Uke 4

Arv og subklasser del 3

I dag

- Repetisjonsquiz!
- Interface
- Exceptions
- Interfacene List og Map

Quiz

Interface

Interface (=grensesnitt)

- Det er ikke mulig for en klasse å arve fra flere klasser
- MEN en klasse kan implementere flere interface
- Ofte har klasser felles egenskaper, men det er unaturlig å arve
  - Feks kan disse fly: et fly, et helikopter, en fugl og en sommerfugl
  - Men det gir ikke mening at alle disse skal arve fra samme klasse
- Kan se på interface som en «rolle» klassen har
  - Fly, helikopter, sommerfugl og fugl har rollen at de kan fly



Interface

```java
public interface KanFly {

    int hoyde();

    int fart();

}
```

- Skriver **interface** (istedenfor class)
- Inneholder kun abstrakte metoder
  - Kun metodesignaturen
  - Interface beskriver metoder som må implementeres
  - Metodene implementeres ikke i interfacet
- Inneholder ikke variabler

Implementere interface

- Bruker **implements** for å implementere interface i klassen.
- Bruker **@Override** når metodene fra interfacet implementeres
- Alle de abstrakte metodene i interfacet må implementeres av klassen
- Interfacet sier **ikke hvordan** metodene skal implementeres, **bare at** de skal implementeres

```java
public interface KanFly {

    int hoyde();

    int fart();

}
```

```java
public class Fugl implements KanFly{

    int hoyde;
    int fart;

    public Fugl(int hoyde, int fart){
        this.hoyde = hoyde;
        this.fart = fart;
    }

    @Override
    public int hoyde() {
       return hoyde;
    }

    @Override
    public int fart() {
        return fart;
    }

}
```

Interface som referansetype

- Kan bruke et interface som referansetype når man skal opprette et objekt.
  - Men kan ikke implementere et interface.
- Dermed kan man lagre objekter som implementerer samme interface i en

```
lic class Hovedprogram {

Run | Debug
public static void m
    KanFly fly = new KanFly();
```

Cannot instantiate the type KanFly Java(16777373)

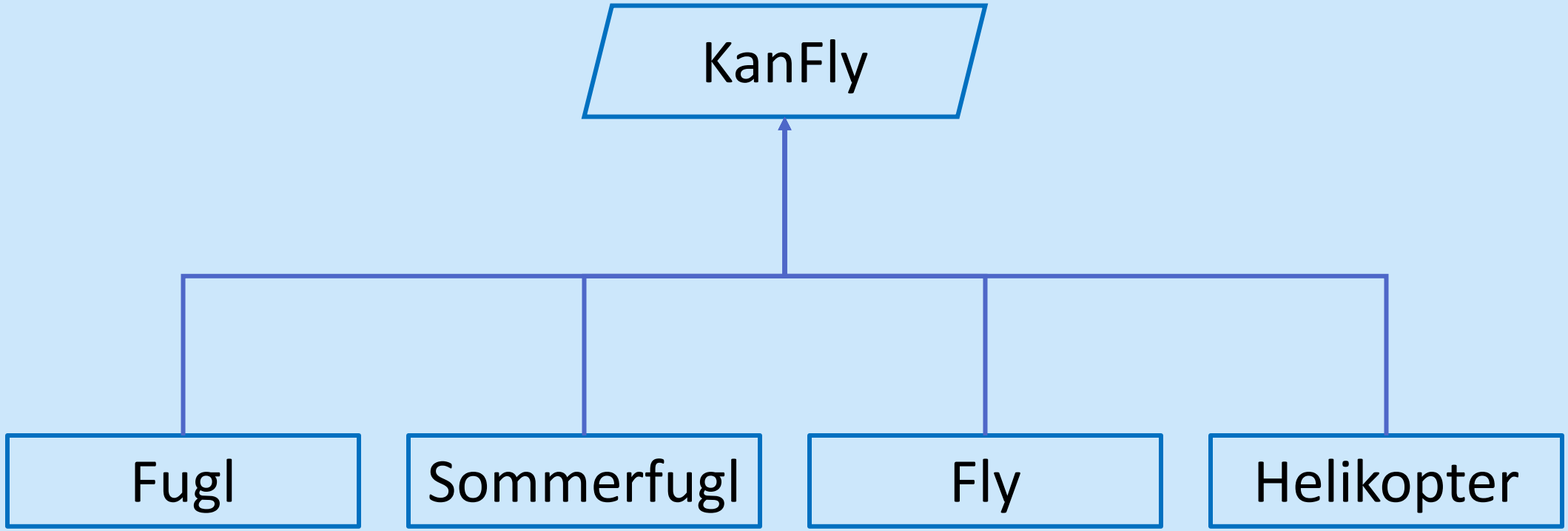
View Problem (⌥F8) No quick fixes available

```
KanFly fly = new Fly();
KanFly fugl = new Fugl(hoyde:80, fart:30);
KanFly sommerfugl = new Sommerfugl();
KanFly helikopter = new Helikopter();

KanFly[] flyvere = new KanFly[4];
flyvere[0] = fly;
flyvere[1] = fugl;
flyvere[2] = sommerfugl;
flyvere[3] = helikopter;
```

