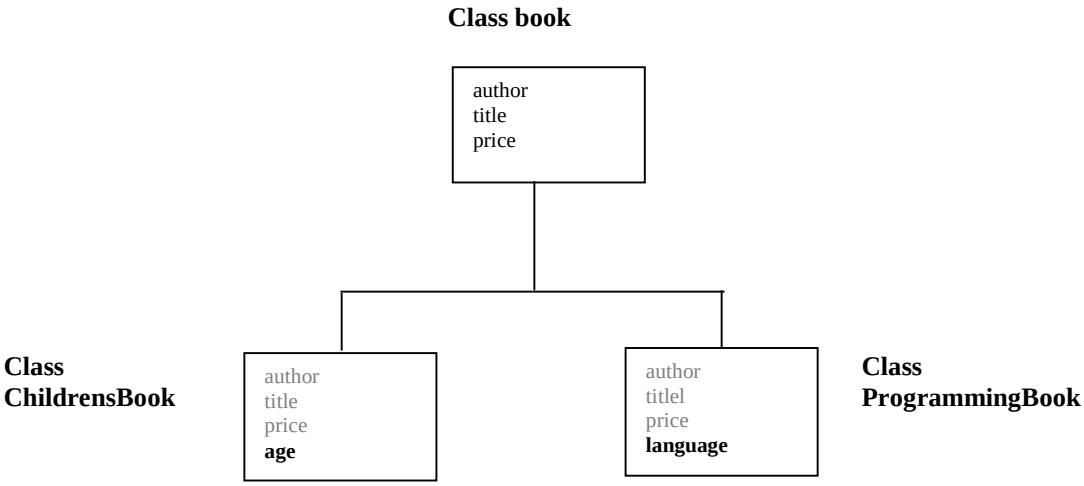
Objektorientering del 3 Java – Arv

När man gör en ny klass behöver man inte alltid börja från början. Genom arv kan man ärva funktionalitet som finns i en existerande klass.

Den klass vi ärver från kallar vi superklass och klassen som ärver för subklass. Det finns dock andra benämningar.



Eftersom barnbok och programmeringsbok ärver från bok så har de samma funktionalitet. Man kan säga att en barnbok eller programmeringsbok också **är** en bok.

Man kan sedan lägga till funktionalitet i klasserna som ärver, som ålder för barnbok eller språk för programmeringsbok. På samma sätt som du ärver egenskaper från dina föräldrar men också har egna egenskaper.

För att ärva skriver man skriver man extends. Extends betyder utöka och det är det vi gör. Vi utökar basklassen eller som det kallas i Java superklassen.

Så här: **class ChildrensBook extends book**

Här är klassen som vi ska ärva från:

```
public class Book
{

                public String title = "N N";
                public String author = "N N";
                public int price = 0;


                public Book()
                {
                }


                //Construktor
                public Bok(String title, String author, int price)
                {
                    setTitle(title);
                    setAuthor(author);
                    setPrice(price);
                }

                //Setters
                public void setTitle(String title)
                {
                      this.title = title;
                }

                public void setAuthor(String author)
                {
                      this.author = author;
                }

                public void setPrice(int price)
                {
                      this.price = price;

                }


                //Getters
                public int getPrice()
                {
                     return price;
                }
                public String getTitle()
                {
                     return title;
                }
                public String getAuthor()
                {
                     return author;
                }


                //Utskriftsmetod
                public void skrivUt()
                {
                     System.out.println("");
                     System.out.println("Titel: " + title);
                     System.out.println("Författare: " + author);
                     System.out.println("Pris: " + price);
                }


}//End superclass
```

Här är en klass som ärver från book:

```
public class programmingBook extends Book {

		protected String language = "";

		//Setter
		public void setLanguage(String language) {
			this.language = language;
		}

		//Getter
		public String getLanguage() {
			return language;
		}

}
```

För att skapa ett objekt av vår nya klass BarnBok skriver vi:

programmingBook **programmingJava** = new programmingBook ();

programmingBook är en book

Eftersom vi ärver från superklassen (book) behöver vi inte deklarera de variabler eller setters och getters som redan finns i vår superklass.

Det innebär att vi kan skriva så här för att sätta värden i vårt nya objekt:

```
programmingJava.setTitle("C++");
programmingJava.setAuthor("Skansholm");
programmingJava.setPrice(500);
```

Vi använder setter som redan finns i superklassen book.

