Programmation en Java

Nicolas Sendrier

INRIA, Domaine de Voluceau, BP 105,
78153 Le Chesnay CEDEX, FRANCE

3 mai 2001

Crible d'Eratosthène

```
class Erato {
  static boolean [] crible(int n) {
    boolean[] elimine = new boolean[n + 1];
    elimine[0] = elimine[1] = true;
    if (n >= 2)
      for (int p = 2; (p * p) <= n; p++)
        if (!elimine[p])
          for (int i = 2 * p; i <= n; i += p)
            elimine[i] = true;

    return elimine;
  }

  public static void main(String args[]) {
    boolean [] elimine =
        crible(Integer.parseInt(args[0]));

    for (int i = 0; i < elimine.length; i++)
      if (!elimine[i]) {
        System.out.println(i);
      }
  }
}
```

Exemple d'objet : `BigInteger`

```
import java.math.BigInteger;

public class FactoBig {

  static BigInteger un = new BigInteger("1");
// static BigInteger un = BigInteger.valueOf(1);

  static BigInteger fact(BigInteger n) {
    if (n.compareTo(un) <= 0)
      return n;
    else
      return n.multiply(fact(n.subtract(un)));
  }

  public static void main (String [] args) {
    BigInteger n = new BigInteger(args[0]);
    System.out.println(n + "! = " + fact(n));
  }
}

% java FactoBig 100
100! = 93326215443944152681699238856266700490715\
15968264381621468592963895217599993229915608941\
14639761565182862536979208272237582511852109168\
8640000000000000000000000000
```

Les objets

```
class Point {
  // variables d'instances
  int x, y;

  // variables globales (origine)
  static int x0 = 0, y0 = 0;

  // constructeur
  Point(int a, int b) {
    x = a; y = b;
  }

  static int norme(Point p) {
    return Math.max(Math.abs(p.x - x0),
                    Math.abs(p.y - y0));
  }

  // la même sans le ‘static’
  int norme() {
    return Math.max(Math.abs(x - x0),
                    Math.abs(y - y0));
  }

  static void main(String [] args) {
    Point p = new Point(1,6);
    System.out.println(norme(p));
    System.out.println(p.norme());
  }
}
```

La méthode `toString()`

Si une méthode `toString()` existe dans la classe `Point`

```
public String toString() {
  return x + "\t" + y;
}
```

alors à l'exécution de

```
  Point p = new Point(4,9);
  System.out.println(p);
```

nous aurons

```
4.0     9.0
```

au lieu de (par exemple)

```
Point@23bcc1
```

Exemple d'utilisation

Depuis une autre classe :

```
Point p1 = new Point(1,6);
Point p2 = new Point(3,3);

System.out.println(p1.norme() + "\t" +
                   Point.norme(p2));
Point.x0 = p2.x;
Point.y0 = p2.y;

System.out.println(p1.norme() + "\t" +
                   Point.norme(p2));
```

À l'exécution on obtiendrait :

```
6.0     3.0
3.0     0.0
```

Le changement d'origine vaut pour toutes les instances de la classe `Point`.

Passage par référence

Si dans la classe `Point` on ajoute

```
void homothetie(double lambda) {
  x = x0 + lambda * (x - x0);
  y = y0 + lambda * (y - y0);
}

static void foo(Point p, double lambda) {
  p.x = x0 + lambda * (p.x - x0);
  p.y = y0 + lambda * (p.y - y0);
}
```

La méthode statique `foo()` pourrait être déclarée dans une autre classe, il faudrait alors remplacer `x0` et `y0` par `Point.x0` et `Point.y0`.

```
Point p = new Point(4,9);

System.out.println(p);
p.homothetie(0.5);
System.out.println(p);
Point.foo(p, 0.5);
System.out.println(p);
```

À l'exécution

```
4.0     9.0
2.0     4.5
1.0     2.25
```

Liste chaînées

```
class Liste {
  // le mot courant
  String tete;
  // la suite de la phrase
  Liste suite;

  // Constructeurs
  public Liste(String mot, Liste phrase) {
    tete = mot;
    suite = phrase;
  }

  public Liste(String mot) {
    tete = mot;
    suite = null; // inutile, valeur par défaut
  }

  // Transformer un tableau en liste
  public Liste(String [] mots) {
    int n = mots.length;
    Liste phrase = new Liste(mots[n-1]);
    for(int i = 2; i <= n; i++)
      phrase = new Liste(mots[n-i],phrase);
    tete = phrase.tete;
    suite = phrase.suite;
  }
```

Listes chaînées (suite)

```
  // Transformer un tableau en liste
  // version récursive
  public Liste(String [] mots, int i) {
    if (i < mots.length - 1) {
      tete = mots[i];
      suite = new Liste(mots, i + 1);
    }
    else
      tete = mots[i];
  }

  // pour l'impression
  public String toString() {
    if (suite == null)
      return tete;
    else
      return tete + " " + suite.toString();
  }

  static String dernierMot(Liste phrase) {
    while (phrase.suite != null)
      phrase = phrase.suite;
    return phrase.tete;
  }
}
```

Gestion des exceptions

```
class Essai {
  static public void main(String [] args) {
    int x = 0;
    try {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      x = a / b;
    }
    catch (ArithmeticException e) {
      System.out.println("division par zéro");
    }
    catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("2 arguments!");
    }
    catch (Exception e) { // autre erreur
      System.out.println(e);
    }

    // et le programme continue ...

    System.out.println(x);

    ...
  }
}
```

```
class NegException extends Exception {
  NegException() {}
}

class RacineCarree {
  static double rac(double x)
                        throws NegException {
    if (x < 0)
      throw new NegException();
    else
      return Math.sqrt(x);
  }

  public static void main(String [] args) {
    double x = ...
    try {
      System.out.println(rac(x));
    }
    catch (NegException e) {
      System.out.println(
               "l'argument doit être positif");
    }
  }
}
```

```
class NegException extends RuntimeException {
  NegException() {}
}

class RacineCarree {
  static double rac(double x) {
    if (x < 0)
      throw new NegException();
    else
      return Math.sqrt(x);
  }

  public static void main(String [] args) {
    double x = ...
    try {
      System.out.println(rac(x));
    }
    catch (NegException e) {
      System.out.println(rac(-x) + " i");
    }
  }
}
```