
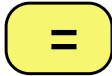





Java

Objekt, klasser, syntax

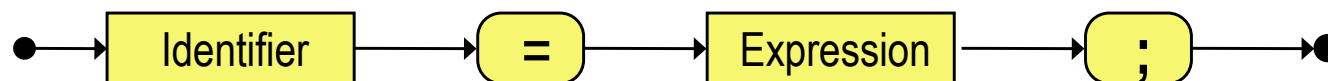
Syntaxdiagram

- Start på en icketerminal 
- Terminal symbol 
- Möjlig väg att följa 
- En annan icketerminal 
- Slut på beskrivning av icketerminal 

Tilldelning av värden

- En variabel innehåller antingen ett *primitivt värde* eller en *referens* till ett objekt
- Referenser som inte “refererar” till ett objekt har värdet `null`.
- Attribut sätts automatiskt till `null/0` övriga variabler måste initieras manuellt.
- En variabel tilldelas ett (nytt) värde genom =

Basic assignment



Lokala variabler

- Attribut/*fields* är en sorts variabler
- De lagrar värden hela objektets “liv”
 - De kan nås i hela klassen.
- Metoder kan innehålla variabler med kortare livslängd
 - De existerar bara medan metoden exekveras
 - De kan bara nås inne i metoden (i det block där de är definierade)
 - Lokala variabler

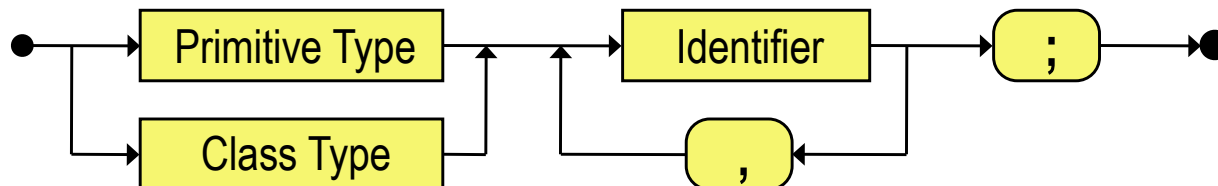
Identifierare och variabler

- Identifierare är namn på olika storheter som definieras av programmeraren
- Lite andra regler gäller i Java än i C
- Identifierare får innehålla bokstäver (då Java använder Unicode så är fler bokstäver än a-z tillåtna som tex åäö), siffror, understrykningstecknet (), och valuta tecknen (t ex dollar tecknet)
 - Jag rekommenderar att ni ej använder andra bokstäver än a-z och namnger era identifierare på engelska
- Identifierare får ej inledas med en siffra
- Java är *case sensitive*, dvs `Total` är olika `total`

Variabler och datatyper

- Variabler används för att spara data
- Variabler måste *deklareras* med datatyp och namn innan de får användas

Basic Variable Declaration



- En primitiv datatyp kan vara heltal (*byte, short, int, long*), flyttal (*float, double*), tecken (*char*) eller *boolean*
- En klasstyp är ett namn på en klass (t ex `Triangle`)

Primitiva datatyper: Hel- och flyttal

- De olika heltals och flyttals typerna har olika storlek. I Java är storlekarna definierade enligt följande:

Typ	Storlek	Minimum	Maximum
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
float	32 bits	+/- 3.4×10^{38} med 7 signifikanta siffror	
double	64 bits	+/- 1.7×10^{308} med 15 signifikanta siffror	

Primitiva datatyper: Tecken

- En variabel av datatypen `char` sparar ett **Unicode tecken**
- Värdemängden är en ordnad uppräkningslista av tecken
- Det finns 65,536 unika Unicode tecken (16 bit) med tecken och symboler från olika språk, t ex 'å' och 'ö'
- Alla tecken är ordnade och varje tecken motsvarar en siffra (detta underlättar att kolla om ett tecken ligger t ex mellan 'a' och 'z')
- Se t.ex. <http://www.unicode.org/> för detaljer

Primitiva datatyper: boolean

- Datatypen boolean har bara 2 värden
 - true
 - false
- Används på alla ställen där sanningsvärden behövs (tex ger jämförelseoperatorerna tillbaka resultat tillhörande denna datatyp)

Objekt

- Hur får vi tillgång till objekt?

```
Triangle bigYellowTriangle;  
Triangle smallBlueTriangle;  
bigYellowTriangle = ???
```

Konstruktörer

```
public TicketMachine(int ticketCost) {  
    price = ticketCost;  
    balance = 0;  
    total = 0;  
}
```

- Konstruktörer ansvarar för att initialisera ett objekt.
 - Alltså ge objektet ett starttillstånd.
- De har samma namn som klassen.
- De lägger in startvärden på attributen.
- Startvärden till attributen kan fås utifrån via parametrar.

