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

DAT050 Objektorienterad programmering

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!***

Uppgift 1: [5 poäng totalt] *grunder i objektorientering och Java*

A. Förklara kort vad det objektorienterade synsättet är. Hur är det man strukturerar objektorienterade program? [2 poäng]

B. Skriv ett kort kodexempel som visar vad lyssnare är i Java. Förklara kort din kod och vad lyssnaren gör. [3 poäng]

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

Denna uppgift handlar om att implementera en klass `BiddingMarket` som fungerar som en databas för information om budgivning för saker folk köper eller säljer.

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 en ny sak som folk kan ge bud på. Saken bör ha ett namn och ett minimum pris.

```
public void addItem(String itemName, int minPrice)
```

- Det ska finnas en metod för att lägga till ett nytt bud för en sak (`itemName`). Denna metod ska kasta en exception ifall den saken inte finns att ge bud på, det redan finns ett högre bud för den saken, eller budet är under minimum priset.

```
public void bidForItem(String itemName, int bidPrice)
```

- Skriv en metod som returnerar all bud hittills för en sak.

```
public List<Integer> getBidHistory(String itemName)
```

Obs: Anta att varje sak har ett unikt namn (`itemName`).

Tips: Det är lättast att lösa uppgiften med hjälp av en hjälpklass vars instanser innehåller all information (minimum pris och budgivnings historik) för *en sak*.

B. Implementera `printBiddingMarket` så att den skriver ut en en rad per sak i databasen. Raden bör ha sakens namn, budhistorik och minimum pris. [5 poäng]

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

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

Följande testkod skriver ut innehållet av en iterator `it`.

```
System.out.print("<");
while (it.hasNext()) {
    System.out.print(it.next());
    if (it.hasNext()) {
        System.out.print(",");
    }
}
System.out.print(">");
```

Exempel: koden ovan skriver ut "<1,2,3>" för en iterator `it` skapad från en lista som består av tre element: 1,2,3. Vi säger att iteratorn `it` *representerar* <1,2,3>.

A. Implementera `EmptyIterator<A>` som alltid representerar <>. [3 poäng]

```
public class EmptyIterator<A> implements Iterator<A> {
    ...
    public EmptyIterator() { ... }
    ...
}
```

B. Implementera `OneIterator<A>` som alltid representerar en iterator som endast innehåller ett element: det elementet som gavs till konstruktorn. [4 poäng]

```
public class OneIterator<A> implements Iterator<A> {
    ...
    public OneIterator(A a) { ... }
    ...
}
```

När testkoden ovan kör på `it` lika med `new OneIterator(5)` bör den skriva ut <5>.

C. Implementera `AppendIterator<A>` som alltid representerar alla element av en iterator `it1` och sedan alla element av en annan `it2`. Givet en iterator som representerar <1,2,3> och en som representerar <4,5> bör `AppendIterator` representera <1,2,3,4,5>. [4 poäng]

```
public class AppendIterator<A> implements Iterator<A> {
    ...
    public AppendIterator(Iterator<A> it1, Iterator<A> it2) { ... }
    ...
}
```

D. Implementera `ReverseIterator<A>` som alltid representerar alla element av en given iterator `it` men i motsatt ordning. Givet en iterator som representerar <1,2,3> bör `ReverseIterator` representera <3,2,1>. [4 poäng]

```
public class ReverseIterator<A> implements Iterator<A> {
    ...
    public ReverseIterator(Iterator<A> it) { ... }
    ...
}
```

Obs: Du kan anta i alla delar att indata inte är null och att alla iterators är ändliga.

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

A. Förklara begreppet *immutable* i (objektorienterad) programmering. [2 poäng]

B. Javas standard bibliotek definierar ett `List` interface. En del av den definitionen är visad nedan. Förklara varför det inte går att implementera detta interface med en immutable klass. [3 poäng]

```
Interface List<E>
  boolean add(E e)
    Appends the specified element to the end of this list.
    Returns true if this collection changed as a result of the call.
  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.
```

C. När man använder `HashMap<K,V>` då brukar `K` vara klasser vars implementationer är immutable. Förklara noggrant varför. Vad kan gå fel ifall man muterar för mycket i klassen som används som `K` i `HashMap<K,V>`? [5 poäng]

Tips: Följande är text från Javas API dokumentation.

```
Class Object
  public int hashCode()
     Returns a hash code value for the object. This method is supported
     for the benefit of hash tables such as those provided by HashMap.
     The general contract of hashCode is: Whenever it is invoked on the
     same object more than once during an execution of a Java application,
     the hashCode method must consistently return the same integer […]
```

För fulla poäng bör du förklara varför det är så viktigt att `hashCode` funktionen returnera samma värde: "`consistently return the same integer`".

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

Din uppgift är att implementera en klass `CountButton` så att den är precis som en `JButton` förutom att texten på knappen alltid inkluderar hur många gånger man klickat på knappen.

Om man skapar en ny `CountButton` med texten "Click me", dvs

```
JButton b = new CountButton("Click me");
```

då ska texten på knappen vara "Click me (0 clicks so far)". Om man sedan klickar på knappen ska texten på knappen automatiskt uppdateras till "Click me (1 clicks so far)". Om man sedan klickar en gång till så då ska det stå "Click me (2 clicks so far)" på knappen.

Obs: Knappens count ska synas också fast användaren anropar på `setText` för knappen. T.ex. om användaren kör (efter två klick)

```
b.setText("New text here");
```

då ska knappen ha texten "New text here (2 clicks so far)".

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

```
Class AbstractButton
  void setText(String text)
     Sets the button's text.
Interface ActionListener
  void actionPerformed(ActionEvent e)
     Invoked when an action occurs.
Class Component extends Object
  void repaint()
     Repaints this component.
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 JButton extends AbstractButton extends … extends Object
  JButton(String text)
     Creates a button with text.
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.
```