Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Ansvarig lärare: Magnus Myréen

DAT050 Objektorienterad programmering

Tentamen, måndag, 2022-01-03, 08:30-12:30

Vitsordsgränser: 3=24p, 4=36p, 5=48p, max 60p.

Handledaren Jessica Barai besöker tentan och svarar på eventuella frågor om uppgifterna.
Kom ihåg att inte fastna på en uppgift. Bestäm i förväg din egen tidsgräns per uppgift. ***Lycka till!***

Uppgift 1: [10 poäng totalt] *grunder i objektorientering*

A. Förklarar kort vad som avses med "black box tänkande" i samband men objektorienterad programmering. [2 poäng]

B. Förklara varför man kan säga att vissa välskrivna objektorienterade program är modulära. [3 poäng]

C. Följande klass tar emot `int` värden med anrop till `addInt` och bör alltid returnera genomsnittet av alla de hittills givna värdena med funktionen `getAverage`. Exempel: om man lagt in värdena 4, 6, 8, 10 med `addInt`, då bör `getAverage` returnera 7.

```
public class Average {

    private int sum = 0;
    private int count = 0;

    public void addInt(int n) {
        sum = sum + n;
        count = count + 1;
    }

    public double getAverage() {
        return (double)sum / (double)count;
    }

}
```

Koden nedan skriver ut `Error` och visar att det finns en svaghet/bug i koden ovan.

```
Average a = new Average();
a.addInt(800000000);
a.addInt(600000000);
a.addInt(500000000);
a.addInt(300000000);
if (a.getAverage() < 0.0) {
    System.out.println("Error");
}
```

Din uppgift är att skriva om `Average` klassen så att denna bug tas bort. [5 poäng]

Uppgift 2: [10 poäng totalt] *gränssnitt*

Ett stort projekt använder följande interface för att sätta på och stänga av grejor.

```
/** An interface for turning on and off a device */
public interface Switch {

    /** Command that turns a device on */
    public void turnOn();

    /** Command that turns a device off */
    public void turnOff();

    /** Returns the current state of the device */
    public boolean isOn();

}
```

Nu anar felsökare att koden i det stora projektet anropar på `turnOn` och `turnOff` i onödan, dvs `turnOn` anropas på objektet som redan är i `on` läge, och lika så för `turnOff` i ett `off` läge. Man bör inte anropa på `turnOn` i `on` läget och inte på `turnOff` i `off` läget.

Din uppgift är att skriva en klass `SafeSwitch` som kan användas som en wrapper runt existerande instanser av `Switch` gränssnittet. Din klass `SafeSwitch` bör implementera `Switch` gränssnittet och ha följande konstruktör.

```
public SafeSwitch(Switch s) {
```

Idéen är att instanser av `SafeSwitch` endast skickar vidare `turnOn` och `turnOff` signaler till `Switch` instansen `s` när `s` är i rätt läge för att ta emot signalen.

Uppgift 3: [15 poäng totalt] *listor*

Pascals triangel ser ut som nedan. ***Obs:*** Rad 2 kan räknas från rad 1; rad 3 från rad 2, osv.

```
rad 1:                1
rad 2:              1   1
rad 3:            1   2   1
rad 4:          1   3   3   1
rad 5:        1   4   6   4   1
rad 6:      1   5   10  10   5   1
rad 7:    1   6   15  20  15   6   1
 ...
```

Din uppgift är att skriva en metod `pascalsTriangleLine` i Java som returnerar rad nummer n från triangeln som en lista, för vilket positivt `n` som helst. Koden bör kasta ett exception ifall det givna nummer n inte är positivt.

```
public List<Integer> pascalsTriangleLine(int n) {
```

Exempel: För input 4 bör metoden returnera en lista som består av 1, 3, 3, 1.

Obs: Koden behöver inte vara minnessnål, men den måste vara lättläst och korrekt.

Obs: Man behöver inte vara varsam med `int` typens begränsningar i den här uppgiften.

