Instuderingsuppgifter läsvecka 3

1.

Implementationsarv är en viktig mekanism för återanvändning. Återanvändning är dock inte i sig ett tillräckligt motiv för att använda implementationsarv. Visa detta påstående med ett exempel.

2.

Förklara varför en subklass har en stark koppling till sin superklass.

3.

Vad säger *The Principle of Least Astonishment*?

4.

Vad är det för skillnad mellan en subtyp i Java och en äkta subtyp (d.v.s. en subtyp som uppfyller *Liskov Substitution Principle*).

5.

Vad innebär delegering?

6.

En grundprincip är att föredra delegering framför implementationsarv.

a) Ge exempel på vilka motiv som finns bakom denna princip.

b) Implementationsarv kan dock ibland vara lämpligt att använda. När?

7.

Vad är funktionell dekomposition? Vilka fördelar finns det med funktionell dekomposition?

8.

Vad innebär *The Separation of Concerns Principle*? Förklara med ett exempel!

9.

Vad är en sidoeffekt? Vad är en oönskad sidoeffekt?

10.

Vad innebär T*he Command-Query Separation Principle*?

11.

Förklara vad *kontraktbaserad design* (*Design by Contract*) innebär. Vad är ett kontrakt? Vem är ansvarig för de olika delarna i kontraktet? Vilka fördelar finns med kontraktbaserad design?

12.

Hur skiljer sig *design by contract* från *defensiv programmering*?

13.

När skall klassinvarianterna kontrolleras?

14.

Eftersom klienten ansvarar för att förvillkoren är uppfyllda måste dessa kunna verifieras mot klassens publika gränssnitt. Måste eftervillkor och klassinvarianter också vara verifierbara mot det publika gränssnittet? Motivera ditt svar!

15.

Vad har en programutvecklare för praktisk nytta av *assertions*?

16.

Koden i metoden chooseSupplier är mindre bra.

```
public static void chooseSupplier(Object obj) {
   if (o instanceof SupplierA) {
      ((SupplierA) obj).doIt();
   } else if ( obj instanceof SupplierB) {
      ((SupplierB) obj).doIt();
   } else if ( obj instanceof SupplierC) {
      ((SupplierC) obj).doIt();
   }
}//chooseSupplier

public class SupplierA {
   public void doIt() {
      System.out.println("Done by supplier A");
   }//doIt
}//SupplierA

public class SupplierB {
   public void doIt() {
      System.out.println("Done by supplier B");
   }//doIt
}//SupplierB

public class SupplierC {
   public void doIt() {
      System.out.println("Done by supplier C");
   }//doIt
}//SupplierC
```

a) Motivera vad som är problemet.

b) Lös problemet. Om du har flera lösningsalternativ, motivera varför du valt den lösning du gjort. Du får ändra fritt i koden bara resultatet blir som det ursprungliga (d.v.s. klasserna skall finnas och anropet chooseSupplier(...) skall producera samma utskrift).

17.

Förklara varför metoden setPaymentAmount i klassen CreditCardPayment bryter mot *Liskov Substitution Principle*.

```
public abstract class Payment {
   /**
      @pre amount >= 0
   **/
   public void setPaymentAmount(int amount) { . . . }
}//Payment

public class CreditCardPayment extends Payment {
   /**
      @pre  amount > = 250
   **/
   public void setPaymentAmount(int amount) { . . . }
}//CreditCardPayment
```

18.

Betrakta följande abstraktioner:

a)

```
/**
   @returns  0 if arr is null or arr is empty, otherwise the minimum value of arr
**/
public int min (int[ ] arr)
```

b)

```
/**
   @returns  the minimum value of arr
   @throws  throws NullPointerException if arr is null and EmptyException if arr is empty
**/
public int min (int[ ] arr) throws NullPointerException, EmptyException
```

c)

```
/**
   @pre arr is not null and arr is not empty
   @returns  the minimum value of arr
**/
public int min (int[ ] arr)
```

Är någon av abstraktionerna att föredra framför de andra, eller är de likvärdiga? Motivera ditt svar!