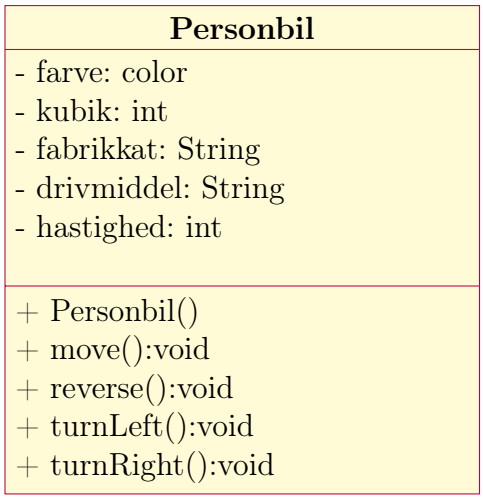
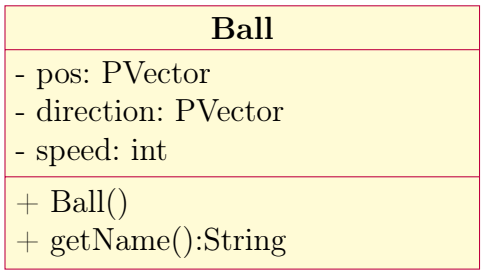
14 Objektorienteret programmering

OOP betyder objekt orienteret programmering og er et programmerings paradigme. Det tager udgangspunkt i et fysisk objekt, som kan være en bil, person, hus, kuglepen etc. Men det kan også være immateriel, som en helligdag eller rettigheder. For at kunne oprette en instans af et objekt, skal vi bruge en beskrivelse af objektes egenskaber og funktionalitet. En personbils egenskaber er f.eks. farve, mærke, motorstørrelse mm. En personbils funktio- nalitet er at den kan køre fremad, bakke, dreje mm. Denne beskrivelse kalder vi for en klasse. Man kan bruge UML diagrammer til at beskrive klasser 25



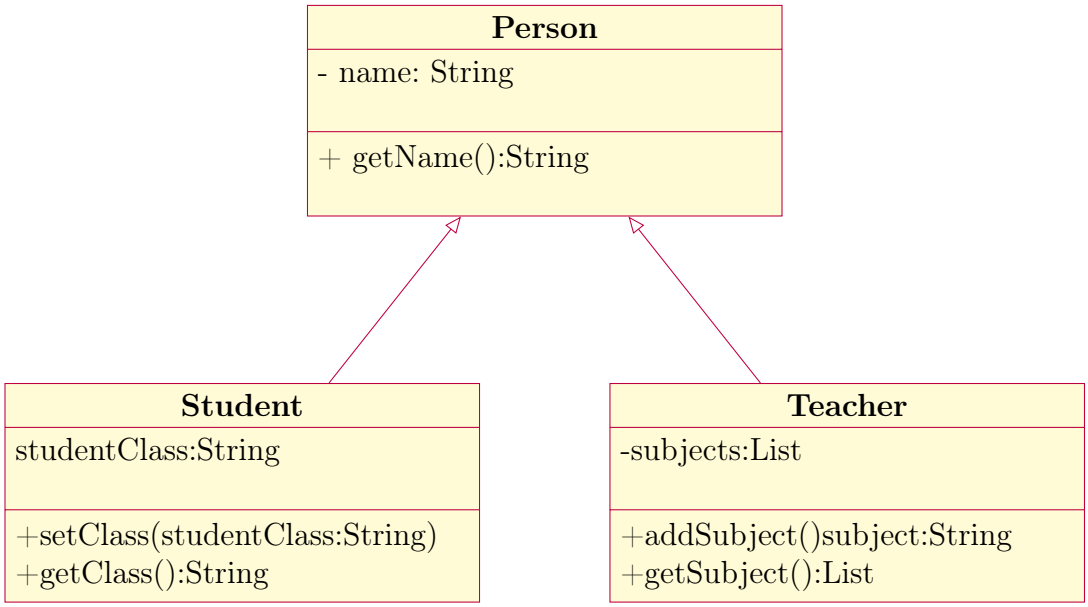
Figur 25: Personbil

Når vi programmerer, forsøger vi at skabe en matematisk model af virkeligheden. Objekter giver os mulighed for at lave vore egne datatyper.



Figur 26: Class diagram

14.1 Nedarvning



Figur 27: Class diagram nedarvning

14.2 Opgave

Her er et eksempel på en klasse. Firkant.pde. Klasser er ikke-primitive datatyper og skrives altid med stort.

Klassen Firkant, kan generere en tilfældig x og y koordinat. Klassen kan også tegne en firkant på dit canvas.