new

- Operator
- Skapar nya objekt
- Ex

```
-Triangle t=new Triangle();
```

Referenser vs värden

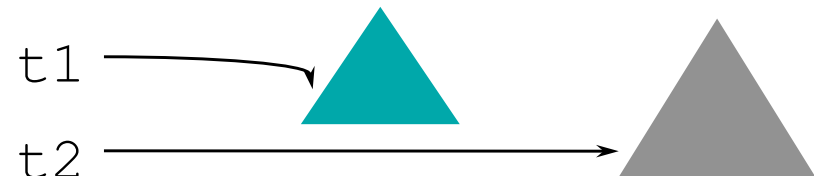
- Tilldelning har lite olika effekt för primitiva datatyper och klasstyper!
 - Primitiva datatyper: Variabeln innehåller själva värdet
 - Klasstyper: Variabel är referens till objektet (jämför med pekare i C)

```
int i1 = 5, i2 = 10;
```

i1: 5 i2: 10

```
i2 = i1;
```

```
Triangle t1 = new Triangle();  
Triangle t2 = new Triangle();
```

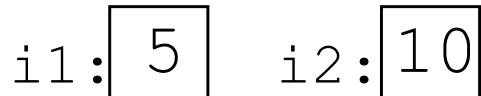


```
t2 = t1;
```

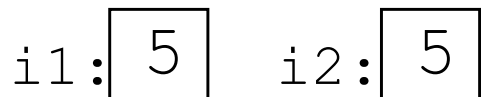
Referenser vs värden

- Tilldelning har lite olika effekt för primitiva datatyper och klasstyper!
 - Primitiva datatyper: Variabeln innehåller själva värdet
 - Klasstyper: Variabel är referens till objektet (jämför med pekare i C)

```
int i1 = 5, i2 = 10;
```



```
i2 = i1;
```



```
Triangle t1 = new Triangle();  
Triangle t2 = new Triangle();
```

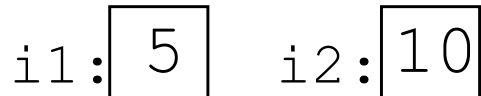


```
t2 = t1;
```

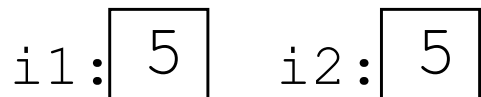
Referenser vs värden

- Tilldelning har lite olika effekt för primitiva datatyper och klasstyper!
 - Primitiva datatyper: Variabeln innehåller själva värdet
 - Klasstyper: Variabel är referens till objektet (jämför med pekare i C)

```
int i1 = 5, i2 = 10;
```



```
i2 = i1;
```



```
Triangle t1 = new Triangle();  
Triangle t2 = new Triangle();
```



```
t2 = t1;
```

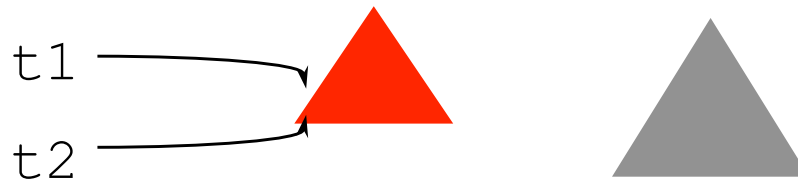


Alias och garbage collection

```
Triangle t1 = new Triangle();  
Triangle t2 = new Triangle();
```

