**IS-110, modul 10 Arild Høyland Mars 2015**

Det er nyttig og viktig å kunne omdefinere metoder ved bruk av arv i Java. Dette for å unngå duplisering av kode og for å kunne benytte metoder som finnes fra før. Dette gjelder både arv i egne klasser, men også metoder som arves fra f. eks. object-klassen (equals og toString er to av de mest brukte). Ved å benytte seg av både arv og omdefinering av metoder er det lett å lage metoder som kan passe til flere subklasser, og som kan tilpasses ytterligere i subklassen, dersom dette trengs.

I mitt prosjekt har jeg laget en metode (print) for å skrive ut informasjon. Denne defineres i kjøretøy-klassen og skriver her ut informasjonen alle kjøretøy har felles. I hver av subklassene kalles denne metoden først, deretter skriver metoden informasjonen som er unik for subklassen. I bil-klassen har jeg også omdefinert toString-metoden som arves fra object. Denne bruker jeg ikke, og det ville nok vært en bedre løsning å implementere denne i en egen print-metode som skriver informasjonen ut til terminalen, i stedet for kun å returnere den som String.

Ellers er det viktig å holde styr på statiske og dynamiske typer når man arbeider med arv. I kallet Kjøretøy kjøretøy1 = new Bil() er Kjøretøy den statiske typen, dvs. klassen som kan lagres i variabelen. Kjøretøy1 er en variabel som kan lagre objekter av klassen Kjøretøy, men ettersom klassen Kjøretøy har to subklasser (Bil og Tog), kan kjøretøy1 både lagre objekter av Bil eller Tog. Dette er den dynamiske typen. Det er viktig å tenke over at dette utsagnet ikke nødvendigvis kan reverseres (Bil bil1 = new Kjøretøy() er ikke akseptabelt ettersom et kjøretøy kan være både Bil og Tog, hvor sistnevnte ikke vil passe inn i bil1-variabelen).