IN1010

UKE 3 - GRUPPE 3

MIAU@IFI.UIO.NO

Plan for dagen

- Implisitt og eksplisitt konvertering
- Polymorfi
- Polymorfe metoder
- Arv/komposisjon
- Overloading

Implisitt konvertering

- En peker har en egen type
  - De kan peke på et objekt som har som type en subklasse av den egen type

```
Gårdsdyr gårdsdyr = new Ku();
```

- Variabelen har kun tilgang til pekeren sin types funksjonalitet
- Kan bruke *instanceof* for å sjekke om objektet er en instans av en spesifikk klasse

Eksplisitt konvertering

- Subklasseobjekter som har blitt implisitt typekonvertert har kun tilgang til superklassens egenskaper
  - Kan få tilbake objekts funksjoner ved eksplitt konvertering

```
if (gårdsdyr instanceof Ku) {
    Ku ku = (Ku) gårdsdyr;
}
```

Polymorfi

- Handler om at metoder med samme navn, kan ha ulik oppførsel
- @Override
  - Gir en metode en ny implementasjon
- Når man kaller en overstyrt metode, vil den nederest implementasjonen av metoden brukes
- Overstyrer med identisk signatur, altså samme navn og parameter

Polymorfe metoder

```java
public class Gårdsdyr {
    String navn;

    public Gårdsdyr(String navn) {
        this.navn = navn;
    }

    public void typeDyr() {
        System.out.println(x: "Gårdsdyr");
    }
}
```

```java
public class Ku extends Gårdsdyr {

    public Ku(String navn) {
        super(navn);
    }

    @Override
    public void typeDyr() {
        System.out.println(x: "Ku");
    }
}
```

```java
public class Sau extends Gårdsdyr {

    public Sau(String navn) {
        super(navn);
    }

    @Override
    public void typeDyr() {
        System.out.println(x: "Sau");
    }

    public void klippUll() {
        System.out.println(x: "Klipper ull");
    }
}
```

```java
Gårdsdyr dyr = new Gårdsdyr(navn: "Mia");
Gårdsdyr ku = new Ku(navn: "Karoline");
Gårdsdyr sau = new Sau(navn: "Philip");

dyr.typeDyr();
ku.typeDyr();
sau.typeDyr();
sau.klippUll();
```

```
Gårdsdyr
Ku
Sau
```

```java
interface Egglegger {
    int antallEgg ();
}

class Hare {
    final String navn;
    Hare (String n) { navn = n; }
}

class Påskehare extends Hare implements Egglegger {
    int antall;
    Påskehare (String n, int ant) { super(n);  antall = ant; }

    @Override public int antallEgg () { return antall; }
}

class Høne implements Egglegger {
    @Override public int antallEgg () { return 1; }
}
```

```java
class BrukHare {
    public static void main (String[] args) {
        Hare skogshare = new Hare("Bjørk");
        Hare sesonghare = new Påskehare("Påsan", 999);
        Høne burhøne = new Høne();

        // Hvilke av disse 10 linjene er lovlige som neste linje?
        Påskehare h1 = sesonghare;
        Egglegger h2 = sesonghare;
        Egglegger h3 = burhøne;
        Egglegger h4 = (Egglegger)skogshare;
        Påskehare h5 = (Påskehare)sesonghare;
        Egglegger h6 = (Hare)sesonghare;
        Hare      h7 = sesonghare;
        Egglegger h8 = (Egglegger)sesonghare;
        Egglegger h9 = skogshare;
        Påskehare h10 = (Påskehare)skogshare;
    }
}
```

	OK	Feil
Påskehare h1 = sesonghare;	○	○ ✔
Egglegger h2 = sesonghare;	○	○ ✔
Egglegger h3 = burhøne;	○ ✔	○
Egglegger h4 = (Egglegger)skogshare;	○	○ ✔
Påskehare h5 = (Påskehare)sesonghare;	○ ✔	○
Egglegger h6 = (Hare)sesonghare;	○	○ ✔
Hare h7 = sesonghare;	○ ✔	○
Egglegger h8 = (Egglegger)sesonghare;	○ ✔	○
Egglegger h9 = skogshare;	○	○ ✔
Påskehare h10 = (Påskehare)skogshare;	○	○ ✔

Arv og komposisjon

- Arv
  - Arver egenskaper
  - Klasser og klassehierarki, extends

- Komposisjon
  - Består av/er en del av noe
  - Variabeler og metoder

```java
public class Ku extends Gårdsdyr {

    public Ku(String navn) {
        super(navn);
    }

    @Override
    public void typeDyr() {
        System.out.println(x: "Ku");
    }
}
```

```java
public class Gårdsdyr {

    String navn;
    int alder;
    String art;

}
```

Overloading

- Hvis man har to metoder med samme navn, men ulike parametre og/eller returtyper
- Override → Krever lik signatur
- Overloading → Krever ulik signatur
- Hvilke implemtasjon som kalles avhenger av verdiene man gir

```java
public int pris() {
    return pris;
}

public int pris(int økning) {
    pris += økning;
    return pris;
}
```

Oppgave

1. **Parkeringshus med motorsykler og biler | Parprogrammering**
   Dere skal nå lage et program som visualiserer et parkeringshus. Et parkeringshus har parkeringsplasser hvor både Motorsykler og Biler kan parkere. Motorsykler og Biler har en del til felles, bl.a. har de begge en sjåfør og et registreringsnummer. Mens en bil også skal kunne holde på alt fra 2 til 5 personer (spesifisert i det bilen lages), derav en av de må være sjåføren.
   - **a)** Lag alle klassene med instansvariabler. Inntil videre skal de ikke ha noen metoder. Det kan være lurt at Motorsykkel og Bil arver av samme klasse - hvorfor tror du det?
   - **b)** Parkeringshus skal ha to metoder: en for å sette inn en bil eller motorsykkel og en for å fjerne en bil eller motorsykkel på en gitt plass. Vi vil at metoden skal returnere bilen/motorsykkelen som står på plassen, og returnere null hvis det hverken finnes bil/motorsykkel på plassen. Når dere skal sette inn en bil eller motorsykkel, skal det gjøres på en ønsket plass. Lag unntakshåndtering til parkeringshuset ditt, for å håndtere tilfeller der man prøver å sette inn når det er fullt, eller på en ugyldig plass. **Hint:** ArrayIndexOutOfBoundsException.
   - **c)** Lag en metode skrivInformasjon(), som skriver ut all informasjon om hvert av kjøretøyene. Husk at motorsykler og biler har noe felles informasjon, siden begge er kjøretøy, men de har også noe særegen tilleggsinformasjon. Når du itererer gjennom kjøretøyene, skal all informasjon om hvert enkelt kjøretøy skrives til terminalen. Du kan ikke bruke instanceof i denne oppgaven, og du skal unngå å repetere kode unødig.