
boolean	Boolean

◁ Les constantes se déclarent avec le mot-clé `final`

```
Ex.: final int nbr_de_cotes = 6;
```

◁ Les opérateurs sont exactement les mêmes qu'en C++
◁ Les structures itératives et conditionnelles du C++ fonctionnent

- Tous les autres types Java sont des classes

Héritage en Java

- Syntaxe :
  - En Java, on dit qu'une classe **B** hérite d'une classe **A** comme ceci :

```
public class B extends A {
...
}
```

- Dès qu'on a écrit le mot-clé **`extends`**, la classe **B** comporte tous les attributs et toutes les méthodes de **A**.
- Remarque : en Java (contrairement à C++), on ne peut hériter que d'une classe à la fois.
- Les paramètres d'une méthode peuvent être dynamiquement d'une classe qui hérite de la classe attendue. <<<<<<

La classe « `Object` »

◁ C'est la classe mère de toutes les classes java : **toutes les classes héritent de `Object`**

◁ Elle contient un certain nombre de méthodes

- ◁ `equals(Object x)` : vérifie l'égalité entre deux objets
- ◁ `clone()` : recopie un objet (*superficiellement*)
- ◁ `toString()` : renvoie une représentation textuelle d'un objet
- ◁ `getClass().getName()` : renvoie la classe de l'objet sous forme de chaîne
- ◁ Il est recommandé de les redéfinir …

The general intent is that, for any object `x`, the expression:
`x.clone() != x`
will be true, and that the expression:
`x.clone().getClass() == x.getClass()`
will be `true`, but these are not absolute requirements. While it is typically the case that:
`x.clone().equals(x)`
will be `true`, this is not an absolute requirement.

Les chaînes de caractères

◁ Il existe un type chaîne de caractères : `String`

```
import java.lang.String;
```

◁ Création et remplissage d'une chaîne de caractères

```
String a = new String("abcdef"); String b = new String("ghijkl");
String c = new String(a+b);
```

Exemples

◁ longueur d'une chaîne de caractères

```
int l = c.length();
```

◁ accès au caractère d'indice `i` ((i + 1)-ème caractère)

```
char ci = c.charAt(3);
```

◁ https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

compact1, compact2, compact3

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

`Serializable, CharSequence, Comparable<String>`

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

The class `String` includes methods for examining individual characters of the sequence, for comparing strings, for string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard

Protection

- Attributs et méthodes d'un objet peuvent être: `private`, `protected`, `public` :

- **`private`** n'est accessible qu'à l'intérieur de la classe qui le contient
- **`protected`** est accessible dans tout le package et dans les sous classes
- **`public`** est bien sûr visible partout

- **`static`** appartient à la classe et non a l'instance de la classe (une sorte de constante de classe)

Des mots-clefs spécifiques : "this"

- permet de lever tout ambiguïté en spécifiant l'objet concerné (l'objet courant dans la classe) sur un exemple:

Exemple

```
class Personne {

    public String nom;

    public Personne(String nom) {

        this.nom = nom;

    }

}
```

```
class Test {

    Personne monsieurX;
    monsieurX = new Personne("Albert");
    . . .
}
```

Des mots-clefs spécifiques : « `super` »

- Constructeur : méthode qui porte le même nom que la classe et qui ne comporte aucun type de retour (même pas `void`)
  (Attention: une classe héritée peut avoir un ou plusieurs constructeurs)
- important de faire appel au constructeur de la classe mère : réutiliser le constructeur comme pour une autre méthode classique

- En Java, utilisation du mot clé `super` :
- `super()` pour appeler le constructeur sans arguments
- `super(…)` peut prendre les arguments d'un constructeur de la classe mère

- `super.uneMethode()` **: `super`, c'est `this` vu avec le type de la super-classe**

Des mots-clefs spécifiques : « `super` » (suite)

Exemple

```
public class SuperClasse {

    public void printMethod() {
        System.out.println("Affiché dans SuperClasse.");
    }
}
public class SousClasse extends SuperClasse {

    // redéfinition de printMethod de la SuperClassse
    public void printMethod() {
        super.printMethod();
        System.out.println("Affiché dans SousClasse");
    }

    public static void main(String[] args) {
        SousClasse s = new SousClasse();
        s.printMethod();
    }
}
```

