Slide 1

Centre International de Mathématiques Pures et Appliquées

International Center for Pure and Applied Mathematics

A CIMPA-UNSA-INRIA-UNESCO-VIETNAM SCHOOL ON

Objects and Network

Part : Threads Exercises

Emmanuel Chailloux

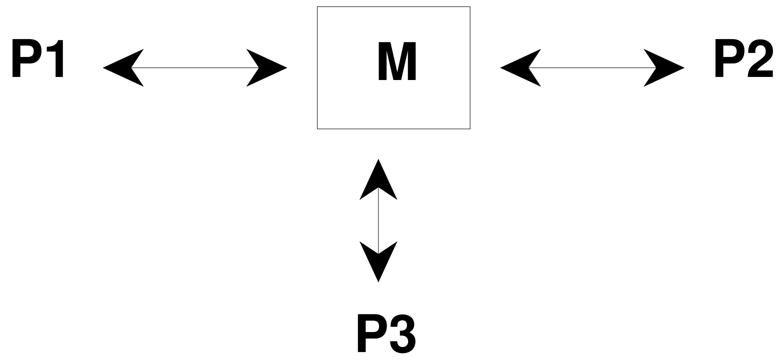
Emmanuel.Chailloux@lip6.fr

http://www-spi.lip6.fr/~emmanuel

Slide 2

Shared memory

All process interact on the same memory :



⇒ implicit communication and explicit synchronization!!!

Slide 3

Relations between Threads

1. Unrelated Threads
2. Related but Unsynchronized Threads
3. Mutually Exclusive Threads
   - mutual exclusion over a block of statements
   - mutual exclusion over a method
4. Communicating Mutually-Exclusive Threads

Slide 4

Threads Exercises

- Game of Life : case 1,2 and 3
- Producer-Consumer : case 3 and case 4
- World of robots : case 3 or case 4

