UKE 4

Arv og subklasser 3

agneshb@uio.no

I DAG

Interface

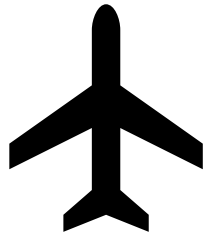
Interfacene List og Map

Repetisjonsquiz!

Exceptions

INTERFACE

INTERFACE (=GRENSESNITT)



Det er ikke mulig for en klasse å arve fra flere klasser

MEN kan implementere flere interface



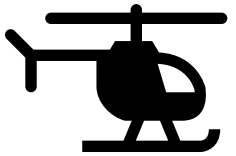
Ofte har klasser **felles egenskaper**, men **arv er unaturlig**

- 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

Skriver **interface** (istedenfor class)

Inneholder kun abstrakte metoder

- Kun metodesignaturen
- Interface beskriver metoder som må implementeres
- Metodene implementeres ikke i interfacet

Inneholder ikke variabler

```
public interface KanFly {

    int hoyde();

    int fart();

}
```

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.

```
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

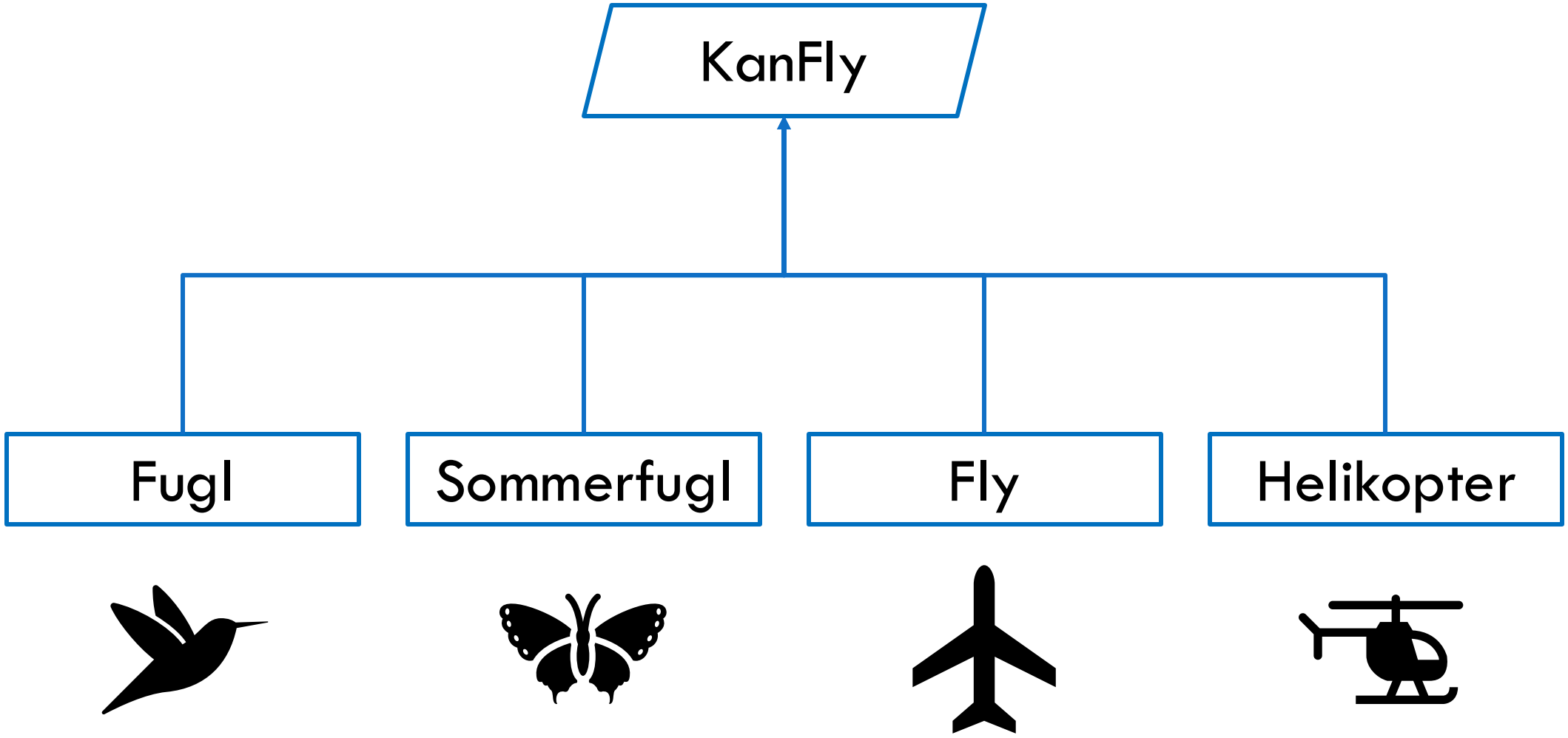
Dermed kan man lagre objekter som implementerer samme interface i en liste.

