Klasser

- Tidigare har vi hanterat objekt av olika slag, t.ex. turtle-objekt via modulen **`turtle`**. I modulen finns pythonkoden som definierar ***klassen*** Turtle. Skall nu beskriva hur egna klasser definieras.

- Objekt kan ha både *egenskaper* (data) och *metoder*. Ett turtle-objekt, till exempel, har bl.a. egenskaperna *position*, *riktning*, *storlek* och *färg* (som lagras i variabler) samt metoder, d.v.s. en sorts funktioner, som hämtar information (exvis data om *position*) om eller påverkar ett speciellt objekt (exvis metoderna *forward*, *left*, ...).

- En klass kan sägas vara en ritning eller beskrivning av vilka data som objekten ska ha och vilka metoder som ska finnas. Beskrivningen är skriven i pythonkod.

Exempel klassen Turtle

```
t1 = turtle.Turtle()
t2 = turtle.Turtle()
```

→ Skapar Turtle-objekten `t1` resp `t2`

```
t1.goto(100,100)
t2.goto(-100,20)
```

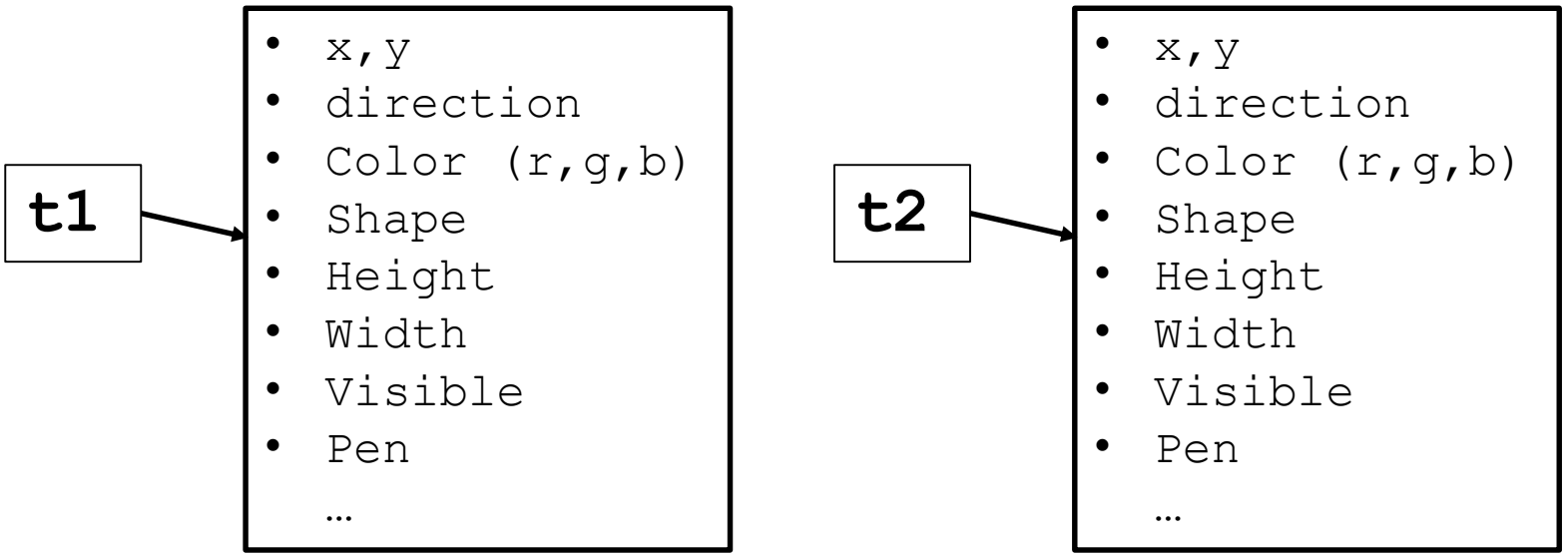
→Uppdaterar pos för `t1` resp `t2`

```
x = t1.xcor()
```

