Übung 7

Entwurfsmuster „Beobachter“ und „Dekorierer“

1. Anhand einer graphischen Anwendung soll der Sinn und Zweck von eventgesteuerten Methoden sichtbar werden. Schreiben Sie ein Java-Programm, das einerseits mit dem Entwurfsmuster „Beobachter“ (`Observable`, `Observer`) arbeitet, andererseits die in Java vorgegebenen „Beobachter-Interfaces“ `MouseListener` und `WindowListener` verwendet. Das Programm soll:

   (a) Ein Fenster mit einem Button darin öffnen,

   (b) beim Klick auf den Button (und nur dann) einen Text ausgeben, graphisch oder auf der Textkonsole,

   (c) beim Schließen des Fensters das Programm beenden,

   (d) mindestens eine weitere Klasse (bzw. ein weiteres Objekt) **darüber informieren**, dass der Benutzer den Button angeklickt hat.

   Siehe ***aufgabe1/HelloButton.java***, ***aufgabe1/Beobachter.java*** *und* ***aufgabe1/Main.java****:*

```
import java.awt.*; // Für Frame & Co.
import java.awt.event.*;
import java.util.Observable; // Observable-KLASSE


public class HelloButton extends Observable
                         implements MouseListener, WindowListener {

 public static void main(String[] args){
   new HelloButton();
 }

 public HelloButton() {
  Frame frame = new Frame("Mein Frame"); // Ein Fenster
  frame.setSize(400,200); // Größe ändern (breite, höhe)
  // "Graphics"-Context zum Zeichnen holen
  Graphics g = frame.getGraphics();
  // Ein Knopf zum Klicken
  Button button = new Button("Hier klicken!");
  // Knopf Zeichensatz und Stil auswählen
  button.setFont(new Font("SansSerif", Font.BOLD, 48));
  // Knopf mit Frame verbinden.
```

```
  frame.add(button);
  // Ein "MouseListener" aktiviert eine Nachricht, wenn
  // ein (Maus-spezifisches) Ereignis eintritt.
  button.addMouseListener(this);
  frame.addWindowListener(this);
  // Noch ist kein Fenster zu sehen (obwohl man schon
  // reinzeichnen kann!), aber gleich:
  frame.setVisible(true);
  System.out.println("HelloButton-Fenster " + this.toString() +
                     " wurde erzeugt, warte auf Events...");
 }

 public void mouseEntered(MouseEvent e) {}
 public void mouseExited(MouseEvent e) {}
 public void mouseReleased(MouseEvent e) {}
 public void mousePressed(MouseEvent e) {}
 public void mouseClicked(MouseEvent e) {
  setChanged();      // "Geändert"-Satus setzen
  notifyObservers(); // Und die Beobachter informieren.
 } // Ende mouseClicked
 public void windowDeactivated(WindowEvent e) {}
 public void windowActivated(WindowEvent e) {}
 public void windowDeiconified(WindowEvent e) {}
 public void windowIconified(WindowEvent e) {}
 public void windowClosing(WindowEvent e) { System.exit(0);}
 public void windowClosed(WindowEvent e) {}
 public void windowOpened(WindowEvent e) {}
}

import java.util.Observable;
import java.util.Observer;

public class Beobachter implements Observer {

 public Beobachter() {
   System.out.println("Beobachter-Objekt " + this.toString()
                      + " wurde erzeugt.");
 }

 public void update(Observable observable, Object arg) {
  System.out.println(this.toString() + ".update() wurde von "
                     + observable.toString() + " aufgerufen.");
 }

}

import java.util.Observable;
```

```
import java.util.Observer;

public class Main {
 public static void main(String[] args) {
  System.out.println("Beginn von main().");

  Observable fenster = new HelloButton(); // Subjekt
  Observer beobachter1 = new Beobachter(); // Beobachter 1
  Observer beobachter2 = new Beobachter(); // Beobachter 2

  // Man beachte die Reihenfolge.
  // notifyObservers() ruft keineswegs update()
  // in der gleichen Reihenfolge auf, in der
  // die Observer abonniert wurden.

  fenster.addObserver(beobachter1);
  fenster.addObserver(beobachter2);

  Eingabe.readString("Bitte Eingabetaste drücken, um Beobachter " + b

  fenster.deleteObserver(beobachter1);

  System.out.println("Ende von main().");
 }
}
```

