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

DAT050 Objektorienterad programmering

Tentamen, torsdag, 2021-10-28, 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: [10 poäng totalt] *immutable, aliasing problem*

A. Vad är aliasing problem och när kan de förekomma i Java program? [2 poäng]

B. Förklara hur användning av *immutable* är ett sätt att undvika aliasing problem. [3 poäng]

C. Är följande implementation av `MyClass` *immutable*? Förklara ditt svar. [5 poäng]

```
import java.util.LinkedList;

public class MyClass {

    private LinkedList<Integer> list;

    public MyClass(LinkedList<Integer> list) {
        this.list = list;
    }

    public int getLength() {
        return list.size();
    }

    public String toString() {
        return list.toString();
    }

}
```

Uppgift 2: [15 poäng totalt] *att skriva klasser*

Denna uppgift handlar om att skriva klasser som kan implementera en enkel modell av ett Ladok liknande system, dvs ett system som håller koll på olika resultat på olika kurser som studenter har avklarat. I denna uppgift antar vi att varje student kan unikt identifieras med en sträng (String) av deras personnummer (kallas studentID nedan). Likaså, är kurser unikt identifierade genom sin kurskod i String format. Betyg representeras av:

```
public enum Grade {

    FIVE, FOUR, THREE, FAIL

}
```

A. Implementera en klass `StudentData` som representerar all data detta Ladok system har sparat *för en student*. Klassen `StudentData` ska inkludera följande metoder. [6 poäng]

```
/** givet en kurskod, returnerar denna students betyg;
    kastar exception ifall studenten inte har tagit denna kurs */
public Grade getGrade(String courseCode) { ... }

/** sparar ett betyg för denna student på den givna kurskoden;
    denna sparar över det gamla betyget om det redan fanns ett */
public void storeResult(String courseCode, Grade result) { ... }

/** returnerar en mängd (Set) som består av alla kurskoder som
    denna student har registrerade betyg på. */
public Set<String> getCoursesTaken() { ... }
```

B. Implementera en klass `MiniLadok` som representerar detta Ladok liknande system. Din implementation bör använda `StudentData` klassen från del A. Klassen `MiniLadok` bör inkludera följande metoder. [9 poäng]

```
/** givet studentens personnummer som string och kurskod,
    returnerar det senast registrerade betyget. kastar exception
    ifall inget resultat finns för den studenten och kursen. */
public Grade getGrade(String studentID, String courseCode) { ... }

/** returnerar, för en given kurskod, en mapping från studenters
    personnummer till deras betyg för den kurskoden */
public Map<String,Grade> getCourseInfo(String courseCode) { ... }
```

Tips: Det lönar sig att använda `HashMap` i både koden för del A och del B.

Obs: Man får anta att indata aldrig är null till koden ovan.

Uppgift 3: [10 poäng totalt] *interface, listor*

I Javas API finns interfacet `Comparable`. Den har en metod och det står följande om den:

```
public interface Comparable<T> {

  /** Returns: a negative integer, zero, or a positive integer as
      this object is less than, equal to, or greater than the
      specified object o. */
  public int compareTo(T o);

}
```

Anta att det finns en klass `Car` som implementerar `Comparable`, dvs:

```
public class Car implements Comparable<Car> { … }
```

Din uppgift är att skriva en metod med följande signatur, som givet en lista av `Car` objekt returnerar den största i listan enligt `compareTo`. Metoden bör kasta en exception ifall det inte finns någon största. Man får anta att listan inte är null och att den inte innehåller null.

```
public static Car getGreatest(List<Car> list) { … din kod … }
```

Uppgift 4: [15 poäng totalt] *arv, arrays*

A. Vad skrivs ut när följande `Test` program körs? [9 poäng]

```
public class Test1 {
    public String f() { return "1"; }
    public String g() { return this.f(); }
}
public class Test2 extends Test1 {
    public String f() { return "2"; }
}
public class Test3 extends Test2 {
    public String f() { return "x"; }
    public String g() { return "3"; }
}
public class Test {
    public static void main(String[] args) {
        Test2 t2 = new Test2();
        System.out.println(t2.g());
        Test1 t1 = (Test1)t2;
        System.out.println(t1.g());
        t2 = new Test3();
        System.out.println(t2.g());
    }
}
```

B. Vad skrivs ut när följande `Main` program körs? [6 poäng]

```
public class Main {
    public static int test(int[] b) {
        int[] c = b;
        c[3] = 0;
        System.out.println(c[0]);
        return 4;
    }
    public static void main(String[] args) {
        int[] a = { 1, 2, 3, 4, 5 };
        System.out.println(a[3]);
        test(a);
        System.out.println(a[3]);
    }
}
```

Uppgift 5: [10 poäng totalt] *händelsehantering, GUI*

Implementera en klass `DiscoButton` som på alla sätt fungerar som en vanlig `JButton` men med den extra egenskapen att `DiscoButton` byter färg varje 100 millisekunder till en slumpmässig ny färg. Man ska kunna skapa en `DiscoButton` med att köra koden `new DiscoButton(text)`, där `text` är texten (String) som ska stå på knappen.

Tips: Använd en `Timer` för köra koden som byter färgen på knappen varje 100 millisekunder. Se utdraget ur Javas API på sista sidan.

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

Class AbstractButton extends JComponent extends Object
  void addActionListener(ActionListener l)
    Adds an ActionListener to the button.
Interface ActionListener
  void actionPerformed(ActionEvent e)
    Invoked when an action occurs.
Class ActionEvent extends AWTEvent extends EventObject extends Object
  ActionEvent(Object source, int id, String command)
    Constructs an ActionEvent object.
  String getActionCommand()
    Returns the command string associated with this action.
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).
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.
  Set<K> keySet()
    Returns a Set view of the keys contained in this map.
  V put(K key, V value)
    Associates the specified value with the specified key in this map.
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 LinkedList<E> extends AbstractCollection<E> extends Object implements List<E>
  LinkedList()
    Constructs an empty list.
  boolean add(E e)
    Appends the specified element to the end of this list.
Interface List<E>
  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 Random extends Object
  Random()
    Creates a new random number generator.
  int nextInt(int bound)
    Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.
Class Timer extends Object
  Timer(int delay, ActionListener listener)
    Creates a Timer and initializes both the initial delay and between-event delay to delay milliseconds.
  void setActionCommand(String command)
    Sets the string that will be delivered as the action command in ActionEvents fired by this timer.
  void setRepeats(boolean flag)
    If flag is false, instructs the Timer to send only one action event to its listeners.
  void start()
    Starts the Timer, causing it to start sending action events to its listeners.