Denne uken

- Dype strukturer ✓
  - Likhet ✓
  - Kopiering ✓
- Immutability ✓
- Arv
  - Grensesnitt
  - Klasser
  - Abstrakte klasser
- Hvis vi får tid: bittelitt grafikk

Recap fra i går: Vårfest

```
/** @return the first day of spring this year */
Date startOfSpring() {
    if (this.snowdropAnswer == null) {
        this.snowdropAnswer = askSnowdrop();
    }
    return snowdropAnswer;
}

// somewhere else in the code...
void partyPlanning() {
    Date partyDate = startOfSpring();
    // ...
    // some other constraint showed up, need to move date
    partyDate.setMonth(partyDate.getMonth() + 1);
    // ... uh-oh. what just happened?
}
```

Recap fra i går: Vårfest

Dårlig løsning 1: endre dokumentasjonen

```
/** @return the first day of spring this year.
The caller may never change the returned object. */
Date startOfSpring() {
    if (this.snowdropAnswer == null) {
        this.snowdropAnswer = askSnowdrop();
    }
    return snowdropAnswer;
}
```

Recap fra i går: Vårfest

Dårlig løsning 2: returnere en kopi

```
/** @return the first day of spring this year.
*/
Date startOfSpring() {
    if (this.snowdropAnswer == null) {
        this.snowdropAnswer = askSnowdrop();
    }
    return snowdropAnswer.clone();
}
```

Recap fra i går: Vårfest

God løsning: bruk en immutable data-type

```
/** @return the first day of spring this year.
*/
LocalDate startOfSpring() {
    if (this.snowdropAnswer == null) {
        this.snowdropAnswer = toLocalDate(askSnowdrop());
    }
    return snowdropAnswer;
}
```

Mutable vs immutable

- Prøv å lage klasser immutable hvis mulig
  - I stedet for å ha metoder som endrer objektet, ha metoder som returnerer en endret kopi av objektet.
  - Godt egnet for klasser med relativt små mengder data
- Dersom du bruker mutable klasser
  - Pass på at du ikke klusser det til for andre
  - Pass på å dokumenter godt alle side-effekter i metoder som muterer objektet
  - Pass på at kopier er tilstrekkelig dype for formålet

Hva er hva?

```java
package inf101v22.forelesning;

public class App <T> implements Runnable {

    public static final String DEFAULT_THING = "Hello World";
    private T thing;

    public static void main(String[] args) {
        Runnable app = new App(DEFAULT_THING);
        app.run();
    }

    App(T thing) {
        this.thing = thing;
    }

    @Override
    public void run() {
        System.out.println(this.thing);
    }
}
```

pakke
klasse
objekt
grensesnitt/interface
instans-variabler
lokale variabler
globale variabler
static metoder
instans -metoder
tilgangsmodifikatorer
konstruktør
main –metode
parameter
argument
array
retur-type
signatur

private
public
Runnable
stack
heap
refererte typer
primitiver
final
immutable
mutable
innkapsling
abstraksjon
overloading
konstant