Vi kan också skriva ut dessa värden genom att använda **skrivUt()**-metoden från vår superklass.

```
programmingJava.skrivUt();
```

Nyckelordet super

Ett problem är dock att vi skriver ju inte ut den extra variabeln **language.** Vi kan inte heller använda konstruktorn från vår superklass för den använder inte den extra variabeln heller.

Kan vi då inte dra nytta av vårt arv ändå på något sätt? Jo, det kan vi! Med hjälp av nyckelordet **super**.

Vi kan tala om att vi vill skriva ut allt som superklassen skriver ut, plus lite till, så här:

```
public void skrivUt()
{
	super.skrivUt();
	System.out.println("Programmeringsspråk: " + language);
}
```

`super.skrivUt();` innebär att vi använder skrivUt()-metoden från vår superklass. Sedan lägger vi till det vi vill skriva ut i subklassens skrivUt()-metod.

Samma sak med konstruktorn. Vi kan använda superklassens konstruktor på samma sätt. Här är hela subklassen med nyckelordet super i konstruktor och skrivUt()-metoden.

```
public class programmingBook extends Book {

		protected String language = "";

		public programmingBook() {
		}

		public programmingBook(String title, String author, int price,  String language) {
			super(title, author, price);
			setLanguage(language);
		}

		// Setter
		public void setLanguage(String language) {
			this.language = language;
		}

		// Hämtar attribut
		public String Language() {
			return language;
		}

		@override
		public void skrivUt()
		{
			super.skrivUt();
			System.out.println("Programmeringsspråk: " + language);
		}

}
```

Uppgifter

1. Skapa en ny consoleapplication och döp den till Arv. Skapa en klass **Bok** och tre klasser som ärver från klassen Bok. Klassen bok ska ha fälten titel (string), författare (string), och pris (int) som tidigare. Den första barnklassen ska heta **barnbok** och extra information är **ålder** (age) som är en **int**. Den andra ska heta **FaktaBok** och extra information ska vara **ämne** (subject) som är en **string**.

Varje klass ska ha privata fält och egenskaper som sätter och hämtar värdet. Bokklassen ska ha en konstruktor som tar tre parametrar och de övriga en konstruktor som tar fyra parametrar.

Alla klasser ska ha en metod som heter **skrivUt**. Barnklasserna ska skriva ut den information som finns i basklassen och dessutom sin extra information.

2. En subklass kan även ärva från en annan subklass. Gör en klass som ärver från programmingBook. Kalla den t.ex. programmingWeb och lägg till en variabel som heter t.ex. side och sätts till serverside eller klientside.

3. När du tillverkar en bok så tillverkar du inte bara en bok vilken som helst. Du går inte till bokhandlaren och säger att du vill köpa en bok bara. Då frågar de vilken bok? Bok är bara vilken sorts objekt det är.

Därför är det inte så stor mening att göra objekt av klassen bok. Man ärver bara ifrån den. För att hindra att man gör objekt av klassen bok sätter man till nyckelordet abstract. Då vet kompilatorn att du bara kan ärva från klassen, inte skapa nya objekt.

Prova att sätta till ordet abstract så här: `public abstract class Book`

Alltså mellan **public** och **class** i klassen Book.

Prova sedan att skapa ett Book-objekt

Book **test** = new Book();

Vad händer?

Lite svårare

4. Vi antar att priset är utan moms. Skriv en metod som räknar ut priset inklusive moms och skriver ut det. Vi kan anta att momsen är 25%. Metoden skrivs i superklassen och ärvs i de andra klasserna.

Skriv metoden och testa att skriva ut priset på en programmeringsbok med moms.

5. När man har samma namn på en metod som t.ex. SkrivUt(); men implementeringen, innehållet är annorlunda. Detta kallas **override**. Man kan säga att man skriver över den gamla betydelsen med en ny.Det var det vi gjorde i övningen.

Man kan också sätta s.k. metadata (annotation) på en metod när man gör en override med **@Override**. Det kan man göra för att vara säker på att det fungerar som tänkt.

Om man ändrar inparametrarna så att metodhuvudet och metodanropet är annorlunda men man har samma namn kallas det **overload**.

Anropet till skrivUt() skulle då kunna se ut så här: skrivUt(25); Vi har alltså en inparameter istället för ingen.

Skriv en metod i superklassen som tar en parameter. Typen ska vara double och vara den moms som ska läggas på boken. Använd metoden på subklassen.

Om momsen är 25 ska alltså metodanropet se ut som i exemplet **skrivUt(25);**

Formel för att räkna ut det nya priset kan vara:

Om moms är 25 %:

((25 / 100) + 1) * pris innan moms;