1 Exercises : Threads

1.1 A simulation for cellulars automaton

The following source program is a text version for the *Game of Life* (a simulation for cellulars automats).

```
/**
  @author Emmanuel Chailloux
  @organization Universite Pierre et Marie Curie
  @version 1.0
*/


/**
    World representation for cells, contains
      world size
      Cells Array
      generation number
*/

class World {

  int maxx, maxy;
  Cell m[][];
  int num=1;

/**
  constructor without parameter : builds a fixed size world : size 20x20
*/

  World() {
    maxx = 20;
    maxy = 8;
    m=new Cell[maxx][maxy];
    emptyWorld();

  }

/**
  constructor with size : builds a variable size world
*/

  World(int x, int y) {
     maxx=x;
     maxy=y;
     m = new Cell[maxx][maxy];
     emptyWorld();
  }

/**
  allocates each cell
*/

  void emptyWorld() {
    for (int i=0; i<maxx; i++) {
      for (int j=0; j<maxy; j++) {
        m[i][j]=new Cell();
      }
    }
  }

/**
  random filling
  empty cells (80%)
*/

  void initRandom() {
    for (int i=0; i<maxx; i++) {
      for (int j=0; j<maxy; j++) {
        if (Math.random() > 0.8) setCell(i,j,new Cell(true));
      }
    }
  }

/**
  filling at hand
  needing to indicate lived cells (giving its coordinates)
  to exit, you indicate an impossible coordinate
*/

  void initHand() {
    int x,y;
    while(true) {
      System.out.println("Enter coordinates (2 lines) ");
      System.out.println("or -1 -1 to exit");
      x=ES.lire_entier();
      y=ES.lire_entier();
      if ((x>=0)&&(x<maxx)&&(y>=0)&&(y<maxy))
```

```
        {setCell(x,y,new Cell(true));}
      else {break;}
    }
  }

/**
  modifies Cell value
*/

  void setCell(int x,int y, Cell c) {
    m[x][y]=c;
  }


/**
  asks Cell value
*/

  Cell getCell(int x, int y) {
    return m[x][y];
  }


/**
  returns the cells array
*/

  Cell [][] getCells(){return m;}

/**
  computes the neighbors number for a cell.
  supposes each cell has 8 neighbors
  computes all alive cells on a square 3x3 without itself.

*/

  int neighbors(int x, int y) {
    int r=0;
    for (int i = x-1;i≤x+1;i++) {
      int k = (i+maxx) % maxx;
      for (int j = y-1;j≤y+1;j++) {
        int l = (j+maxy) % maxy;
        if (m[k][l].isAlive()) r++;
      }
    }
    if (m[x][y].isAlive()) r--;
    return r;
  }

/**
  computes the next generation
     for each cell, computes the neighbors number
     then applies the life and death rules
*/

  void nextGen() {
    World m2 = new World(maxx,maxy);                    // all cells are empty.


    for (int i=0;i<maxx;i++) {
      for (int j=0; j<maxy; j++) {
        int n = neighbors(i,j);
        if (m[i][j].isAlive()) {
          if ((n== 2) || (n==3)) {m2.setCell(i,j,new Cell(true));}
        }
        else {
          if (n==3) {m2.setCell(i,j,new Cell(true));}
        }
      }
    }
    m=m2.getCells();
    num++;

  }

/**
  display nth generation
*/

  void displayGen() {
    for (int j=0; j<maxy; j++) {
      for (int i=0; i<maxx; i++) {
        if (m[i][j].isAlive())
          System.out.print("* ");
        else
          System.out.print(".  ");
      }
      System.out.println();
    }
    System.out.println("Generation : "+num);
  }

}
```

```
/**
  minimal classe for cells
*/

class Cell {

  boolean val;

  Cell() {val=false;}
  Cell (boolean b) {val = b;}

  boolean isAlive() {return val;}

}



/**
  classe de l'application GameOfLife
*/

class GameOfLife {


/**
  world initiazation
     size and filling kind
     all is inside the init method

*/

/**
   determine la taille du World et le type de remplissage
*/

  public static World init() {
    String s;

    World m;

    System.out.print("Do you want to give world size?  (Y/N)");
    s=ES.lire_chaine();
    System.out.println();
    if (!(s.equals("Y"))) {m = new World(30,10);}
    else {
      int x,y;
      System.out.print("width :  ");
      x=ES.lire_entier();
      System.out.print("height :  ");
      y=ES.lire_entier();
      m = new World(x,y);
    }

    do {
      System.out.println("Press key <R> for a random filling");
      System.out.println(" or <H> filling at hand");
      s=ES.lire_chaine();
    } while ((!(s.equals("R"))) && (!(s.equals("H"))));

    if (s.equals("R")) m.initRandom(); else m.initHand();
    return m;
  }


/**
  loop to wait for events
  press the key <E> to exit (break)
*/

  public static void loop(World m) {
    String s;

    System.out.println("Press the key <E> to exit");
    System.out.println(" or another key to continue");

    while (true) {
      m.displayGen();
      s=ES.lire_chaine();
      if (s.equals("E")) break;
      else m.nextGen();
    }
  }

/**
  last method called before to exit
*/

  public static void end() {
    System.out.println("\n ...  See you soon \n");
  }

/**
```

```
   entry point for the application
*/

   public static void main(String args[]) {
      World m;
      m=init();
      loop(m);
      end();
   }

}
```

To follow the different tasks :

1. To extend the inheritence tree to build an *applet* from this application. You don't have to modify the original source. You can display the old and the new generations.

2. To write an HTML file which starts two applets of "Game of Life". To test it with `appletviewer` and a *browser*.

3. To modify the code to run four (independent) threads on the same world (you slice the world in four parts). Each thread can display its cells with a different color.

4. To modify this program with overlap between threads. You need to synchronize threads.

1.2 A Producer-Consumer

It's a classical example.

- To create basic classes to represent `Product` and a `Shop` (represented by an Array) where Product instances are stored.

- To create a `Producer` class which can build Product instances and store them inside a Shop instance.

- To create a `Consumer` class which can take `Product` instances from a `Shop` instance.

- What are the different problems of synchronisation?

- To integrate a solution and to run your program with a `Producer` instance and three Consumer instances, synchronized to a `Shop` instance.

1.3 A World of robots

In a special world, represented by a plane, somes robots move. The world contains cells where robots can stay or move. There are different kinds of robots. Each robot will be represented by a thread.

1. To define a `World` *abstract* class.

2. To define a concrete sub-class for a plane world.

3. To define an abstract class for the robots.
4. To define a concrete class for an "hazardeous"robot. It'll decide by random what will its next position.
5. To integrate these different classes to build a text application, where robot movements will be displayed.
6. To build an *applet* to have a graphical display. A robot will be represented by a colored rectangle.

If you have time, to add others kinds of robots :

- a stayed robot (without movement)
- a *right* robot, which turns right when it can't move.
- a *friendly* robot, which follows the closest robot. (in this case it needs to known positions of others robots).
- To try also a world which communicates at boards.