IN1010 Gruppetime – Uke 3

Gruppe 2

Repetisjon

Arv og subklasser

Extends

- Extends er nøkkelordet for at en klasse arver fra en annen
- En klasse som arves fra er superklassen, mens klassen som arver fra superklassen kalles en subklasse

Implisitt konvertering

- *Pekere* har sin egen type
- En peker med en type kan peke på objekter som har som type en subklasse av dens type

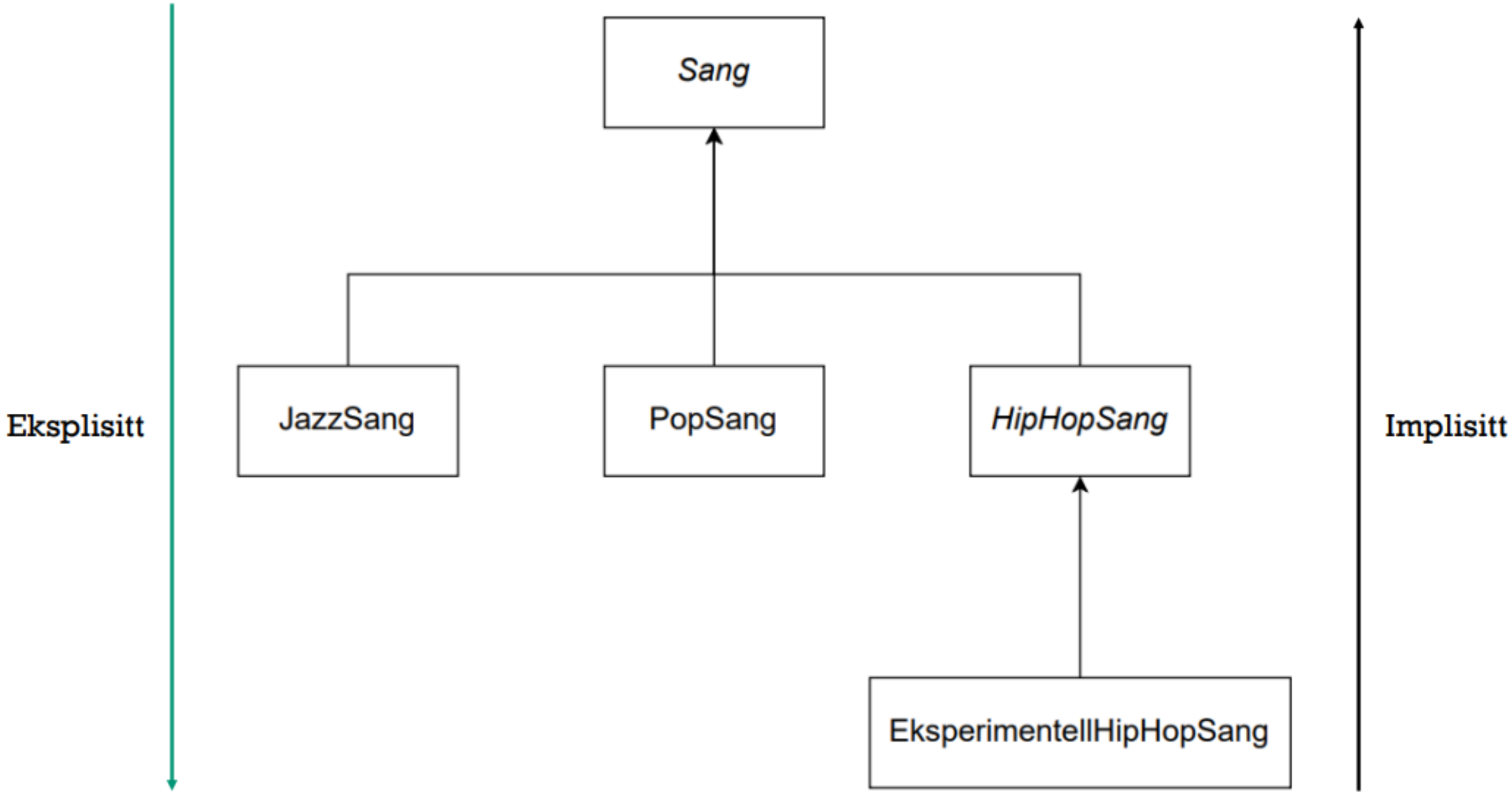
```
Sang sang = new PopSang("etc", "etc", 0, "etc", true);
```

