Chapter 4: Writing Classes/ Att skriva egna klasser.

I dessa uppgifter kommer du att lära dig om hur man definierar egna objekt genom att skriva klasser. Detta är grunden för att förstå objekt orienterad programmering. Det är något mer abstract än tidigare programmering som ni har gjort så det är mycket viktigt att du läser i boken eller online innan du börjar med uppgifterna.

Klasser beskriver typer av objekt. När vi skapar objekt av en klass, säger vi att vi ”instansierar” ett objekt av en klass.

Skapa objekt, görs på samma sätt som du skapar objekt från java bibliotek klasser.

Datatyp (klassen) + variabelnamn + new + konstruerare, se nedan.

```
Random generator = new Random() ;
```

```
Account konto1 = new Account(??);
```

Observera tekniken som används, den är alltid desamma.

Del 1: Övningar

Om du aldrig har programmerat är det obligatoriskt att du läser kapitlen 4 och 6 i boken samt skriver 1-2 programexempel från boken eller i online litteratur. Du måste ha förstått någorlunda klasser / objekt innan du börjar med uppgifterna.

Del 2: Labb

Uppgift 1 :

1. Klassen Account, se nedan, innehåller en definition för en klass som representerar ett bankkonto. Spara den i din mapp och undersök den väl. Vilka metoder finns där och vad de gör. Modifiera sedan klassen på följande sätt:

a) Fyll i koden för metoden toString(), vilket bör returnera en sträng som innehåller namn, kontonummer och balans för kontot, alltså objektets innehåll.

b) Fyll i koden för metoden chargeFee(), vilket bör dra av en serviceavgift från kontot (alltså modifierar balance med fee)

c) Ändra chargeFee() så att istället för att den returnerar *void* skall den returnera balance.

d) Fyll i koden för metoden changeName(String newName) som tar en sträng som parameter och ändrar namnet till den nya.

```
//*******************************************************
// Account.java
//
// A bank account class with methods to deposit to, withdraw from,
// change the name on, charge a fee to, and print a summary of the account.
//*******************************************************

public class Account
{
  private double balance;
  private String name;
  private long acctNum;

  //-----------------------------------------------
  //Constructor -- initializes balance, owner, and account number
  //-----------------------------------------------


public Account(double initBal, String owner, long number){
    balance = initBal;
    name = owner;
    acctNum = number;
  }

  //-----------------------------------------------
  // Checks to see if balance is sufficient for withdrawal.
  // If so, decrements balance by amount; if not, prints message.
  //-----------------------------------------------
  public void withdraw(double amount){
    if (balance >= amount)
       balance -= amount;
    else
       System.out.println("Insufficient funds");
  }
```

```
//-----------------------------------------------
  // Adds deposit amount to balance.
  //-----------------------------------------------
  public void deposit(double amount){
    balance += amount;
  }

  //-----------------------------------------------
  // Returns balance.
  //-----------------------------------------------
  public double getBalance() {
    return balance;
  }


//-----------------------------------------------
  // Returns a string containing the name, account number, and balance.
  //-----------------------------------------------
  public String toString(){

 }

  //-----------------------------------------------
  // Deducts $10 service fee
  //-----------------------------------------------
  public void chargeFee() {
  }

  //-----------------------------------------------
  // Changes the name on the account
  //-----------------------------------------------
  public void changeName(String ){

  }
  }
```

2. Klassen ManageAccounts (se nedan) innehåller ett skalprogram som använder Account klassen ovan. Spara den i samma mapp, och slutför det enligt kommentarerna.

```
public class ManageAccounts
{
    public static void main(String[] args)
    {
   Account acct1, acct2;

   //create account1 for Sally with $1000
   acct1 = new Account(1000, "Sally", 1111);
```

```
    //create account2 for Joe with $500

    //deposit $100 to Joe's account

    //print Joe's new balance (use getBalance())

    //withdraw $50 from Sally's account

    //print Sally's new balance (use getBalance())

    //charge fees to both accounts

    //change the name on Joe's account to Joseph

    //print summary for both accounts

     }
}
```

3. Ändra klassen Account så att metoderna inte tillåter insättning av negativt belopp eller uttag av större belopp än saldot. Metoderna skall returnera lämpligt meddelande om dessa problem uppstår. Testa metoderna i programmet ManageAccounts.java.

4. Om man vill ha uppsikt över hur många objekt som skapas utifrån en klass (i våra fall hur många konton) skall man lägga i klassen en variabel, till exempel, (**`static int nr`**). Variabeln uppdateras (increment) i konstruktor.

Lägg i klassen Account metoden nedan. Skriv färdig metoden.

```
public static int getNrofAccounts()
  {
    //metoden skall returnera variabeln nr.
  }
```

Testa metoden genom att lägga till 2 extra Account- objekt i programmet ManageAccounts.java. samt anropa nu metoden **`getNrofAccounts().`** Stämmer det?

