

Objekter II

INF101 forelesning 25. Januar 2022

Torstein Strømme

Stikkord: Klasser, objekter, primitive og refererte typer, minne, static vs ikke-static

Eksempel: FireProtectionSystem

- Lag et system for brannvarsling
- Det skal være mulighet for flere alarmer og flere sensorer
- Alle alarmer skal være på så lenge minst én sensor er aktiv

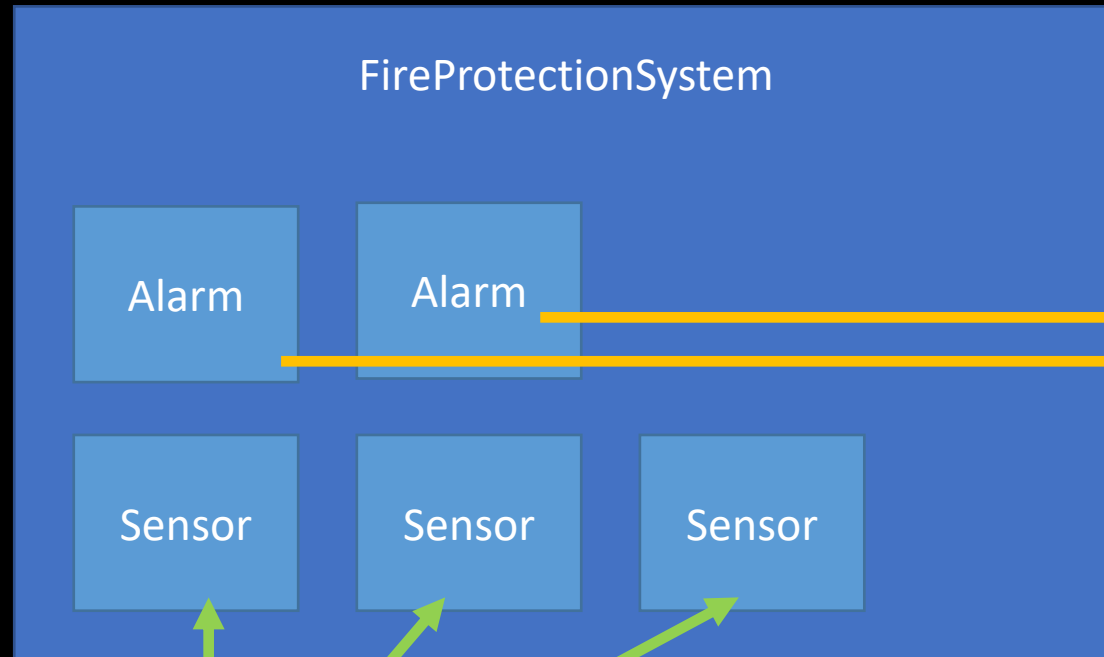
- Prinsipp for objektorientert programmering:
- En klasse er en abstraksjon av en (fysisk eller logisk) enhet

”Single-responsibility principle”

”do one thing, and do it well”