Conversion ("cast") et «instanceof"

- `instanceof` permet de savoir à quelle classe appartient un objet

  (voir aussi : `getClass().getName()` qui renvoie la classe de l'objet sous forme de chaîne)

```
if( x instanceof Double) {

  System.out.println("x is a Double");

}

else if( param instanceof Integer) {

  System.out.println(« x is an Integer");...
```

Exemples

- Le transtypage (cast) permet de changer le type d'un objet

```
class A { int x = 1; }
class B extends A { String x = "zz";}
class C extends B { boolean x = true;}


public static void main(String[] args) {
C c = new C();

System.out.println(c.x); // true
System.out.println(((B)c).x); // zz
System.out.println(((A)c).x); // 1
```

Autre exemple d'utilisation de "instanceof"

```
class B{ …}
class D extends B{…}
class C {…}

B b = new B();
D d = new D();
C c = new C();

b instanceof B // ??
b instanceof D // ??
d instanceof B // ??
d instanceof D // ??

b = d;

b instanceof B // ??
b instanceof D // ??
c instanceof B // ??
```

Autre exemple d'utilisation de "`instanceof`" (suite et solution)

Exemple

```
class B{ …}
class D extends B{…}
class C {…}

B b = new B();
D d = new D();
C c = new C();

b instanceof B // true
b instanceof D // false
d instanceof B // true
d instanceof D // true

b = d;

b instanceof B // true
b instanceof D // true
c instanceof B // erreur de compilation Erreur No. 365 :
        // impossible de comparer C avec B  en ligne …, colonne ..
```

Quelques instructions spécifiques

- `for()` : permet d'effectuer un traitement sur tous les éléments d'un ensemble

```
for(String sousTab[] : tab) {  j = 0; }
```

- L'affectation conditionnelle :

```
x = bool ? a : b;
```

```
int g = existeFilsGauche() ? filsGauche.hauteur() : 0;
```

Les classes abstraites

◁ Une **méthode** est abstraite si elle est marquée du mot-clé ***abstract***

◁ alors, la **classe** qui la contient doit aussi être abstraite par le mot-clé ***abstract***

◁ Aucun objet de cette classe ne pourra être créée

◁ Les classes dérivées devront surcharger la méthode abstraite pour pouvoir être intanciées

(voir l'exemple à suivre)

classes abstraites (suite)

```
class Rectangle{
    int x, y, largeur, longueur;
    Rectangle(int x, int y, int la, int lo){
       this.x = x; this.y = y;
       largeur = la; longueur = lo;
    }
     void deplaceDe( int dx, int dy){
        x+=dx; y+=dy;
     }
    double perimetre(){
       return 2*(largeur+longueur);
    }
    double surface(){
       return largeur*longueur;
    }
 }

 class Cercle{
    int x, y, rayon;
    Cercle(int x, int y, int r){
       this.x = x; this.y = y;
       rayon = r;
    }
     void deplaceDe( int dx, int dy){
        x+=dx; y+=dy;
     }
    double perimetre(){
       return 2*Math.PI*rayon;
    }
    double surface(){
```

Exemple

classes abstraites (suite)

```
class Rectangle{
    int x, y, largeur, longueur;
    Rectangle(int x, int y, int la, int lo){
       this.x = x; this.y = y;
       largeur = la; longueur = lo;
    }
    void deplaceDe( int dx, int dy){
       x+=dx; y+=dy;
    }
    double perimetre(){
       return 2*(largeur+longueur);
    }
    double surface(){
       return largeur*longueur;
    }
 }

 class Cercle{
    int x, y, rayon;
    Cercle(int x, int y, int r){
       this.x = x; this.y = y;
       rayon = r;
    }
    void deplaceDe( int dx, int dy){
       x+=dx; y+=dy;
    }
    double perimetre(){
       return 2*Math.PI*rayon;
    }
    double surface(){
```