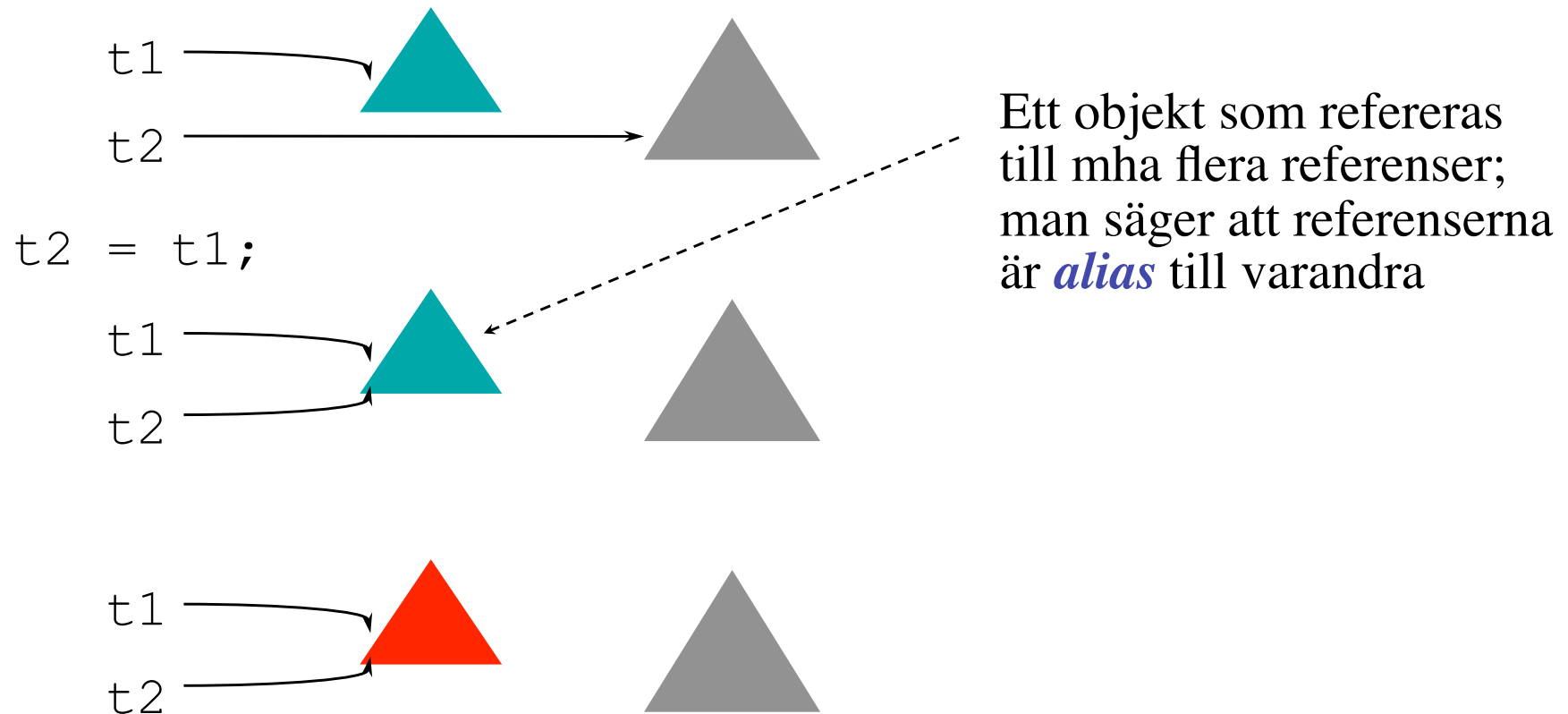


```
t2 = t1;
```



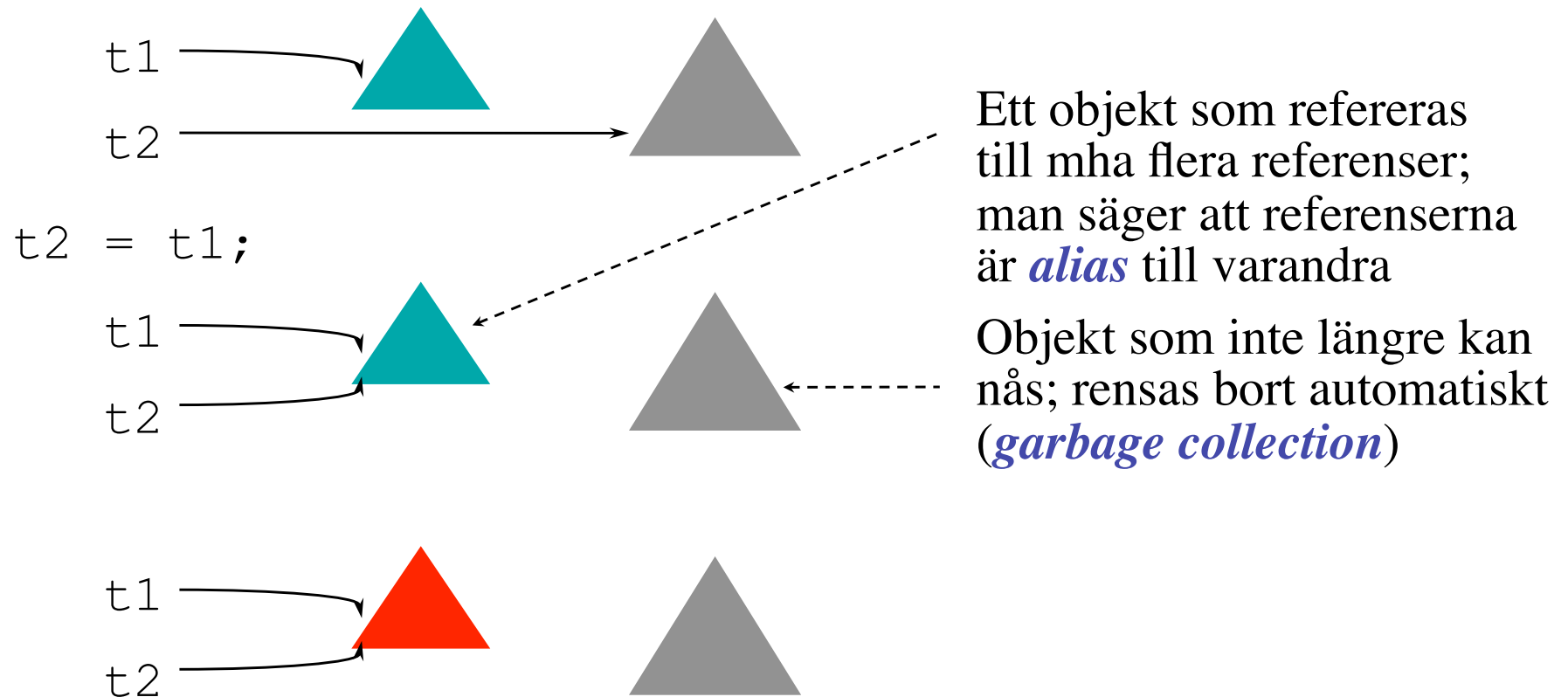
Alias och garbage collection

```
Triangle t1 = new Triangle();  
Triangle t2 = new Triangle();
```



