

Programmering Fortsättning

Övning 6, Generiska Typer, Parallellism och Reflexiv programmering

Joachim von Hacht

1 Generiska typer

1. För varje rad i main nedan ange om;

- a) Raden kompilerar,
- b) Kompilerar med en varning,
- c) Ger ett run-time fel,
- d) Kompileras och exekveras utan problem.

Vid varningar eller fel ange vad som är orsaken.

1p

```
package _1_1;
public class Main611 {
    public static void main(String[] args) {
        FoodBag<Sandwich> bag1 = new FoodBag<HerringSandwich>();
        FoodBag<?> bag2 = new FoodBag<CheeseSandwich>();
        bag2.setFood(new CheeseSandwich());
        FoodBag<HerringSandwich> bag3 = new FoodBag<Sandwich>();
        FoodBag bag4 = new FoodBag();
        bag4.setFood(new CheeseSandwich());
        CheeseSandwich cs = bag2.getFood();
        Object ocs = bag4.getFood();
        FoodBag<Sandwich> bag5 = new ExtendedFoodBag<Sandwich>();
        FoodBag<? extends Sandwich> bag6 = new FoodBag<HerringSandwich>();
    }
}

package _1_1;
public class FoodBag<T> {
```

```
private T food;
public void setFood(T t) {food = t;}
public T getFood() {return food;}
}
```

```
package _1_1;
public class ExtendedFoodBag<T> extends FoodBag<T> {}
```

```
package _1_1;
public class Sandwich {}
```

```
package _1_1;
public class HerringSandwich extends Sandwich {}
```

```
package _1_1;
public class CheeseSandwich extends Sandwich {}
```

2. Koden nedan fungerar inte. Förklara och rätta till. Tips: Begränsade jokrar.

1p

```
package _1_2;
import java.util.Vector;
public class Main612 {
    public static void main(String[] args) {
        // Create and populate a vector of frames
        Vector<Frame> frames = new Vector<Frame>();
        frames.add(new Frame());
        frames.add(new Frame());
        // Create and populate a vector of dialogs
        Vector<Dialog> dialogs = new Vector<Dialog>();
        dialogs.add(new Dialog());
        dialogs.add(new Dialog());
        dialogs.add(new Dialog());
        //Loop through all frames and draw them
        Vector<Window> windows = frames;
        for (Window w: windows) {
            w.draw();
        }
        // Loop through all dialogs and draw them
        windows = dialogs;
        for (Window w: windows) {
            w.draw();
        }
    }
}
```

```
package _1_2;
public abstract class Window {
    public abstract void draw();
}

package _1_2;
public class Frame extends Window {
    @Override
    public void draw() {
        System.out.println("Frame drawn.");
    }
}

package _1_2;
public class Dialog extends Window {
    @Override
    public void draw() {
        System.out.println("Dialog drawn.");
    }
}
```

2 Simulering av högre ordningens funktioner¹

1. I Java kan man kan simulera högre ordningen funktioner. 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 eller 1 om talet är negativt, noll respektive positivt. Låt Sign implementera ett generisk gränssnitt, UnaryOp. Implementera gränssnittet, Sign och BinaryMapper.

1p

```
package _2_1;
import java.util.ArrayList;
import java.util.List;
public class Main621 {
    public static void main(String[] args) {
        List<Integer> ini = new ArrayList<Integer>();
        ini.add(2);
        ini.add(3);
        ini.add(9);
    }
}
```

¹Kommer inte att behöva simuleras i framtiden, Java 8 löser detta.

```

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 _2_1;
public interface UnaryPred<T> {
    public boolean p(T o);
}

```

3 Trådar

Uppgifterna här innehåller mer text än svårighet....

1. **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.

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:

- a) Construct a series of interleavings so that, when thread B returns, the color of the Color object is red (and not blue).
 - b) 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
2. 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) 1p
3. **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 sitt 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
```

2p

Programmet²;

```
package _3_3;
public class Person extends Thread {
    private String name;
    private Person whomToAsk;
    public Person(String name) {
        this.name = name;
    }
    public void setWhomToAsk(Person whomToAsk) {
```

²Tack till Daniel E. W. och Pelle E.

```
        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();
    }
}
```

4 Reflection

1. Testning av privata metoder är problematiskt. Antag att vi trots allt har goda skäl till att vilja testa några privata metoder. Vi vill inte ändra i befintlig kod (till public). Lös det hela genom att skapa en hjälpklass PrivateAccessor (som använder reflection). Ungefär så här är det tänkt att fungera;

2p

```
Adder a = new Adder(); // Class with private sum-method
PrivateAccessor pa = new PrivateAccessor( a, "sum",
                                           Integer.class, Integer.class);
pa.setParams(1, 3);
```

```
// Will execute sum method  
Integer i = (Integer) pa.test();  
Assert.assertTrue( i == 4);
```