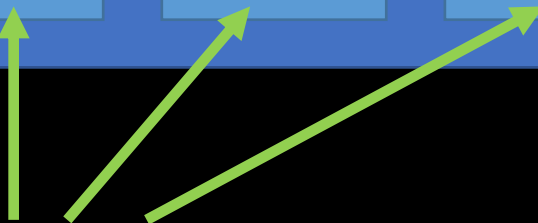
Eksempel: FireProtectionSystem

Installere sensorer
og alarmer



Sirener og blinklys

Røyk og flammer



Visualisering av stack'en/kontrollflyt

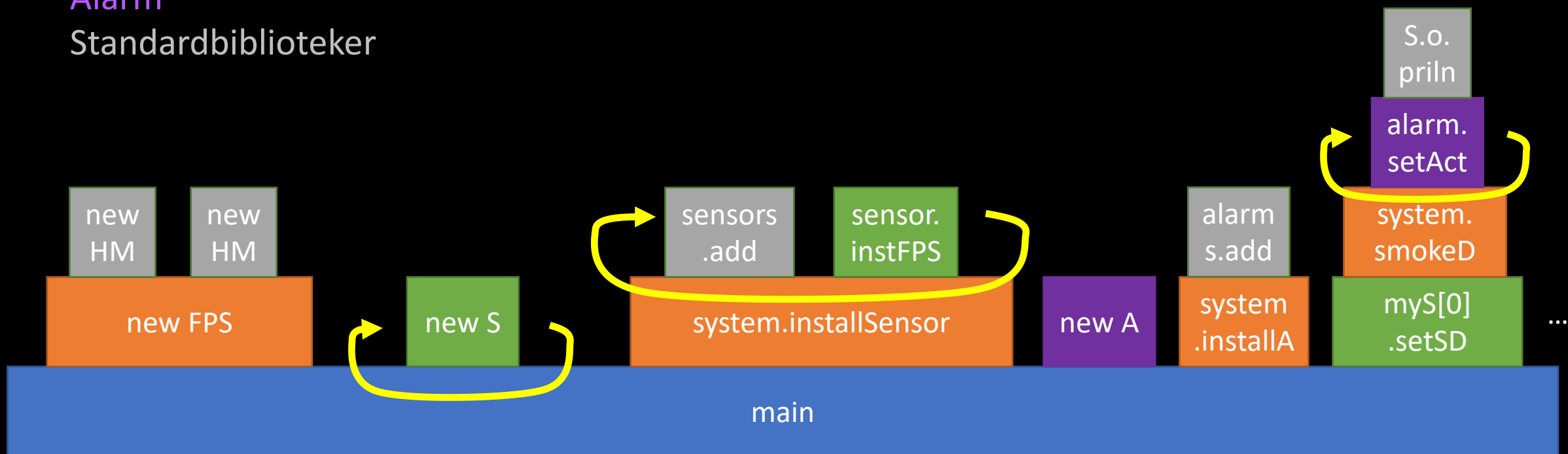
Main

FireProtectionSystem

Sensor

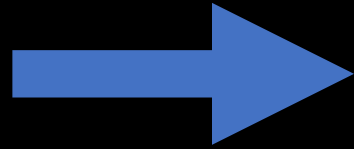
Alarm

Standardbiblioteker



Et objekt

- Har *feltvariabler*
- Kan utføre *metoder*



er
(en instans av)

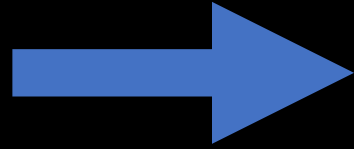
En klasse

- Beskriver *typen* feltvariabler
- Beskriver oppførselen til metoder

Objekt	Klasse
Per, f. nr: 240122 32123	Person
DE83556	Bil
42 + 5i	Komplekst tall
0xDEADBEEF	Minneadresse

Et objekt

- Har *feltvariabler*
- Kan utføre *metoder*



er
(en instans av)

En klasse

- Beskriver *typen* feltvariabler
- Beskriver oppførselen til metoder

```
public static void main(String[] args) {  
    ScoreAndPlayer winner = new ScoreAndPlayer(0, "Player A");  
    winner.increaseScore(10);  
    System.out.printf("Player %s won with %d points\n", winner.p  
}
```

```
class ScoreAndPlayer {  
    int score;  
    String player;  
  
    ScoreAndPlayer(int score, String player) {  
        this.score = score;  
        this.player = player;  
    }  
  
    void increaseScore(int scoreToAdd) {  
        this.score += scoreToAdd;  
    }  
}
```

feltvariabler

konstruktør

metode

```
class ScoreAndPlayer {  
    int score;  
    String player;  
  
    ScoreAndPlayer(int score, String player) {  
        this.score = score;  
        this.player = player;  
    }  
  
    void increaseScore(int scoreToAdd) {  
        this.score += scoreToAdd;  
    }  
}
```

En klasse

- Beskriver *typen* feltvariabler
- Beskriver oppførselen til metoder
- Konstruktør-metoder oppretter nye objekter
 - Konstruktører har kobinert navn og returverdi

