Programmering Fortsättning 2010

Övning 6

Joachim von Hacht

1 Likhet

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

```
package equals;
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 equals;
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 equals;
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;
    }
}
```

1p

2. Skapa en klass ColorPointDelegate motsvarande ColorPoint ovan m.h.a. delegering. Lös problemen med equals. 1p

2 Kopiering

1. Klassen DoubleBoxClonable är en variant av klassen DoubleBox från övning 1 (klassen Box är identisk med den från övning 1 och visas inte här). Implementera clone och en kopieringskontruktor för DoubleBoxCloneable och subklassen Extend-DoubleBoxCloneabl så att allt i main fungerar.

```
package clone;
public class DoubleBoxCloneable implements Cloneable {
```

```
   private Box innerBox = new Box();
   public DoubleBoxCloneable() {
   }
   public DoubleBoxCloneable(DoubleBoxCloneable orig) {
      // TODO
   }
   public DoubleBoxCloneable clone() {
      // TODO
      return null;
   }
   public void setValue(int v) {
      innerBox.setValue(v);
   }
   public int getValue() {
      return innerBox.getValue();
   }
}

package clone;
public class ExtendDoubleBoxCloneable extends DoubleBoxCloneable {
   public Box extendedInnerBox = new Box();
   public ExtendDoubleBoxCloneable clone() {
      // TODO
      return null;
   }
   public ExtendDoubleBoxCloneable(ExtendDoubleBoxCloneable orig) {
      // TODO
   }
   public ExtendDoubleBoxCloneable() {
   }
   public void setExtendedValue(int v) {
      extendedInnerBox.setValue(v);
   }
   public int getExtendedValue() {
      return extendedInnerBox.getValue();
   }
}

package clone;
public class Main {
   public static void main(String[] args) {
      DoubleBoxCloneable d1 = new DoubleBoxCloneable();
      d1.setValue(111);
      DoubleBoxCloneable d2 = (DoubleBoxCloneable) d1.clone();
```

```
        d2.setValue(222);
        System.out.println("d1 is " + d1.getValue());// Should be the same!
        System.out.println("d2 is " + d2.getValue());
        ExtendDoubleBoxCloneable e1 = new ExtendDoubleBoxCloneable();
        e1.setValue(333);
        e1.setExtendedValue(333000);
        ExtendDoubleBoxCloneable e2 = e1.clone();
        e2.setValue(444);
        e2.setExtendedValue(444000);
        // Should be the same
        System.out
              .println("e1 is " + e1.getValue() + ":" + e1.getExtendedValue());
        System.out
              .println("e2 is " + e2.getValue() + ":" + e2.getExtendedValue());
        // Using copyctor
        DoubleBoxCloneable d3 = new DoubleBoxCloneable(d1);
        d3.setValue(555);
        System.out.println("d1 is " + d1.getValue());// Should be the same!
        System.out.println("d3 is " + d3.getValue());

        // Using copyctor and subclass
        ExtendDoubleBoxCloneable e3 = new ExtendDoubleBoxCloneable(e1);
        e3.setValue(666);
        e3.setExtendedValue(666000);
        // Should NOT be the same
        System.out
              .println("e1 is " + e1.getValue() + ":" + e1.getExtendedValue());
        System.out
              .println("e3 is " + e3.getValue() + ":" + e3.getExtendedValue());
    }
}
```

2p

3 Generiska typer

1. For each line in the main program determine whether it

   a) fails to compile,

   b) compiles with a warning,

   c) generates an error at run time,

   d) compiles and runs without problems.

   In case of a warning or error, explain what is wrong.

```
package generic.types;
public class Main {
    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());
    }
}
```

```
package generic.types;
public class FoodBag<T> {
    private T food;
    public void setFood(T t) {food = t;}
    public T getFood() {return food;}
}
```

```
package generic.types;
public class Sandwich {}
```

```
package generic.types;
public class HerringSandwich extends Sandwich {}
```

```
package generic.types;
public class CheeseSandwich extends Sandwich {}
```

2p

2. The following code will fail. Desribe and explain the error, fix the code using wildcards, and explain your fix. Hint: Bounded wildcards.

```
package generic.types2;
import java.util.Vector;
public class Main {
    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 generic.types2;
    public abstract class Window {
        public abstract void draw();
    }

    package generic.types2;
    public class Frame extends Window {
        @Override
        public void draw() {
            System.out.println("Frame drawn.");
        }
    }

    package generic.types2;
    public class Dialog extends Window {
        @Override
        public void draw() {
            System.out.println("Dialog drawn.");
        }
    }
```

2p

4 Jämförelse[1]

1. Antag att vi vill sortera klassen Customer nedan. Det kan tänkas att man inte kan förutspå alla totala ordningar som kan vara av intresse vid en viss tidpunkt. Ett

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

vanligt fall är då vi i en applikation visar en tabell och man genom att klicka på kolumnernas rubriker bestämmer om posterna ska sorteras i stigande eller fallande ordning med avseende pa värdet i kolumnen. Givet att vi har n kolumner sa kan dessa ordnas pa $n!$ olika sätt. Om vi dessutom lägger till egenskapen att vi ska kunna ha både stigande och fallande ordning för varje fält sa får vi $2^n n!$ möjliga ordningar. Även för små n blir detta snabbt ohanterligt. Vi skulle t.ex. behöva 384 olika komparatorer för att få alla totala ordningar av 4 instansvariabler (16x24 = 384).

Ett flexiblare sätt är att skapa komparatorer för varje instansvariabel och (dynamiskt) seriekoppla dessa efter behov (då behöver vi bara 4 st). Använd denna idé för att implementera en flexibel sorteringslösning för Customer.

TIPS Använd gränssnittet Comparator kombinerat med Decorator (se Web) och Template mönstren.

```
package comparator.template;
public class Customer {
   private final String pNumb;
   private final String firstName;
   private final String lastName;
   public Customer(String pnumb, String firstName, String lastName) {
      this.pNumb = pnumb;
      this.firstName = firstName;
      this.lastName = lastName;
   }
   public String getPNumb() {
      return pNumb;
   }
   public String getFirstName() {
      return firstName;
   }
   public String getLastName() {
      return lastName;
   }

   @Override
   public int hashCode() {
      final int prime = 31;
      int result = 1;
      result = prime * result
            + ((firstName == null) ? 0 : firstName.hashCode());
      result = prime * result + ((lastName == null) ? 0 : lastName.hashCode());
      return result;
   }
```

```
   // Note different behavior for null
   // Equals return false
   @Override
   public boolean equals(Object obj) {
      if (this == obj)
         return true;
      if (obj == null)
         return false;
      if (getClass() != obj.getClass())
         return false;
      Customer other = (Customer) obj;
      if (firstName == null) {
         if (other.firstName != null)
            return false;
      } else if (!firstName.equals(other.firstName))
         return false;
      if (lastName == null) {
         if (other.lastName != null)
            return false;
      } else if (!lastName.equals(other.lastName))
         return false;
      return true;
   }

   @Override
   public String toString(){
      return pNumb + " " + firstName + " " + lastName;
   }
}
```

2p