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, onsdag, 2023-08-23, 14:00-18:00

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: [5 poäng totalt] *grunder i objektorientering och Java*

A. I objektorienterad programmering är datan det centrala och funktioner som manipulerar datan hänger med data. Datan och datans funktioner lever i *objekt*. Väldefinierade gränssnitt mellan insidan och utsidan av objekt är viktigt för strukturen av objektorienterade program.

B. Här är kod som kopplar en lyssnare till en knapp. Lyssnaren är ett objekt som har en funktion `actionPerformed` som anropas varje gång knappen trycks.

```
JButton b = new JButton("Hi");
b.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.out.println("Button clicked!");
        }
    });
```

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

A.

```
public class BiddingMarket {

    private HashMap<String,ItemInfo> data;

    public BiddingMarket() {
        data = new HashMap<String,ItemInfo>();
    }

    public void addItem(String itemName, int minPrice) {
        ItemInfo ii = data.get(itemName);
        if (ii != null) {
            throw new IllegalArgumentException("Item already exists");
        }
        data.put(itemName,new ItemInfo(itemName,minPrice));
    }

    private ItemInfo getInfo(String itemName) {
        ItemInfo ii = data.get(itemName);
        if (ii == null) {
            throw new IllegalArgumentException("No such item");
        }
        return ii;
    }

    public void bidForItem(String itemName, int bidPrice) {
        getInfo(itemName).bid(bidPrice);
    }

    public List<Integer> getBidHistory(String itemName) {
        return getInfo(itemName).getBids();
    }

}

public class ItemInfo {

    private String itemName;
    private int minPrice;
    private List<Integer> bids; // latest is last

    public ItemInfo(String itemName, int minPrice) {
        this.itemName = itemName;
        this.minPrice = minPrice;
        bids = new LinkedList<Integer>();
    }
    public List<Integer> getBids() {
        return bids; // might want to clone the list here
    }

    public void bid(int bidPrice) {
        int min = minPrice;
        if (bids.size() > 0) {
            min = bids.get(bids.size()-1);
        }
        if (bidPrice < min) {
            throw new IllegalArgumentException("Too low");
        }
        bids.add(bidPrice);
    }
}
```

B.

in BiddingMarket:

```
    public void printBiddingMarket() {
        for (ItemInfo item : data.values()) {
            item.print();
        }
    }
```

in ItemInfo:

```
    public void print() {
        String str = itemName;
        str += " minPrice=";
        str += String.valueOf(minPrice);
        str += " bids=";
        for(Integer i : bids) {
            str += String.valueOf(i);
            str += ",";
        }
        System.out.println(str);
    }
```

Modellsvar för Uppgift 3: [15 poäng] *interface, iterators, listor*

A.

```
public class EmptyIterator<A> implements Iterator<A> {

    public EmptyIterator() { }

    public boolean hasNext() {
        return false;
    }

    public A next() {
        return null;
    }

}
```

B.

```
public class OneIterator<A> implements Iterator<A> {

    private A a;
    private boolean hasLeft = true;

    public OneIterator(A a) {
        this.a = a;
    }

    public boolean hasNext() {
        if (hasLeft) {
            hasLeft = false;
            return true;
        } else {
            return false;
        }
    }

    public A next() {
        return a;
    }

}
```

C.

```
public class AppendIterator<A> implements Iterator<A> {

    private Iterator<A> a;
    private Iterator<A> b;

    public AppendIterator(Iterator<A> a,Iterator<A> b) {
        this.a = a;
        this.b = b;
    }

    public boolean hasNext() {
        return a.hasNext() || b.hasNext();
    }

    public A next() {
        if (a.hasNext()) {
            return a.next();
        } else {
            return b.next();
        }
    }

}
```

D.

```
public class ReverseIterator<A> implements Iterator<A> {

    private Iterator<A> a;

    public ReverseIterator(Iterator<A> it) {
        List<A> l = new LinkedList<A>();
        while (it.hasNext()) {
            l.add(0,it.next()); // insert element at front
        }
        this.a = l.iterator();
    }

    public boolean hasNext() {
        return a.hasNext();
    }

    public A next() {
        return a.next();
    }

}
```

En annan lösning:

```
public class ReverseIterator<A> implements Iterator<A> {

    private List<A> l = new LinkedList<A>();
    private int index;

    public ReverseIterator(Iterator<A> it) {
        while (it.hasNext()) {
            l.add(it.next());
        }
        index = l.size(); // one past last element
    }

    public boolean hasNext() {
        return (0 < index);
    }

    public A next() {
        if (0 < index) { index--; }
        return l.get(index);
    }

}
```

Modellsvar för Uppgift 4: [10 poäng totalt] *immutable, standard klasser*

A. Ett objekt är immutable om den *inte går att ändra* på efter att den har skapats. Vid operationer som skulle ändra på objektet skapas istället nya objekt. Standard klasserna String och Integer är immutable.

B. Man ser från typsignaturen för `add` metoden att tanken är att `add` ändrar på objektet. Det går inte att implementera `add` metoden i en immutable klass för att typsignaturen inte tillåter en att returnera en ny lista.

C. Dokumentationen visar tydligt att HashMap implementationen litar på att värdet av hashCode funktionen inte ändrar sig. Med andra ord: HashMap är implementerat med antagandet att hashCode funktionen alltid returnerar samma värde för samma objekt. Internt organiserar en HashMap all sin data i luckor enligt hashCode värdet för nycklarna. Om ett objekt i luckan för hashCode värden 3 plötsligt ändras till hashCode värdet 5 då kommer objektet att ligga i fel lucka och HashMap koden kommer inte att titta i rätt lucka.

Modellsvar för Uppgift 5: [15 poäng] *arv, användning av standard bibliotek*

```
public class CountButton extends JButton {

    private int clickCount = 0;
    private String text;

    public CountButton(String text) {
        super(text); // anropar på setText nedan
        addActionListener(e -> { clickCount++; updateText(); });
    }

    private void updateText() {
        super.setText(text + " (" + clickCount + " clicks so far)");
    }

    public void setText(String text) {
        this.text = text;
        updateText();
    }

}
```

Koden ovan är tillräcklig för fulla poäng, men här är lite test kod:

```
    public static void main(String[] args) {
        JFrame f = new JFrame();
        JPanel p = new JPanel();
        JButton b = new CountButton("Click me");
        p.add(b);
        f.add(p);
        f.pack();
        f.setVisible(true);
    }
```