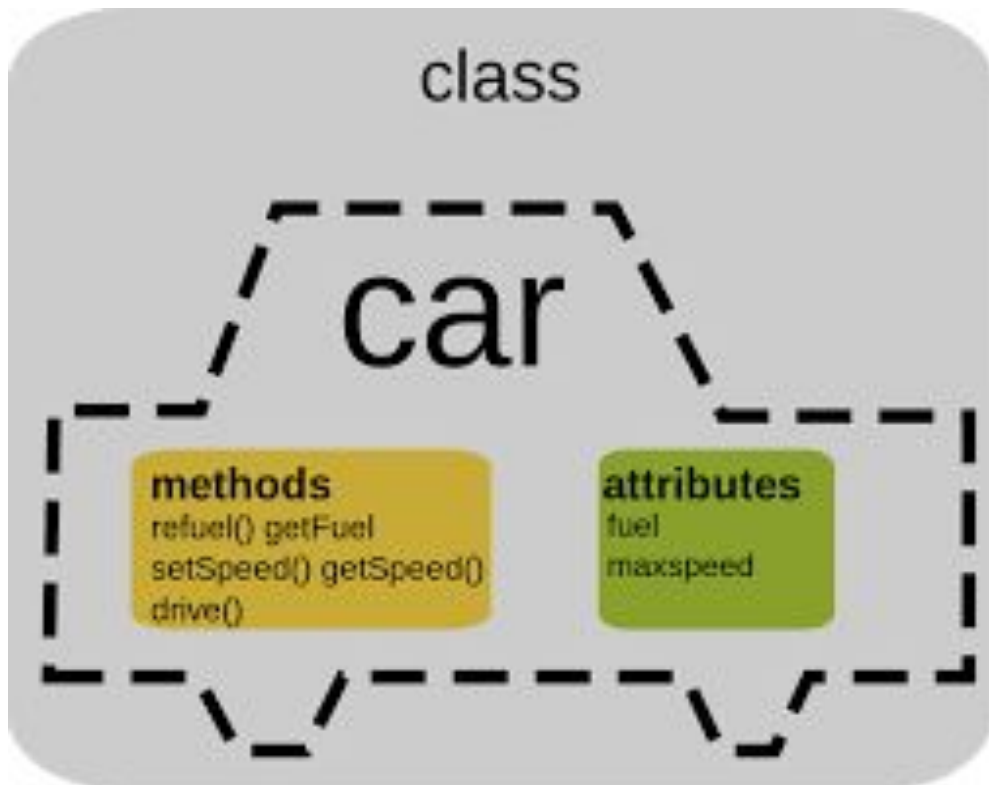


1.3 Grundläggande objektorienterad programmering

Holger Rosencrantz

Klasser definieras bl.a. av *metoder* (beteenden) och *attribut* (egenskaper)



