Programmering Fortsättning

Övning 3, Tillstånd och Beroenden

Joachim von Hacht

1 Klass och objekt

1. För varje relevant rad i koden nedan avgör om den kompilerar och motivera varför eller varför inte. 1p

```
package _1_1;
public class Mix {
   public static int i = 1;
   public int j = 2;
   public static void doIt(){
      i = 5;
      i = j;
      j = i;
      doOther();
      this.doOther();
      new Mix().j = 5;
      new Mix().doOther();
   }
   public void doOther(){
      i = 5;
      i = j;
      j = i;
      doIt();
   }
}
```

2. Vad händer och vad händer inte? Förklara! 1p

```
package _1_2;
public class StaticCall {
    public static void main(String[] args) {
```

```
        Math m = null;
        System.out.println(m.random());
    }
}
```

2 Inre klasser

1. Vilka rader kommer inte att kompilera? Om man kommenterar bort dessa och kör vad kommer att skrivas ut? 2p

```
package _2_1;

//Thanks to Wayne Pollock, Tampa Florida USA.
class Outer {
  int num = 1;
  static int staticNum = 1;
  class Inner {
    int num = 2;
    void aMethod() {
      System.out.println("num=" + num);
      System.out.println("this.num=" + this.num);
      System.out.println("Inner.this.num=" + Inner.this.num);
      System.out.println("Outer.this.num=" + Outer.this.num);
      System.out.println("staticNum=" + staticNum);
      System.out.println("this.staticNum=" + this.staticNum);
      System.out.println("Inner.this.staticNum=" + Inner.this.staticNum);
      System.out.println("Outer.this.staticNum=" + Outer.this.staticNum);//Waring
      System.out.println("Outer.staticNum=" + Outer.staticNum);
    }
  }

  static class Nested {  //Note static
    int num = 3;
    void aMethod() {
      System.out.println("num=" + num);
      System.out.println("this.num=" + this.num);
      System.out.println("Nested.this.num=" + Nested.this.num);
      System.out.println("Outer.this.num=" + Outer.this.num);
      System.out.println("staticNum=" + staticNum);
      System.out.println("Outer.staticNum=" + Outer.staticNum);
      System.out.println("Nested.staticNum=" + Nested.staticNum);
      System.out.println("this.staticNum=" + this.staticNum);
      System.out.println("Nested.this.staticNum=" + Nested.this.staticNum);
```

```
      System.out.println("Outer.this.staticNum=" + Outer.this.staticNum);
    }
  }
}
```

3 Initiering

1. Vid en första anblick verkar programmet nedan skriva ut "TheKing wears a size ...(år från 1930)...belt." Initiering kan dock ställa till det. Vad händer och varför? 1p

```
package _3_1;
import java.util.Calendar;
public class TheKing {
  public static final TheKing theKing = new TheKing();
  private final int beltSize;
  private static final int this_year = Calendar.getInstance()
      .get(Calendar.YEAR);

  private TheKing() {
    beltSize = this_year - 1930;
  }

  public int beltSize() {
    return beltSize;
  }

  public static void main(String[] args) {
    System.out.println("TheKing wears a size " +
        theKing.beltSize() + " belt.");
  }
}
```

2. Vad skrivs ut i main? Förklara! Vad händer om man avkommenterar /* static */? 1p

```
package _3_2;
public class Parent {
    private int val;
    public Parent() {
        val = lookup();
    }
    public int value() {
        return val;
    }
    public int lookup() {
```

```
        return 5;// Silly
    }
}

package _3_2;
public class Child extends Parent {
    private /*static*/ int num = 10;
    public int lookup() {
        return num;
    }
}

package _3_2;
public class Main332 {
    public static void main(String args[]) {
        Child child = new Child();
        System.out.println("child.value() returns " + child.value());

        Parent parent = new Child();
        System.out.println("parent.value() returns " + parent.value());

        Parent parent2 = new Parent();
        System.out.println("parent2.value() returns " + parent2.value());
    }
}
```

4 Mutabilitet

1. Förbättra klassen nedan. Ett krav är att den skall vara icke-muterbar (Date är muterbar). Viss funktionalitet får modifieras men så mycket som möjligt skall finnas kvar. Även tillåtet att lägga till klasser. Tips: java.util.Collections. 2p

```
package _4_1;
import java.util.Date;
import java.util.List;
public class Holidays {
    List<Date> dates;
    public Holidays() {
    }
    public Holidays(List<Date> dates) {
        this.dates = dates;
    }
    public void add(Date d) {
```

```
        dates.add(d);
    }
    public Date getDate(Date d) {
        if (dates.contains(d)) {
            return d;
        } else {
            return null;
        }
    }
    public List<Date> getDates() {
        return dates;
    }
}
```

5 En iterator

1. Implementera en iterator för klassen nedan. Man kan implementera en iterator på två principiellt olika sätt; failfast eller failsafe. Välj en av dessa för din implementation. Eventuellt måste du själv göra vissa designval. 2p

```
package _5_1;

import java.util.Arrays;
import java.util.Iterator;

public class DataStructure implements Iterable<Elem> {
    // Bad but used for now
    private Elem[] elementData;
    // Actual number of elements
    private int size = 0;
    // Max number of elements
    private int capacity = 4;

    public DataStructure() {
        elementData = new Elem[capacity];
    }
    public boolean add(Elem e) {
        if (size == capacity - 1) {
            capacity *= 2;
            elementData = Arrays.copyOf(elementData, capacity);
        }
        elementData[size++] = e;
        return true;
```

```
    }
    public void clear() {
        for (int i = 0; i < size; i++) {
            elementData[i] = null; // Let gc do its work
        }
        capacity = 4;
        elementData = new Elem[capacity];
        size = 0;
    }
    public int size() {
        return size;
    }
    @Override
    public Iterator<Elem> iterator() {
        // TODO Auto-generated method stub
        return null;
    }

}

package _5_1;

public final class Elem {

    private final int value;

    public Elem(int value){
        this.value = value;
    }

    public int getValue(){
        return value;
    }
}
```