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

DAT050 Objektorienterad programmering

Modellsvar för 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!***

Modellsvar för Uppgift 1: [10 poäng totalt] *immutable, aliasing problem*

Obs: Svaren behöver inte vara så här långa, men de bör kommunicera samma punkter.

A. Aliasing sker när två referenser pekar på samma objekt. Problem med aliasing uppstår när man glömmer att flera saker pekar på samma objekt och någon del av programmet muterar (dvs ändrar) på objektet som flera delar av programmet pekar på. Detta händer lätt i Java program.

B. Man kan skriva sina klasser så att deras objekt inte går att ändra på. Sådana klasser kallas *immutable*. Sådana objekt skapar nya objekt istället för att mutera de existerande objekten. Med att använda immutable klasser så gör det inget om två referenser pekar på samma objekt för att objekten kan aldrig ändras och då kan inte en del av programmet ändra i misstag på objekt som en annan del av programmet använder.

C. MyClass är **inte** immutable för att dess tillstånd, dvs `list`, kan ändras utifrån pga att objekt av typen `LinkedList` är muterbara. I följande exempel ser vi att ett `MyClass` objekt kan ändras och är därför **inte** immutable.

```
LinkedList<Integer> l = new LinkedList<Integer>();
l.add(5);
MyClass m = new MyClass(l);
System.out.println(m.getLength());   // skriver ut 1
l.add(6);
System.out.println(m.getLength());   // skriver ut 2
```

Frivillig extra förklaring:

Man kan göra `MyClass` till en immutable klass med att ändra konstruktion till följande:

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

Denna version ser till att ingen utanför har en pekar till instansvariabeln `list` som i `MyClass`. Med denna korrigerade version kommer kodexemplet ovan att skriva ut `1` för båda `System.out.println` anropen.

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

A.

```
public class StudentData {

    // mapping från kurskoder till betyg
    private HashMap<String,Grade> grades = new HashMap<>();

    public Grade getGrade(String courseCode) {
        Grade g = grades.get(courseCode);
        if (g == null) {
            throw new IllegalArgumentException("Course not taken");
        }
        return g;
    }

    public void storeResult(String courseCode, Grade result) {
        grades.put(courseCode, result);
    }

    public Set<String> getCoursesTaken() {
        return grades.keySet();
    }

}
```

B.

```
public class MiniLadok {

    // mapping från studentID till StudentData (för den studenten)
    private HashMap<String,StudentData> allData = new HashMap<>();

    public Grade getGrade(String studentID, String courseCode) {
        StudentData sd = allData.get(studentID);
        if (sd == null) {
            throw new IllegalArgumentException("No such student");
        }
        return sd.getGrade(courseCode);
    }

    public Map<String,Grade> getCourseInfo(String courseCode) {
        Map<String,Grade> res = new HashMap<>();
        for (String studentID : allData.keySet()) {
            try {
                StudentData sd = allData.get(studentID);
                Grade g = sd.getGrade(courseCode);
                res.put(studentID,g);
            } catch (Exception e) {
                // do nothing, the student has not taken this course
            }
        }
        return res;
    }

}
```

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

Följande svar räcker:

```
public static Car getGreatest(List<Car> list) {
   if (list.size() == 0) {
       throw new IllegalArgumentException("No such element");
   }
   Iterator<Car> it = list.iterator();
   Car max = it.next();
   while (it.hasNext()) {
       Car temp = it.next();
       if (max.compareTo(temp) != 1) {
           max = temp;
       }
   }
   // extra kod skulle läggas här, se nedan
   return max;
}
```

Det går att läsa uppgiftens text på två olika sätt, specifikt kan man läsa "Metoden bör kasta en exception ifall det inte finns någon största" så att den bör kasta en exception ifall det inte finns ett element som är större än alla andra. Det är OK att förstå texten på det sättet. Detta kan man implementera med att lägga till följande rader kod innan `return`.

```
Iterator<Car> it2 = list.iterator();
int count = 0;
while (it2.hasNext()) {
    Car temp = it2.next();
    if (max.compareTo(temp) == 0) {
        count = count + 1;
    }
}
if (count > 1) {
    throw new IllegalArgumentException("No single max element.");
}
```

Oberoende hur man läser texten bör man kasta exception ifall den givna listan är tom, dvs `list.size() == 0`, som i koden högst uppe. Det finns ju inget största element i en tom lista.

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

(Ingen förklaring behövs för fulla poäng ifall svaren är korrekt.)

A.

```
2
2
3
```

B.

```
4
1
0
```

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

```
public class DiscoButton extends JButton implements ActionListener {

    private Random r = new Random();

    public DiscoButton(String text) {
        super(text);
        Timer t = new Timer(100,this);
        t.setRepeats(true);
        t.start();
    }

    public void actionPerformed(ActionEvent e) {
        Color c = new Color(r.nextInt(256),
                            r.nextInt(256),
                            r.nextInt(256));
        setBackground(c);
    }

}
```