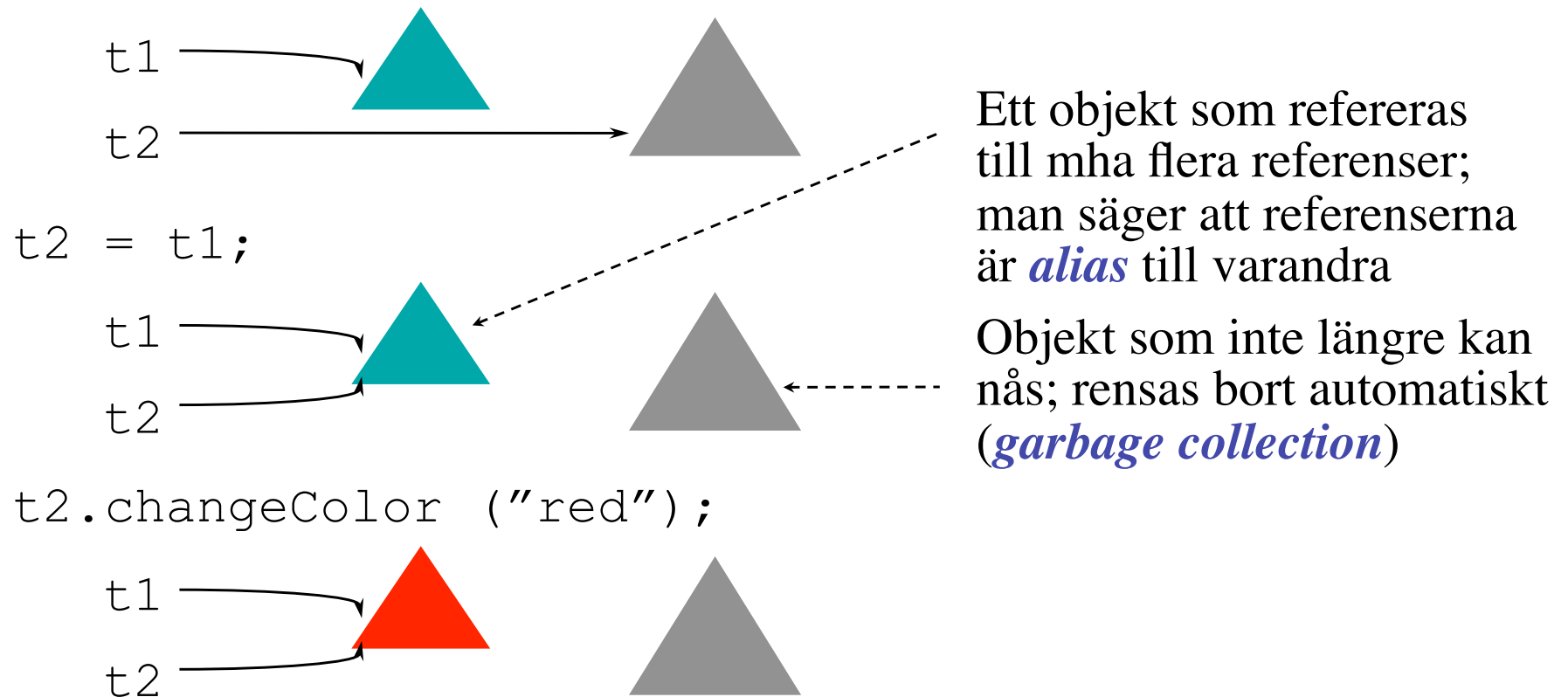
Alias och garbage collection

```
Triangle t1 = new Triangle();  
Triangle t2 = new Triangle();
```