→ Ta reda på xcoord för `t1`

```
d = t1.distance(t2)
```

→Beräkna avstånd `t1` vs `t2`



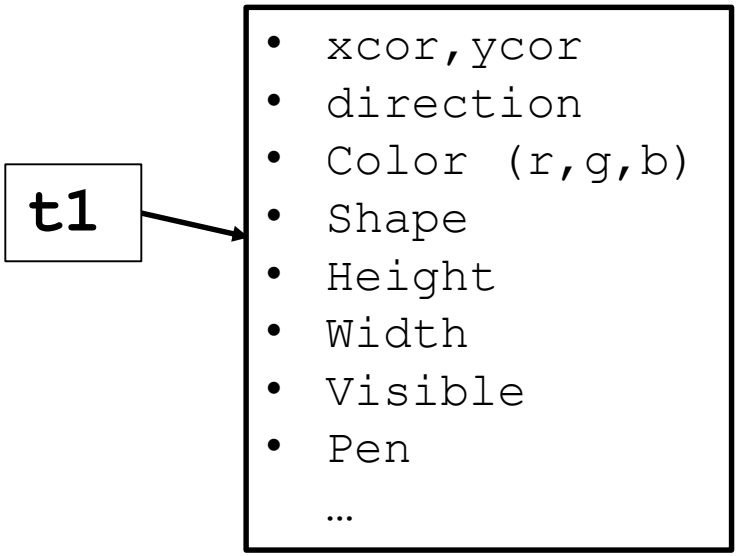
Ett Turtle-objekt refererar till många variabler som definierar objektets egenskaper (data).

Det finns speciella ”funktioner” som man kan anropa dessa objekt för. Dessa ”funktioner” kallas för ***metoder***.

Objekten anropas av metoderna med punknotation:

```
t1.goto(100,100)
x = t1.xcor()
```

objekt, punkt, metodnamn, ev. argument



Följande händer vid metodanropen:

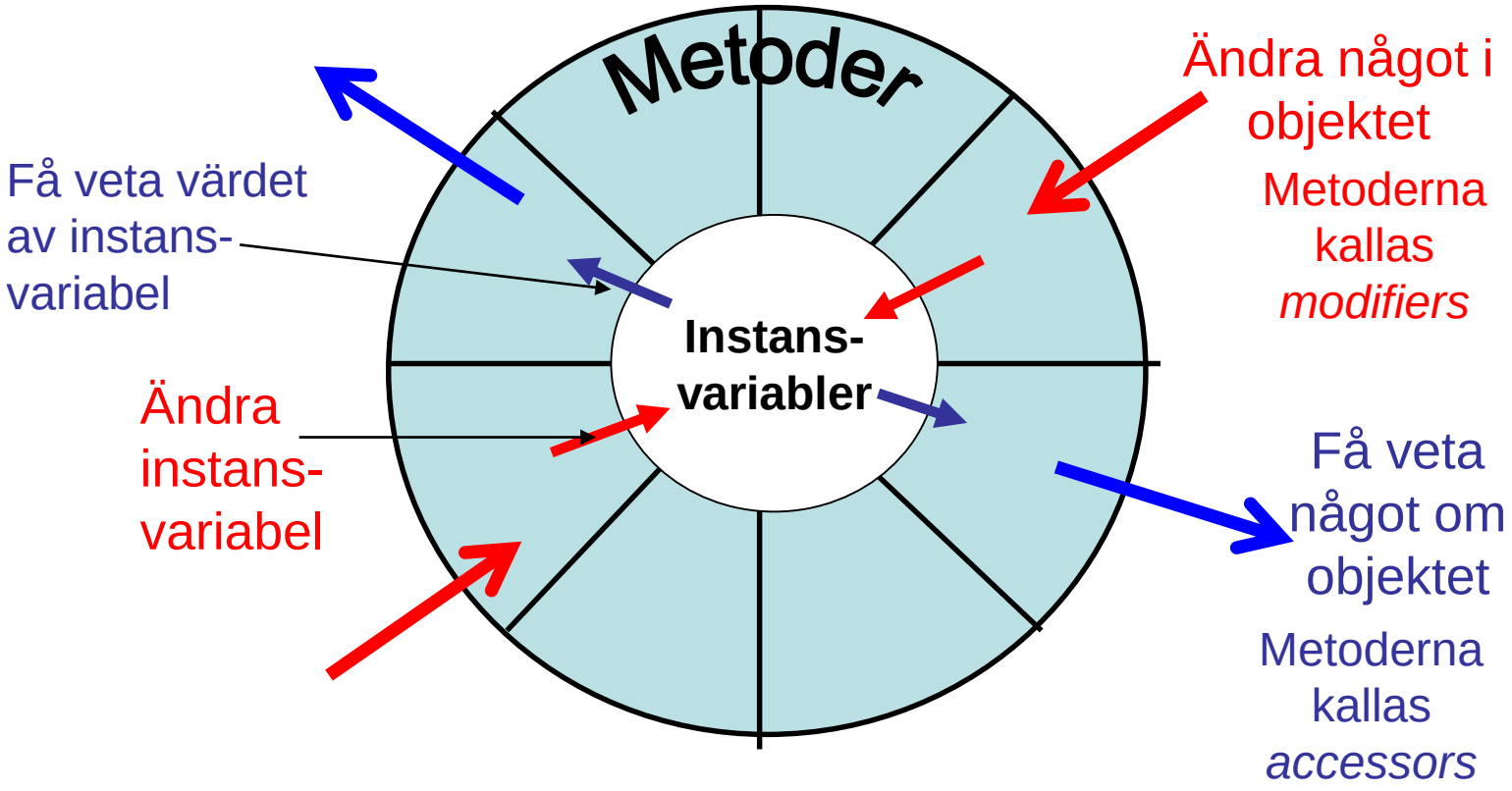
- `t1's` data, nämligen variablerna `xcor, ycor` kommer att uppdateras till (100,100)
- Variabeln `x` får värdet av `t1's xcor`