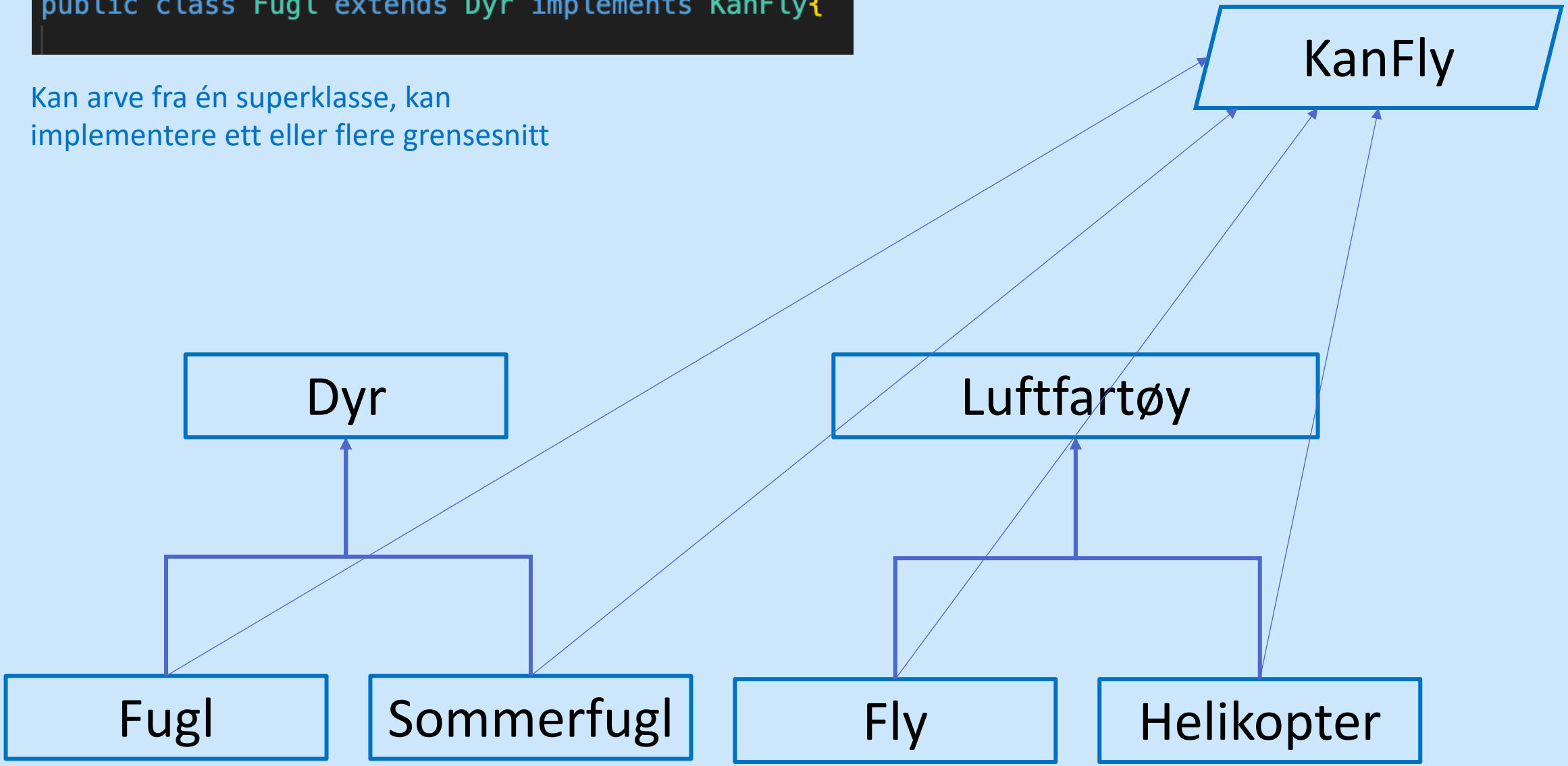
Flyvere = [fly, fugl, sommerfugl, helikopter]

