Objektorientert design av kode. Refaktorering.

DEL 1

INF1010-forelesning 2. mars

Ragnhild Kobro Runde

Læringsmål

- Kjenne til og kunne bruke viktige prinsipper for god kodedesign.
- Kunne finne alternative løsninger for et gitt problem.
- Kunne vurdere fordeler og ulemper ved ulike løsninger.

Objektorientert design i INF1050 og INF1010

- INF1050: Fokus på hele systemutviklingsprosessen.
- INF1010: Programmering i Java.

- Sommerville:
  Object-oriented design is concerned with developing an object-oriented model of a software system to implement the identified requirements. The objects in an object-oriented design are related to the solution to the problem that is being solved.

Bilhierarki

Eksempelet fra oblig 2

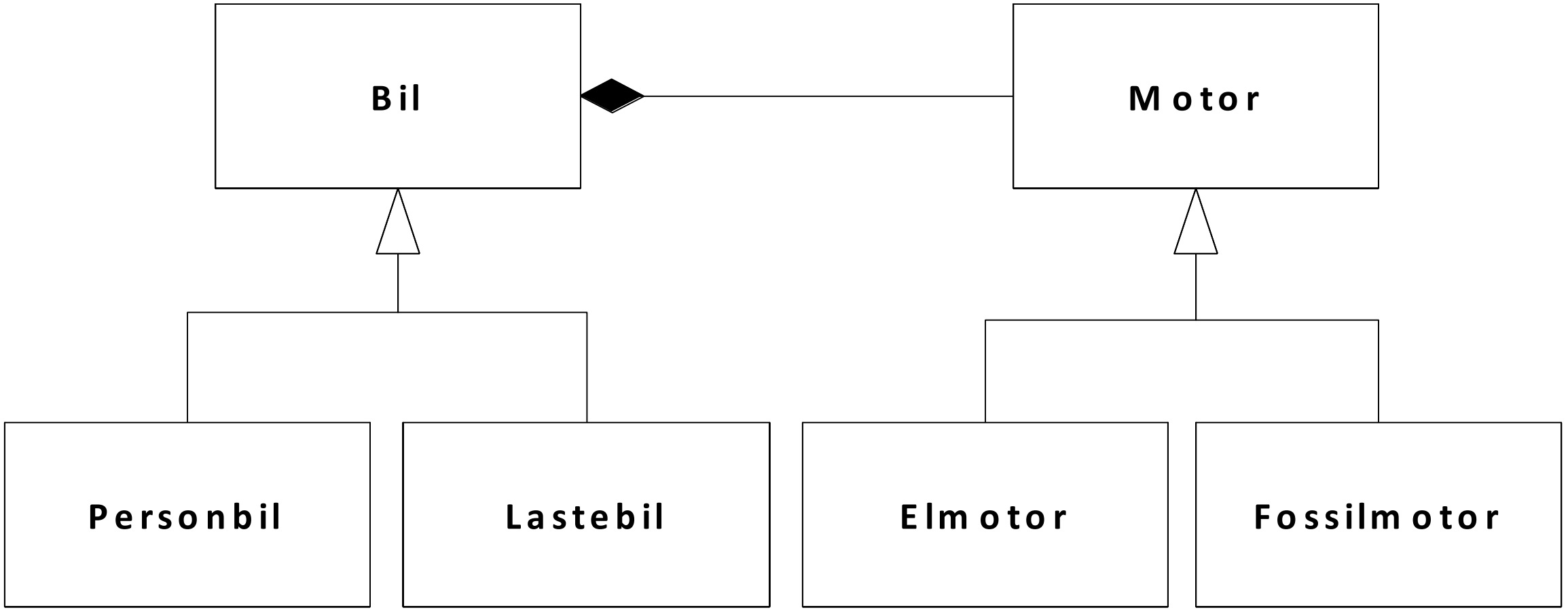
Oblig2: Bilhierarki

- Alle biler er enten elbiler eller fossilbiler.
- Alle fossilbiler er enten personbiler eller lastebiler.

- Alle Bil-objektene ble lagt inn i en ArrayList<Bil> (eller Bil[]).
- Programmet skulle lese inn data fra fil og deretter skrive ut alle biler, alle elbiler eller alle fossilbiler.

