

Instuderingsuppgifter läsvecka 6 - LÖSNINGAR

1.

- a) Att en klass inte är trådsäker innebär att flera trådar, som använder ett objekt av klassen, samtidigt tillåts förändra tillståndet hos objektet.

I klassen `Product` är inte metoderna `addToStock` och `removeFromStock` trådsäkra. Följande kan inträffa:

Anta att instansvariabeln `nrInStock`, som anger antalet produkter i lager, har värdet 12 när två trådar `t1` och `t2` samtidigt exekverar metoden `addToStock` respektive `removeFromStock`. I beräkningen av satserna

`nrInStock = nrInStock + quantity;`

respektive

`nrInStock = nrInStock - quantity;`

läser både `t1` och `t2` av värdet 12 på instansvariabeln `nrInStock`. Således händer följande:

`t1` adderar 20 till värdet 12 och får resultatet 32

`t2` subtraherar 10 från värdet 12 och får resultatet 2

`t2` lagrar värdet 2 i instansvariabel `nrInStock`

`t1` lagrar värdet 32 i instansvariabeln `nrInStock`

Effekten blir att instansvariabeln `nrInStock` nu har värdet 32, men det faktiska antalet produkter i lager är 22.

- b) Mutator-metoderna `addToStock` och `removeFromStock` måste synkroniseras:

```
public synchronized void addToStock(int quantity) {  
    nrInStock = nrInStock + quantity;  
} // addToStock  
  
public synchronized void removeFromStock(int quantity) {  
    if (quantity > nrInStock)  
        throw new IllegalArgumentException("To big quantity");  
    nrInStock = nrInStock - quantity;  
} // addToStock
```

2.

a)

- `wait()` – blockerar den anropande tråden samt låser upp den aktuella monitorns lås så att andra trådar kan gå in i monitorn. Monitorn låses igen innan tråden exekverar vidare efter anropet.
- `notify()` – gör att en tråd som väntar i ett anrop av `wait()` i den aktuella monitorn görs körbar och lämnar anropet av `wait()` så fort som monitorn inte längre är låst.
- `notifyAll()` – som `notify()`, fast den gör alla väntande trådar körbara.

- b) För att någon av de nämnda metoderna ska kunna anropas måste exekveringen befinna sig inne i en metod som är deklarerad `synchronized` i objektet som vi anropar metoden på (eller i ett motsvarande `synchronized`-block).

3.

- a) Ett problem Kalle kan få är att meddelanden försvinner eftersom **set()** skulle kunna anropas två gånger i följd av samma tråd. Ett annat problem som kan uppstå är att den läsande tråden stjälar all tillgänglig CPU-tid (busy-wait) vilket kan göra att andra trådar, t.ex. den som anropar **set()** aldrig får köra.

b)

```
public class Buffer {
    private String message;
    public synchronized void set(String m) {
        while ( message != null) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        message = m;
        notifyAll();
    } //set
    public synchronized String get() {
        while ( message == null) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        String m = message;
        message = null;
        notifyAll();
        return m;
    } //get
} //Buffer
```

4.

Inre klasser använd då en klass behöver en "hjälpklass" som inte används utanför klassen eller då klassen behöver tillgång till den omslutande klassens attribut (representation) t.ex. en iterator.

5.

- a) möjlig b) omöjlig c) möjlig d) omöjlig

Allmänt gäller: R måste föregå G, G måste föregå I, I måste föregå V och V måste föregå K, eftersom dessa utskrifter utförs i denna ordning av tråden som exekverar **main**-metoden. Vidare måste O föregå Y, eftersom dessa utskrifter utförs i denna ordning i **run**-metoden i tråden **t1**.

6.

En statisk inre klass har inte access till instansvariabler i den yttre klassen. Lösningen är antingen att göra klassen **Inner** till en icke-statisk klass, eller att göra instansvariabeln **str** till en klassvariabel.

```
public class Problem {
    private String str;
    private class Inner {
        private void testMethod() {
            str = "Set from Inner";
        } //testMethod
    } //Inner
} //Problem
```

```
public class Problem {
    private static String str;
    private static class Inner {
        private void testMethod() {
            str = "Set from Inner";
        } //testMethod
    } //Inner
} //Problem
```

7.