Klassehierarki



```
public class Fugl extends Dyr implements KanFly{
```

Kan arve fra én superklasse, kan implementere ett eller flere grensesnitt



implements vs extends

- className **implements** interfaceName
- interfaceOne **extends** interfaceTwo
- classOne **extends** classTwo

Exceptions

Her er noen oppgaver: https://trix.ifi.uio.no/course/16?tags=exception

Kan implementere egne exceptions

- Arver fra Exception eller RuntimeException
- Exception → checked
- RuntimeException → unchecked
- Bruker konstruktøren til superklassen med super()

```java
public class Person {

    String navn;
    String fnr; //skal være 11 siffer
```

```java
public class FeilAntallSifferException extends RuntimeException{

    public FeilAntallSifferException () {
        super(message:"Feil antall siffer i fødelsnummere! Det skal være 11 siffer.");
    }
```

```java
public class FeilAntallSifferException extends Exception{

    public FeilAntallSifferException () {
        super(message:"Feil antall siffer i fødelsnummere! Det skal være 11 siffer.");
    }
```

Checked vs. unchecked

Exception → checked

- unntakene «sjekkes» ved kompilering. Hvis det utløses unntak må man håndtere unntaket (med try-catch) eller bruke nøkkelordet "throws".
- Eks:
  - FileNotFoundException
  - IOException
  - ClassNotFoundException

RuntimeException → unchecked

- ikke er nødvendig å håndtere eller bruke "throws"-klausul.
- Eks:
  - NullPointerException
  - IndexOutOfBoundsException
  - IllegalArgumentException

Extends RuntimeException (unchecked)

```java
public class FeilAntallSifferException extends RuntimeException{

    public FeilAntallSifferException () {
        super(message:"Feil antall siffer i fødelsnummere! Det skal være 11 siffer.");
    }
```

```java
public Person(String navn, String fnr){
    this.navn = navn;
    if (fnr.length() != 11) {
        throw new FeilAntallSifferException();
    }
    this.fnr = fnr;
}
```

Extends exception (checked) - throws

```java
public class FeilAntallSifferException extends Exception{

    public FeilAntallSifferException () {
        super(message:"Feil antall siffer i fødelsnummere! Det skal være 11 siffer.");
    }
```

```java
public Person(String navn, String fnr) throws FeilAntallSifferException{
    this.navn = navn;
    if (fnr.length() != 11) {
        throw new FeilAntallSifferException();
    }
    this.fnr = fnr;
}
```

Extends exception (checked) – try-catch

```java
public class FeilAntallSifferException extends Exception{

    public FeilAntallSifferException () {
        super(message:"Feil antall siffer i fødelsnummere! Det skal være 11 siffer.");
    }
```

List og Map

- Map og List er interfacer
- Kan dermed ikke implementeres
- Men kan brukes som referansetype

```
Map<String, String> hashmap = new HashMap<>();
HashMap<String, String> hashMap2 = new HashMap<>();
//går ikke:
//Map<String, String> hashmap3 = new Map<>();

List<String> list = new ArrayList<>();
ArrayList<String> arraylist = new ArrayList<>();
//går ikke:
//List<String> list2 = new List<>();
```

Jobbe med oppgaver

- ukesoppgaver
- innlevering 1