Objektens variabler kallas ***instansvariabler***.

Man paketerar allt, instansvariabler och metoder med pythonkod i en s.k. ***klass***.

Detta kallas också för klassens ***definition***.

- Till ett objekt hör metoder och instansvariabler.
- All kommunikation med objektet sker via metoder.
- Endast objektet vet om sina instansvariabler.
- Via metoderna kan man ändra instansvariablerna och få veta dess värden.



Ett exempel vi skall designa en klass BankAccount

För att hantera ett enkelt bankkonto behöver vi uppgift (data) om kontonumret och saldot.

Kan kanske lösas med två st enkla variabler så här:

```
balance = 10000.0      # saldot
accountNumber = 'ABC-123'
```

Hur gör vi om vi har flera bankkonton?

Införa många variabler: `balance1, balance2,…`

och `accountNumber1, accountNumber2, …`

Det blir väldigt knöligt och om vi tänker efter, egentligen ohållbart. Exvis söka efter ett visst kontonummer.

Bättre: En lista för saldo resp för kontonummer ...

```
balance = []
accountNumber = []
```

De båda listorna är nu tomma.
Skapa konton:

```
balance.append(999.0)
accountNumber.append('123456-33')
```

Första kontot

```
balance.append(100.0)
accountNumber.append('955521-46')
```

Andra kontot

Men, några problem:

- Ett specifikt bankkonto består av två element i varsin lista. Det är indexet (positionen) i listan som kopplar ihop dessa två element.

- Tänk om man önskar ha mer info om ett bankkonto, ex.vis kreditgräns, kontotyp, etc → Ger fler listor att hålla reda på.

- Hur löser man uttag och insättning på kontona på ett säkert sätt? Exvis göra felaktiga uttag

Bättre med lista med listelement

```
bankAccount = [ ['123456-33',999.0] ]

bankAccount.append( ['955521-46', 100.0] )
```

Eller ett lexicon (dictionary):

```
bankAccount = {}

bankAccount['123456-33'] = 999.0

bankAccount['955521-46'] = 100.0
```

Bättre, en ”variabel” som representerar ett bankkonto, och som refererar till två värden: saldot och kontonumret.

Denna variabel skall vi kunna hantera med sådant som associeras med ett bankkonto. Den bör också vara skyddad mot felaktig användning.

Vi designar/skriver en klass BankAccount med vilken vi kan skapa BankAccount-variabler som representerar ett bankkonto. Dessa variabler kallar vi BankAccount-objekt.

