# Vad är Objektorienterad programmering (OOP) ?

- Ett tillvägagångssätt att i kod modellera verkligheten
- Ex: Person, hus, cirkel, osv.
- Skiljer sig från procedurell programmering (t.ex. C, Pascal, Basic)
- Kapslar in (eng. encapsulate) data och beteende i Objekt

# Fördelar med objektorienterad programmering

- Koden lätt att återanvända – spar utvecklingstid
- Gömmer data – ger säkrare kod
- Lätt att underhålla – ändringar kan ske på ett ställe
- Ger bra struktur på koden – lättare att förstå

# Klasser

- Klass – en ritning över hur Objekt av klassen kommer att se ut
- Grupperar och definierar funktionalitet och egenskaper
- Java – använder nyckelordet **class** för att definiera en klass

```
public class SimpleClass
{

}
```

Skapar en klass med namnet **SimpleClass**

# Objekt

- En realisering av en klass

  Exempel på objektskapande (instansiering) :

  **SimpleClass myObejct = new SimpleClass();**

  Förklaring:
  **myObject** är ett objekt av klassen **SimpleClass**

  *Notera:*
  Objekt och Instans – två ord för samma sak

# Objekt och Objekt-referensvariabler

- Objekt-referensvariabel – en ”fjärrkontroll” för att komma åt ett objekt
- För att skapa ett objekt av en klass används operatorn **new** följt av klassnamnet:

  **new SimpleClass();**

  - Skapar ett objekt av typen SimpleClass i minnet
- För att skapa en objekt-referensvariabel skriver man **klassnamnet** följt av ett **godtyckligt namn**:

  **SimpleClass myObject;**

  - Skapar en objekt-referensvariabel (fjärrkontroll) till ett objekt av klasstypen SimpleClass
- Går att göra allt på en rad:

  **SimpleClass myObject = new SimpleClass();**

# Primitiva variabler och Objekt-referensvariabler – vad är skillnaden?

- Variabler av primitiver (ex. int, float, double) innehåller värden

  **int i = 0;**
  - i innehåller *värdet* 0

- Variabler av referenstyp innehåller ett *referensvärde* till ett objekt

  **SimpleClass myObject = new SimpleClass();**
  - Här innehåller myObject *referensen* till ett objekt av typen SimpleClass inte själva objektet

# *null* – är ingenting

- När man skapar en objektreferens-variabel utan att tilldela den någon referens till ett objekt innehåller den värdet **null**

  **SimpleClass myObject;**
  - Värdet i myObject är ***null*** eftersom den inte har tilldelats någon objektreferens

  **myObject = new SimpleClass();**
  - Nu innehåller objekt-referensvariabeln myObject en referens till ett SimpleClass-objekt

# Metoder – vad objektet av klassen kan göra

- En klass använder sig av *metoder* för att definiera den funktionalitet som klassen har

```
public class SimpleClass
{
        public int getNumber()
        {
                return 1;
        }
}
```

- När ett objekt av klassen SimpleClass skapas kommer den att ha metoden **getNumber()**

```
SimpleClass myObject = new SimpleClass();
int number = myObject.getNumber();
```

# Instansvariabler – håller data för objektet

- Håller värden som är unika för ett visst objekt

Kallas för *datamedlem*

```
public class SimpleClass
{
        int number = 1;

        public int getNumber()
        {
                return number;
        }

        public void setNumber(int newNumber)
        {
                number = newNumber;
        }
}
```

- Objekt av klassen SimpleClass kommer att ha *instansvariabeln* **number**.
  Manipulering av **number** påverkar **bara** aktuellt objekts instansvariabel

# Klassvariabler – håller data för klassen

- Klassvariabler – har samma värde hos alla objekt av klassen
- Kallas också för datamedlem

```
public class SimpleClass
{
   static int antotherNumber = 2;
  int number = 1;

    public int getAnotherNumber()
    {
         return anotherNumber;
    }

    public void setAnotherNumber(int newAnotherNumber)
    {
         anotherNumber = newAnotherNumber;
    }
}
```

- Objekt av klassen SimpelClass kommer att ha *klassvariabeln* **anotherNumber**. Manipulering av den klassvariabeln kommer att gälla för **alla** objekt av klassen

- **static** framför en datamedlem gör den till en klassvariabel

# Åtkomst av instansvariabler

- Ett objekt av klassen **måste** existera
- Objektreferns-variabeln följt av en punkt ger åtkomst till instanvariablerna

```
SimpleClass myObject = new SimpleClass();
int aNumber = myObject.number;
```

- **aNumber** kommer nu att hålla värdet som finns i myObject's datamedlem **number**

# Åtkomst av klassvariabler

- Objekt av klassen behöver inte existera för att komma åt klassvariabler

  **int aNumber = SimpleClass.anotherNumber;**

  - **aNumber** kommer nu att hålla värdet som finns i SimpleClass datamedlem **anotherNumber**

