Programmering Fortsättning 2010

Övning 5

Joachim von Hacht

1 Trådar

1. The following program involves 3 threads: a Student thread, who has to workLikeCrazy() until it is time to go home, a good teacher, and an evil teacher. The good teacher, in the "main" thread, calls comeToEnd() to set goHome to true and then waits for the student to finish. At the same time, the timer task, representing the bad teacher, tries to prevent the student from going home by calling workLikeCrazy. Obviously, it is of high practical interest from a student's standpoint to know whether the evil teacher succeeds.

   - What does the keyword volatile mean? Look up the language specification.
   - What is the specification of Thread.join and of Thread.sleep? Does a thread that sleeps X ms continue exactly after X ms? Lastly, how does the method Timer.schedule work in the main program below?
   - Does the evil teacher win or not (it does not suffice to run the program, you need to explain why)?

```
package threads.primitives;
// Not really good to extend
public class Student extends Thread {
    static private final int studentDelay = 150;
    private volatile boolean goHome = false;
    private final Object lock = new Object();
    public void run() {
        while (!goHome) {
            System.out.println("Working..");
            workLikeCrazy();
        }
        System.out.println("School's out!");
    }
```

```
    private void workLikeCrazy() {
        try {
            Thread.sleep(studentDelay); // Student works...
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }
    void keepWorking() {
        while (true) {
            synchronized (lock) {
                goHome = false;
            }
        }
    }
    void comeToEnd() throws InterruptedException {
        synchronized (lock) {
            goHome = true;
            join();
        }
    }
}

package threads.primitives;
import java.util.Timer;
import java.util.TimerTask;
public class Main {
    static private final int evilTeacherDelay = 100;
    static private final int goodTeacherDelay = 500;
    public static void main(String[] arg) throws InterruptedException {
        final Student student = new Student();
        student.start();

        Timer t = new Timer(true);
        t.schedule(new TimerTask() {
                public void run() {student.keepWorking(); }
            }, evilTeacherDelay);

        Thread.sleep (goodTeacherDelay);
        student.comeToEnd();
    }
}
```

2p

2. **Race conditions:** In the RGB ("Red-Green-Blue") color model, a color is represented as the triple of (red, green, blue) values, where each of the three primary

colors is encoded with a value $v, 0 \leq v \leq 255$; the higher the value of an entry of v, the higher the intensity of the corresponding primary color (255 means full intensity). An illustration, the following table lists the RGB encoding of various colors:

RGB	Color
(0,0,0)	Black
(255,0,0)	Red
(255,255,0)	Yellow
(220,20,60)	Crimson red

Consider the class Color, a simplified version of java.awt.Color.

```
package threads.racecondition;
public class Color {
    private int red;
    private int green;
    private int blue;
    public Color(int r, int g, int b) {
        if (!isRBGColor(r, g, b)) {
            throw new IllegalArgumentException();
        }
        this.red = r;
        this.green = g;
        this.blue = b;
    }
    public void setColor(int r, int g, int b) {
        if (!isRBGColor(r, g, b)) {
            throw new IllegalArgumentException();
        }
        this.red = r;
        this.green = g;
        this.blue = b;
    }
    public int[] getColor() {
        int[] retVal = new int[3];
        retVal[0] = red;
        retVal[1] = green;
        retVal[2] = blue;
        return retVal;
    }

    public void invert() {
        red = 255 - red;
```

```
        green = 255 - green;
        blue = 255 - blue;
    }
    private static boolean isRBGColor(int red, int green, int blue) {
        return (0 <= red && red <= 255) && ( 0 <= green &&
                green <= 255 ) &&  ( 0<= blue && blue <= 255);
    }
}
```

Assume that there is one thread, B, which tries to set the color of a Color object to blue (setColor(0,0,255)), and another second thread, R, which tries to set the color of the same object to red. Show that the class is not thread-safe by constructing series of interleavings that leave the Color object illustrating two different race conditions:

