Instuderingsuppgifter läsvecka 5

1.

Följande interface och klass är givna:

```
public interface Set<T> {
    public void add(T e);
    public void delete(T e);
    public int size();
    public boolean has(T e);
}//Set

public class NewSet<T> {
    public void insert(T e) { . . . }
    public void remove(T e) { . . . }
    public int size() { . . . }
    public boolean contains(T e) { . . . }
}//NewSet
```

Skriv en klass NewSetAdapter som implementerar designmönstret *Adapter* för att anpassa gränssnittet för typen NewSet till typen Set.

2.

a) Vad är syftet med designmönstret *Decorator*?

b) Betrakta nedanstående kod:

```
public class A {
    public void f() {
        System.out.println("A");
    }
}//A

public class B extends A {
    public void f() {
        super.f();
        System.out.println("B");
    }
}//B

public class C extends A {
    public void f() {
        super.f();
        System.out.println("C");
    }
}//C

public class BC extends A {
    public void f() {
        super.f();
        System.out.println("B");
        System.out.println("C");
    }
}//BC

public class CB extends A {
    public void f() {
        super.f();
        System.out.println("C");
        System.out.println("B");
    }
}//CB

public static void main(String[] args) {
    A a = new A();
    A b = new B();
    A c = new C();
     A bc = new BC();
    A cb = new CB();
    a.f();
    b.f();
    c.f();
    bc.f();
    cb.f();
}//main
```

Skriv om koden genom att använda designmönstret *Decorator*.

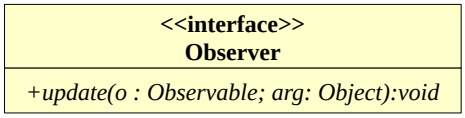
3.

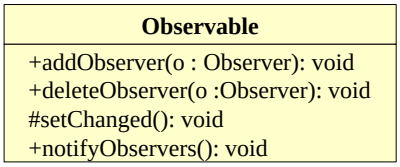
Antag att klassen B är intresserad av alla tillståndsförändringar som sker i A. När instansvariabeln value i A ändras skall B skriva ut det nya värdet på konsolen. Komplettera koden så att designmönstret *Observer* realiseras.

```
public class A {
   private int value = 0;
   public void compute(int x) {
      value += x;
   }
   public int getValue() {
      return value;
   }
}//A
```

```
public class B {
   private A theAObject;
   public B(A anAObject) {
      theAObject = anAObject;
   }
}//B
```

Interfacet Observer och klassen Observable har följande utseende:





4.

Förklara kortfattat syftet med designmönstret *Model-View-Control* och vad de tre komponenterna (Model, View och Control) har för funktion.

5.

Vad är syftet med designmönstret *Façade*?

6.

Vad är det för skillnad mellan kontrollerande (*checked*) respektive icke-kontrollerande (*undchecked*) undantag? När skall kontrollerande respektive icke-kontrollerande undantag användas?

7.

Vad har de olika delarna i **try-catch-finally**-konstrukten för uppgifter? Vilka av dessa delar kan utelämnas?

8.

Utöka nedanstående program med lämplig undantagshantering.

```
public class Maximum {
   public static void main(String[] args) {
      if (args.length >= 2 ) {
         int t1 = Integer.parseInt(args[0]);
         int t2 = Integer.parseInt(args[1]);
         int max = t1 + t2;
         System.out.println("The maximum of " + t1 + " and " + t2 + " is " + max);
      }
      else
         System.out.println("Usage: java Maximum interger1 integer2");
   }//main
}//Maximum
```

9.

Undantagsklasserna FirstException, SecondException och ThirdException samt klasserna Base och Sub definieras enligt:

```
public class FirstException  extends Exception {...}

public class SecondException  extends FirstException {...}

public class ThirdException  extends SecondException {...}

public class Base {
   public void method()  throws SecondException {
      ...
   }
}//Base

public class Sub  extends Base {
   @Override
   public void method()  throws ...
}//Sub
```

Ange, för var och en av nedanstående överskuggningar av metoden method, om överskuggningen är korrekt eller inte:

a) **public void** method() **throws** SecondException, ThirdException {...}

b) **public void** method() **throws** SecondException {...}

c) **public void** method() **throws** ThirdException {...}

d) **public void** method() **throws** FirstException {…}

10.

Är nedanstående deklarationer korrekta? Om inte, förklara varför!

a)
```
public interface IB {
   public void doIt() throws AnException;
}//IB

public class ImpIB implements IB {
   public void doIt() {
      //the code not shown
   }//doIt
}//ImpIB
```

b)
```
public class AnException extends RuntimeException {. . .}

public interface IC {
   public void doIt();
}//IC

public class ImpIC implements IC {
   public void doIt() throws AnException{
      //the code not shown
   }//doIt
}//ImpIC
```

c)
```
public class AnException extends Exception {. . .}

public interface ID {
   public void doIt();
}//ID

public class ImpID implements ID {
   public void doIt() throws AnException{
      //the code not shown
   }//doIt
}//ImpID
```