- Dette kalles for *implisitt typekonvertering*
- Variablene har kun tilgang til superklassens funksjonalitet
  - *Objektet* endrer seg ikke
- Vi kan bruke *instanceof* for å sjekke om et objekt er en instans av en spesifikk klasse

Eksplisitt konvertering

- Subklasseobjekter som har blitt implisitt typekonvertert har kun tilgang til superklassens egenskaper, men de kan vi få tilbake med *eksplisitt typekonvertering*
- Vi kan bruke *instanceof* for å sikre for at vi ikke får kompilatorfeil

```
if (sang instanceof PopSang) {
    PopSang pop = (PopSang) sang;
}
```


Sang
JazzSang
PopSang
HipHopSang
EksperimentellHipHopSang
Eksplisitt
Implisitt

Ukas tema: Polymorfi

Polymorfi

- Polymorfi handler om at metoder med samme signatur (på tvers av superklasser og subklasser) har ulik oppførsel
- @Override
  - Overstyrer superklassens metodedefinisjon og gir ny implementasjon
- Når man kaller en overstyrt metode på et objekt, så vil den *nederste* implementasjonen av metoden brukes
- Man kan bruke superklassens implementasjon av metoden ved å kalle på *super.<metodenavn her>()*
- Husk at man overstyrer metoden med eksakt lik *signatur*, altså samme parametre og navn

```java
public class Kafe {
    public void viSelger() {
        System.out.print("Her selger vi kaffe");
    }
}
```

```java
public class KatteKafe extends Kafe {
    @Override
    public void viSelger() {
        super.viSelger();
        System.out.println(" og du kan klemme katter!");
        // Printer ut "Her selger vi kaffe og du kan klemme katter!"
    }
}
```

Overloading

- Hvis man har to metoder som heter det samme, men med forskjellige parametre og/eller returtype, kalles dette *overloading*
- Overriding krever lik signatur, Overloading krever forskjellig signatur
- Hvilken implementasjon som blir kalt på avhenger av verdiene man gir når man kaller på metoden

```java
public class Kafe {
    public void viSelger() {
        System.out.print("Her selger vi kaffe");
    }

    public void viSelger(String kafeNavn) {
        System.out.print("På kafe " + kafeNavn + " selger vi kaffe");
    }
}
```

```java
public class KatteKafe extends Kafe {
    @Override
    public void viSelger() {
        super.viSelger();
        System.out.println(" og du kan klemme katter!");
        // Printer ut "Her selger vi kaffe og du kan klemme katter!"
    }

    @Override
    public void viSelger(String kafeNavn) {
        super.viSelger(kafeNavn);
        System.out.println(" og du kan klemme katter!");
    }
}
```

toString() og equals()

- toString() og equals() er to metoder som defineres i klassen Object
  - Husk at Object er alle klassers superklasse
- toString() returnerer en streng og kalles på automatisk der nødvendig (print-setninger)
  - Lik __*str*__ i Python
- equals(Object o) tar inn et annet objekt og sjekker at disse er like
  - Lik __*eq*__ i Python
- Kodeeksempel

Oppgave 1

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.

Oppgave 2 (ekstra)

- Lag en egen oppgave der du bruker polymorfi. Du står fritt til å velge tema selv, men her har du noen tips: iPhone og forskjellige iPhone-modeller, kafé og ulike kafétyper, teknologibedrift med ulike avdelinger, diverse klesplagg