

## Lösningsförslag till tentamen

<b>Kurs</b>	<b>Objektorienterad programmering</b>
<b>Tentamensdatum</b>	<b>2009-08-28</b>
<b>Program</b>	<b>D2</b>
<b>Läsår</b>	<b>2008/2009, lp 1</b>
<b>Examinator</b>	<b>Uno Holmer</b>

---

### Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

### Uppgift 2 (5+5 p)

a) Lägg till metoden

```
private void quit() { System.exit(0); }
```

och följande kod i makeMenuBar:

```
JMenuItem quitItem = new JMenuItem("Quit");  
quitItem.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) { quit(); }  
});  
menu.add(quitItem);
```

b) Lägg till metoden

```
private void sekToDollars() {  
    String digits =  
        JOptionPane.showInputDialog(null,  
                                     "Type in the price in SEK",  
                                     "Currency converter",  
                                     JOptionPane.PLAIN_MESSAGE);  
  
    if ( digits == null )  
        return;  
    Double sekValue = Double.parseDouble(digits);  
    sekDigits.setText(sekValue.toString());  
    Double dollarValue = sekValue/dollarToSek;  
    dollarDigits.setText(dollarValue.toString());  
}
```

och följande kod i makeMenuBar:

```
JMenuItem sekToDollarsItem = new JMenuItem("SEK to dollars");  
sekToDollarsItem.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        sekToDollars();  
    }  
});  
menu.add(sekToDollarsItem);
```

**Uppgift 3** (7+4 p)

a)

1-3: obj har statisk typ Base och dynamisk typ Sub

1: Sub.f1	f1 definieras om i Sub
2: Base.f2++Sub.f2	f2 definieras om i Sub och den börjar med att anropa Base.f2 explicit
3: Base.f3	f3 definieras ej om i Sub utan ärvs från Base

4-7: obj2 har statisk och dynamisk typ Sub. Subklassens metoder anropas utom i fall 6 där f3 ärvs från basklassen.

4: Sub.f1  
5: Base.f2++Sub.f2  
6: Base.f3  
7: Sub.f4

b)

Int obj1 = new Int();

Gränssnitt är abstrakta klasser och sådana får ej instansieras.

Base obj2 = new Base();                      Base är abstrakt.

Sub1 obj3 = new Sub1();                      Se nedan.

Sub2 obj4 = new Sub2();                      Korrekt.

Klassen Sub1 kan ej kompileras eftersom den abstrakta metoden h ej är implementerad vare sig i Base eller i Sub1. Notera att h ej måste definieras i Base. Eftersom klassen är abstrakt kan definitionen av h skjutas upp till någon subklass längre ner i hierarkin. Alla mellanliggande klasser blir då abstrakta.

**Uppgift 4** (3+4+4 p)

- a) Metoderna equals och hashCode överskuggas troligen ej i klassen AddressBookEntry. Då ärvs dessa metoder från Object. Likhet blir då pekarlikhet och distinkta objekt får olika hashkoder, även om de är lika.
- b) Inför metoderna equals och hashCode i HashBookEntry:

```
public class AddressBookEntry {
    ...
    public final boolean equals(Object other) {
        if ( this == other )
            return true;
        if ( other instanceof AddressBookEntry ) {
            AddressBookEntry e = (AddressBookEntry)other;
            return name == null ?
                e.name == null :
                name.equals(e.name);
        } else
            return false;
    }
    public int hashCode() { return name.hashCode(); }
}
```

c)

```
public class AddressBookEntry implements Cloneable {
    ...
    public AddressBookEntry clone()
        throws CloneNotSupportedException
    {
        return (AddressBookEntry)super.clone();
    }
}

public class AddressBook implements Cloneable {
    ...
    // Copy constructor
    public AddressBook1(AddressBook1 other)
        throws CloneNotSupportedException
    {
        book = (LinkedList<AddressBookEntry>)other.book.clone();
        for (int i = 0; i < other.book.size(); i++)
            book.set(i, other.book.get(i).clone());
    }

    // Alternative copy constructor
    public AddressBook1(AddressBook1 other)
        throws CloneNotSupportedException
    {
        book = new LinkedList<AddressBookEntry>();
        for ( AddressBookEntry e : other.book )
            book.add(e.clone());
    }

    public AddressBook clone() throws CloneNotSupportedException {
        return new AddressBook(this);
    }
    ...
}
```

**Uppgift 5** (8 p)

```
public class Resource {
    private static Resource instance = null;

    private Resource() {}

    public static Resource getInstance() {
        if ( instance == null )
            instance = new Resource();

        return instance;
    }

    public void someMethod() { ... }
}

public class Client1 {
    private Resource r;

    public Client1() {
        this.r = Resource.getInstance();
    }

    public void clientMethod() {
        ...
        r.someMethod();
        ...
    }
}
```

(analogt Client2)

```
public class Main {
    static public void main(String[] arg) {
```

Observera att klienterna inte explicit skapar objekt av resursklassen.  
Det får bara resursklassen själv göra.

```
        Client1 c1 = new Client1();
        Client2 c2 = new Client2();
        ...
    }
}
```

En nackdel med metoden är att klienterna ser att resursen är en Singleton. Ett alternativ är endast att ändra koden för main i tesen till:

```
public class Main {
    static public void main(String[] arg) {
        Resource r = Resource.getInstance();
        Client1 c1 = new Client1(r);
        Client2 c2 = new Client2(r);
        // ...
    }
}
```

**Uppgift 6** (7+3 p)

a)

```
public class PersistentQueue implements Queue {
    private Queue q = null;

    public PersistentQueue(Queue q) {
        this.q = q;
    }

    // Delegation
    public void add(String element) {
        q.add(element);
    }

    public boolean isEmpty() {
        return q.isEmpty();
    }

    public String get() throws NoSuchElementException {
        return q.get();
    }

    public void clear() {
        q.clear();
    }

    // New methods
    public void save(String fileName) throws IOException {
        try {
            ObjectOutputStream outStream =
                new ObjectOutputStream(
                    new FileOutputStream(fileName));
            outStream.writeObject(q);
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }

    public Queue load(String fileName) throws IOException {
        q.clear();
        try {
            ObjectInputStream inStream =
                new ObjectInputStream(
                    new FileInputStream(fileName));
            q = (Queue)inStream.readObject();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return q;
    }
}
```

b)

```
PersistentQueue pq = new PersistentQueue(new SomeQueue());
pq.add("A");
pq.add("B");
pq.add("C");

try {
    pq.save("savedq.dat");
} catch (IOException e) {
    e.printStackTrace();
}

try {
    pq.load("savedq.dat");
    while ( ! pq.isEmpty() )
        System.out.println(pq.get());
} catch (IOException e) {
    e.printStackTrace();
}
```