https://padlet.com/torsteins/inf101keywords passord: abstraksjon

Arv

INF101 forelesning 22. februar 2022

Torstein Strømme

Stikkord: arv for grensesnitt, arv for klasser, abstrakte klasser

Arv

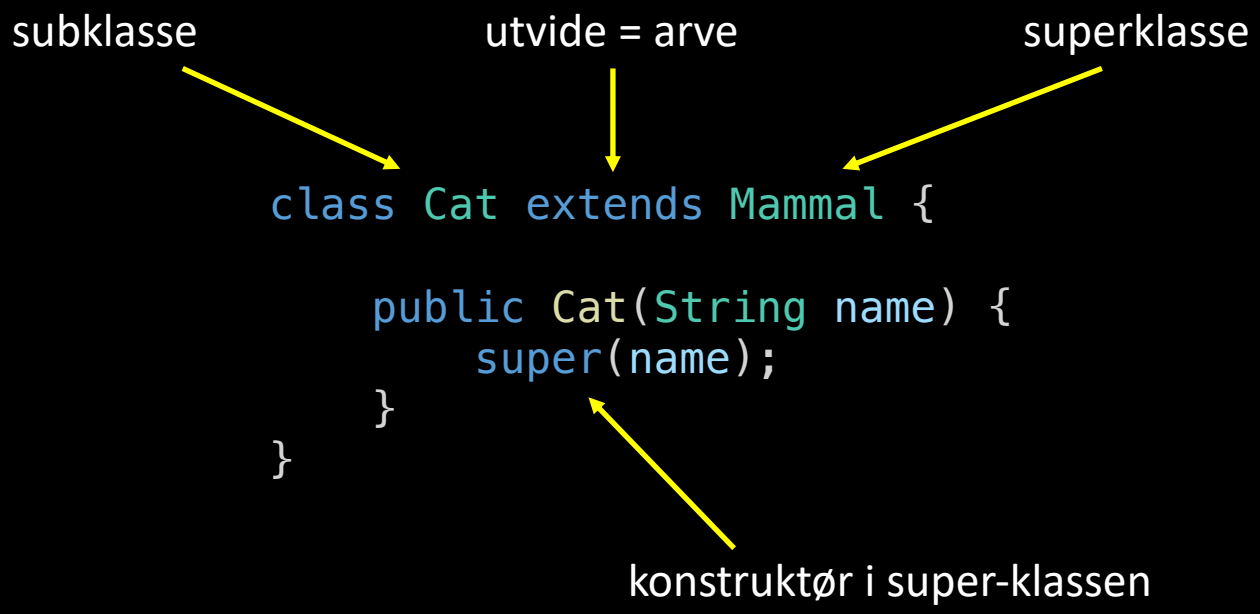
- Et grensenitt kan *utvide* et annet grensesnitt
  - Aable utvider Bable → implementerer du Aable må også implementere Bable
  - Aable -objekter kan lagres i en variabel av type Bable

```
interface IGrid<E> extends Iterable<E> { ... }
```

- En klasse kan *utvide* en annen klasse
  - Hvis Aclass utvider Bclass → Aclass har automatisk alle protected/public metoder som Bclass har
  - Aclass- objekter kan lagres i en variabel av typen Bclass.

```
public class Dog extends Animal { ... }
```

Arv




```
class Mammal {
    private String name;

    public Mammal(String name) {
        this.name = name;
    }

    void sleep() {
        System.out.println("The " +
            this.getClass().getSimpleName().toLowerCase() +
            " " + this.name + " goes to sleep”
        );
    }
}
```

```
public static void main(String[] args) {
    Mammal a = new Animal("Angela");
    Cat b = new Cat("Boris");
    Mammal c = new Cat("Xi");
    a.sleep(); // The mammal Angela goes to sleep
    b.sleep(); // The cat Boris goes to sleep
    c.sleep(); // The cat Xi goes to sleep
}
```

Arv: Override

```
public static void main(String[] args) {
    Mammal a = new Mammal("Angela");
    Mammal b = new BlackPanther("Boris");

    a.sleep(); // The animal Angela goes to sleep
    b.sleep(); // The black panther Boris goes to sleep
}
```

```
class Mammal {
    private String name;

    public Mammal(String name) {
        this.name = name;
    }

    void sleep() {
        System.out.println("The " + this.animalType() +
        " " + this.name + " goes to sleep");
    }

    String mammalType() {
        return this.getClass()
            .getSimpleName()
            .toLowerCase();
    }
}
```

```
class BlackPanther extends Mammal {

    public BlackPanther(String name) {
        super(name);
    }

    @Override
    String mammalType() {
        return "black panther";
    }
}
```

Abstrakte klasser

- En mellomting mellom et grensesnitt og en klasse
- En klasse som bare er delvis implementert
- Designet for å bli utvidet

Abstrakte klasser

```
abstract class Mammal {
    private String name;

    public Mammal(String name) {
        this.name = name;
    }

    void speak() {
        System.out.println(this.name + " says: " + this.getNoise());
    }

    abstract String getNoise();
}
```

```
class Cat extends Mammal {

    public Cat(String name) {
        super(name);
    }

    @Override
    String getNoise() {
        return "miaow";
    }
}
```

```
public static void main(String[] args) {
    Mammal a = new Cat("Angela");
    Mammal b = new BlackPanther("Boris");

    a.speak(); // Angela says: miaow
    b.speak(); // Boris says: roar
}
```

Arv

- Grensesnitt kan arve fra/kombinere andre grensesnitt
- Klasser kan implementere mer enn ett grensesnitt
- Klasser kan kun arve fra én klasse

