29 juli Programmering, Nao - Pass 2

Vi ska idag lära oss om objektorienterad programmering. När man jobbar med större projekt är det vanligt att använda sig utav klasser. Istället för att se dem som att programmet flödar och hoppar mellan funktioner och loopar, är istället hela programmet uppbyggt av objekt. Ett objekt är en instans av en klass, det vill säga en slags simulering av en företeelse som används inom objektorienterad programmering för att samla data och kod som hör ihop. Ett objekt kan till exempel vara en bok i ett program som används på ett bibliotek för att hantera utlåning. Boken har ett antal egenskaper som beskrivs av dess attribut (författare, titel, antal sidor, egenskapen att vara utlånad eller tillgänglig till exempel) och ett antal metoder som kan användas för att hämta eller ändra information (man kanske vill veta författare av boken, eller ändra status som utlånad eller tillgänglig). Ett annat objekt skulle kunna vara en låntagare som då kan ha en samling böcker som den lånat som attribut och metoder för att låna och lämna tillbaka böcker.
I exemplet under har vi skrivit en klass som hanterar fordon (vehicle). Ett fordon har ett registreringsnummer (plate number) som vi måste ange för att kunna skapa ett fordon.

```
class Vehicle: #Fordon
    def __init__(self, plate_nr):
        self.plate_number = plate_nr #registreringnummer
```

Det kan även vara bra att skapa en funktion som hämtar ut registreringsnumret, och det kan man göra genom att skriva en funktion ”getPlateNumber” i klassen Vehicle.

```
class Vehicle:
    def __init__(self, plate_nr):
        self.plate_number = plate_nr

    def getPlateNumber(self):
        return self.plate_number
```

För att skapa ett fordon måste vi veta registreringsnumret till fordonet. Fordonet skapas därefter genom att lägga till följande kod utanför klassen.

```
my_vehicle = Vehicle("ABC123")
```

Ett fordon är då skapat och vi kan ta reda på registreringsnumret till fordonet genom att kalla på funktionen som hämtar ut registreringsnumret och därefter skriva ut det. Det skrivs på följande sätt:

```
my_vehicle = Vehicle("ABC123")
plate_number = my_vehicle.getPlateNumber()
print(plate_number)
```

1. Skriv in exempelkoden ovanför i en python-fil och se att det fungerar. För att öppna en python-fil kan du först öppna IDLE (pytonterminalen) och därefter trycka ctrl+N, då öppnas en python-fil automatiskt. Se till att koden ovanför fungerar innan du fortsätter med de kommande uppgifterna. Fråga om hjälp om det inte fungerar. Spara filen!

2. Du ska nu själv få skapa en klass som hanterar böcker. Vad är speciellt med böcker och vad borde man ta in när man skapar ett bok-objekt? När man programmerar är det vanligt (och i stort sätt obligatoriskt) att skriva på engelska, annars kan programmet ge felmeddelanden och inte köra. Kom därför ihåg att skriva era funktioner och klasser på engelska. Om ni är osäkra på ett ord kan ni fråga eller skriva in det i Google translate.

3. (a) Skapa funktioner i klassen som tar ut de parametrar du satt in i klassen som hanterar böcker.

(b) Skapa ett bok-objekt och hämta de parametrar du satt in. Skriv ut parametrarna i terminalen. Spara filen!

I exemplen ovanför har vi skrivit en klass som hanterar fordon, men det finns många olika typer av fordon. Det är därför vanligt att skriva underklasser till en klass som är mer generell för att kunna specificera och hantera ett specialfall av de allmänna begreppen. Några exempel på underklasser till Fordon kan t.ex. vara bil, lastbil, traktor, motorcykel etc. För att skapa en underklass till en mer generell klass skriver man

```
class Car(Vehicle):
    seats = 5

    def __init__(self, regnr):
        super().__init__(regnr)
```