- Diskuter fordeler og ulemper ved denne løsningen fremfor å ha Bil-objektene i to separate lister (ArrayList<Elbil> og ArrayList<Fossilbil>).

Alternativt design Bil-oppgaven: Komposisjon



Klassestruktur basert på komposisjon

Har i tillegg

- konstruktører
- metoder for utskrift
- subklassene Personbil og Lastebil

```
abstract class Bil {
    private Motor motor;
}

abstract class Motor {
}

class Elmotor extends Motor {
    private double batterikapasistet;
}

class Fossilmotor extends Motor {
    private double utslipp;
}
```

Filtrert utskrift vha instanceof

```
public void skrivAlle(String filter) {
    for (Bil bil:biler) {
        skrivHvisMotorErLik(filter);
    }
}

// I klassen Bil:
public void skrivHvisMotorErLik(type) {
    if ((type.equals("EL") &&
            motor instanceof Elmotor)) ||
         (type.equals("FOSSIL")&&
            motor instanceof Fossilmotor))
        skrivUtInfo();
    }
}
```

String

Filtrert utskrift vha getMetode

```
// I klassen Bil:
public void skrivHvisMotorErLik(type) {
    if (type.equals(motor.hentType()))
        skrivUtInfo();
    }
}

// I klassen Motor:
abstract String hentType();

// I klassen Elmotor:
public String hentType() {
    return "EL";
}
// Tilsvarende for Fossilmotor
```

Filtrert utskrift vha egen sjekk

```
// I klassen Bil:
public void skrivHvisMotorErLik(type) {
    if (motor.er(type))
        skrivUtInfo();
    }
}

// I klassen Motor:
abstract boolean er(String type);

// I klassen Elmotor:
public boolean er(String type) {
    return type.equals("EL");
}
// Tilsvarende for Fossilmotor
```

Utskrift til fil

- Filformatet fra obligen:

```
EL <reg.nr.> <batterikapasitet>
PERSONBIL <reg.nr.> <utslipp> <seter>
LASTEBIL <reg.nr.> <utslipp> <nyttevekt>
```

Hvor bør metoden(e) for å skrive til fil legges?

Alt 1: Hvert objekt skriver sine data til fil

```
public void skrivTilFil(String filnavn) {
  try {
    PrintWriter writer = new PrintWriter(filnavn);
    for (Bil bil:biler) {
      bil.skrivBilTilFil(writer);
    }
    writer.close();
  } catch (FileNotFoundException e) {
    // Feilhåndtering
  }
}

// I klassen Bil:
public void skrivBilTilFil(writer) {
  motor.skrivMotorTilFil(writer);
  writer.print(registreringsnummer);
}
// Osv...
```

Alt 2: Hvert objekt lager teksten som skal skrives til fil

```
public void skrivTilFil(String filnavn) {
  try {
    PrintWriter writer = new PrintWriter(filnavn);
    for (Bil bil:biler) {
      writer.println(bil.tilFil());
    }
    writer.close();
  } catch (FileNotFoundException e) {
    // Feilhåndtering
  }
}

// I klassen Bil:
public String tilFil() {
  return motor.tilFil() + registreringsnummer;
}
// Osv...
```

Noen generelle prinsipper

Ekspertprinsippet, kohesjon og kobling

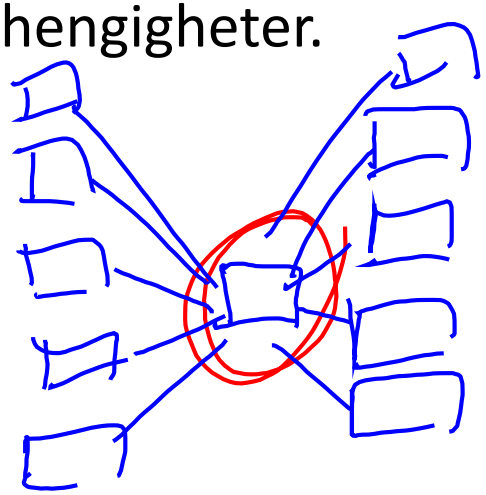
Ekspertprinsippet («Gjøre selv»)

- **Problem:** Hvordan tilordne ansvar til objekter?
- **Løsning:** Det objektet som har kunnskapen (dataene) bør ha ansvaret.