Arv

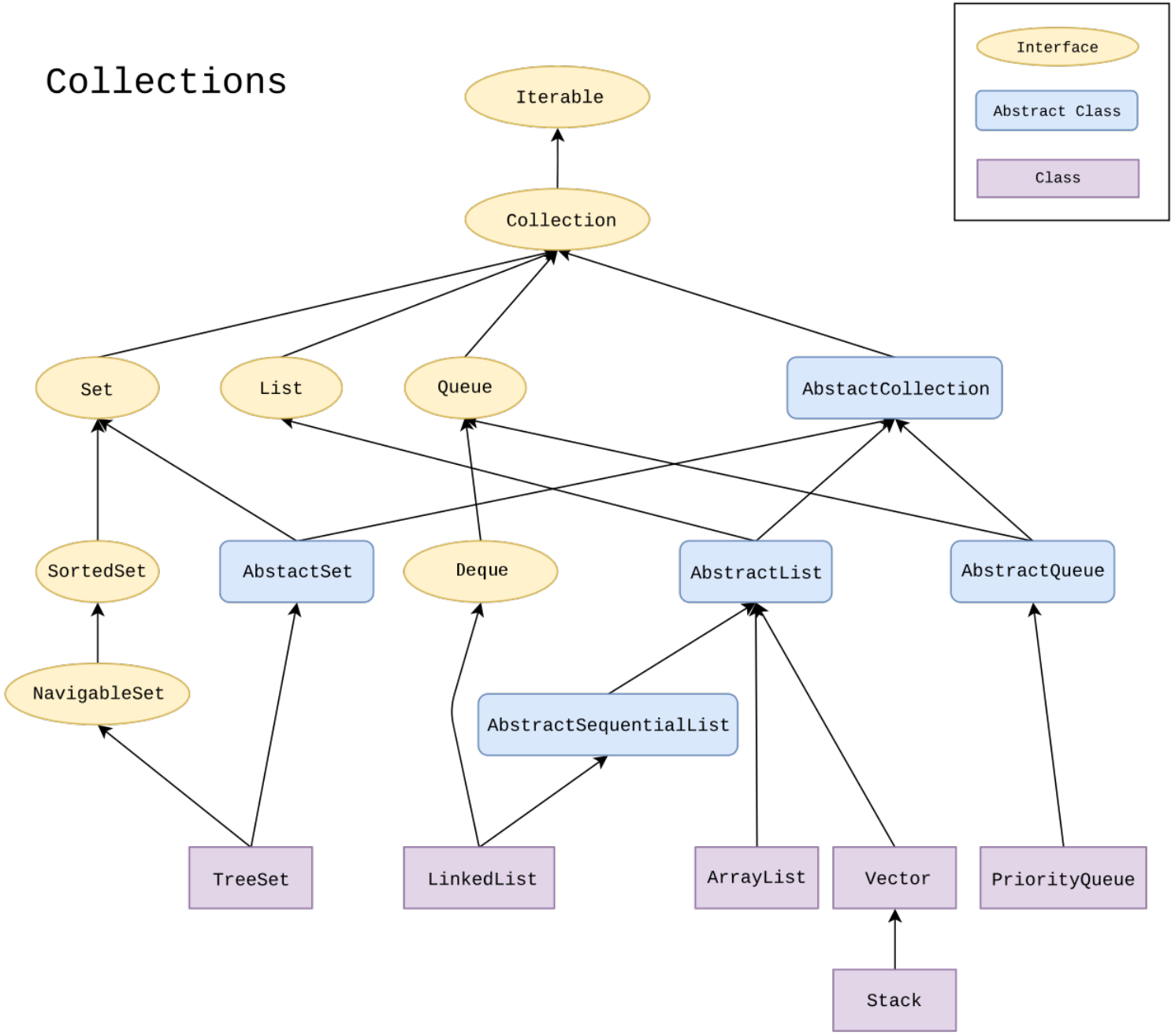
- Fordeler
  - DRY – don’t repeat yourself
- Ulemper
  - Bryter med single responsibility principle
  - Arv kan ødelegge immutability
    - Løsning: gjør hele klassen “final” – da er det ikke mulig å utvide klassen
  - Lavere modularitet

Arv

- Arv for grensesnitt 🥳
- Arv for klasser 🤨
  - Vurder om komposisjon er et bedre alternativ
  - Er vanlig i rammeverk, som f. eks. Swing og JavaFX
  - Kan brukes dersom enten:
    - Klassen du utvider er designet spesielt for å bli utvidet (f. eks. i rammeverk)
    - Du ikke *endrer* oppførselen for metoder som allerede er definert (med andre ord, en metode som tror den jobber med et objekt av type Bclass skal ikke få noen overraskelser dersom objektet egentlig er en Aclass).