2. Verwenden Sie das Entwurfsmuster „Dekorierer“, um für zwei von einer abstrakten Basisklasse `Kuchen` abgeleitete Klassen (z.B. `Kaesekuchen` und `Sachertorte`) kombinierbare Dekorationen wie `Schokostreusel`, `Sahne`, `Marzipanrosen` oder `Rosinen` zu erzeugen, und schreiben Sie ein Testprogramm, das einen Käsekuchen mit Rosinen und doppelt Schokostreuseln erzeugt.

3. Erweitern Sie das Programm aus der vorigen Aufgabe so, dass auch der Preis berechnet wird, der sich aus dem „undekorierten“ Kuchen und den verwendeten Dekorierern ergibt.

   Siehe ***aufgabe2-3/Kuchen.java***, ***aufgabe2-3/Kaesekuchen.java***, ***aufgabe2-3/Sachertorte.java***, ***aufgabe2-3/Option.java***, ***aufgabe2-3/Marzipanrosen.java***, ***aufgabe2-3/Rosinen.java***, ***aufgabe2-3/Sahne.java***, ***aufgabe2-3/Schokostreusel.java*** *und* ***aufgabe2-3/Main.java****:*

```
public abstract class Kuchen {
 String beschreibung = "Kuchen";

 public String getBeschreibung() {
  return beschreibung;
 }
```

```
 public abstract float preis();
}
```

```
public class Kaesekuchen extends Kuchen {
 public Kaesekuchen() {
  beschreibung = "Käsekuchen";
 }

 public float preis() {
  return 2.0F;
 }
}
```

```
public class Sachertorte extends Kuchen {
 public Sachertorte() {
  beschreibung = "Sachertorte";
 }

 public float preis() {
  return 4.0F;
 }
}
```

```
public abstract class Option extends Kuchen {
 public abstract String getBeschreibung();
}
```

```
public class Marzipanrosen extends Option {
 Kuchen kuchen;

 public Marzipanrosen(Kuchen kuchen) {
  this.kuchen = kuchen;
 }

 public String getBeschreibung() {
  return kuchen.getBeschreibung() + ", Marzipanrosen";
 }

 public float preis() {
  return 0.35F + kuchen.preis();
 }
}
```

```
public class Rosinen extends Option {
 Kuchen kuchen;

 public Rosinen(Kuchen kuchen) {
  this.kuchen = kuchen;
 }

 public String getBeschreibung() {
  return kuchen.getBeschreibung() + ", Rosinen";
 }

 public float preis() {
  return 0.15F + kuchen.preis();
 }
}
```

```
public class Sahne extends Option {
 Kuchen kuchen;

 public Sahne(Kuchen kuchen) {
  this.kuchen = kuchen;
 }

 public String getBeschreibung() {
  return kuchen.getBeschreibung() + ", Sahne";
 }

 public float preis() {
  return 0.25F + kuchen.preis();
 }
}

public class Schokostreusel extends Option {
 Kuchen kuchen;

 public Schokostreusel(Kuchen kuchen) {
  this.kuchen = kuchen;
 }

 public String getBeschreibung() {
  return kuchen.getBeschreibung() + ", Schokostreusel";
 }

 public float preis() {
  return 0.20F + kuchen.preis();
```

```
 }
}
```

```
import java.text.DecimalFormat; // Für Klasse DecimalFormat.

public class Main {
 public static void main(String[] args) {
  Kuchen kuchen1 = new Kaesekuchen();
  // Kaesekuchen mit Rosinen dekorieren
  kuchen1 = new Rosinen(kuchen1);
  // mit Schokostreuseln dekorieren
  kuchen1 = new Schokostreusel(kuchen1);
  // nochmal mit Schokostreuseln dekorieren
  kuchen1 = new Schokostreusel(kuchen1);

  // Nur zwei Nachkommastellen ausgeben, Formatdefinition
  DecimalFormat zweistellen = new DecimalFormat("0.00");

  System.out.println(kuchen1.getBeschreibung() + " kostet " +
              zweistellen.format(kuchen1.preis()) + " Euro.");
 }
}
```