Konstruktør

- Kalles når objektet opprettes
- Dersom det ikke er en konstruktør i klassen, er det det samme som å ha en tom konstruktør uten parametere.
- Det kan finnes mer enn én, dersom de tar ulike parametere (dvs. de må ha ulike *signaturer*)

```
class ScoreAndPlayer {  
    int score;  
    String player;  
  
    ScoreAndPlayer(int score, String player) {  
        this.score = score;  
        this.player = player;  
    }  
  
    void increaseScore(int scoreToAdd) {  
        this.score += scoreToAdd;  
    }  
}
```


Hva skjer?

Når et objekt opprettes

- Først opprettes objektet med standard verdier
- Så initieres verdiene
- Så kalles konstruktør

Når en metode kalles

- Feltvariabler er tilgjengelig (som om *this* var gitt som argument)

Minne

64-bit arkitektur → 64 bits per celle

8GB RAM → $64 \cdot 10^9$ bits → 10^9 celler

A diagram illustrating memory cells. A large cyan bracket on the left groups the entire table. A yellow bracket at the top highlights the first three rows. The table consists of 20 rows, each with a hexadecimal address on the left and a decimal value on the right. The values decrease from 63999999936 at the top to 0 at the bottom, with some rows containing ellipses to indicate intermediate values.

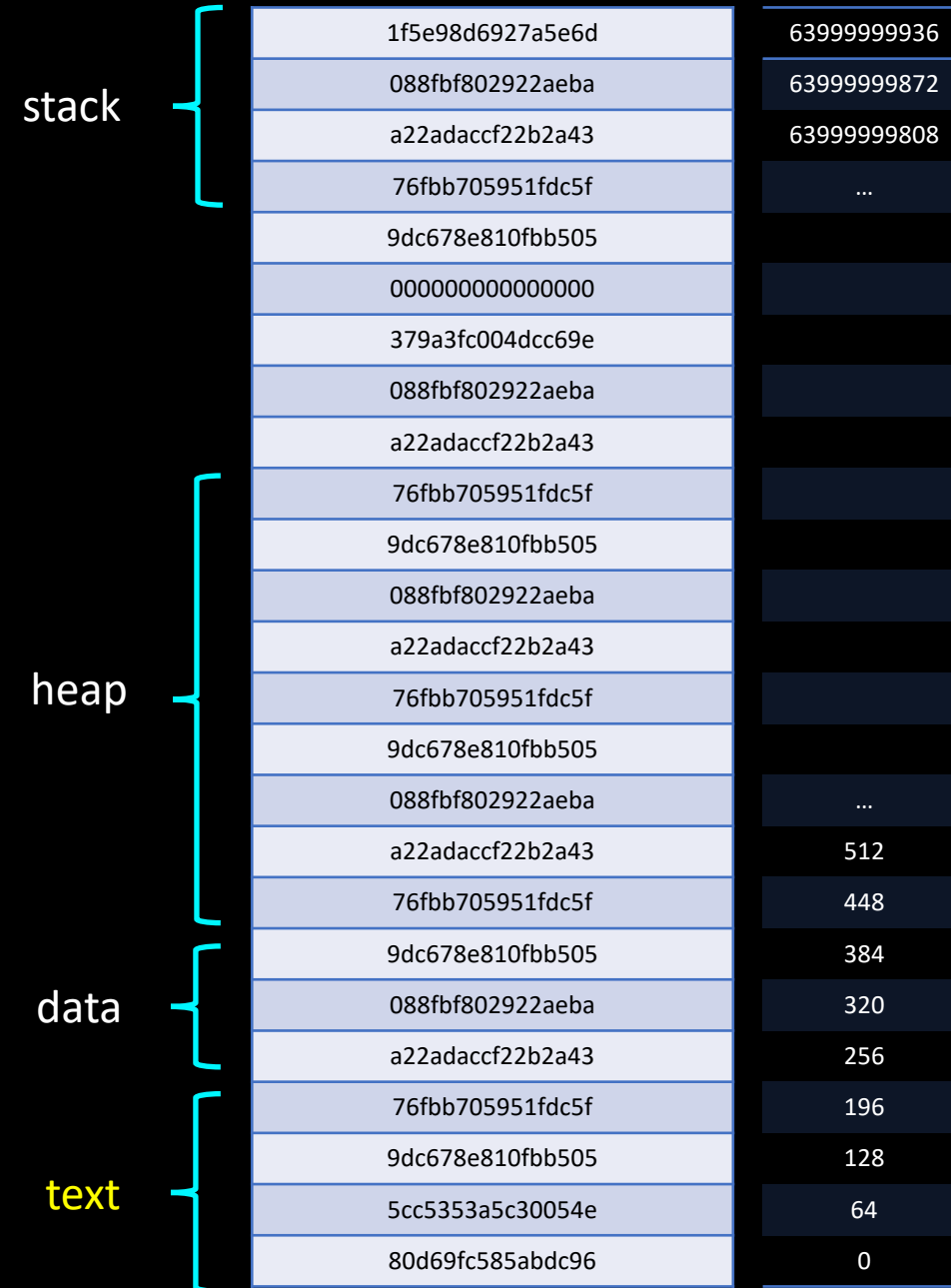
1f5e98d6927a5e6d	63999999936
088fbf802922aeba	63999999872
a22adaccf22b2a43	63999999808
76fbb705951fdc5f	...
9dc678e810fbb505	
0000000000000000	
379a3fc004dcc69e	
088fbf802922aeba	
a22adaccf22b2a43	
76fbb705951fdc5f	
9dc678e810fbb505	
088fbf802922aeba	
a22adaccf22b2a43	
76fbb705951fdc5f	
9dc678e810fbb505	
088fbf802922aeba	...
a22adaccf22b2a43	512
76fbb705951fdc5f	448
9dc678e810fbb505	384
088fbf802922aeba	320
a22adaccf22b2a43	256
76fbb705951fdc5f	196
9dc678e810fbb505	128
5cc5353a5c30054e	64
80d69fc585abdc96	0