Uppgift 4: [10 poäng totalt] *string, equals, referensvärden*

A. Vad skrivs ut när följande `PrintStuff` program körs? [5 poäng]

```
public class PrintStuff {

    public static void main(String[] args) {
        String a = "Hello";
        String b = "hello";
        String c = "HELLO";
        a.toUpperCase();
        String d = a.toLowerCase();
        c = a;
        System.out.println(a == b);
        System.out.println(a == c);
        System.out.println(b == d);
        System.out.println(a.equals(b));
        System.out.println(b.equals(d));
    }

}
```

B. Implementera en standard `equals` metod för följande klass. [5 poäng]

```
public class StringPair {

    private String a, b;

    public StringPair(String a, String b) {
        this.a = a;
        this.b = b;
    }

    public String getFirst() { return a; }
    public String getSecond() { return b; }

}
```

Uppgift 5: [15 poäng totalt] *model-view-controller, grafik*

Din uppgift är att implementera en klass `SnakeView` som ska ärva standard klassen `JPanel` och fungera som en View klass i ett Snake spel som är implementerat enligt Model-View-Controller design mönstret. ***Obs:*** implementera endast `SnakeView` klassen.

- Din klass får en instans av `SnakeModel` som argument till konstruktören. Denna `SnakeModel` representerar spelytan som ett 20x20-stort rutfält av celler. Klassen `SnakeModel` har en metod `getCellAt(int i, int j)` som returnerar en `Cell`.

  ```
  public enum Cell { SNAKE, FOOD, EMPTY }
  ```

- Din `SnakeView` klass bör rita bakgrunden som svart. Varje `SnakeModel` cell bör vara 30 pixel hög och bred. En cell som `SnakeModel` instansen säger innehåller `SNAKE` bör fyllas med en vit rektangel. En cell med `FOOD` bör innehålla en fylld röd cirkel. En tom cell, dvs `EMPTY`, ska förbli svart. Anta att `SnakeView` är tillräckligt stor för att allt ska synas.

Obs: Javas `import` satser kan man lämna bort i lösningen.

Utdrag ur Javas API. **Obs:** Man behöver inte använda alla dessa klasser. Man får också använda annat från Javas API.

Class Color extends Object
- Color(int r, int g, int b)
  Creates an opaque sRGB color with the specified red, green, and blue values in the range (0 - 255).
- BLACK
  The color black.
- RED
  The color red.
- WHITE
  The color white.

Class Component extends Object
- void repaint()
  Repaints this component.

Class Graphics extends Object
- void fillOval(int x, int y, int width, int height)
  Fills an oval bounded by the specified rectangle with the current color.
- void fillRect(int x, int y, int width, int height)
  Fills the specified rectangle.
- void setColor(Color c)
  Sets this graphics context's current color to the specified color.

Class HashMap<K,V> extends AbstractMap<K,V> extends Object implements Map<K,V>
- HashMap()
  Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).
- V get(Object key)
  Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
- V put(K key, V value)
  Associates the specified value with the specified key in this map.

Interface Iterator<E>
- boolean hasNext()
  Returns true if the iteration has more elements.
- E next()
  Returns the next element in the iteration.

Class JButton extends AbstractButton extends JComponent extends Object
- JButton(String text)
  Creates a button with text.
- void setBackground(Color bg)
  Sets the background color of this component.

Class JComponent extends Container extends Component extends Object
- void paintComponent(Graphics g)
  Calls the UI delegate's paint method, if the UI delegate is non-null.

Class JPanel extends JComponent extends Container extends Component extends Object
- JPanel()
  Creates a new JPanel with a double buffer and a flow layout.

Class LinkedList<E> extends AbstractCollection<E> extends Object implements List<E>
- LinkedList()
  Constructs an empty list.

Interface List<E>
- boolean add(E e)
  Appends the specified element to the end of this list.
- E get(int index)
  Returns the element at the specified position in this list.
- Iterator<E> iterator()
  Returns an iterator over the elements in this list in proper sequence.
- int size()
  Returns the number of elements in this list.

Class Object
- boolean equals(Object obj)
  Indicates whether some other object is "equal to" this one.