- Construct a series of interleavings so that, when thread B returns, the color of the Color object is red (and not blue).
- Construct a series of interleaving so that, when the two threads B, R return, the color is neither red nor blue (the object, thus, corrupted). 1p

3. Demonstrate the following techniques to make the previous class Color threadsafe.

   a) Use synchronized blocks (not methods).
   b) Make class immutable.
   c) Create a threadsafe wrapper class (i.e. all calls to Color goes thru the wrapper class) 3p

4. **Deadlock:** När man kör Person nedan får man följande utskrift;

```
Daniel says: I got a question from Pelle start thinking...
Pelle says: I got a question from Daniel start thinking...
Daniel finished thinking
Pelle says: I got an answer from Daniel (<---)
Pelle finished thinking
Daniel says: I got an answer from Pelle
```

Som synes blir Pelle avbruten i stitt tänkande när Daniel svarar honom (<—). Gör så att varken Pelle eller Daniel blir avbrutna under sitt tänkande. Så här skall det se ut;

```
Pelle says: I got a question from Daniel start thinking...
Daniel says: I got a question from Pelle start thinking...
Daniel finished thinking
Pelle finished thinking
```

```
Daniel says: I got an answer from Pelle
Pelle says: I got an answer from Daniel
```

Programmet[1];

```
package threads.deadlock;
public class Person extends Thread {
    private String name;
    private Person whomToAsk;
    public Person(String name) {
        this.name = name;
    }
    public void setWhomToAsk(Person whomToAsk) {
        this.whomToAsk = whomToAsk;
    }
    public String toString() {
        return name;
    }
    private void getAQuestionFrom(Person p) {
        try {
            System.out.println(this + " says: I got a question from " +
                    p + " start thinking...");
            // Think a while before answering...
            Thread.sleep((int) (Math.random() * 1000) + 1000);
            System.out.println(this + " finished thinking");
            p.getAnAnswerFrom(this);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
    private void getAnAnswerFrom(Person p) {
        System.out.println(this + " says: I got an answer from " + p);
    }
    public void run() {
        whomToAsk.getAQuestionFrom(this);
    }
    public static void main(String[] args) {
        Person pelle = new Person("Pelle");
        Person daniel = new Person("Daniel");
        pelle.setWhomToAsk(daniel);
        daniel.setWhomToAsk(pelle);
        pelle.start();
        daniel.start();
```

[1] Tack till Daniel E. W. och Pelle E.

```
    }
}
```
1p

2 Generiska klasser

1. Man kan simulera högre ordningen funktioner m.h.a (generiska) klasser. Antag att vi vill att det skall fungera som i main nedan.

   a) Even implementerar gränssnittet UnaryPred och avgör om ett tal är jämnt. Implementera Even och UnaryMapper. UnaryMapper behöver inte ta hänsyn till runtime-typen på listan (LinkedList, ArrayList,...). 1p

   b) Sign tar en double och returnerar -1, 0 elle 1 om talet är negativt, noll respektive positivt. Låt Sign implementera ett generisk gränssnitt, UnaryOp. Implementera gränssnittet, Sign och BinaryMapper. 2p

```
package generics.higherorder;
import java.util.ArrayList;
import java.util.List;
public class Main {
   public static void main(String[] args) {
      List<Integer> ini = new ArrayList<Integer>();
      ini.add(2);
      ini.add(3);
      ini.add(9);
      UnaryMapper<Integer> um = new UnaryMapper<Integer>();
      List<Boolean> bout = um.map(new Even(), ini);
      for(boolean i : bout){
         System.out.println(i);//true,false,false
      }
      List<Double> ind = new ArrayList<Double>();
      BinaryMapper<Double, Integer> bm = new BinaryMapper<Double, Integer>();
      ind.add(-2.3);
      ind.add(0.0);
      ind.add(333.45);
      List<Integer> out = bm.map(new Sign(), ind);
      for(Integer i : out){
         System.out.println(i);//-1,0,1
      }
   }
}

package generics.higherorder;
public interface UnaryPred<T> {
```

```
    public boolean p(T o);
}
```