Vilka data skall ett bankkonto ha koll på?
Jo, saldo och kontonummer, vi kallar dem lämpligen

- `balance`, (en float)
- `accountNumber`, (en sträng)

Detta definierar klassens s.k. *instansvariabler* (eller *data*)

Hur skapar vi ett BankAccount- objekt? På vilket (vilka) sätt önskar vi kunna skapa bankkonton?

Detta bestämmer hur de s.k. *konstruktorerna* skall se ut. Konstruktorerna är speciella metoder som skapar och initierar objekt. (kallas även för initieringsmetoder)

- Skapa ett bankkonto där vi bestämmer kontonumret och sätter saldot till 0 kr (eller en där man preciserar startsaldot)
- Något annat sätt att skapa skapa ett bankkonto? Nej...

Dvs vi behöver bara EN *konstruktor.*

```
b1 = BankAccount('5511-15')
```

Jämför med: `t = turtle.Turtle()`

Denna konstruktor har inga parametrar. Det blir en standard turtle

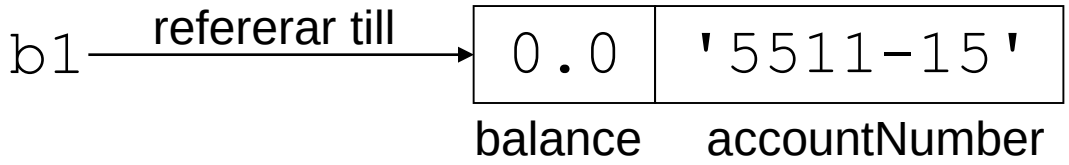
Modulen (filen) där klassen Turtle är definierad

Anrop av konstruktorn i klassen Turtle

Att skapa ett BankAccount-objekt

```
b1 = BankAccount('5511-15')
```

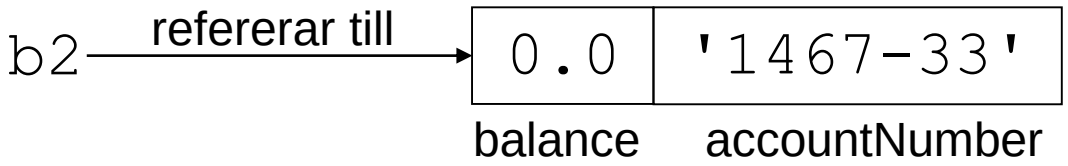
`b1` blir ett objekt av typen `BankAccount`



`b1` refererar till sina instansvariabler: `balance, accountNumber`

Skapa ett annat BankAccount-objekt

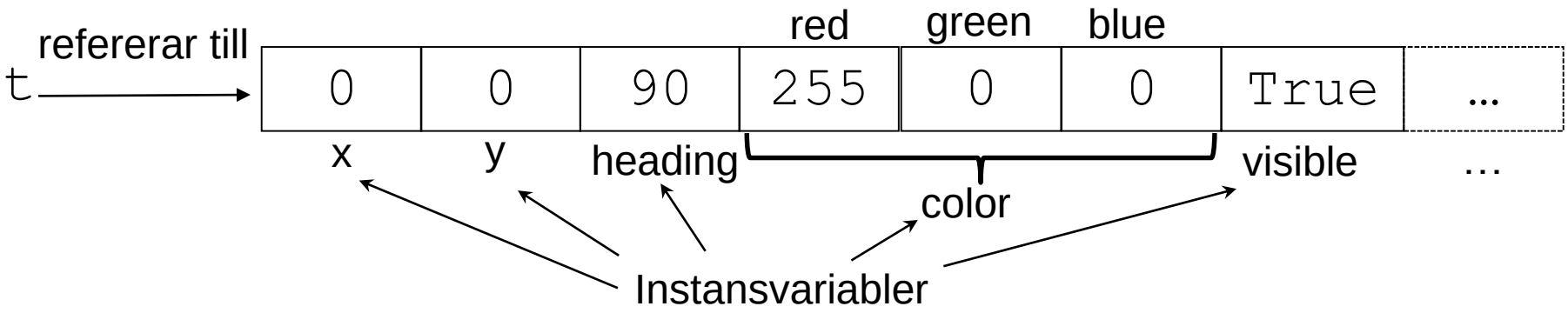
```
b2 = BankAccount('1467-33')
```



Varje objekt har sin egen uppsättning instansvariabler som den har kontroll över.