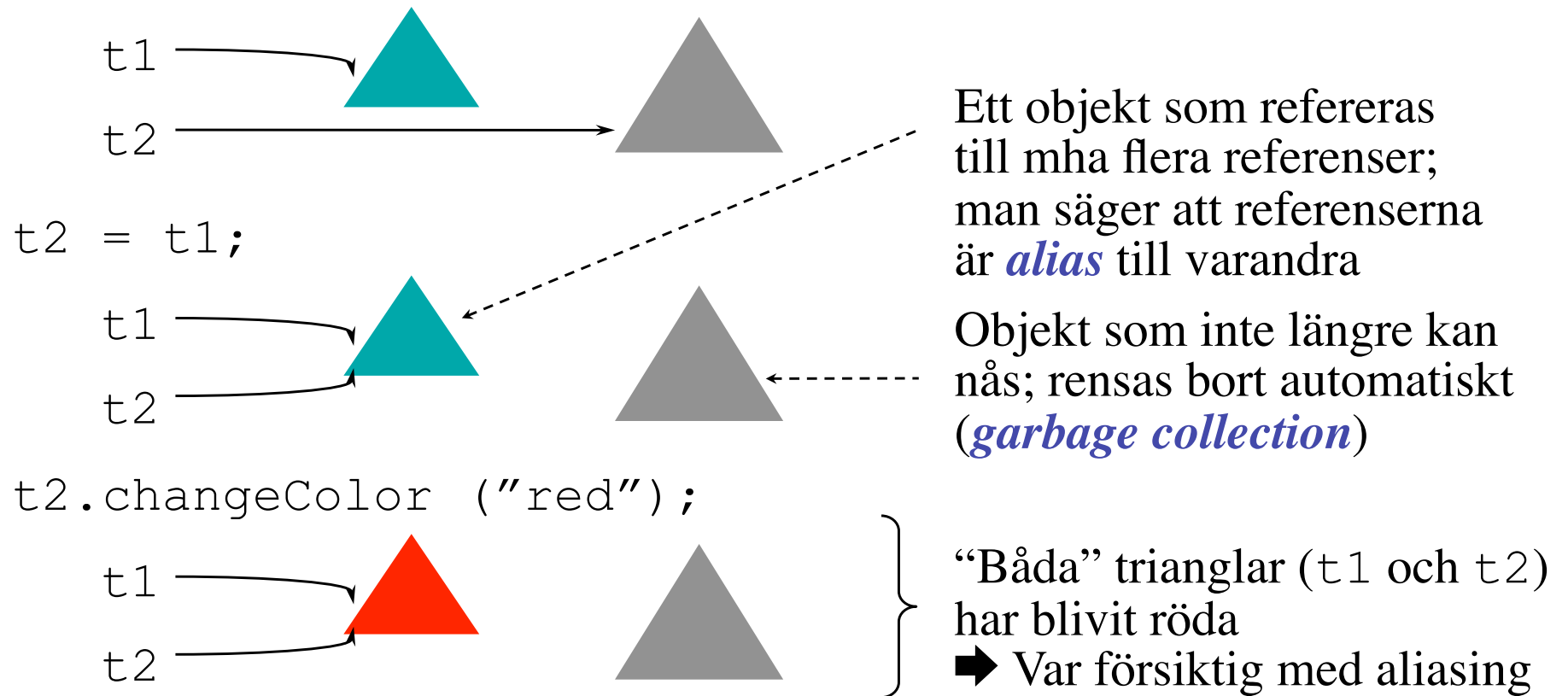
Alias och garbage collection

```
Triangle t1 = new Triangle();  
Triangle t2 = new Triangle();
```



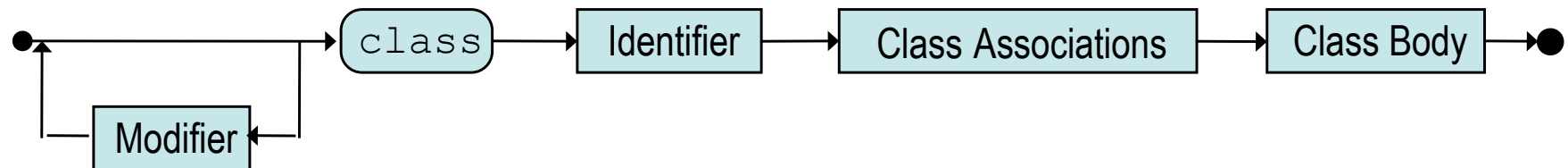
Alias och garbage collection

```
Triangle t1 = new Triangle();  
Triangle t2 = new Triangle();
```