(Høy) kohesjon og (lav) kobling

- **Kohesjon** er et mål på hva slags ansvar et objekt har og hvor fokusert ansvaret er.
  - Mål: objekter som har **moderat ansvar** og utfører et **begrenset antall oppgaver** innenfor **ett funksjonelt område**.

- **Kobling** er et mål på hvor sterkt et objekt er knyttet til andre objekter.
  - Mål: Objekter med begrenset antall avhengigheter.



Ja, men, det virker jo…!

- Hvorfor er det viktig med god kodedesign?
- Når er det viktig med god kodedesign?
- Hva er god kodedesign?

- Generelt viktig:
  - Lesbarhet.
  - Vedlikeholdbarhet.
  - Gjenbrukbarhet.
  - Effektivitet.
  - Robusthet.
  - Pålitelighet.
  - Korrekthet.

Kjennetegn på god design (fra INF1050)

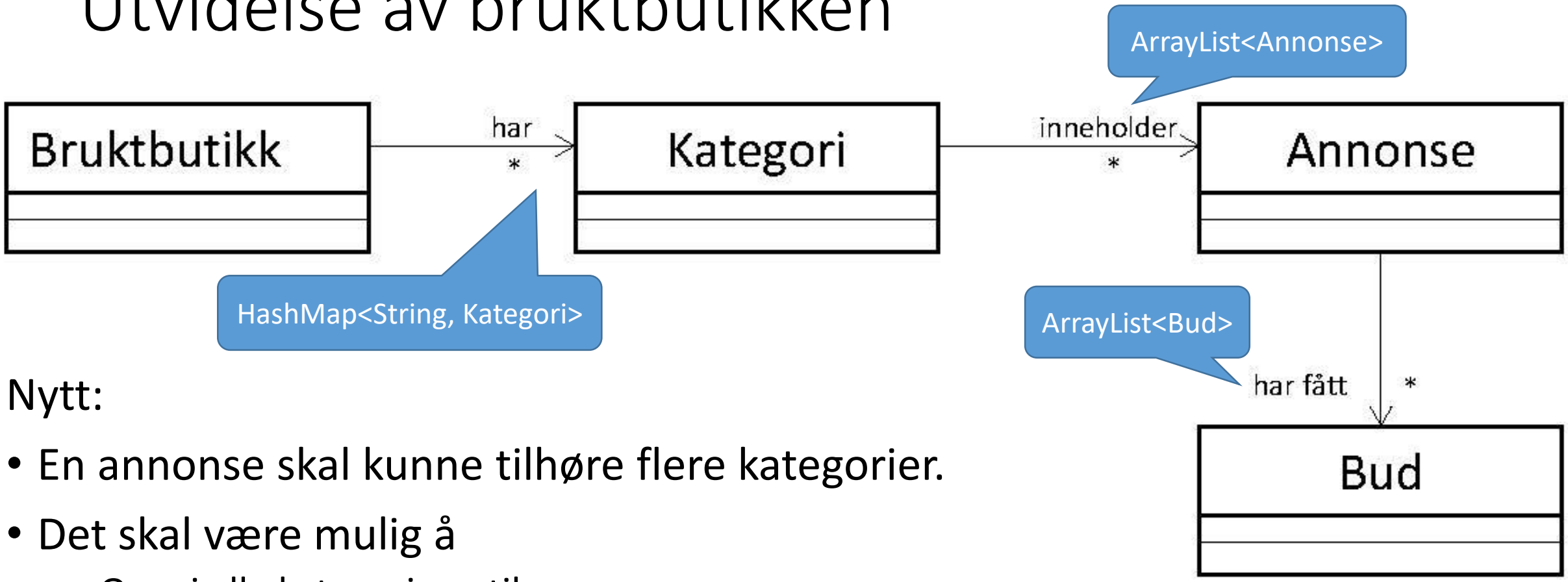
- En god utforming gjør den jobben den er ment å gjøre.
- En god utforming er enkel og elegant.
  - Eleganse innebærer å finne akkurat riktig abstraksjonsnivå.
- En god utforming er gjenbrukbar, utvidbar og enkel å forstå.
- Et godt objekt har et lite og veldefinert ansvarsområde.
- Et godt objekt skjuler implementasjonsdetaljer fra andre objekter.

