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

DAT050 Objektorienterad programmering

Tentamen, tisdag, 2023-01-03, 08:30-12:30

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

Kom ihåg att inte fastna på en uppgift. Bestäm i förväg din egen tidsgräns per uppgift. ***Lycka till!***

Uppgift 1: [8 poäng totalt] *grunder i objektorientering och Java*

A. Förklara vad klassvariabler och instansvariabler är. Hur skiljer de sig från varandra? Använd exempel i din förklaring. [4 poäng]

B. Förklara vad JPanel och JFrame är. Hur skiljer de sig från varandra? Använd exempel i din förklaring. [4 poäng]

Uppgift 2: [15 poäng totalt] *att skriva klass, användning av standard klasser*

Denna uppgift handlar om att implementera en klass `PupilDatabase` som fungerar som en databas för information om elever i en skola.

A. Skriv all kod som behövs för en klass som har följande funktionalitet. [10 poäng]

- När man skapar en ny instans av klassen bör den representera en tom databas.
- Man ska kunna lägga till information (elev id, namn, klass) med en metod med följande signatur..

```
public void setPupilInfo(int pupilID, String name, String className)
```

- Klassen bör ha metoder för att hämta ut information, som nedan. Dessa metoder ska returnera null ifall det inte finns information lagrat för det givna `pupilID`.

```
public String getPupilName(int pupilID)

public String getPupilClassName(int pupilID)
```

Obs: Anta att varje elev har ett unikt elev id (`pupilID`).

Tips: Det är lättast att lösa uppgiften med hjälp av en hjälpklass vars instanser innehåller all information för *en elev*.

B. Lägg till en metod som returnerar namnen på alla elever i en given klass. [5 poäng]

```
public List<String> pupilsInClass(String className)
```

Uppgift 3: [10 poäng] *interface, iterators, listor, exceptions*

Denna uppgift handlar om iterators. Java har ett standard `Iterator` interface med metoder `hasNext` och `next`, som man kan använda för att stiga (framåt) igenom en samling element.

Denna uppgift handlar om att implementera en ny iterator variant `NextPrev` med vilken man kan gå *framåt och bakåt* i en samling element. Det här är definitionen av `NextPrev`:

```
public interface NextPrev<A> {

    /** Returns true if there is an element at this position */
    public boolean hasCurr();

    /** Returns element at current position, if hasCurr() returns true.
        Throws an exception if hasCurr() returns false */
    public A curr();

    /** Moves this iterator one step forward */
    public void next();

    /** Moves this iterator one step backward */
    public void prev();

}
```

Skriv kod för en klass `NextPrevOfList` som är en `NextPrev` iterator för listan som man ger till `new NextPrevOfList`. Kommentarerna nedan förklarar hur koden nedan bör fungera med din implementation av `NextPrevOfList` klassen.

```
List<String> l = new LinkedList<String>();
l.add("A");                                    // l är ["A"]
l.add("B");                                    // l är ["A","B"]
l.add("C");                                    // l är ["A","B","C"]
l.add("D");                                    // l är ["A","B","C","D"]
NextPrev<String> np = new NextPrevOfList(l);
System.out.println(np.curr());                 // bör skriva ut A
System.out.println(np.curr());                 // bör skriva ut A
np.next();
System.out.println(np.curr());                 // bör skriva ut B
np.next();
System.out.println(np.curr());                 // bör skriva ut C
while (np.hasCurr()) {
    System.out.println(np.curr());
    np.prev();
}                           // loopen bör skriva ut C, sedan B, sedan A
```

Obs: Du kan anta att ingen ändrar i listan som `NextPrevOfList` får som indata.

Uppgift 4: [12 poäng totalt] *specifikation och testning*

Denna uppgift handlar om att hitta *svagheter* i testkod. Testkoden är följande.

```
Random r = new Random();
int[] a = new int[r.nextInt(2000)+1];
for (int i=0; i<a.length; i++) {
    a[i] = r.nextInt();
}
try {
    sort(a);                // det är denna sort metod som testas
} catch (Exception e) {
    assertTrue(false);
}
for (int i=0; i<a.length-1; i++) {
    assertTrue(a[i] <= a[i+1]);
}
```

Koden ovan testar metoden `sort` som har följande signatur och specifikation.

```
/** Rearranges elements of array a so that a[i] <= a[i+1], for every i.
    This method does not throw any exception. */
public static void sort(int[] a)
```

Här är uppgifterna:

A. Förklara varför testkoden kan missa fel i implementationen av `sort`. [4 poäng]
Tips: Nyckeln till gåtan är ordet "rearranges" i specifikationen, dvs "arrangerar om".

B. Skriv en implementation av `sort` som testkoden alltid accepterar, men som *inte* följer specifikationen för `sort`. [4 poäng]

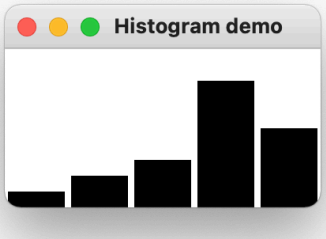
C. Beskriv *med ord* hur testkoden kan förbättras så att den verkligen testar det som står i specifikationen för `sort`. [4 poäng]

Uppgift 5: [15 poäng] *grafiskt användargränssnitt, ritning*

Din uppgift är att implementera en klass `Histrogram` så att varje `Histrogram` *är en* `JPanel` och så att den ritar ut ett histogram som visualisera datan `new Histrogram` är given.

Exempel: Med din implementation av `Histrogram` bör koden nedan få fönstret `f` att se ungefär ut som på bilden till höger.

```
double[] data = { 0.1, 0.2, 0.3, 0.8, 0.5 };
JFrame f = new JFrame("Histogram demo");
JPanel p = new Histrogram(data);
p.setPreferredSize(new Dimension(200,100));
f.add(p);
f.pack();
f.setVisible(true);
```



Obs: Se till att `Histrogram` alltid ritar ut datan så som den fick den när den skapades.

Obs: Din lösning bör fungera också för annan indata än den som är given i exemplet ovan, men du kan anta att double arrayn aldrig är tom och att den aldrig är null.

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

Class Color
  static Color WHITE
  static Color BLACK
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. **Class**
IllegalArgumentException extends Exception
  IllegalArgumentException()
     Creates a new IllegalArgumentException.
Interface Iterator<E>
  boolean hasNext()
     Returns true if the iteration has more elements.
  E next()
     Returns the next element in the iteration.
Class JComponent extends Container extends Component extends Object
  int getHeight()
     Returns the current height of this component.
  int getWidth()
     Returns the current width of this component.
  void paintComponent(Graphics g)
     Calls the UI delegate's paint method, if the UI delegate is non-null.
Class JFrame extends Frame extends Container extends Component extends Object
  JFrame(String title)
     Creates a new, initially invisible Frame with the specified title.
  void setLayout(LayoutManager manager)
     Sets the LayoutManager.
Class JPanel extends JComponent extends Container extends Component extends Object
  JPanel()
     Creates a new JPanel with a double buffer and a flow layout.
Interface List<E>
  boolean add(E e)
     Appends the specified element to the end of this list.
  Iterator<E> iterator()
     Returns an iterator over the elements in this list in proper sequence.
  E get(int index)
     Returns the element at the specified position in this list.
  int size()
     Returns the number of elements in this list.
Class LinkedList<E> extends AbstractCollection<E> extends Object implements List<E>
  LinkedList()
     Constructs an empty list.
Class Object
  boolean equals(Object obj)
     Indicates whether some other object is "equal to" this one.