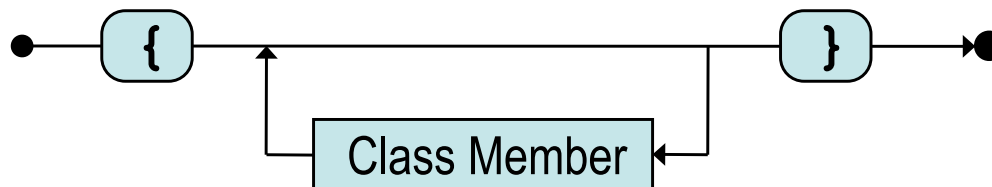


Klassdeklaration

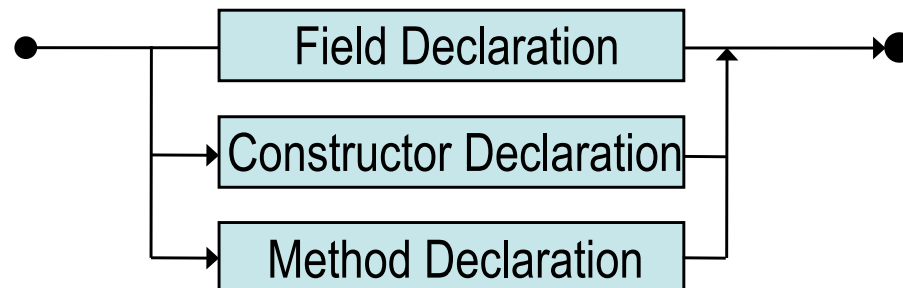
Class Declaration



Class Body

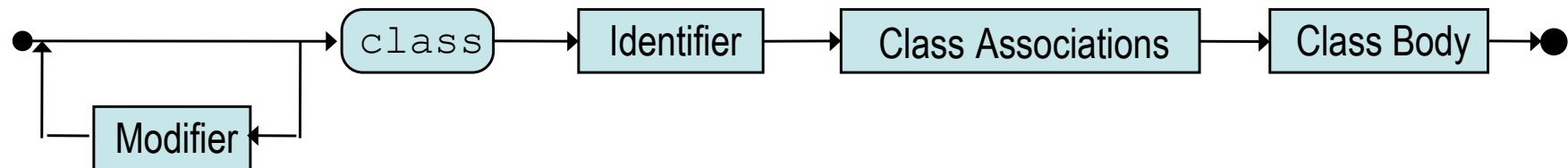


Basic Class Member

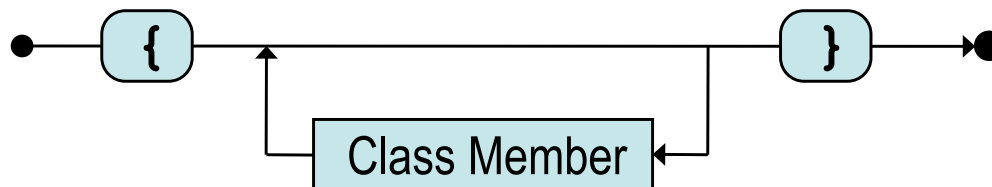


Klassdeklaration

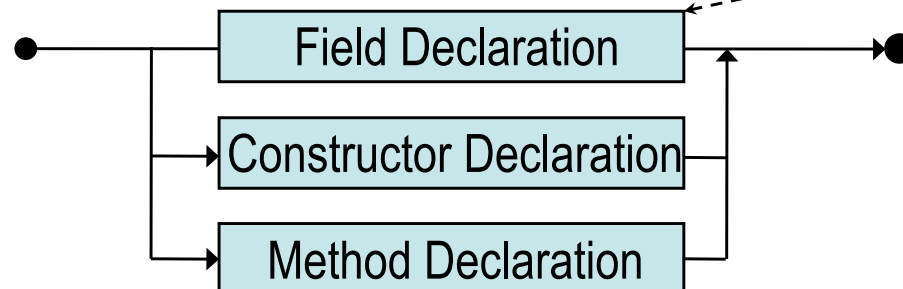
Class Declaration



Class Body



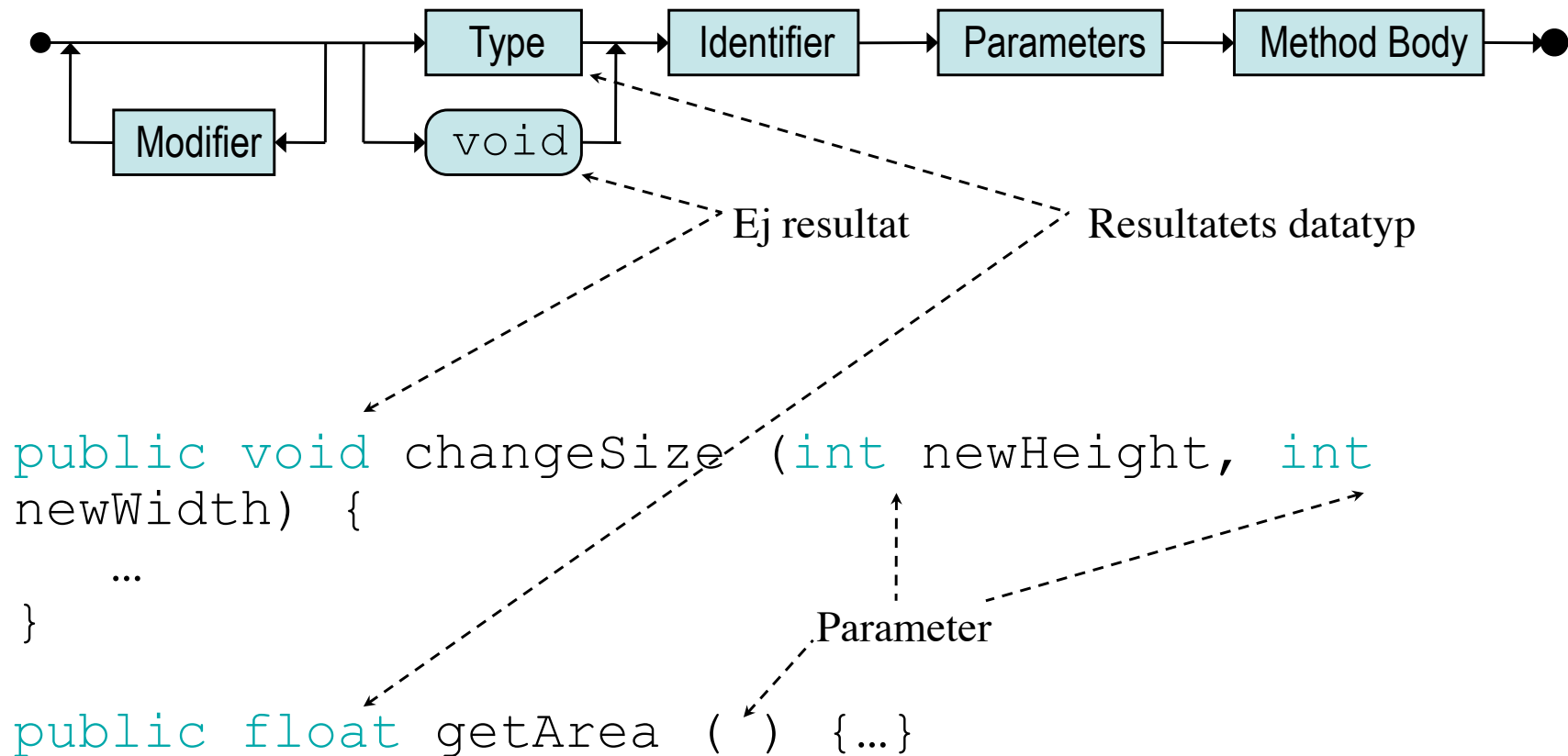
Basic Class Member



Motsvarar variabel
deklaration

Metoddeklaration

Basic Method Declaration



Instantiering och manipulation

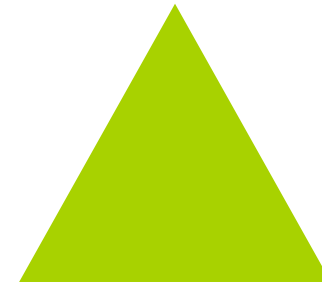
Triangle
height width xPosition yPosition color
changeColor changeSize move ...

```
Triangle aTriangle = new Triangle();
```


Instantiering och manipulation

Triangle
height width xPosition yPosition color
changeColor changeSize move ...

```
Triangle aTriangle = new Triangle();
```

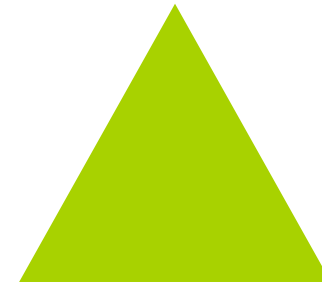


Instantiering och manipulation

Triangle
height width xPosition yPosition color
changeColor changeSize move ...

```
Triangle aTriangle = new Triangle();
