**Tentamen 101213 - LÖSNINGSFÖRSLAG**

**Uppgift 1.**

a) Ger utskriften ”c( ) in D”

b) Ger utskriften ”true”

c) Tilldelningen C x = **new** D() ger kompileringsfel eftersom klassen D är abstrakt.

d) Ger utskriften ”b( ) in E”

e) Anrope x.b() ger kompileringsfel eftersom typen C inte har operationen b().

f) Ger exekveringsfel. Typen F kan inte typomvandlas till typen D.

g) Tilldelningen D x = **new** C() ger kompileringfel, eftersom typen C inte är subtyp till D.

h) Tilldelningen C y = x ger kompileringsfel, eftersom x är av typen A och A är ingen subtyp till C.

**Uppgift 2.**

a) Nej! En subklass får inte ha ett starkare förvillkor på en överskuggad metod än vad superklassen har.

b) Ja! En subklass får ha ett starkare eftervillkor på en överskuggad metod än vad superklassen har.

c) Den andra specifikationen

```
/**
  @pre  none
  @return If arr is null or empty throws IllegalArgumentException. Otherwise, the mean of the numbers in arr
**/
public static double mean(int[] arr) { . . . }
```

är att föredra, eftersom denna metod inte har några förvillkor som användaren måste uppfylla för att kunna använda metoden.

**Uppgift 3.**

a) Satsen

```
catch (DomsException e)
```

kommer aldrig kan nås eftersom DomsException är en subklass till Exception och därmed fångas undantag av typen DomsException av satsen

```
catch (Exception e)
```

Koden skall ha följande utseende:

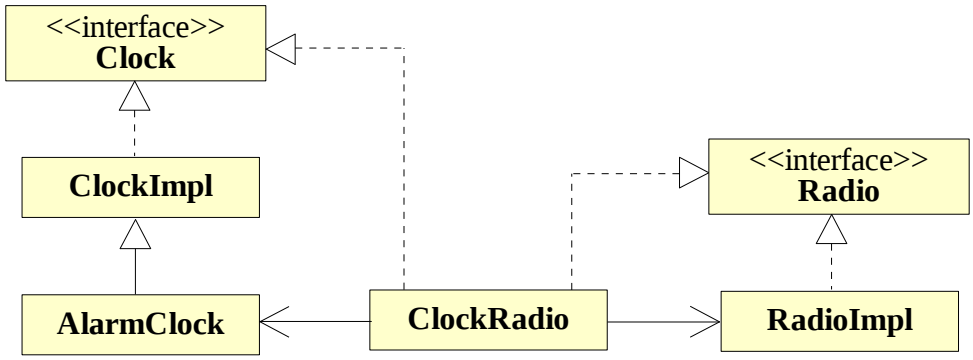
```
try {
    throw new DomsException();
} catch (DomsException e) {
    System.out.println("Caught Dom's exception!");
} catch (Exception e) {
    System.out.println("Caught an exception!");
} finally {
    System.out.println("This is the finally block!");
}
```

b) `SubClass` kan inte referera till privata instansvariabler i sin superklass `SubClass`. En korrekt lösning har följande utseende:

```
public class SubClass extends SuperClass {
    private int secondValue;
    public SubClass(int val, String desc, int val2) {
        super(val, desc);
        secondValue = val2;
    }//constructor
    public String toString(){
        return (super.toString() + " & second value " + secondValue);
    }//toString
}//SubClass
```

**Uppgift 4.**

Lösning presenterad som UML-diagram:




Lösning som kodskelett:

```
public interface Clock {
    . . .
}//Clock

public class ClockImpl implements Clock {
    . . .
}// ClockImpl

public class AlarmClock extends ClockImpl {
    . . .
}// AlarmClock

public interface Radio {
    . . .
}//Radio

public class RadioImpl implements Radio {
    . . .
}// RadioImpl

public class ClockRadio implements Clock, Radio {
    private Clock clock = new AlarmClock();
    private Radio radio = new RadioImpl();
    . . .
}
```

**Uppgift 5.**

```
public abstract class Vehicle {
    . . .
    public boolean isSensitive();
}// Vehicle

public class Car extends Vehicle {
    . . .
    public boolean isSensitive() {
        return false;
    }// isSensitve
}//Car

public class Lorry extends Vehicle {
    . . .
    public boolean isSensitive() {
        return false;
    }// isSensitve
}//Lorry

public class Motorcycle extends Vehicle {
    . . .
    public boolean isSensitive() {
        return true;
    }// isSensitive
}// Motorcycle

public class MobileHome extends Vehicle {
    . . .
    public boolean isSensitive() {
        return true;
    }// isSensitve
}// MobileHome

public class VehicleUtil {
    // omissions
    public static void removeWindSensitiveVehicles(List<Vehicle> list) {
        Iterator<Vehicle> iterator = list.iterator();
        while(iterator.hasNext()) {
            Vehicle vehicle = iterator.next();
            if(vehicle.isSensitive ()) {
                iterator.remove();
            }
        }
    }// removeWindSensitiveVehicles
}// VehicleUtil
```

Man skulle också kunna införa ett interface

```
public interface Sensitive {
    public boolean isSensitive();
}// Sensitive
```

som klassen `Vehicle` implementerar

```
public abstract class Vehicle implements Sensitive {
    . . .
}// Vehicle
```

I övrigt blir klasserna som ovan.