Minne

stack	1f5e98d6927a5e6d	63999999936
	088fbf802922aeba	63999999872
	a22adaccf22b2a43	63999999808
	76fbb705951fdc5f	...
	9dc678e810fbb505	
	0000000000000000	
	379a3fc004dcc69e	
heap	088fbf802922aeba	
	a22adaccf22b2a43	
	76fbb705951fdc5f	
	9dc678e810fbb505	
	088fbf802922aeba	
	a22adaccf22b2a43	
	76fbb705951fdc5f	
	9dc678e810fbb505	
	088fbf802922aeba	...
	a22adaccf22b2a43	512
data	76fbb705951fdc5f	448
	9dc678e810fbb505	384
	088fbf802922aeba	320
text	a22adaccf22b2a43	256
	76fbb705951fdc5f	196
	9dc678e810fbb505	128
	5cc5353a5c30054e	64
	80d69fc585abdc96	0

Minne: text

- Selve logikken i kildekoden
- Klasser



Minne: data

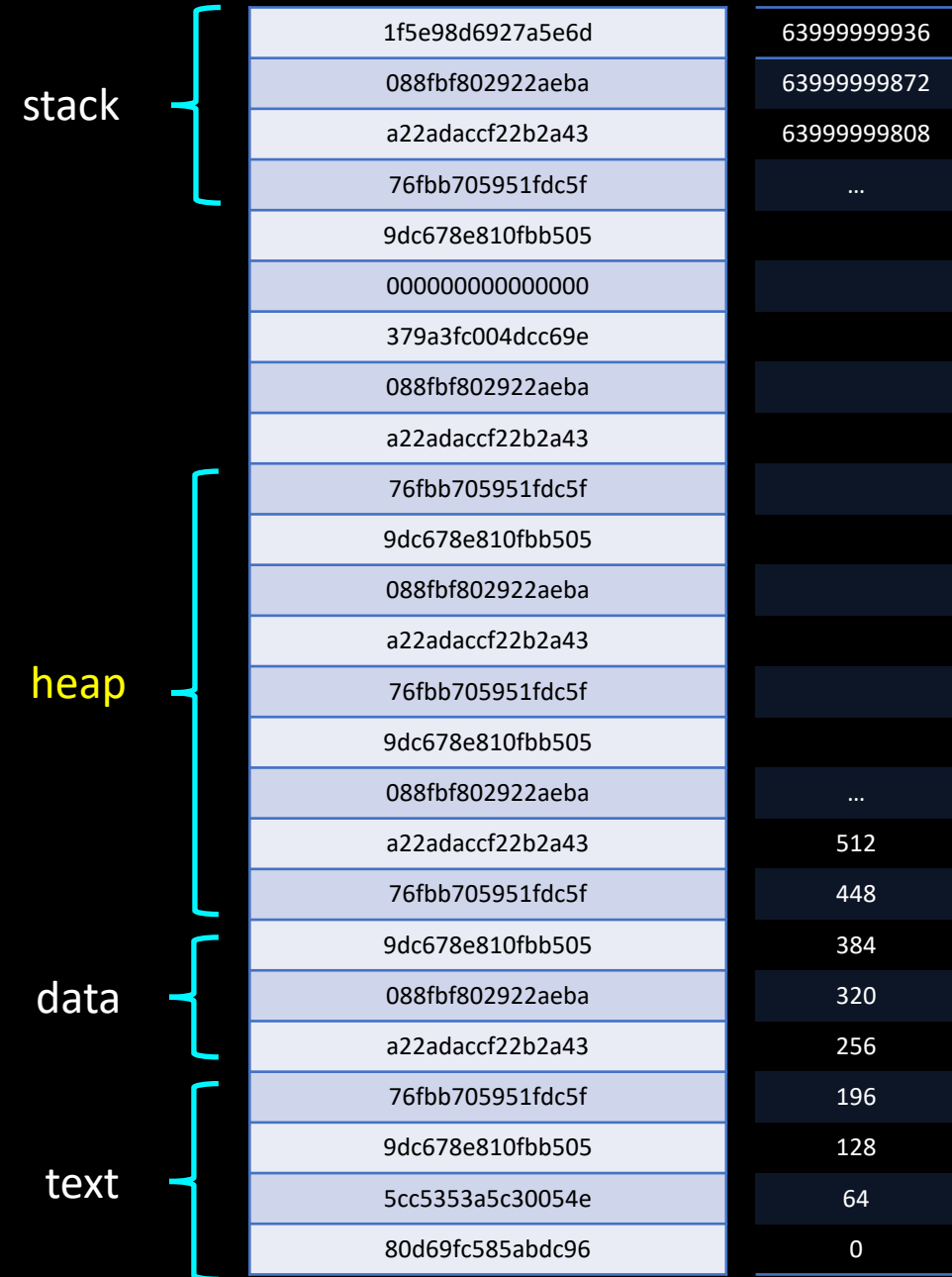
- Konstanter
- Globale/static variabler

The diagram illustrates the memory layout of a program, divided into four segments: stack, heap, data, and text. Each segment contains a list of memory addresses and their corresponding sizes. The stack grows downwards from high addresses, while the heap grows upwards from low addresses. The data segment contains global and static variables, and the text segment contains the program's code.

Segment	Address	Size
stack	1f5e98d6927a5e6d	63999999936
	088fbf802922aeba	63999999872
	a22adaccf22b2a43	63999999808
	76fbb705951fdc5f	...
	9dc678e810fbb505	
	0000000000000000	
	379a3fc004dcc69e	
	088fbf802922aeba	
	a22adaccf22b2a43	
	76fbb705951fdc5f	
heap	9dc678e810fbb505	
	088fbf802922aeba	
	a22adaccf22b2a43	
	76fbb705951fdc5f	
	9dc678e810fbb505	
	088fbf802922aeba	...
	a22adaccf22b2a43	512
	76fbb705951fdc5f	448
	9dc678e810fbb505	384
	088fbf802922aeba	320
data	a22adaccf22b2a43	256
	76fbb705951fdc5f	196
	9dc678e810fbb505	128
text	5cc5353a5c30054e	64
	80d69fc585abdc96	0