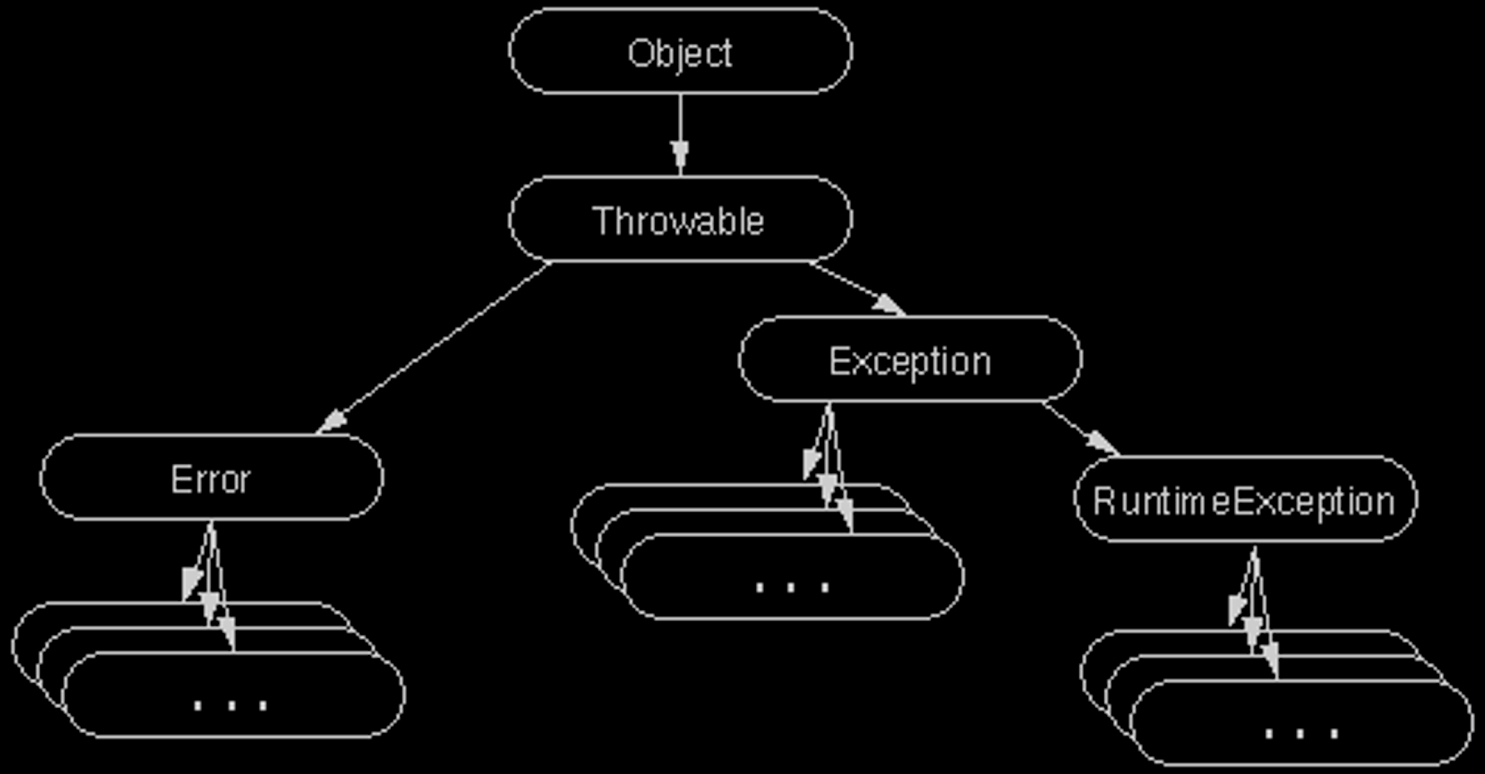
Arv

i Java Collections



Arv

Java Throwable



Grafikk: Java swing

INF101 forelesning 22. februar 2022

Torstein Strømme

```java
public static void main(String[] args) {
    JFrame frame = new JFrame("INF101");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(600, 400);
    frame.setVisible(true);
}
```

```java
public static void main(String[] args) {
    View view = new View();

    JFrame frame = new JFrame("INF101");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setContentPane(view);
    frame.setSize(600, 400);
    frame.setVisible(true);
}
```

```java
class View extends JComponent {

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        g.fillRect(20, 40, 200, 60);
    }
}
```

Livekoding: sjakkbrett