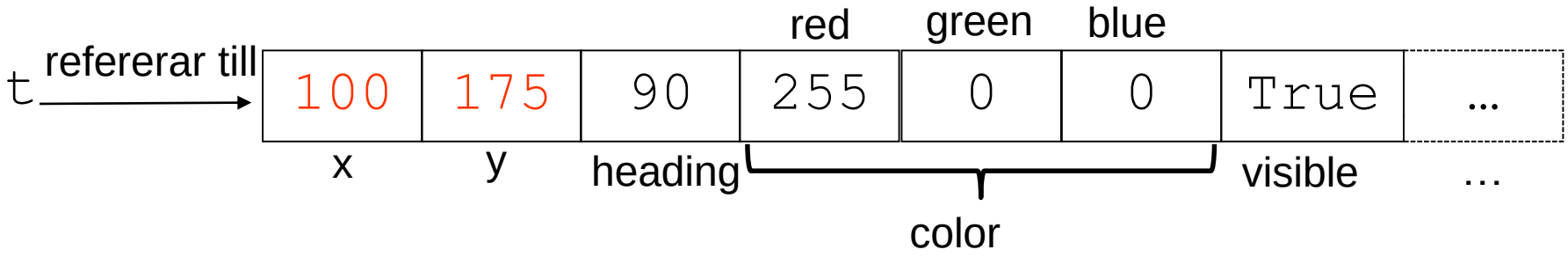
Jämför:

`t = turtle.Turtle() # t` är ett objekt av typen `Turtle`




```
t.goto(100,175) # Flytta paddan
```

Notera punktnotationen: Anropa metoden `goto` med två argument (den nya positionen) för de instansvariabler som `t` refererar till. Metoden ändrar x,y



Nu har vi definierat vilka data ett BankAccount-objekt skall ha.

Men vad vi vill kunna göra med ett bankkonto?

Detta definierar vilka *metoder* klassen skall ha

- Ta ut pengar
- Sätta in pengar
- Få veta saldot
- Få veta kontonumret

} 4 metoder

- (Ändra kontonumret, Nej, behövs ej!)

Metod som ändrar instansvariabel kallas *modifier*

Metod med vilken man får veta värdet på instansvariabel kallas *accessor*

Vilken typ är ovan fyra metoder?