Även andra lösningar är möjliga.

**Uppgift 6.**

Att använda LinkedList i implementationen är att föredra då implementationen blir enklare och möjligen också mera effektiv.

Implementation med LinkedList:

```
import java.util.*;
public class ListStack<E> implements Stack<E> {
    private LinkedList<E> elements;
    public ListStack() {
        elements = new LinkedList<E>();
    }//constructor
    public void push(E e) {
        elements.addFirst(e);
    }//push
    public E top() {
        if (elements.isEmpty())
            throw new EmptyStackException();
        return elements.getFirst();
    }//top
    public E pop() {
        if (elements.isEmpty())
            throw new EmptyStackException();
        return elements.removeFirst();
    }//pop
    public boolean isEmpty() {
        return elements.isEmpty();
    }//isEmpty
}//ListStack
```

Implementation med ArrayList:

```
import java.util.*;
public class ListStack<E> implements Stack<E> {
    private List<E> elements;
    public ListStack() {
        elements = new ArrayList<E>();
    }//constructor
    public void push(E e) {
        elements.add(e);
    }//push
    public E top() {
        if (elements.isEmpty())
            throw new EmptyStackException();
        return elements.get(elements.size()-1);
    }//top
    public E pop() {
        if (elements.isEmpty())
            throw new EmptyStackException();
        return elements.remove(elements.size()-1);
    }//pop
    public boolean isEmpty() {
        return elements.isEmpty();
    }//isEmpty
}//ListStack
```

**Uppgift 7.**

```
import java.util.Observable;
public class Clock extends Observable {
    private Time time;
    public void tick() {
        time.increase();
        setChanged();
        notifyObservers();
    }//constructor
    public Time getTime() {
        return time;
    }//getTime
}//Clock

import java.util.Observer;
import java.util.Observable;
public class Display implements Observer {
    private Clock clock;
    public Display(Clock clock) {
        this.clock = clock;
        clock.addObserver( this );
    }//constructor
    private void show(Time time) {
        ...
    }//show
    public void update(Observable obs, Object o) {
        show(clock.getTime());
    }//update
}//Display
```

Behövde inte skrivas

```
import java. Observer;
import java. Observable;
public class Console implements Observer {
    private Clock clock;
    public Console(Clock clock) {
        this.clock = clock;
        clock.addObserver(this);
    }//constructor
    private void print(Time time) {
        ...
    }//print
    public void update(Observable obs, Object o) {
        print(clock.getTime());
    }//update
}//Console
```

**Uppgift 8.**

```
public class ShutdownThread extends Thread {
    private int secs;
    private String msg;

    public ShutdownThread(String msg, int secs) {
        this.secs = secs;
        this.msg = msg;
    }//constructor

    public void run() {
        System.out.println(msg);
        try {
            Thread.sleep(secs*1000);
        } catch (InterruptedException e) {
            return;
        }
        shutdownNow();
    }//run

    public static void shutdownNow() {
        System.out.println("You failed!");
        System.exit(0); // cause entire JVM to shut down
    }// shutdownNow
}// ShutdownThread
```

**Uppgift 9.**

a) Flera trådar kan samtidigt anropa metoden `report`. Effekten kan bli att ett resultat som rapporteras in inte kommer med i slutresultatet.

Exempel:

Anta att instansvariabeln totalStep har värdet 123456 när två trådar t1 och t2 samtidigt exekverar metoden report. I beräkningen av satsen

```
totalSteps = totalSteps + steps;
```

läser både t1 och t2 av värdet 123456 på instansvariabeln totalStep.

t1 adderar 1234 till värdet 123456 och får resultatet 124690.

t2 adderar 4321 till värdet 123456 och får resultatet 127777.

t2 lagrar värdet 127777 i instansvariabel `totalStep`.

t1 lagrar värdet 124690 i instansvariabeln `totalStep`.

Effekten blir att de 4321 stegen som t1 skulle registrera aldrig blev registrerade i slutresultatet.

b)

```
public class Team {
    private int totalSteps = 0;
    public synchronized  void report(int steps) {
        totalSteps = totalSteps + steps;
    }//report
    public int read() {
        return totalSteps;
    }//read
}//Team
```