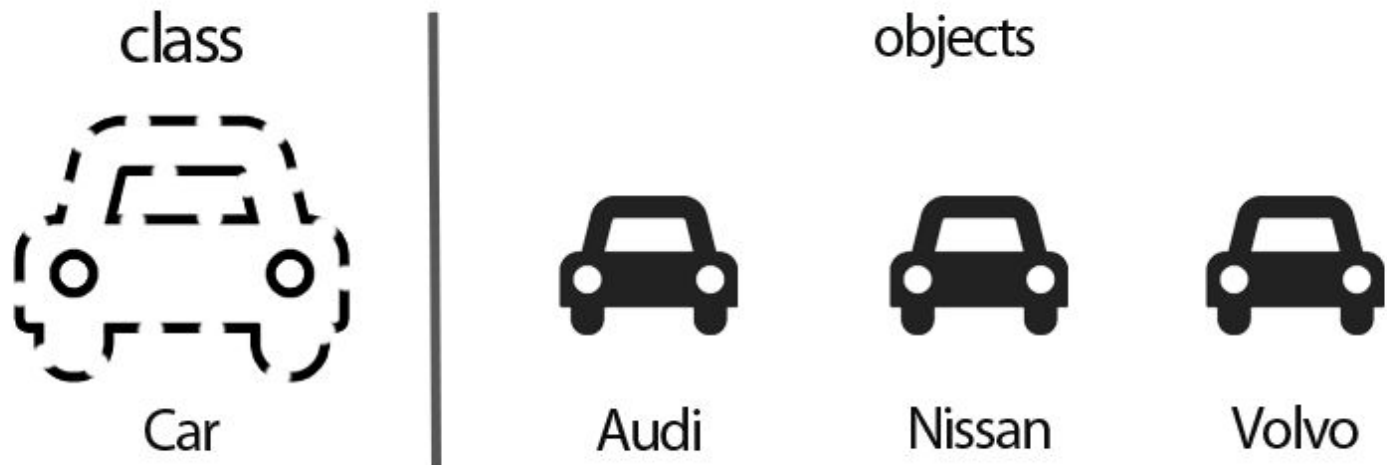
Rafael: Skolan i Aten



Platon pekar upp, Aristoteles pekar ner



Objekt är *instanser* av klasser



Konstruktör

- En metod som skapar och returnerar en ny instans av en klass (dvs. ett nytt objekt)
- Har i Python alltid namnet `__init__`, men anropas med klassens namn
- Varje klass har en default-konstruktör som inte tar några argument och returnerar en instans av klassen som inte har några attributvärden

```
class Bil:
    pass
volvo = Bil()
print(volvo)      # <__main__.Bil object at 0x0000028A6629DFD0>
```

Implementera en klass med en egen konstruktor-metod

```
class Bil:
    def __init__(self, speed, brand):
        self.speed = speed
        self.brand = brand
    def honk(self):
        print("TUUT!!")
    def drive(self):
        print("Bilen körs i", self.speed, "kilometer i timmen.")

bil1 = Bil(50, "Volvo")
bil1.honk()
bil1.drive()
bil2 = bil1          # variabeln bil2 "pekar" på samma objekt som
bil1 - vi har alltså inte skapat något nytt objekt
bil2.speed = 70      # ändrar både bil1 och bil2
print(bil1.speed)    # 70
```

Implementera en klass med en egen konstruktor-metod

```
class Bil:
    def __init__(self, speed, brand):
        self.speed = speed
        self.brand = brand
    def honk(self):
        print("TUUT!!")
    def drive(self):
        print("Bilen körs i", self.speed, "kilometer i timmen.")


bil1 = Bil(50, "Volvo")
bil1.honk()
bil1.drive()
bil2 = bil1          # variabeln bil2 "pekar" på samma objekt som
bil1 - vi har alltså inte skapat något nytt objekt
bil2.speed = 70      # ändrar både bil1 och bil2
print(bil1.speed)    # 70
```

Tilldela attributvärden

Implementera en klass med en egen konstruktor-metod

```
class Bil:
    def __init__(self, speed, brand):
        self.speed = speed
        self.brand = brand
    def honk(self):
        print("TUUT!!")
    def drive(self):
        print("Bilen körs i", self.speed, "kilometer i timmen.")

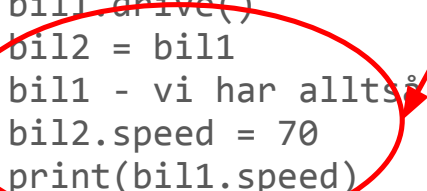

bil1 = Bil(50, "Volvo")
bil1.honk()
bil1.drive()
bil2 = bil1 # variabeln bil2 "pekar" på samma objekt som bil1 - vi har alltså inte skapat något nytt objekt
bil2.speed = 70 # ändrar både bil1 och bil2
print(bil1.speed) # 70
```