Uppgift 2:

Program består oftast av flera klasser. Ni ska implementera ett enkelt spel som vi kallar ”**24Spelet** ” där reglerna är mycket enkla. Alla spelare kastar 4 tärningar på en gång. Spelaren som får mest poäng vinner.
Programmet behöver inget grafiskt gränssnitt. Programmet består av två klasser, se nedan.
a)Klassen Spelare där instansvariablerna, konstrueraren och metoden getNamn() är klara.
Din första uppgift är att implementera metoderna ***spela(), getPoäng()*** *och* ***nollStäll()*** enligt beskrivningen som du finner framför metoden.

```
public class Spelare
{
private String namn;  // instansvariabler
private int poäng;
/*konstrueraren */
public Spelare( String nm)
{
namn=nm;
poäng= 0;
}
/* returnerar namn */
public String getNamn( )
{
return namn;
}
/* metoden ska simulera tärningskastet genom att slumpa 4 st
värden mellan 1 och 6. Summan av alla dessa värden skall tilldelas
till klassen variabel poäng*/
public void spela( )
{
// skriv din kod här
}
/* metoden skall returnera antalet poäng */
public int getPoäng()
{
// skriv din kod här
}
/* metoden sätter variabel poäng till noll */
public void nollStäll ()
{
// skriv din kod här
}
}
```

b) **Din andra uppgift** är att skriva själva spelet alltså klassen SpelProgram.
Där skall du använda klassen Spelare och skapa två Spelare-objekt.
Låt de spela tre omgångar och utse vinnaren genom att skriva ut ett meddelande som innehåller texten ” Vinnaren är ” och vinnarens namn förstås. Vinnare är den spelare som har vunnit två av de tre omgångar spel.

Glöm inte att nollställa Spelare-objekten inför varje omgång och spara antalet vinster för varje spelare.

```
public class SpelProgram
{
public static void main (String [] arg)
{
// skriv din kod här
}
}
```

OBS! Spelarens namn skall läsas in från tangentbordet.
Slumpa tal med hjälp av ett Random-objekt. Konstrueraren från klassen Random och metoden nextInt() som ska användas för att slumpa tal är definierade som nedan:

Uppgift 3:

Bank Dispenser

Nu skall vi göra ett enkelt bank program. Det mesta är färdig implementerad.

1. Lägg i klassen Account en ny metod *public long getAcctNum ()*{ // body }. Metoden skall returnera acctNum.

2. I programmet nedan finns ett enkelt bank program. Banken har bara ett Account- objekt och en meny för att göra bank operationer.

Spara klassen nedan (i samma mapp med klassen Account), kompilera och kör programmet nedan. Försök förstå hur programmet fungerar. Case 1 and 3 är implementerad. Du skall skriva färdig case 2.

```
import java.util.Scanner;

public class BankDispenser
{
      static Scanner scan = new Scanner(System.in);
      static Account account1=new Account( 1000, "Svensson Dan", 750123);


public static void main(String[] args)
    {
      printMenu();
      int choice = scan.nextInt();



    while (choice != 0)
      {
      dispatch(choice);
      printMenu();
      choice = scan.nextInt();
       }

}

public static void dispatch(int choice)
    {
     switch(choice)
       {
      case 0:
           System.out.println("Bye!");

           System.exit(0);
      break;
      case 1:
```

```
            System.out.println("Insert your account number");
            int nr = scan.nextInt();


// the program checks if the account number i corect. If yes the
//programme ask for the deposit amount and execute it
      if( account1.getAcctNum()==nr)

      {
            System.out.println("Insert the amount to deposit");
            int amount=scan.nextInt();
            account1.deposit(amount);
      }
      else
            System.out.println( " Wrong account number");
      break;
      case 2:
            // your code for withdraw
      break;

       case 3:

            System.out.println("Insert your account number");
            int nr = scan.nextInt();
            if( account1.getAcctNum()==nr)
            System.out.println( account1.toString());
       break;
       default:
            System.out.println("Sorry, invalid choice");
       }
    }



    //----------------------------
    // Print the user's choices
    //----------------------------


   public static void printMenu()
    {
   System.out.println("\n  Welcome to MY BANK ");
   System.out.println("   ====");
   System.out.println("0: Quit");
   System.out.println("1: Deposit" );
   System.out.println("2: Withdraw");
   System.out.println("3: Show account information");

   System.out.print("\nEnter your choice: ");
    }
}
```

2. Som du ser, finns det bara ett konto objekt i programmet. Nu skall du skapa ett konto till i ”din” BankDispenser, t.ex.

```
static Account account2=new Account( 2000, "Anderson Amilia",
881203);
```

Ändra i programmet så att du kan göra konto operationer på account2 också.

Hur skulle programmet ändras om du skulle ha 100 konto? Det behövs ett annat sätt att strukturera programmet. Tänk hur?

.