```
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;
```

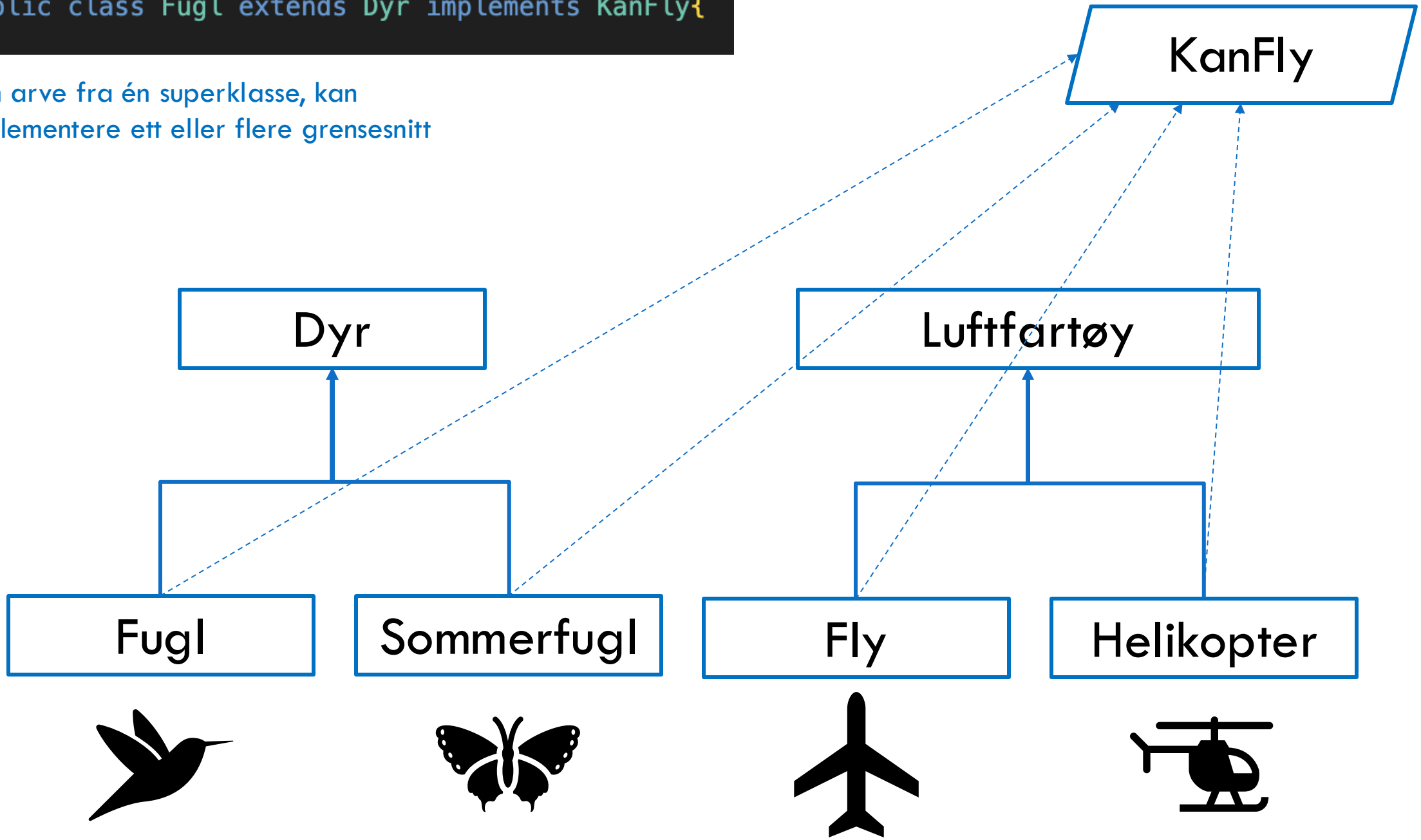
Flyvere = [fly, fugl, sommerfugl, helikopter]

KLASSEHIERARKI



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

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



IMPLEMENTS VS EXTENDS

klasse1 **implements** interface1

interface1 **extends** interface2

klasse1 **extends** klasse0

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<>();
```