Döp metoderna och bestäm parametrar samt returvärde

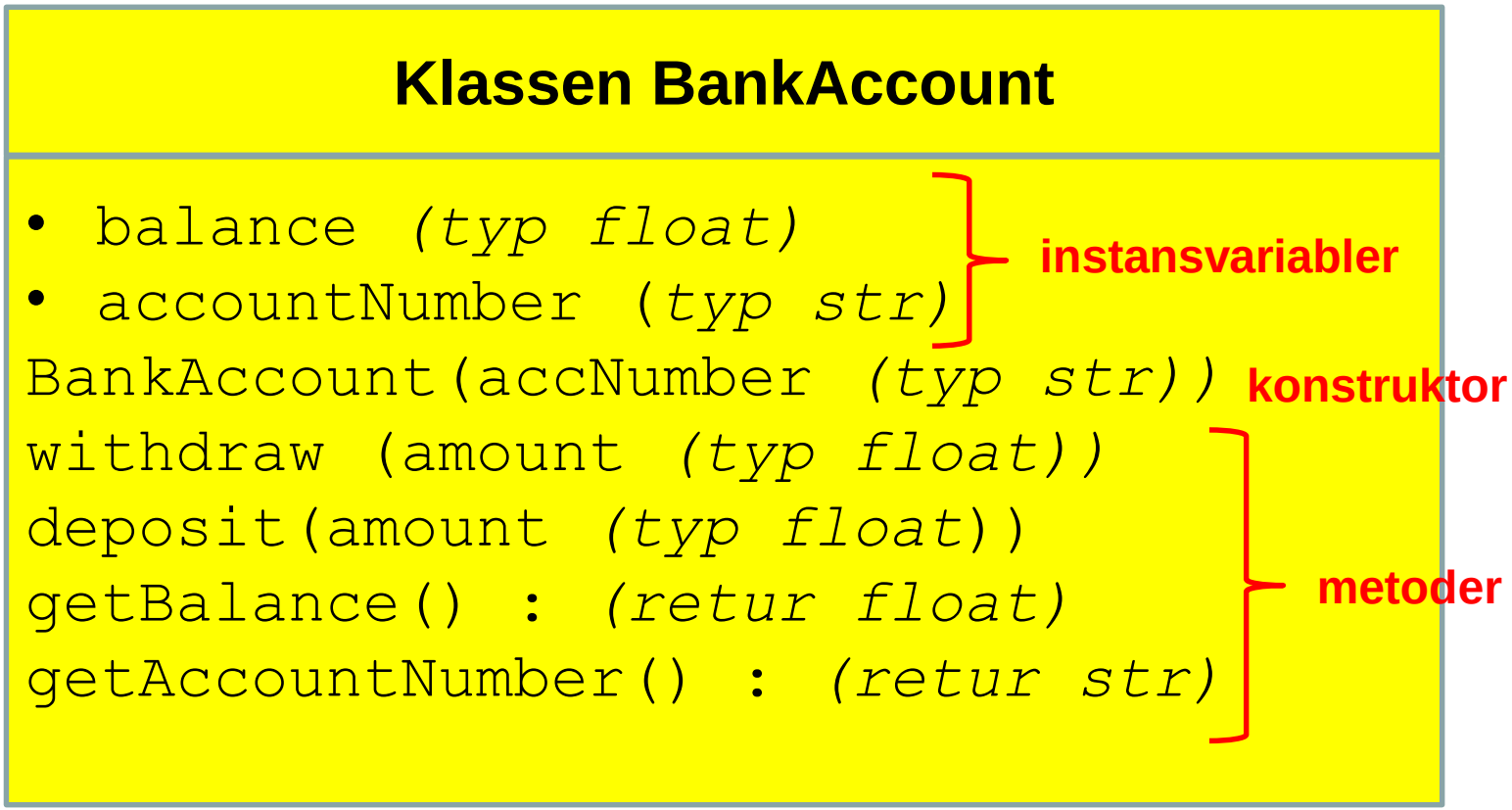
- Ta ut pengar :
  Namn: withdraw
  Parametrar? Returvärde?
  amount **Nej**

- Sätta in pengar
  Namn: deposit
  Parametrar? Returvärde?
  amount **Nej**

- Få veta saldot
  Namn: getBalance
  Parametrar? Returvärde?
  Nej **Ja, *balance***

- Få veta kontonumret
  Namn: getAccountNumber
  Parametrar? Returvärde?
  Nej **Ja, *accountNumber***

Så här kan klassen BankAccount sammanfattas.