```

```
aTriangle.move();
```

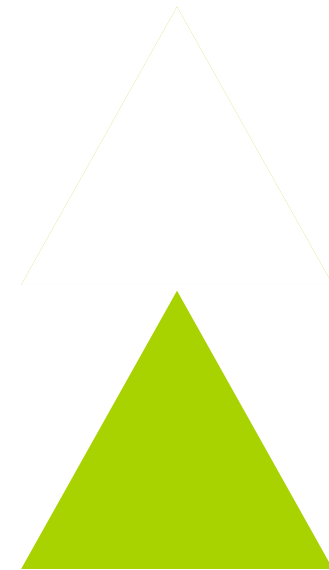


Instantiering och manipulation

Triangle
height width xPosition yPosition color
changeColor changeSize move ...

```
Triangle aTriangle = new Triangle();
```

```
aTriangle.move();
```



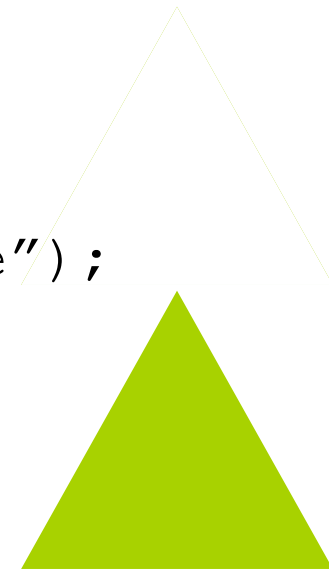
Instantiering och manipulation

Triangle
height width xPosition yPosition color
changeColor changeSize move ...

```
Triangle aTriangle = new Triangle();
```

```
aTriangle.move();
```

```
aTriangle.changeColor("blue");
```



Instantiering och manipulation

Triangle
height width xPosition yPosition color
changeColor changeSize move ...

```
Triangle aTriangle = new Triangle();
```

```
aTriangle.move();
```

```
aTriangle.changeColor("blue");
```



Instantiering och manipulation

Triangle
height width xPosition yPosition color
changeColor changeSize move ...

```
Triangle aTriangle = new Triangle();
```

```
aTriangle.move();
```

```
aTriangle.changeColor("blue");
```

```
aTriangle.move();
```