- Går även att komma åt genom objekt av klassen (rekommenderas inte)

  SimpleClass myObject = new SimpleClass();
  **int aNumber = myObject.anotherNumber;**

  - **aNumber** kommer nu att hålla värdet som finns i myObject's datamedlem **anotherNumber**

# Konstruktor

- Används för att initiera ett objekts **instansvariabler** när objektet skapas
- Har alltid samma namn som klassen
- Om klassen saknar konstruktor ges en ”default” konstruktor som inte gör något alls

```
public class SimpleClass
{
    int number;

    public SimpleClass(int aNumber)
    {
        number = aNumber;
    }
}
```

Skapa ett objekt av SimpleClass:

SimpleClass myObject = new SimpleClass(10);

- myObejct’s instansvariabel **number** kommer att ha **aNumber**’s värde dvs. 10

# Överlagring av konstruktorer

- En klass kan ha flera konstruktorer – kallas **överlagring**

```
public class SimpleClass
{
        int number;
        String message;

        public SimpleClass(int aNumber, String aMessage)
        {
                number = aNumber;
                message = aMessage;
        }

        public SimpleClass(int aNumber)
        {
                number = aNumber;
        }
}
```

Skapa ett objekt av SimpleClass:

SimpleClass myObject = new SimpleClass(10, ”OOP – Rules!”);

- myObejcts instansvariabel **number** kommer att ha värdet 10 och **message** kommer att vara “OOP – Rules!”

# Åtkomstattribut fort.

- Dålig idé med direktåtkomst av instansvariabler – använd åtkomstattributet private tillsammans med en åtkomstmetod

```
public class SimpleClass
{
        private int number;

        public int getNumber()
        {
            return number;
        }
}
```

- Nu går det bara att komma åt **number** genom att anropa **myObject.getNumber();**

- Ogiltigt att anropa **myObject.number;**
  **number** är inte *åtkomlig* utanför klassen

- Detta kallas *datagömning (Data Hiding)*

# Åtkomstattribut

- Används bl.a. för att dölja en klass metoder och datamedlemmar
- public ,protected, default, private
- **public** – ger alla åtkomst
- **protected** – ger bara subklasser åtkomst och klasser i samma paket
- **default** – ger bara klasser i samma paket åtkomst
  Notera: default skrivs aldrig – utelämnande av åtkomstattribut ger default
- **private** – ger bara klassen åtkomst

# Objekt i klasser

- En klass kan innehålla objekt-refernsvariabler som datamedlemmar

```
public class SimpleClass
{

}

public class ComplexClass
{
        private SimpleClass myObject = new SimpleClass();
}
```

- ComplexClass har en datamedlem av typen SimpleClass som heter myObject

# Skicka objekt som argument

- Objekt kan skickas som argument till en metod

```
public void storeObject(SimpleClass myObject)
{
    …
}
```

- Metod som tar ett objekt av klasstypen SimpleClass som argument

Notera: Det är **inte** objektet som skickas utan bara en kopia av objekt-referensvariabeln

# Inre klasser

- Är en klass som deklareras *i* en annan klass
- Blir en datamedlem av den yttre klassen
- Kan vara anonym - behöver inte ha något deklarerat namn
- Kommer åt alla datamedlemmar hos den yttre klassen
- Kan vara private, default, protected eller public - precis som andra datamedlemmar
- Går bara att komma åt om ett objekt av yttre klassen finns (gäller inte inre static-klasser)
- Kan vara static

# Definiera en inre klass

- Skapar en inre klass som blir en datamedlem av den yttre klassen

```
//Yttre klass
public class Outer
{
        //Inre klass
        public class Inner
        {
        }
}

//Instansiera ett objekt av klassen Inner
Inner inner = new Outer().new Inner();

--- eller ---

Outer outer = new Outer();
Inner inner = outer.new Inner();
```

# Att tänka på

- Om den inre klassen deklareras som static kommer den **INTE** åt några datamedlemmar eller metoder i den yttre klassen

```
//Yttre klass
public class Outer
{
        public void doStuff()
        {
                ...
        }

        //Inre klass
        public class static Inner
        {
                doStuff(); ← Fungerar inte. Det finns inget objekt av Outer
        }
}
```

# Övning – skapa klasser

- Börja med att plocka ut minst 5 klasskandidater från något visst område (Bank, Mataffär, Bil, osv)
- Skapa enkla klasser från dessa
- Lägg till datamedlemmar
- Ge klasserna en eller flera konstruktorer
- Lägg sedan till metoder till klasserna
- Ge metoder och datamedlemmar rätt åtkomstattribut (public, default, protected, private)
- Testa dina klasser från en klass som har en main-metod
- Ge någon klass en *static* variabel och metod
- Överlagra någon klass konstruktor
- Testa klasserna igen