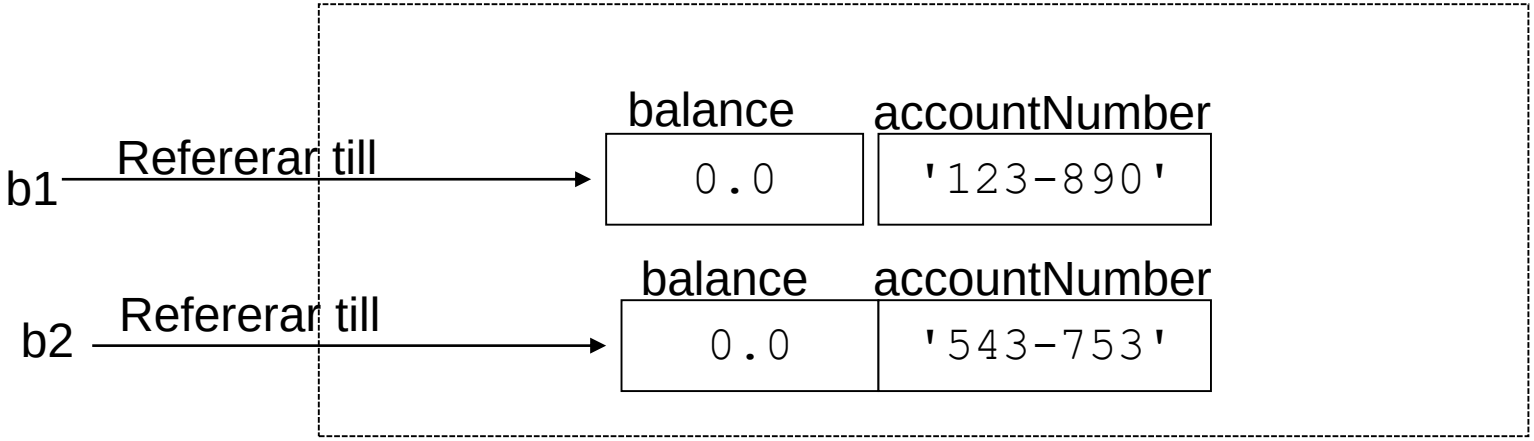


- Instansvariabeln `balance` skall ha typen `float`.
- Instansvariabeln `accountNumber` skall ha typen `str`.
- Konstruktorn skall ha en parameter av typen `str`.
- Metoderna `withdraw` och `deposit` skall ha en parameter av typen `float`. Båda är *modifiers,* ty de ändrar instansvariabler, dvs ändrar objektet.
- Metoderna `getBalance` och `getAccountNumber` skall ej ha någon parameter och returnera en `float` respektive en `str`. Båda är *accessors,* ty de skickar ut data från objektet.

Pythonsatser som skapar två BankAccount-objekt

```
b1 = BankAccount('123-890')
b2 = BankAccount('543-753')
```

Ser ut så här i minnet



Vid anropet av konstruktorn reserveras minnesutrymme som behövs för instansvariablerna som får värden
Vi kommer åt instansvariablernas värden via metoderna

Vi studerar klassen BankAccount som finns i pythonfilen `BankAccount.py` som finns att ladda ned från kurswebben. Testa att köra med Run. Vad händer?

Studera programmet `Test_BankAccount.py` som använder klassen `BankAccount`. Finns att ladda ned från kurswebben. Testkör programmet.

Vi ser att de två sista satserna i `Test_BankAccount`,dvs utskrifterna av `b1` och `b2` ger konstigt resultat. Återkommer till detta.

Ändra `Test_BankAccount` så att vi på ett konto tar ut för mycket pengar. Vad händer?

Vi ändrar i programmet `BankAccount` så att man ej kan ta ut pengar som det ej finns täckning för.

Vad kan ändras?

Kan klassen ”meddela” på något sätt att man försöker ta ut för mycket?

Vi ändrar programmet `Test_BankAccount` så att det tar hänsyn till att klassen är ändrad. Programmet skall skriva ut en text huruvida det gick att ta ut pengar eller inte.

Vi såg tidigare att de två sista satserna i `Test_BankAccount`, dvs utskrifterna av `b1` och `b2` ger konstiga resultat, det blev:

```
b1: <BankAccount.BankAccount object at 0x03C98DC0>
b2: <BankAccount.BankAccount object at 0x03EE6598>
```

Python vet ej hur objektet skall skrivas ut, därför skrivs det att objekten är `BankAccount`-objekt och att klassen är definierad i filen BankAccount samt objektens referens (minnesadress).

Vi skall nu definiera metoden `__str__` som ser till att man får en mer informativ utskrift. Vi kikar i klassen `BankAccount.py` där metoden finns längst ned.

- Skulle man kunna slå ihop metoderna `withdraw` och `deposit`?

- `changeAmount(amount)`:
  om `amount < 0`: ta ut
  om `amount > 0`: sätt in

Vad händer vid anropen…

Skapa ett BankAccount-objekt, dvs anrop av init-metoden

```
b1 = BankAccount('123-xxx')
```

Kopieras till parametern `accNumber`

```
def __init__(self, accNumber):
    self.accountNumber = accNumber
    self.balance = 0
```

`self` refererar till det aktuella objektet

Det är `self` som gör att vi kan komma åt värdena som vi skapar i objekten.

Anrop av metoden deposit…