Instantiering och manipulation

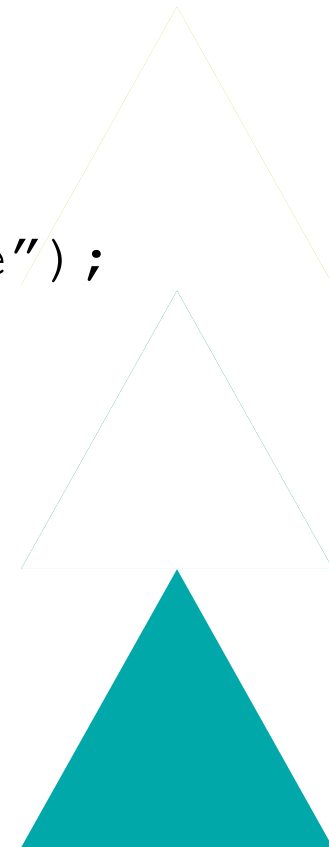
Triangle
height width xPosition yPosition color
changeColor changeSize move ...

```
Triangle aTriangle = new Triangle();
```

```
aTriangle.move();
```

```
aTriangle.changeColor("blue");
```

```
aTriangle.move();
```



Instantiering och manipulation

Triangle
height width xPosition yPosition color
changeColor changeSize move ...

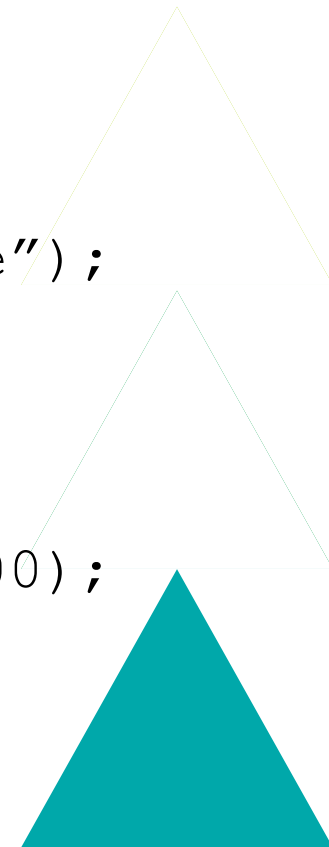
```
Triangle aTriangle = new Triangle();
```

```
aTriangle.move();
```

```
aTriangle.changeColor("blue");
```

```
aTriangle.move();
```

```
aTriangle.changeSize(50, 100);
```



Instantiering och manipulation

Triangle
height width xPosition yPosition color
changeColor changeSize move ...

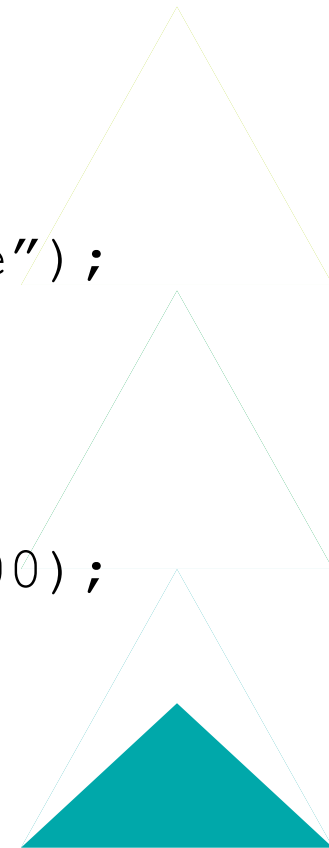
```
Triangle aTriangle = new Triangle();
```

```
aTriangle.move();
```

```
aTriangle.changeColor("blue");
```

```
aTriangle.move();
```

```
aTriangle.changeSize(50, 100);
```



Instantiering och manipulation

Triangle
height width xPosition yPosition color
changeColor changeSize move ...

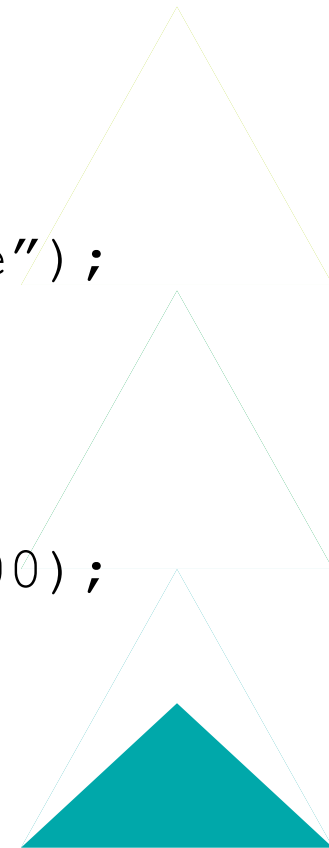
```
Triangle aTriangle = new Triangle();
```

```
aTriangle.move();
```

```
aTriangle.changeColor("blue");
```

```
aTriangle.move();
```

```
aTriangle.changeSize(50, 100);
```



En vanlig metodanrop ser ut så här

```
<objektreferens>.<metodnamn> (<parameterlista>);
```

Objektorienterad Programmeringsmetodik, 5DV133

© Johan Eliasson

Anrop av metod i den egna klassen

- Är man i en metod i en klass och vill anropa en annan metod i samma klass så kan objektreferensen utelämnas vid anrop
 - Implicit läggs då referensen `this` till som refererar till objektet vars metod just anropas.
 - Om man vill så kan man explicit ange `this` som objektreferens i dessa fall.