Exemple

```
abstract class Forme{
    int x, y;
    Forme(int x, int y){
       this.x = x; this.y = y;
    }
    void deplaceDe( int dx, int dy){
      x+=dx; y+=dy;
    }
    abstract double perimetre();
    abstract double surface();
}

class Rectangle extends Forme{
    int largeur, longueur;
    Rectangle(int x, int y, int la, int lo){
       super(x, y);
       largeur = la; longueur = lo;
    }
    double perimetre(){
      return 2*(largeur+longueur);
    }
    double surface(){
      return largeur*longueur;
    }
}
class Cercle extends Forme{
    int rayon;
    Cercle(int x, int y, int r){
        super(x, y);
        rayon = r;
    }
```

Les exceptions

- Permettent de détecter et de gérer les **situations exceptionnelles**
- Elles sont typées : e.g. `ArrayIndexOutOfBoundsException`
- Peuvent donner des infos complémentaires
- Localisation de l'exception dans le code

- Les exceptions remplacent les « plantages » des programmes C++
- Beaucoup plus facile à débugger mais obligent le programmeur à prendre en compte des cas particuliers d'exécution du programme
- Inconvénient : traitement des exceptions lent
- Remarque : elles existent aussi en C++, mais peu utilisées
- Toutes les **exceptions personnalisées** héritent de la classe **`Exception`**
- Mécanisme très performant puisqu'on peut créer ses propres exceptions

Les exceptions (suite): traitements

Des parties de code exécutées seulement dans des circonstances particulières (e.g. évitement de division par zéro) et provoquant un déroutement du programme normal.

Exemple

```
if (t == null)
throw new NullPointerException()
```

Trois étapes:

◁ Générer (**`throw`**) une exception lorsqu'une situation "anormale" se présente.

permet d'éviter pour le moment de gérer ce problème du pointeur nul (il sera donc traité autre part).

◁ Surveiller une région (bloc **`try`**) qui est une partie de code dans laquelle des exceptions peuvent être générées.

◁ capter les exceptions (**`catch`**) pour les traiter dans un bloc spécifique de gestion d'exceptions (un bloc pour chaque type d'exception possible).

```
try {
// Code susceptible de générer des exceptions
}
```

```
try { // Code susceptible de générer des
exceptions
        } catch(Type1 id1) {
        // Traitement des exceptions de type Type1
        } catch(Type2 id2) {
        // Traitement des exceptions de type Type2
        } catch(NullPointerException e) {
        System.out.println("NullPointerException");
        }
```

Exceptions (suite) : un exemple simple

```
class Essai{
  public static void main(String[] args) {
        int j = 1, i = 0;

        try {
        System.out.println(j/i);
        } catch (ArithmeticException e) {
        System.out.println("Division par zéro !");
        }
        System.out.println("Fin de traitement");
  }
}
```

Remarque :

la génération (**throw**) n'apparaît pas ici car elle est faite lors de la division

Exemple

```
>>> javac Essai.java
>>> java Essai
Division par zéro !
Fin de traitement
```

Les exceptions personnalisées

Les exceptions personnalisées doivent hériter de la classe `Exception`

Exemple :

Exemple

```
class NombreNegatifException extends Exception{
  public NombreNegatifException(){
    System.out.println("Vous avez un nombre négatif !");
  }
}
```

Les fichiers en Java

Les opérations à effectuer

- Ouvrir un fichier :
  - *en lecture* : création d'un objet de la classe `FileReader`
  - *en écriture* : objet de la classe `FileWriter`

  qui héritent de la classe `File`
- Lecture ou écriture
  par les méthodes `read()` ou `write(x)`
- Fermer un fichier : par la méthode `close()`

Traitement des fichiers en Java

Exemple : copie simple de fichier

FileCopy.java

```
import java.io.*;
public class FileCopy {
    public static void main(String[] args) throws IOException {
        FileReader in = new FileReader(new File("test.txt"));
        FileWriter out = new FileWriter(new File("test_copie.txt"));
        int c;
        // lecture caractere par caractere
        while ((c = in.read()) != -1) { // lecture du caractere c
                // dans le flux in et verification de non-fin-de-fichier

                out.write(c);  // ecriture dans le flux out
            }
        in.close();
        out.close();
    }
}
```

