Programmering Fortsättning

Övning 5, Implementationsarv

Joachim von Hacht

1 Överskuggning och döljning[1]

1. Vad kommer att skrivas ut i Main? Förklara! 1p

```
package _1_1;
public class A {
    int i = 1;
    int f() { return i; }
    static char g() { return 'A'; }
}

package _1_1;
public class B extends A {
    int i = 2;
    int f() { return -i; }
    static char g() { return 'B'; }
}

package _1_1;
public class Main511 {
    public static void main(String args[]) {
        B b = new B();
        System.out.println(b.i);
        System.out.println(b.f());
        System.out.println(b.g());
        System.out.println(B.g());

        A a = b;
        System.out.println(a.i);
```

[1] Eng. Overriding and Hiding

```
        System.out.println(a.f());
        System.out.println(a.g());
        System.out.println(A.g());
    }
}
```

2 Arv och information hiding

1. Antag att du inte har tillgång till koden nedan. Kan man trots detta skapa objekt som kan förändra tillståndet (d.v.s. man ersätter de eventuellt icke-muterbara objekten med muterbara varianter) 1p

```
package _2_1;
import java.util.ArrayList;
import java.util.List;
public class Cart {
    protected final List<SomeType> items;
    public Cart(List<SomeType> items) {
        this.items = new ArrayList<SomeType>();
        for(SomeType i : items){
            // copyOf makes a deep copy
            this.items.add(i.copyOf());
        }
    }
    public SomeType getItem(int index) {
        return items.get(index).copyOf();
    }
}
```

3 FBC

1. (Fragile baseclass problem) Vi har skapat en subklass MonitorableStack med MyStack som basklass;

```
package _3_1;
import java.util.ArrayList;
/**
 * Simple stack.
 * @inv this.pop(this.push(s)) == s && this.stackPointer >= 0 &&
 *      this.size() == this.stackPointer
 */
class MyStack extends ArrayList<Object> {
    private static final long serialVersionUID = 1L;
```

```
    private int stackPointer = 0;
    public void push(Object article) {
        add(stackPointer, article);
        stackPointer++;
    }
    public Object pop() {
        stackPointer--;
        return remove(stackPointer);
    }
    public void pushMany(Object[] o) {
        for (int i = 0; i < o.length; ++i)
            push(o[i]);
    }
}
```

```
package _3_1;
/**
 *  A stack that stores the maximum size over its entire lifetime
 *  @inv highWaterMark >= currentSize (&& MyStack invariants)
 */
class MonitorableStack extends MyStack {
    private static final long serialVersionUID = 1L;
    private int highWaterMark = 0;
    private int currentSize;
    public void push(Object article) {
        currentSize++;
        if (currentSize > highWaterMark) {
            highWaterMark = currentSize;
        }
        super.push(article);
    }
    public Object pop() {
        currentSize--;
        return super.pop();
    }
    public int maximumSizeSoFar() {
        return highWaterMark;
    }
}
```

Vi vill kunna göra följande t.ex.;

```
package _3_1;
```

```
public class Main531 {
    public static void main(String[] args) {
        Object[] strs = new Object[] {"a", "b", "c", "d", "e"};
        MonitorableStack m = new MonitorableStack();
        m.pushMany(strs);
        System.out.println(m.maximumSizeSoFar());// 5 ok
        m.push("f");
        System.out.println(m.maximumSizeSoFar()); // 6 ok
    }
}
```

Visa att man kan knäcka MonitorableStack genom att byta representation och ändra i metoden pushMany i basklassen. 2p

4 Arv kontra delegering

1. Betrakta klasserna nedan. Kommer programmet att fungera (titta eventuellt i Javadoc:en för ArrayList)?[2] 1p

```
package _4_1;
import java.util.ArrayList;
/**
 * Simple stack.
 * @inv this.pop(this.push(s)) == s && this.stackPointer >= 0 &&
 *      this.size() == this.stackPointer
 */
class StackExtend extends ArrayList<Object> {
    private static final long serialVersionUID = 1L;
    private int stackPointer = 0;

    public void push(Object article) {
        add(stackPointer, article);
        stackPointer++;
    }
    public Object pop() {
        stackPointer--;
        return remove(stackPointer);
    }
    public void pushMany(Object[] o) {
        for (int i = 0; i < o.length; ++i)
            push(o[i]);
    }
```

[2]Normalt skall man aldrig ärva en behållare, så detta är enbart en illustration.

```
    public int size(){
        return stackPointer;
    }
}

package _4_1;
public class Main541 {
    public static void main(String[] args) {
        StackExtend s1 = new StackExtend();
        s1.push("1");
        s1.push("2");
        s1.clear();
        s1.push("3");
        s1.push("4");
        System.out.println(s1.size());
    }
}
```

2. Skapa en Stack-klass som kan användas på samma sätt som StackExtend ovan men som använder delegering istället för arv (använder ArrayList). Om det uppstod några problem i uppgift 1 se till att den nya stacken löser dessa. 1p

5 Likhet

1. Visa att klasserna ColorPoint och ColorPoint2 båda har problem med equals-metoden. Ett par kodrader räcker (TIPS: Symmetri, transitivitet). 2p

```
package _5_1;
public class Point {
    private final int x;
    private final int y;
    public Point(int x, int y) {this.x = x;this.y = y;}
    public boolean equals(Object o) {
        if (!(o instanceof Point))
            return false;
        Point p = (Point) o;
        return p.x == x && p.y == y;
    }
}

package _5_1;
import java.awt.Color;
public class ColorPoint extends Point {
```

```
   private Color color; //Significant field
   public ColorPoint(int x, int y, Color color) {
      super(x, y);
      this.color = color;
   }
   public Color getColor() {
      return color;
   }
   @Override
   public boolean equals(Object o) {
      if (!(o instanceof ColorPoint)) {
         return false;
      }
      ColorPoint cp = (ColorPoint) o;
      return super.equals(o) && cp.color == color;
   }
}
```

```
package _5_1;
import java.awt.Color;
public class ColorPoint2 extends Point {
    private Color color; // new field
    public ColorPoint2(int x, int y, Color color) {
        super(x, y);
        this.color = color;
    }
    public Color getColor() {
        return color;
    }
    public boolean equals(Object o) {
        if (!(o instanceof Point)) {
            return false;
        }
        if (!(o instanceof ColorPoint2)) {
            return o.equals(this);
        }
        ColorPoint2 cp = (ColorPoint2) o;
        return super.equals(o) && cp.color == color;
    }
}
```

2. Skapa en klass ColorPointDelegate motsvarande ColorPoint ovan m.h.a. delegering i stället för arv. Löser det alla (något) problem med equals? Någon annan lösning? 2p