Instuderingsuppgifter läsvecka 3 - LÖSNINGAR

1.

Se föreläsningsanteckningarna.

2.

Se föreläsningsanteckningarna.

3.

Se föreläsningsanteckningarna.

4.

Se föreläsningsanteckningarna.

5.

Se föreläsningsanteckningarna.

6.

Se föreläsningsanteckningarna.

7.

Se föreläsningsanteckningarna.

8.

Se föreläsningsanteckningarna.

9.

Se föreläsningsanteckningarna.

10.

Se föreläsningsanteckningarna.

11.

Se föreläsningsanteckningarna.

12.

Defensiv programmering förespråkar att man lägger in kontroller i varje programmodul för att upptäcka och hantera oförutsedda situationer. Detta leder till redundanta kontroller (t.ex. kan både klient och server utföra samma kontroller). Stora mängder kontroller gör att programmet bli mer komplext, svårare att underhålla och långsammare.

Vid design by contract är ansvarsfördelningen tydlig och är en del av gränssnittet för modulen, vilket förebygger redundanta kontroller och förenklar underhåll.

13.

Innan man lämnar varje konstruktor och innan man lämnar varje publik mutator-metod.

14.

Nej! Det är metoden (servern) som ansvarar för eftervillkor och klassinvarianter varför dessa kan uttryckas med hjälp av den interna implementationen eller det publika gränssnittet.

15.

Assertions är till stor hjälp i felsökningsprocessen för att kunna rätta fel som beror på att för- och eftervillkor samt klassinvarianter inte har uppfyllts.

16.

a) Designen bryter mot Open/Close-principen. Om man skulle införa ytterligare en klass måste man ändra i metoden chooseSupplier.

b) Inför ett gränssnitt som klasserna SupplierA, SupplierB och SupplierC implementerar.

```
public interface Supplier {
   public void doIt();
}

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

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

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

public static void chooseSupplier(Supplier obj) {
   obj.doIt();
}
```

Alternativt kan man införa en abstrakt klass som klasserna SupplierA, SupplierB och SupplierC utökar.

17.

Förvillkoret amount > 250 i klassen CreditCardPayment är starkare än i dess superklass Payment.

18.

Specifikation a) är undermålig och särklassigt sämst. Värdet 0 som returneras om fältet är **null** eller är tomt kan ju även vara ett giltigt värde för ett fält som innehåller element.

Specifikationerna b) och c) är båda fullvärdiga och vilken som är att föredra beror på om man utgår från klientens eller serverns perspektiv. Från klientens perspektiv är specifikation b) att föredra.