Traitement des fichiers en Java

Ecriture dans un fichier

◁ L'écriture dans un fichier est assurée par la classe *FileWriter*

```
import java.io.*;
...
FileWriter fichierSortie = new FileWriter("fichier.txt"); ...
fichierSortie.write("chaine a ecrire");
...
fichierSortie.close();
```

◁ L'utilisation de la classe *File* permet de s'abstraire des problèmes du système

Une nouvelle commande `dir`

```
/*  Listage du contenu d'un repertoire.
   Classe File pour determiner le contenu d'un repertoire */

public static void dir(File pathName) {

 System.out.println("Contenu du repertoire "+pathName.getName());

 List<File> lf = Arrays.asList(pathName.listFiles());


 for (File f : lf) {


    if (f.isDirectory()) {

        System.out.println("R - "+f.getName());
     }

    else  {

        System.out.println(f.getName()+" (« +f.length()+")");
    }
  }

}
```

TestFichiers.java

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


public class TestFichiers {

public static void main(String[] argv) {

File file = new File("/Users/roux");

dir(file); } Exemple : copie simple de fichier
```

FileCopy.java

```
import java.io.*; public class FileCopy {

public static void main(String[] args) throws IOException
{ FileReader in = new FileReader(new File("test.txt"));
FileWriter out = new FileWriter(new
File("test_copie.txt")); int c;

// lecture caractere par caractere
while ((c = in.read()) != -1) { // lecture du caractere c

                // dans le flux in et verification de non-
fin-de-fichier
                out.write(c);  // ecriture dans le flux
out
            }
in.close();

        out.close();
    }
}
```

Lecture/Ecriture de fichiers (au travers d'un exemple)

Exemple

```
// Contenu d'un fichier texte
class Contenu extends LinkedList<String>{
   // Constructeur : Afficher le contenu d'un fichier texte
   public Contenu(String texte) throws IOException    {
        Reader reader = new FileReader(texte+ ».txt");      // Lecture du fichier texte
        BufferedReader in = new BufferedReader(reader);
        String ligne = in.readLine();          // Prise en compte d'une ligne
        while (ligne != null)  {
             // traitement de la ligne courante: decouper les mots separes par des delimiteurs
             String delimiteurs = "/0123456789∞~^%$£*-_=+|#'.,;:?!() {}[]`<>\"\t\\<>´ª";
             StringTokenizer st = new StringTokenizer (ligne, delimiteurs);
             while (st.hasMoreTokens())  {
                  String mot = st.nextToken();                  // Traitement du mot courant
                  mot=mot.toLowerCase();
                  System.out.println(mot);
                  this.add((String)mot);
             }
             ligne = in.readLine();          // Lecture de la prochaine ligne
        }
   }
```

Lecture/Ecriture de fichiers (suite)

```
// Ecrire tous les mots dans un second fichier texte
public void ecri() throws IOException{
    Writer writer = new FileWriter(this.nom+"_modifie.txt");
    writer.write("##\n#\n# Contenu de "+this.nom+".txt :
                  \n#\n#\t"+this.size()+" mots.\n#\n###\n\n");
    // l'entête puis tous les mots de la liste qui a été constituée
    for(Object k : this)  {  // this est une LinkedList<String>
        writer.write(k+"\n");
        writer.flush();
    }
    writer.close();
}
}
```

Les types énumérés
(une idée simple introduite au travers d'un exemple)

```
public enum Jour {
   LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE;
}

class EssaiJour {
   public static void main(String[] arg) {
   Jour jour = Jour.valueOf(arg[0]);
      if (jour == Jour.SAMEDI) System.out.print("fin de semaine : ");
      switch(jour)  {
         case SAMEDI :
         case DIMANCHE :
            System.out.println("se reposer");
            break;
         default :
            System.out.println("travailler");
            break;
      }
   }
}
```