INDKAPSLING

Programmering 1

Lektion 6

...

4. Klasser og objekter
5. Arv og komposition
6. Indkapsling ← I dag
7. Abstrakte klasser og interfaces
8. Polymorfi

...

Dagens program

- Repetition af klasser og objekter
- Hvad er indkapsling
- Hvorfor bruge indkapsling?
- Øvelser i grupper
- Opsamling

Repetition af klasser og objekter

En klasse er en skabelon for objekter.

```
class SavingsAccount {
  // fields
  double balance;
  double rate = 0.05;
  String owner;

  // constructor
  SavingsAccount(double balance, String owner) {
    this.balance = balance;
    this.owner = owner;
  }

  // method
  void applyInterest() {
    balance = balance * (1 + rate);
  }
}
```

Et objekt har en tilstand og adfærd

```
SavingsAccount kimsAccount = new SavingsAccount(1000, "Kim");
kimsAccount.applyInterest();

SavingsAccount annesAccount = new SavingsAccount(2000, "Anne");
annesAccount.applyInterest();

System.out.println(kimsAccount.balance); // => 1050
System.out.println(annesAccount.balance); // => 2100
```

Hvad er indkapsling

Ny adfærd: Hæve penge fra kontoen

```
SavingsAccount kimsAccount = new SavingsAccount(1000, "Kim");
kimsAccount.balance = kimsAccount.balance - 1200;
System.out.println(kimsAccount.balance); // => -200
```

Hvad er problemet her?

Demo

```
git clone https://github.com/jakobjanot/programmering1
```

Øvelse (20 min): At hæve/indsætte penge

```
class SavingsAccount {
  private double balance;
  private double rate = 0.05;
  private String owner;

  public SavingsAccount(double balance, String owner) {
    this.balance = balance;
    this.owner = owner;
  }

  public void applyInterest() {
    balance = balance * (1 + rate);
  }

  public double getBalance() {
    return balance;
  }

  // TODO: Tilføj ny adfærd
```

Diskussion af øvelse: At hæve/indsætte penge

```
class SavingsAccount {
  private double balance;
  private double rate = 0.05;
  private String owner;

  SavingsAccount(double balance, String owner) {
    this.balance = balance;
    this.owner = owner;
  }

  public void applyInterest() {
    double interest = balance * (1 + rate);
    deposit(interest);
  }

  public double getBalance() {
    return balance;
  }

  public void withdraw(double amount) {
    if (amount > balance)
      return;

    balance = balance - amount;
  }

  public void deposit(double amount) {
    if (amount < 0)
      return;

    balance = balance + amount;
  }
}
```

```
SavingsAccount kimsAccount = new SavingsAccount(1000, "Kim");
kimsAccount.withdraw(1200);
System.out.println(kimsAccount.getBalance()); // => 1000

kimsAccount.withdraw(200);
System.out.println(kimsAccount.getBalance()); // => 800
```

Resultat

- Mere kontrol over objektets tilstand
- Mere fleksibilitet til at ændre klassens implementering
- Mere læsbart og vedligeholdeligt kode

Hvorfor bruge indkapsling?

API - Application Programming Interface

a.k.a. Klassens grænseflade til omverdenen

- `public` metoder er klassens API (eller kontrakt)
- `private` metoder er klassens interne implementering

Øvelse: Refaktorering af `SavingsAccount`

- Kan vi gøre koden mere læsbar uden at ændre klassens API?

Diskussion af øvelse: Refaktorering af `SavingsAccount`

```
class SavingsAccount {
  private double balance;
  private double rate = 0.05;
  private String owner;

  SavingsAccount(double balance, String owner) {
    this.balance = balance;
    this.owner = owner;
  }

  public void applyInterest() {
    deposit(calculateInterest());
  }

  [...]

  private double calculateYearlyInterest() {
    return balance * (1 + rate);
  }
}
```

Gruppearbejde (2 time)

Opgave - Beregning af rente

1. Implementer daglig rente
2. Flyt implementeringen til en ny klasse til beregning af rente - API'et skal være `calculateInterest(double balance)`
3. Refaktorer `SavingsAccount` til at bruge den nye klasse
4. Lav en ny klasse `CheckingAccount` med samme funktionalitet som `SavingsAccount`, der tillader overtræk (negativ saldo).
5. Implementer strafrente på overtrækket.
6. (Valgfri) Lav en ny klasse `Bank`, der kan tilskrive renter på liste

Diskussion af gruppearbejde

Opsamling - hvad har vi lært?

- Indkapsling er en måde at skjule data og metoder fra omverdenen
- Indkapsling giver mere kontrol over objektets tilstand
- `public` metoder er klassens API
- `private` metoder er klassens interne implementering
- Indkapsling gør koden mere læsbar og vedligeholdelig, da vi kan tilføje så mange `private` metoder som vi vil uden at påvirke klassens brugere

Næste gang

- Hvordan lader vi `Bank` have en liste med forskellige typer konti, dvs. `CheckingAccount`, `SavingsAccount` m.fl.?
- Hvordan sikrer vi os at alle konti har metoden `applyInterest()` til at tilføje renter?

Næste gang - Abstrakte klasser og interfaces

- Hvordan kan vi bruge interfaces til at definere en kontrakt for klasser?
- Hvordan kan vi bruge abstrakte klasser til at dele kode mellem klasser?