CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
DAT050	Objektorienterad programmering			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
DAT050-0014-OPU		23/8-23	7	4

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1			
2	X	14	
3	X	12	
4	X	8	
5	X	3	
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng		37	

[redacted]

3

2 a)

```
public class BiddingMarket {
    public HashMap<Item> market;

    private class Item {
        public String itemName;
        public int minPrice;
        public List<Integer> bidHistory;

        public Item(String itemName, int minPrice, List<Integer> bidHistory){
            this.itemName = itemName;
            this.minPrice = minPrice;
            this.bidHistory.size = bidHistory.size;
            for(Integer i : bidHistory){
                this.bidHistory[i] = bidHistory[i];
            }
        }
    }

    public void addItem(String itemName, int minPrice){
        market.put(itemName, minPrice);
    }

    public void bidForItem(String itemName, int bidPrice){
        Item i = market.get(itemName);
        if(i == null) throw new IllegalArgumentException;
        if((i.minPrice > bidPrice) || (i.bidHistory[bidHistory.size()-1] > bidPrice)){
            throw new IllegalArgumentException;
        }
        i.bidHistory.add(bidPrice);
    }
}
```

OK!

2

2

2

3

2a)

```
public List<Integer> getBidHistory(String itemName) {
    Item i = market.get(itemName);
    if(i == null) throw new IllegalArgumentException;
    return i.bidHistory;
}
```

9/10

b)

```
public void printBiddingMarket() {
    for(Item i : market) {
        System.out.println(i.itemName + ", " + i.bidHistory + ", " + i.minPrice);
    }
}
```

5/5

3

3a)

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

  public EmptyIterator() {

  }
  public boolean hasNext() {
    return false;
  }
}
```

next?

2/3

b)

```
public class OneIterator<A> implements Iterator<A> {
  public A a;
  public int i=0;
  public OneIterator(A a) {
    this.a = a;
  }

  public boolean hasNext() {
  if(i == 0) {
    i++;
    return true;
  }
   return false;
  }

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

4/4

3

3c)

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

    public A[] arr;
    public int pos = -1;
    public Iterator<A> it1;
    public Iterator<A> it2;

    public AppendIterator(Iterator<A> it1, Iterator<A> it2){

        this.it1 = it1;
        this.it2 = it2;
        this.arr = appendIt(this.it1, this.it2);
    }

    public boolean hasNext(){

        if (it1.hasNext() || it2.hasNext()) return true;
        return false;

    }

    public A next(){

        pos++;
        return arr[pos];
    }

    public A[] appendIt(Iterator<A> it1, Iterator<A> it2){
        A[] a;
        int i = 0;

        while(it1.hasNext()){
            a[i] = it1.next();
            i++;
        }
        while(it2.hasNext()){
            a[i] = it2.next();
            i++;
        }
        return a;
    }

}
```

null

crash!

3/4

3

3d)

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

    public Iterator<A> it;
    public int pos;
    public A[] arr;

    public ReverseIterator(Iterator<A> it){
      this.it = it;
      arr = helper(this.it);
    }

    public boolean hasNext() {
        if (it.hasNext()) return true;
        return false;
    }

    public A next() {
      pos--;
      return arr[pos];
    }

    public A[] helper (Iterator<A>it) {
        A[] a;
        int i = 0;
        while(it.hasNext()) {
          a[i] = it.next();
          i++;
        }
        pos = a.length;
    }
```

3/4

CHALMERS | Anonymous code / Anonym kod: DAT050-0014-OPU | Points for question (to be filled in by teacher) / Poäng på uppgiften (ifylles av lärare) | Consecutive page no. / Löpande sid nr 6 | Question no. / Uppgift nr 4

4a) Immutable betyder att något inte kan muteras. För att göra en klass immutable kan man exempelvis deklarera variabler med: private static...

~~Att~~ göra klasser immutable hindrar aliasing problem. 1/2

b) För att add ska kunna returnera true måste listan ha ändrat sig. Hade klassen varit immutable hade add alltid returnerat false. 3/3

c) Muteras klassen för mycket kommer hashCode metoden inte kunna "consistently return the same integer". Det är viktigt för att man ska få rätt Key-Value par. 4/5

5)

```
public class CountButton extends JButton {
  public String standard = " clicks so far)";
  public int count = 0;
  public String text;
  public CountButton(String text){
    this.text = text + "(" + count + standard;
  }
  public void setText(String text){
    this.text = text + "(" + count + standard;
  }
  public void actionPerformed(ActionEvent e){
    count++;
  }
}
```

3/15