- a) Man låter metoden `clone()` kasta ett undantag enligt:

```
public class NoCopy {  
    ...  
    public NoCopy clone() throws CloneNotSupportedException {  
        throw new CloneNotSupportedException();  
    } //clone  
} //NoCopy
```

b)

```
public class C implements Cloneable {  
    private int x;  
    private B b;  
    ...  
    public C clone() {  
        try {  
            C copy = (C) super.clone();  
            copy.b = b.clone();  
            return copy;  
        } catch (CloneNotSupportedException e) {  
            return null; //never invoked  
        }  
    }  
}
```

c)

```
import java.util.ArrayList;  
public class D implements Cloneable {  
    private int x;  
    private ArrayList<C> cs;  
    ...  
    public D(D other) {  
        this.x = other.x;  
        cs = new ArrayList<C>();  
        for (C item: other.cs) {  
            cs.add(item.clone());  
        }  
    }  
    public D clone() {  
        return new D(this);  
    }  
}
```

8.

- a) Ström som används för att lagra data på ett skrivminne: `FileOutputStream`
- b) Superklass för alla klasser som representerar utgående strömmar: `OutputStream`
- c) Superklass för flera av de klasser som utökar funktionaliteten hos en existerande ström: `FilterOutputStream`
- d) Innehåller metoder för att skicka olika typer av primitiva datatyper, t.ex. heltal, flyttal: `DataOutputStream`

```
a) import java.io.*;
public class Faulty {
    public static void main(String[] arg) throws IOException {
        BufferedReader infil = new BufferedReader(new FileReader("data.txt"));
        int totally = 0;
        int faultyBefore = 0;
        int faultyAfter = 0;
        while(true) {
            String rad = infil.readLine();
            if (rad == null)
                break;
            totally++;
            double diff = Double.parseDouble(rad);
            if (Math.abs(diff) >= 0.0001)
                faultyBefore++;
            if (Math.abs(diff) >= 0.00001)
                faultyAfter++;
        }
        infil.close();
        System.out.println("Ej accepterade axlar ökar från " + 100* (double) faultyBefore/ totally
                           + " % till " + 100 * (double) faultyAfter/ totally + " %.");
    }
}
```

```
b) import java.io.*;
public class Faulty {
    public static void main(String[] arg) throws IOException {
        DataInputStream infil = new DataInputStream(new FileInputStream("data.bin"));
        int totally = 0;
        int faultyBefore = 0;
        int faultyAfter = 0;
        while(true) {
            try {
                double diff = infil.readDouble();
                totally++;
                if (Math.abs(diff) >= 0.0001)
                    faultyBefore++;
                if (Math.abs(diff) >= 0.00001)
                    faultyAfter++;
            } catch (EOFException e) {
                //reached end of file
                break;
            }
        }
        infil.close();
        System.out.println("Ej accepterade axlar ökar från " + 100*(double) faultyBefore/ totally
            + " % till " + 100 * (double) faultyAfter/ totally + " %.");
    }
}
```

c) Därför att binärfiler kräver mindre minnesutrymme, samt är enklare och effektivare att bearbeta.

10.

a) Klassen måste implementera interfacet `Serializable`.

```
import java.io.Serializable;
public class Person implements Serializable {
```

b)

```
import java.io.*;
public class Splitt {
    public static void main(String[] arg) throws IOException, ClassNotFoundException {
        try {
            ObjectInputStream infile = new ObjectInputStream(new FileInputStream(arg[0]));
            ObjectOutputStream femalefile = new ObjectOutputStream(new FileOutputStream(arg[1]));
            ObjectOutputStream malefile = new ObjectOutputStream(new FileOutputStream(arg[2]));
            Person p;
            while (true) {
                try {
                    p = (Person) infile.readObject();
                }
                catch (EOFException e) {
                    break;
                }
                if (p.isFemale()) {
                    femalefile.writeObject(p);
                } else {
                    malefile.writeObject(p);
                }
            }
            infile.close();
            femalefile.close();
            malefile.close();
        }
        catch (FileNotFoundException e) {
            System.out.println("Öppningen av filerna misslyckades!");
            System.exit(0);
        }
    }
}
```