Vi har då skapat en klass som hanterar en bil som en underklass till ett fordon. Vi har här även bestämt att en bil har exakt 5 säten. Klassen Vehicle kallas för en superklass och klassen Car kallas för en underklass. Underklassen måste ta in samma parametrar som superklassen (eller på något sätt definiera dem i underklassen) och skicka dom vidare till superklassen. Detta görs genom att kalla på super() (se exemplet). I underklassen kan man även skapa en funktion som hämtar ut antal säten i bilen.

```
class Car(Vehicle):
    seats = 5

    def __init__(self, regnr):
        super().__init__(regnr)

    def getNumberOfSeats(self):
        return self.seats
```

Vi kan därefter skapa bil-objekt och hämta ut registreringsnummer och antal säten. Vi har här skapat 2 bil-objekt.

```
# Skapar bil-objekt:
car_Fredrik = Car("ABC123")
car_Jonatan = Car("XYZ987")

# hämtar antal säten:
seats_Fredrik = car_Fredrik.getNumberOfSeats()
seats_Jonatan = car_Jonatan.getNumberOfSeats()

# hämtar registreringsnummer:
plate_number_Fredrik = car_Fredrik.getPlateNumber()
plate_number_Jonatan = car_Jonatan.getPlateNumber()

# Skriver ut:
print("Fredrik: seats - ", seats_Fredrik, ", plate number - ", plate_number_Fredrik)
print("Jonatan: seats - ", seats_Jonatan, ", plate number - ", plate_number_Jonatan)
```

4. Skriv in exempelkoden ovanför i samma python-fil som ni skrev klassen till fordon. Testa att det fungerar.

5. Skriv minst 2 underklasser till superklassen böcker. Exempel på underklasser till böcker är barnböcker, faktaböcker och skönlitterära böcker. Kommer du på fler? Välj själv vilka underklasser du vill att superklassen ska ha. Skriv dessa i samma fil som du skrev klassen till böcker.

6. (a) Vad är speciellt med de underklasser du valt att skriva? Måste de ta in några extra parametrar? Har de något som är samma för alla i den underklassen som kan skrivas konstant? Skapa funktioner som hämtar ut dessa.

(b) Skapa flera objekt av de underklasser du nu har programmerat. Hämta ut informationen om objekten och skriv ut dessa.

En underklass kan i sin tur vara en superklass till en annan klass. T.ex. kan en bil ha underklasserna personbil och taxi. En personbil kan bl.a. ha egenskapen att man kan sätta dit en takbox. Vi vill då skapa en funktion som har möjlighet att sätta dit en takbox med ett innehåll och en funktion som kollar om vi har en takbox eller inte, och vad takboxen i sådana fall innehåller. Ett sådant program kan se ut som detta:

```
class PrivateCar(Car):
    item = ""
    def __init__(self, regnr):
        super().__init__(regnr)
    def setRoofbox(self, item):
        self.item = item
    def getRoofbox(self):
        if self.item=="":
            return "No roofbox"
        else:
            return self.item
```

För att skapa ett privatbil-objekt och lägga till och se innehållet i takboxen kan man skriva:

```
private_car_Fredrik = PrivateCar("ABC123")
# Kollar att vi inte har en takbox när vi startar:
roofbox = private_car_Fredrik.getRoofbox()
print(roofbox)
# Lägger till en takbox med innehåll:
private_car_Fredrik.setRoofbox("Skis")
# Kollar om vi har en takbox igen. Denna gång med innehåll.
roofbox = private_car_Fredrik.getRoofbox()
print(roofbox)
```

7. Skriv återigen in exempelkoden i python-filen där du skrev Vehicle och Car. Testa att det funkar.
Extrauppgift: Kan du lägga till något i takboxen när du tidigare redan lagt något i den? Hur löser du problemet? Hur tar du bort takboxen?

8. Skriv underklasser till de underklasser du redan skrivit som hanterar böcker. Vad är speciellt med dessa? Skriv funktioner i de nya underklasserna. Testa att de fungerar och skriv ut all information.

9. (a) Skriv en ny klass som hanterar personer. Vad är speciellt med personer? Skriv denna klassen i en ny fil.
(b) skriv underklasser till personer.
(c) Skriv en funktion som gör att en person kan använda en personbil.

10. (a) Kan klassen böcker, person och/eller fordon ha superklasser?
(b) Ge exempel. Skriv dessa.

11. Har du fler exempel (förutom böcker, personer och fordon) som kan vara klasser och ha underklasser?