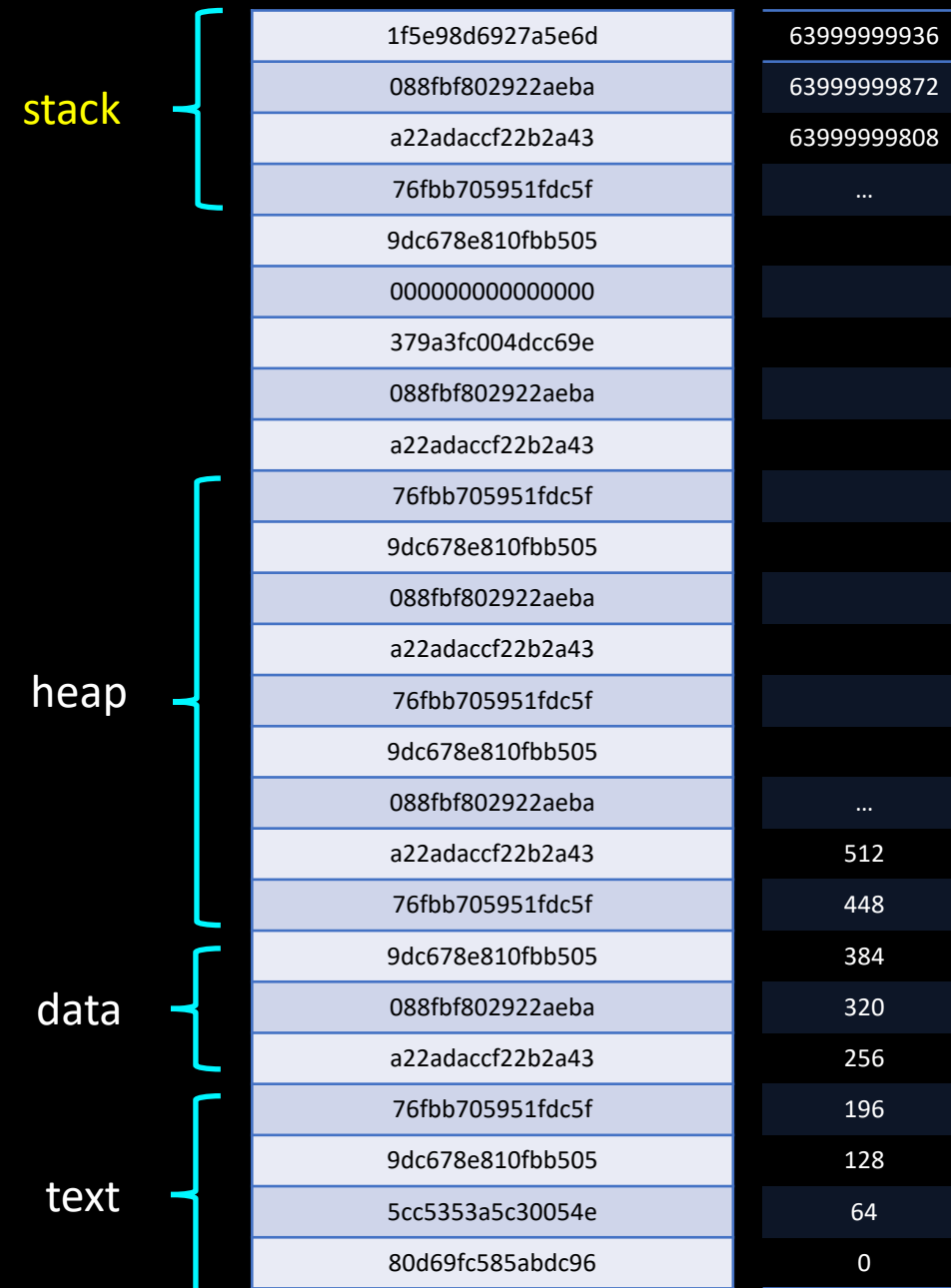
Minne: heap

- Objekter
- Arrays
- Dynamisk



Minne: stack


- Primitive variabler
- Referanser til objekter
- Referanser til arrays
- (INF113 pensum: referanser til text; hva som er neste steg etter et return statement)
- Dynamisk basert på programflyt