QUIZ

EXCEPTIONS

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 fødselsnummeret! Det skal være 11 siffer.");
    }
```

```java
public class FeilAntallSifferException extends Exception{

    public FeilAntallSifferException () {
        super(message:"Feil antall siffer i fødselsnummeret! 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

- Unntakene sjekkes under kjøring. Ikke nødvendig å håndtere eller bruke "throws"-klausul.
- Eks:
  - NullPointerException
  - IndexOutOfBoundsException
  - IllegalArgumentException

EXTENDS RUNTIMEEXCEPTION (UNCHECKED)

```java
public class Person {

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

    public Person(String navn, String fnr){
        this.navn = navn;
        settFnr(fnr);
    }

    private void settFnr(String fnr) {
        if (fnr.length() != 11) {
            // Utløser usjekket unntak som indikerer problemet
            throw new FeilAntallSifferException();
        }
        this.fnr = fnr;
    }
```

```java
public class FeilAntallSifferException extends RuntimeException{

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

EXTENDS EXCEPTION (CHECKED) – THROWS OG TRY-CATCH

```java
public class Person {

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

    public Person(String navn, String fnr){
        this.navn = navn;
        try {
            settFnr(fnr);
        } catch (FeilAntallSifferException e) {
            System.out.println(x:"Ugyldig fødselsnummer! Må være 11 siffer.");
        }
    }

    private void settFnr(String fnr) throws FeilAntallSifferException {
        if (fnr.length() != 11) {
            // Utløser usjekket unntak som indikerer problemet
            throw new FeilAntallSifferException();
        }
        this.fnr = fnr;
    }
```

← Kall på metode som kan utløse unntak (catch)

← Metode som kan utløse unntak (throw)

```java
public class FeilAntallSifferException extends Exception{

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

EXTENDS EXCEPTION (CHECKED) – TRY-CATCH

```java
public class FeilAntallSifferException extends Exception{

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

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

1. INTERFACE OG ARV | PROGRAMMERING

a) Lag et interface, Drikkbar. Interfacet skal ha to metoder: drikkGlass(); og fyllOpp();

b) Lag deretter en klasse «Drikke» som implementerer grensesnittet Drikkbar. Drikke må holde på hvor mye drikke det er plass til i flasken (dl) og hvor mye drikke det er i flasken nå (dl). Maksstørrelsen på flasken settes i konstruktøren og vi kan anta at flasken er full når den opprettes.

c) Dersom metoden «drikkGlass» kalles skal flasken reduseres med 2 dl. Dersom det er mindre enn 2 dl igjen, skal flasken tømmes (til 0 dl). «fyllOpp» skal fylle flasken opp til maksstørrelse.

d) Utvid programmet ditt med subklassen Ol (øl). Subklassen skal være lik drikke, men i stedet for at flasken reduseres med 2 dl, skal den reduseres med en halv liter når et glass drikkes. Lag også en toString()-metode for både drikke og øl. Hint: husk @Override der du mener det er riktig.

e) Tegn klassehierarkiet for oppgaven.

f) Implementer en liten test av programmene dine.