Implementera en klass med en egen konstruktor-metod

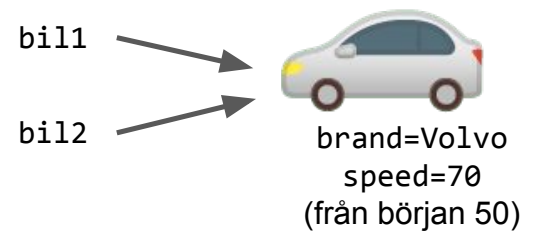
```
class Bil:
    def __init__(self, speed, brand):
        self.speed = speed
        self.brand = brand
    def honk(self):
        print("TUUT!!")
    def drive(self):
        print("Bilen körs i", self.speed, "kilometer i timmen.")

bil1 = Bil(50, "Volvo")
bil1.honk()
bil1.drive()
bil2 = bil1 # variabeln bil2 "pekar" på samma objekt som bil1 - vi har alltså inte skapat något nytt objekt
bil2.speed = 70 # ändrar både bil1 och bil2
print(bil1.speed) # 70
```

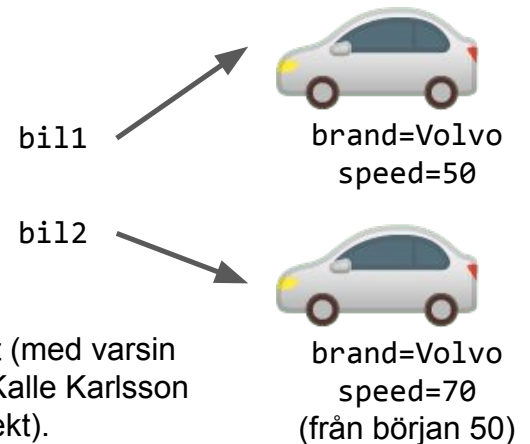


Objektreferens: Två olika variabler kan "peka" på samma objekt

```
bil1 = Bil(50, "Volvo")
bil2 = bil1
bil2.speed = 70
```



```
bil1 = Bil(50, "Volvo")
bil2 = Bil(50, "Volvo")
bil2.speed = 70
```



Variabler refererar till objekt! Det kan även vara så att två olika objekt (med varsin variabel) har samma attributvärden; det kan t.ex. finnas två stycken Kalle Karlsson i telefonkatalogen - de har samma namn, men är olika personer (objekt).

Main-metod och import mellan olika filer

```
from Bil import Bil # kör koden i filen Bil.py
class Motorväg:
    def main():
        minBil = Bil(60, "Saab")
        minBil.drive()
    if __name__ == "__main__": # True ifall det är den aktuella
        # filen som "körs"
        main()
```

- Metoden main körs endast ifall det är den aktuella filen som körs
- Testa att omsluta den föregående koden med ett liknande villkor för att få bort irriterande utskrifter när man importerar en klass

Denna vecka: Övningar objektorienterad programmering

Studera fler exempel från w3schools enligt länkar i lektionsanteckningarna:

- https://www.w3schools.com/python/gloss_python_class_init.asp

Övningar:

1. Gör en klass som heter "Elev". När man anropar konstruktorn tilldelas värden på namn (textsträng) och ålder (heltal) som två klassvariabler.
2. Ändra så att eleven har klassvariabeln glad. Lägg sedan till ett argument i konstruktorn godkänd så att klassvariabeln glad tilldelas värdet True ifall värdet på godkänd är True. Konstruktorns "signatur" ska alltså se ut så här:

```
def __init__(self, namn, ålder, godkänd)
```

Jobba i övrigt med valfria uppgifter på Kattis och ev. progolymp.se

Pusha ditt arbete i slutet av lektionen till GitHub

Statiska metoder och variabler

- En statisk metod eller variabel tillhör en klass och är tillgänglig utan att man behöver skapa något objekt
- Vanligen är metoder och attribut inte statiska, vilket innebär att man får felmeddelande ifall man anropar dem utan att först ha skapat ett objekt

icke-statiska metoder kräver att man först skapar ett objekt

```
class Bil:
    def honk(self):          # en Bil-metod
        print("TUUT!!")
honk()                      # fel!!
Bil.honk()                  # också fel!!
```