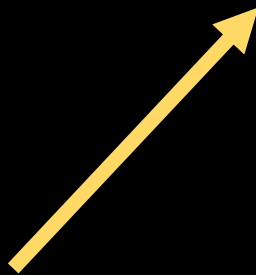
Primitive typer

- byte, short, int, long
- float, double
- boolean
- char

på stacken



i heap
(på stacken lagres
kun en referanse)



Refererte typer

- Klasser
 - Innebygde
 - Arrays
 - String
 - Integer
 - System
 - ...
 - I biblioteker
 - ArrayList
 - HashMap
 - ...
 - Våre egne klasser

Likhet for primitive typer

```
int x = 5;
int y = 5;

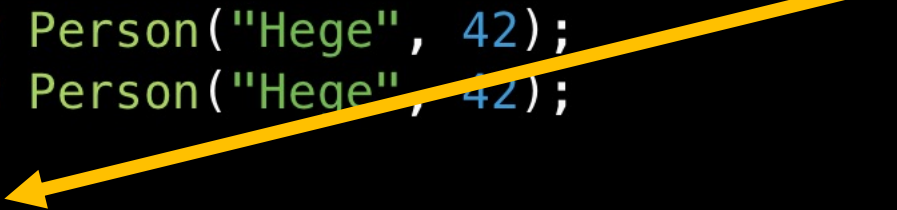
if (x == y) {
    System.out.println("x and y are the same");
}
else {
    System.out.println("x and y are different");
}
```

Likhet for refererte typer

Sammenligner *referansen*

```
Person x = new Person("Hege", 42);
Person y = new Person("Hege", 42);

if (x == y) {
    System.out.println("x and y are the same");
}
else {
    System.out.println("x and y are different");
}
```



Likhet for refererte typer

```
Person x = new Person("Hege", 42);
Person y = new Person("Hege", 42);

if (x.equals(y)) {
    System.out.println("x and y are
}
else {
    System.out.println("x and y are
}
```

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    boolean equals(Person that) {
        if (this.age != that.age) {
            return false;
        }
        return this.name.equals(that.name);
    }
}
```

hashCode

- Krav:

`a.equals(b)` \rightarrow `a.hashCode() == b.hashCode()`

Hvis kravet brytes, vil ikke f. eks. HashSet fungere.

Tilbake til FireProtectionSystem

- Et Sensor -objekt er en abstraksjon av en fysisk sensor
- Vi ønsker:
 - Gi beskjed dersom bruker av systemet forsøker å installere to Sensor – objekter som representerer den samme fysiske sensoren.
- Plan:
 - La Sensor -objekter ha informasjon om hvilken fysisk sensor de representerer
 - Dersom to Sensor –objekter representerer samme fysiske sensor, skal de være like med hensyn til `.equals` og `.hashCode`
 - Skriv ut en feilmelding dersom bruker forsøker å installere to like sensorer.

Sammendrag

<code>new</code>	Oppretter et nytt objekt i heap. Kaller en konstruktør.
<code>b = a;</code>	b er en referanse til <i>det samme</i> objektet som a.
<code>(a == b)</code>	uttrykket er true hvis a og b er <i>det samme</i> objektet.
<code>a.equals(b)</code>	uttrykket er true hvis a og b er <i>like</i> . NB! Må implementeres i våre egne klasser.
<code>.hashCode()</code>	Dersom to objekter a og b er like med <code>.equals</code> , <i>skal</i> også <code>a.hashCode()</code> være lik <code>b.hashCode()</code> . Må implementeres parallelt med <code>.equals()</code> .