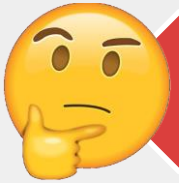
Forelesning 7
Abstraksjon og Interface

INF101 - 2023

UNIVERSITETET I BERGEN

Objektorientert programmering



Abstraction

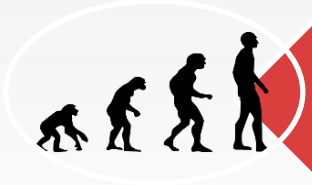
- Fokuser på det viktigste og gjør det enkelt

Modularity

- Lag kode som er byggeklosser

Encapsulation

- Skjul detaljer, koden blir lettere å bruke/tryggere



Gjenbruk av kode

- Arv - Gjenbruk av klasser
- Polymorphism – metoder virker på flere typer

Abstraksjon

- Når man skal lage et program er det ofte begreper og objekter i virkeligheten som skal representeres som objekter i koden.
- Vi representerer ikke alle detaljer, bare det viktigste.

Abstraksjon

- En hver klasse har et visst ansvar, å legge rett funksjonalitet til rett klasse er viktig.
- Når rett funksjonalitet er i rett klasse blir systemet lett å bruke og lett å forstå
- Ikke for mye detaljer, ikke for lite funksjonalitet.

- Single responsibility prinsiple
  - Når hver klasse kun har ansvar for en ting har du full abstraksjon

Abstraksjon

- Abstraksjon handler også om å finne ut hva som er felles for flere klasser
- Felles kode bør trekkes ut og gjenbrukes.

Interface

- Definerer hva en klasse skal gjøre
  - Metodenavn uten kode
  - Gode kommentarer
- Gir frihet til å endre implementasjon av interface uten å påvirke andre deler av programmet.
- Samme metode kan gjenbrukes på flere typer objekter så lenge de implementerer interfacet. Dette kalles polymorhism.

Interface navngiving

- Slutter ofte med -able
  - Playable
  - Drawable
- Starter ofte med I
  - ICamera
  - ICar
- Starter ofte med Can
  - CanSwim
  - CanShoot

Det viktigste er at det er lett å lese navnet og skjønne hva dette er. E.g. Interface with play() method

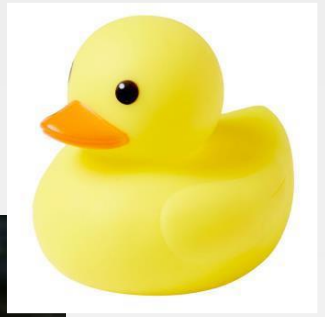
Lab 2 - Pokemon

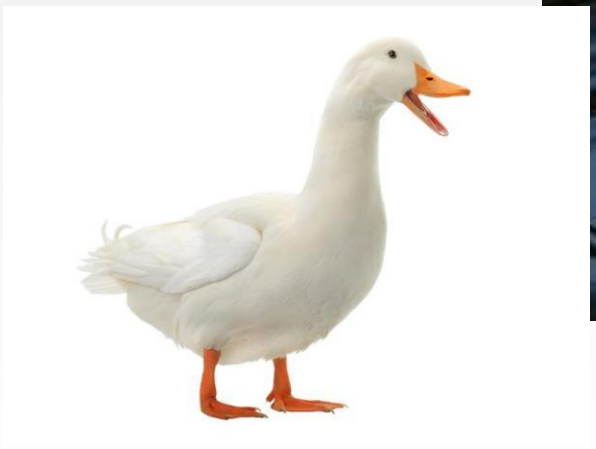
- La oss se litt på Lab2

- Vi kan lage forskjellige Pokemon objekter, men hva er det vi kan endre på?

Abstraksjon







Sitat – duck-typing

James Whitcomb Riley (1849–1916):

“When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.”

Duck-typing betyr at vi trenger ikke alltid beskrive virkelige objekter nøyaktig, bare de egenskapene som er nødvendige.

Duck-typing

Python bruker duck typing

- Så lenge objektet har en metode med samme signatur vil koden kjøre

Java bruker interface

- Kun de metodene som har implementert det rette interfacet kan gies som input.

Skal interface ha mange metoder?

- Hvis du vil lage ny funksjonalitet kan du implementere en ny versjon av interfacet.
- Stort interface krever mye koding
- Lite interface krever mindre koding

- Del opp interfacene i deler som kun har ansvar for en ting, så kan du bytte ut akkurat den funksjonaliteten.

Lab 2 - Pokemon

- Kan vi redusere duplikat kode ?
- Er det slik at hver klasse kun har ansvar for en ting?

Polymorfisme

- I Java må du definere typen på objekter
- Hvis du skal lage en metode må du bestemme hvilken type objekter den kan ta inn.
  - Hva hvis du vil ha samme metode tilgjengelig for flere typer
  - max(int a, int b)
    max(double a, double b)
    …
- Vi har sett polymorfisme allerede.

Polymorfisme

```
Object obj = getData();
System.out.println(obj);
```

- Hvilken kode kalles her?
  - I hvilken klasse finner vi toString() metoden?
- toString() metoden finnes i mange klasser, hvilken kode velges?
- Det kommer an på hvilken type obj har.

Objekt klassen

- Alle klasser er av typen Objekt
  - Vi lærer mer om dette når vi kommer til arv
- Noen viktige metoder finnes i alle klasser
  - toString()
  - equals()

- Forrige uke lærte dere forskjellen på equals() og ==
- Denne uken lærte dere toString()
- Neste uke er det mer om equals()

La oss implementere Bibliotek

- Biblioteket trenger et system for å holde styr på alle bøkene sine.
- Kan vi hjelpe dem?

- Lage en klasse Book
- Lage et system for å holde styr på mange bøker

```java
public interface Thanker {

    /**
     * This method should say thanks
     */
    void sayThanks();

    public static void main(String[] args) {
        Thanker terminal = new TerminalThanker();
        terminal.sayThanks();
    }
}

class TerminalThanker implements Thanker {
    @Override
    public void sayThanks() {
        System.out.println("Thank you");
    }
}
```