- Grady Booch

Bruktbutikk

Eksempel fra høstens eksamen i INF1000/INF1001

Utvidelse av bruktbutikken

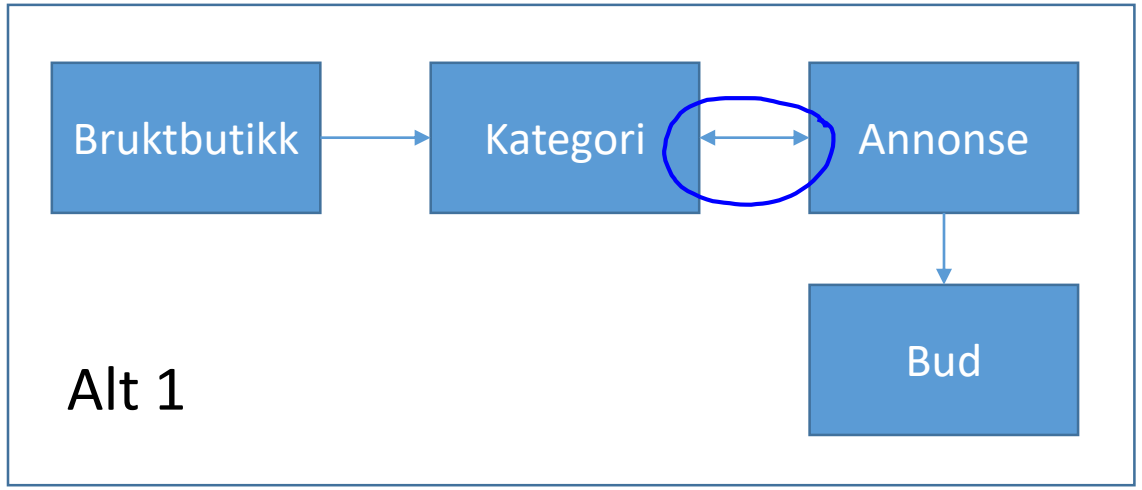


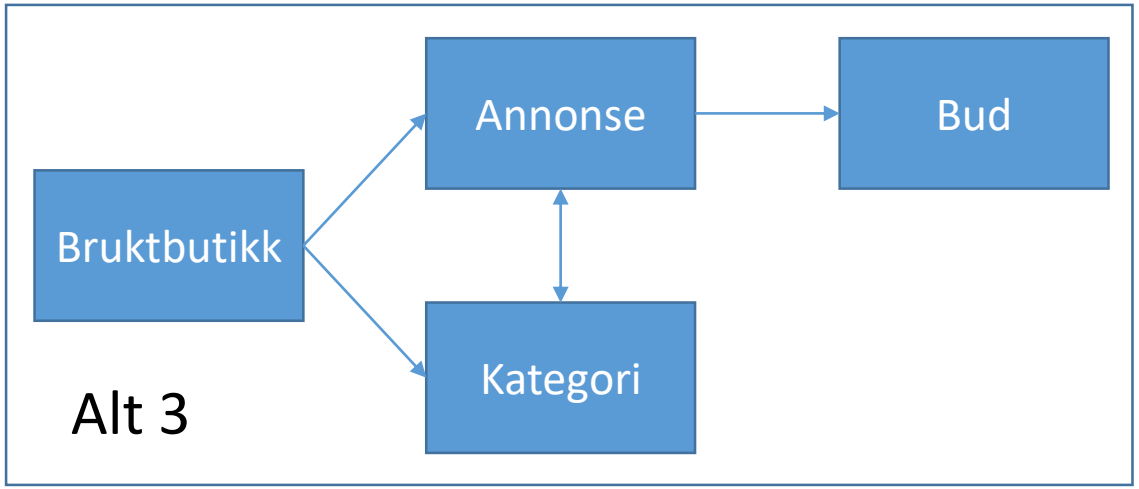
Nytt:

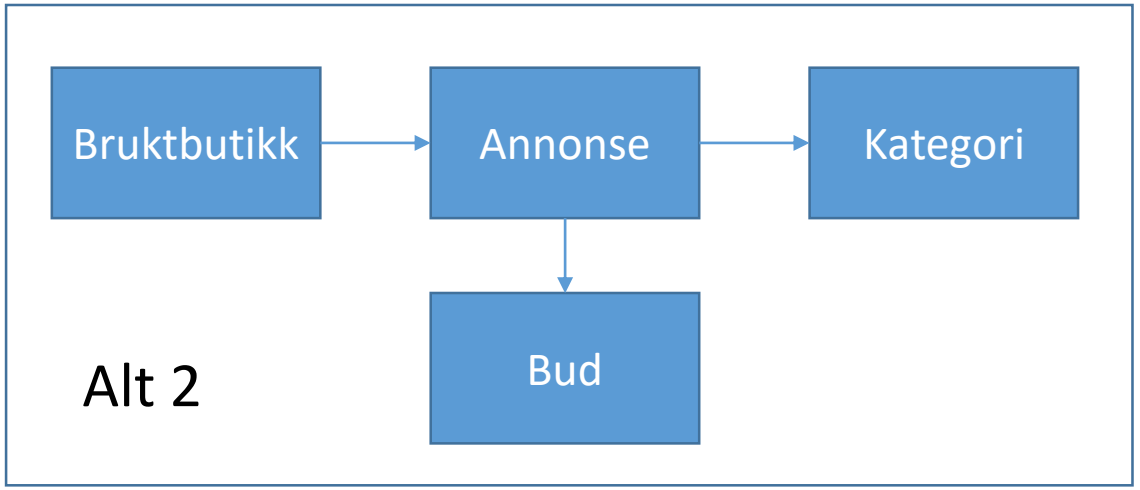
- En annonse skal kunne tilhøre flere kategorier.
- Det skal være mulig å
  - Oppgi alle kategoriene til en annonse.
  - Søke etter alle annonser som er i et gitt sett kategorier.

Hvilke endringer ville du gjort i datastrukturen for å få til dette?

Alternative datastrukturer







Hva er fordelene/ulempene med hver av disse?

Bruktbutikk – kategorier og annonser

Sammenlign følgende beskrivelser:

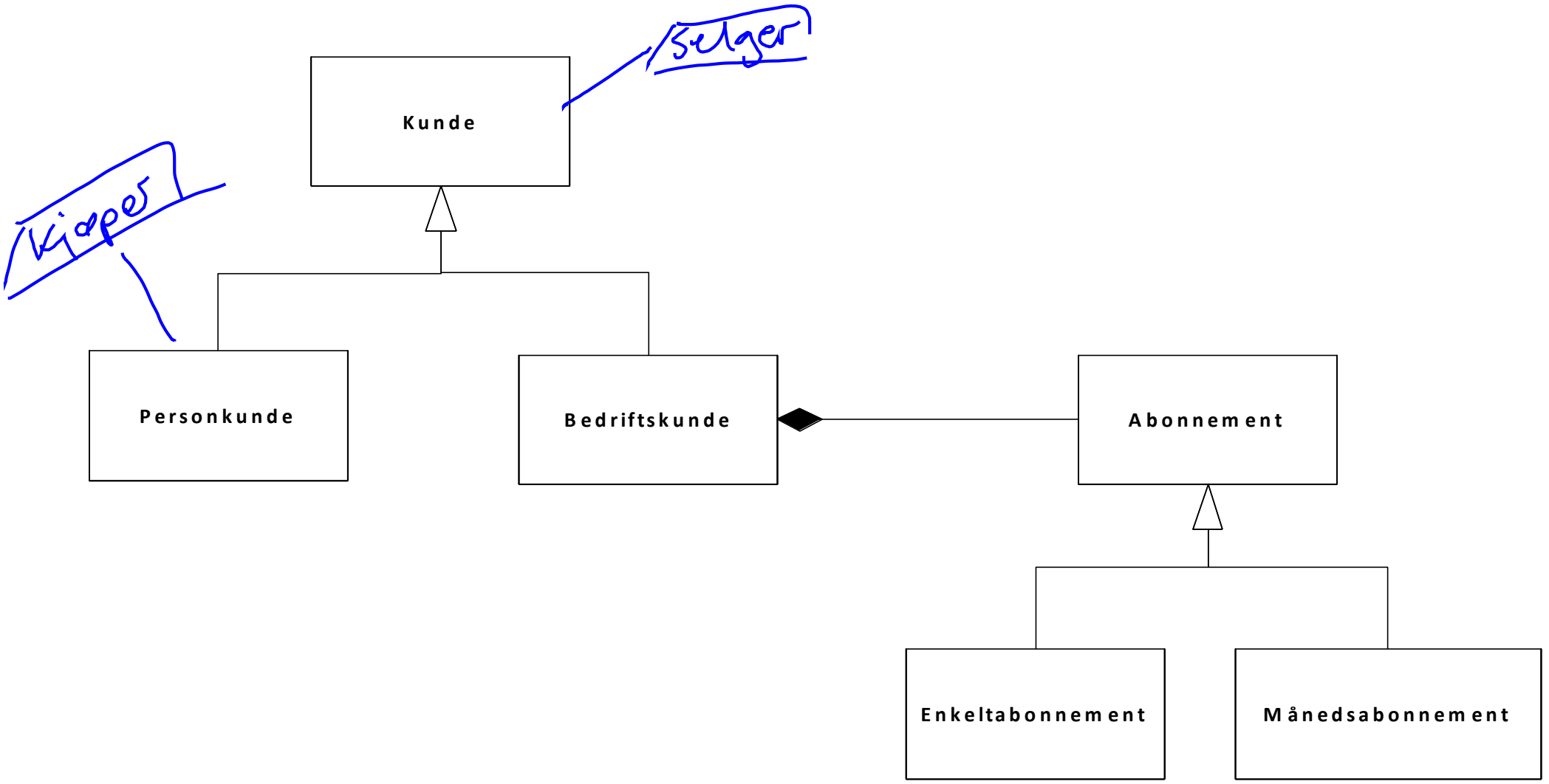
- En bruktbutikk selger ting i en rekke ulike kategorier. Innen hver kategori finnes det en mengde annonser hvor potensielle kjøpere kan legge inn bud på de ulike tingene. En annonse kan tilhøre flere kategorier samtidig.
- En bruktbutikk selger ting via en mengde annonser hvor potensielle kjøpere kan legge inn bud. For hver annonse er det angitt et sett med kategorier som annonsen tilhører.

Bruktbutikk med kunder

- Bruktbutikken ønsker å opprette et kunderegister og samtidig utvide funksjonaliteten slik at også kundene kan legge ut annonser.
- Både privatpersoner og bedrifter kan være kunder.
- Alle kunder kan opprette annonser for å selge ting, men bare privatkunder kan gi bud.
- Å opprette annonser er gratis for privatpersoner. Bedriftskunder kan enten betale per annonse de legger ut eller en fast månedsavgift.

Hvilken datastruktur ville du foreslått her?

Forslag: Bruktbutikk med kunder



Eksempel: Barnehage-programmet

Et større eksempel: Barnehage

Vi skal lage et program for å administrere en barnehage.
I barnehagen finnes det to typer avdelinger:

- Småbarnsavdeling
  - Barn opp til 3 år
  - Plass til 9 barn
- Avdeling for større barn
  - Barn over 3 år
  - Plass til 18 barn

Et barn kan være tatt opp til maksimalt en avdeling. For hver avdeling vedlikeholdes en venteliste. Et barn kan bare stå på en venteliste. Derimot kan et barn både være tatt opp i en småbarnsavdeling, og stå på venteliste til en avdeling for større barn.

Opptak fra ventelisten skjer "ved loddtrekning", men med søskenprioritet ved opptak til småbarnsavdelingene. Til avdelingene for større barn har barn som allerede har plass i en småbarnsavdeling prioritet.

Hvilken datastruktur ville du foreslått her?