En klasse er en skabelon.

```
class navn-paa-klassen{

        klassens tilstand / attributter
        (variabler-kan initieres gennem konstroktor eller setters)

        klassen konstruktor
        (hvad skal der ske i de sekundt at vi initierer klassen)

        klassens metoder
        (getters/setters og alle andre metoder)
}
```

Figur 28: skabelon til en klasse

Hvis du kigger i mappen klasse, så er der to filer. klasse.pde, er den fil som bruger klassen, og klassen er Firkant.pde.

Vi deklarerer en variable af datatypen integer på følgende måde: int x; -på samme måde gør vi med klasser: Firkant f; på denne måde laver vi en spand

som kan indeholde firkanter. Men for at kunne bruge spanden skal vi have noget i den. Vi initierer på følgende måde: f = new firkant();

Nu kan vi tilgå både metode og attributter ved at skrive f.generate(); eller f.x = 0; I vores tilfælde giver det ingen mening at ændre på klassens attributter, fordi klassen selv genererer nogle tilfældige værdier. Reglen er, at klassen skal være autonom. Dvs. at der ikke må være andre som ændrer på klassens tilstand end klassen selv.

14.2.1 Opgave

Ændre klassen Firkant således, at firkantens tilfældig x og y position generes i konstruktøren og ikke skal kaldes fra hovedprogrammet.

14.2.2 Opgave

Tilføj en attribut farve, til klassen. Farven skal være tilfældig. Brug RGB farver. Hvis du ikke kan huske hvad komandoen hedder, kan slå det op i dokumentationen.

14.2.3 Opgave

Tilføj en metode som kan tegne en cirkel i stedet for en firkant.

14.2.4 Opgave

ændre hovedprogrammet klasse.pde så den skiftevis tegner en firkant og en cirkel.

14.2.5 Opgave

Det er jo lidt dumt at have en klasse som hedder firkant, hvis den tegner en cirkel. Så lave en ny klasse så du deler firkant og cirkel hver for sig. Tilpas klasse.pde til dine nye klasser.

14.2.6 Opgave

Udvid med en trekant og en rektangel.

14.2.7 Opgave

lav så figuerne bliver af variabel størrelse.

14.2.8 Opgave

opload dit program til din github.

14.3 Opgave

Læringsmål: Du skal i denne opgave arbejde med klasser, arrays og input fra mus. På github, kan du finde et program house, med en klasse Room. Programmet tegner et hus. For hvert objekt af klassen Room, du opretter, tegner programmet et ekstra værelse. Et værelse er 100x100 pixels. Der er i alt plads til tre rum i bredden og tre rum i højden.

1. Undersøg programmet house. Hvor deklarerer jeg en variabel af datatypen Room, og hvad hedder min variabel?

2. Hvor mange funktioner er der i programmet house?

3. Når jeg oprettet objektet, altså initierer variablen med en værdi, hvor mang parameter skal der så bruges?

4. Undersøg klassen Room, attributter.

   (a) Hvor mange er der og hvad bruges de til?

   (b) Hvor mange er konstanter og hvor mange er variabler?

   (c) Hvad er sammenhængen mellem antallet af parameter og klassens tilstand?

5. Find i programmet house, den eller de linijer kode som tegner taget på huset. Dette ansvar bør ligge i klassen. Opret en funktion drawRoof() i klassen Room og flyt de linijer kode over i klassen.

6. For at kunne håndtere flere rum, kan vi oprette et array. Ændre variablen house, til et array af datatype Room. Jeg kunne godt tænke mig to etager, så derfor skal længden på house være 6. Room[] house = new Room[6];

7. Tilføj nu følgende rum: Kitchen, Livingroom, Toilet, Bedroom og Bathroom. Husk at korrigere X og Y positionen.

8. For at få programmet til at udskrive alle rum, skal du loop'e igennem dit array og kalde "house[i].drawRoom();"brug "for(int i = 0;i<house.length;i++)"som løkke struktur.

9. Find ud af, hvad du skal ændre i funktionen mouseClicked() for at kunne tænde og slukke lyset i alle rum.

10. Hvor vil du tilføje funktionen checkHouse(); (se Figure 14.3.), som kan finde ud af om du kan gå fra huset med ro i sindet og vide at alt lys er slukket?

11. I mousedClicked() er det funktionen selv, som udskriver om lyset er tændt eller slukket. Dette ansvar vil jeg gerne have skrevet over i klassen.

```
void  checkHouse(){
        boolean found=false;
        for (int i=0; i$<$ house.length; i++) {
          if (house[i].isLightOn() == true) {
            println ("turn off light in"+
                 house[i].getRoomName());
                 found = true;
          }
        }
         if (!found){
                 println("All right! You're ready to go!");
        } else {
                 println("you forgot something");
        }
}
```

Figur 29: funktion som undersøger om alt lys er slukket