Exempel statisk metod

```
class Bil:
    @staticmethod
    def milestokm(miles):
        return 1.6093*miles
print(Bil.milestokm(15))    # 24.1395
```

Statiska metoder används lämpligen för t.ex. allmänna beräkningar och liknande hjälpmetoder som implementeras på samma sätt för alla objekt

Icke-statiska metoder bör användas ifall det är ett enskilt objekt som "gör något" (t.ex. en enskild bil som tutar)

Exempel statiskt attribut: räkna antalet instanser av en klass

```
class Bil:
    antalBilar = 0
    def __init__(self):
        Bil.antalBilar += 1
bil1 = Bil()
bil2 = Bil()
bil3 = Bil()
print(Bil.antalBilar)    # 3
```


Åtkomstnivåer: public, protected och private

Access Modifiers	Same Class	Same Package	Sub Class	Other Packages
<i>Public</i>	Y	Y	Y	Y
<i>Protected</i>	Y	Y	Y	N
<i>Private</i>	Y	N	N	N

Exempel privat attribut

```
class Person:
    __bmi = 0          # privat attribut
    def __init__(self, weight, height):
        self.weight = weight
        self.height = height
        self.__bmi = weight/(height**2)
    def getBmi(self):   # publik åtkomstmetod, s.k. getter-metod
        return self.__bmi

p = Person(70, 1.78)
print(p.getBmi())
print(p.__bmi)        # FEL - variabeln __bmi är privat!
```

Publika getter- och setter-metoder till privata attribut

```
class Person:
    __weight = 0                # privat attribut
    def setWeight(self, weight): # publik setter-metod
        if type(weight) == int and weight > 0:
            self.__weight = weight
        else:
            print("Du måste ange ett positivt heltal!")
    def getWeight(self):        # publik getter-metod
        return self.__weight

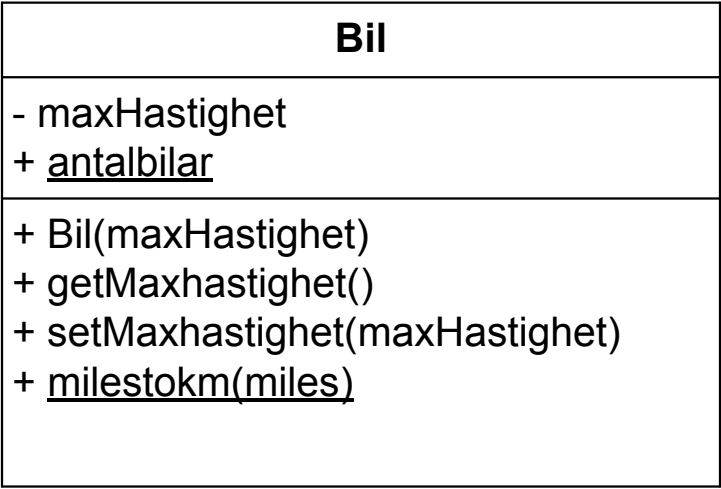
p = Person()
p.setWeight(-2)                # Du måste ange ett positivt heltal
p.setWeight(70)
print(p.getWeight())           # 70
```

Privata metoder

```
class Person:
    def __calculateBmi(self, weight, height): # privat metod
        return weight/(height**2)
    def __init__(self, weight, height):
        bmi = self.__calculateBmi(weight, height)
        print("Ditt BMI är", bmi)

p = Person(70, 1.78)           # Ditt BMI är 22.09...
p.__calculateBmi(70, 1.78)    # FEL - kan ej anropa privat metod
```

Klassdiagram



Klassdiagram

