Objekter II

INF101 forelesning 25. Januar 2022

Torstein Strømme

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

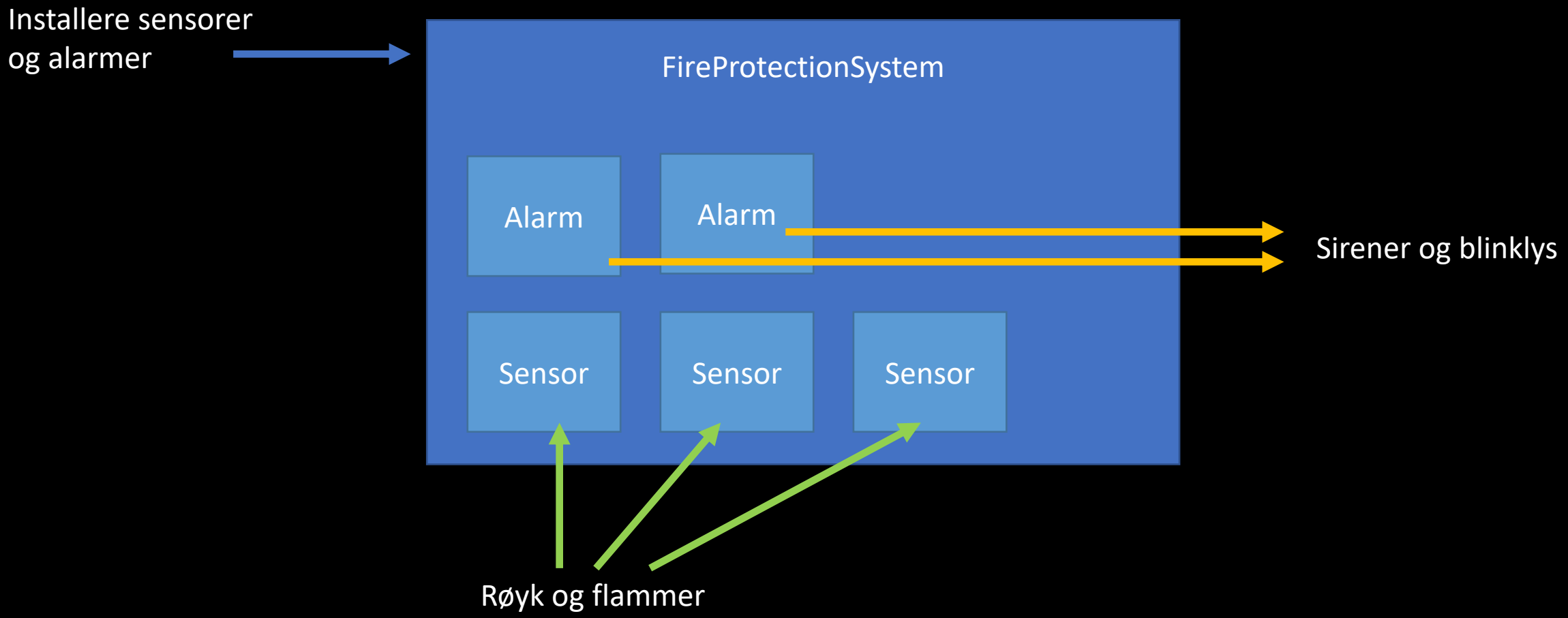
Eksempel: FireProtectionSystem

- 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

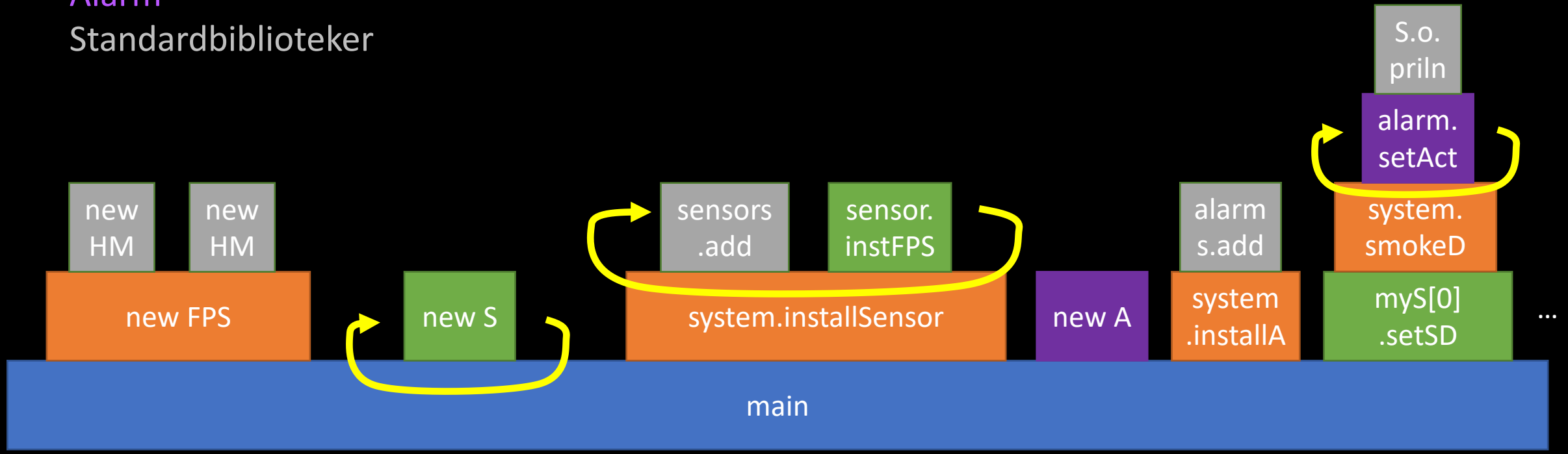
”Single-responsibility principle” ”do one thing, and do it well”

Eksempel: FireProtectionSystem

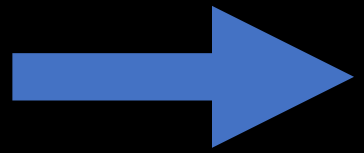


Visualisering av stack'en/kontrollflyt

Main
FireProtectionSystem
Sensor
Alarm
Standardbiblioteker



Et objekt



er
(en instans av)

En klasse

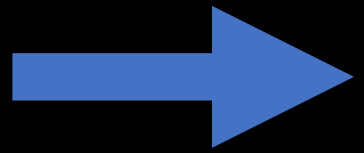
- Har *feltvariabler*
- Kan utføre *metoder*

- 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

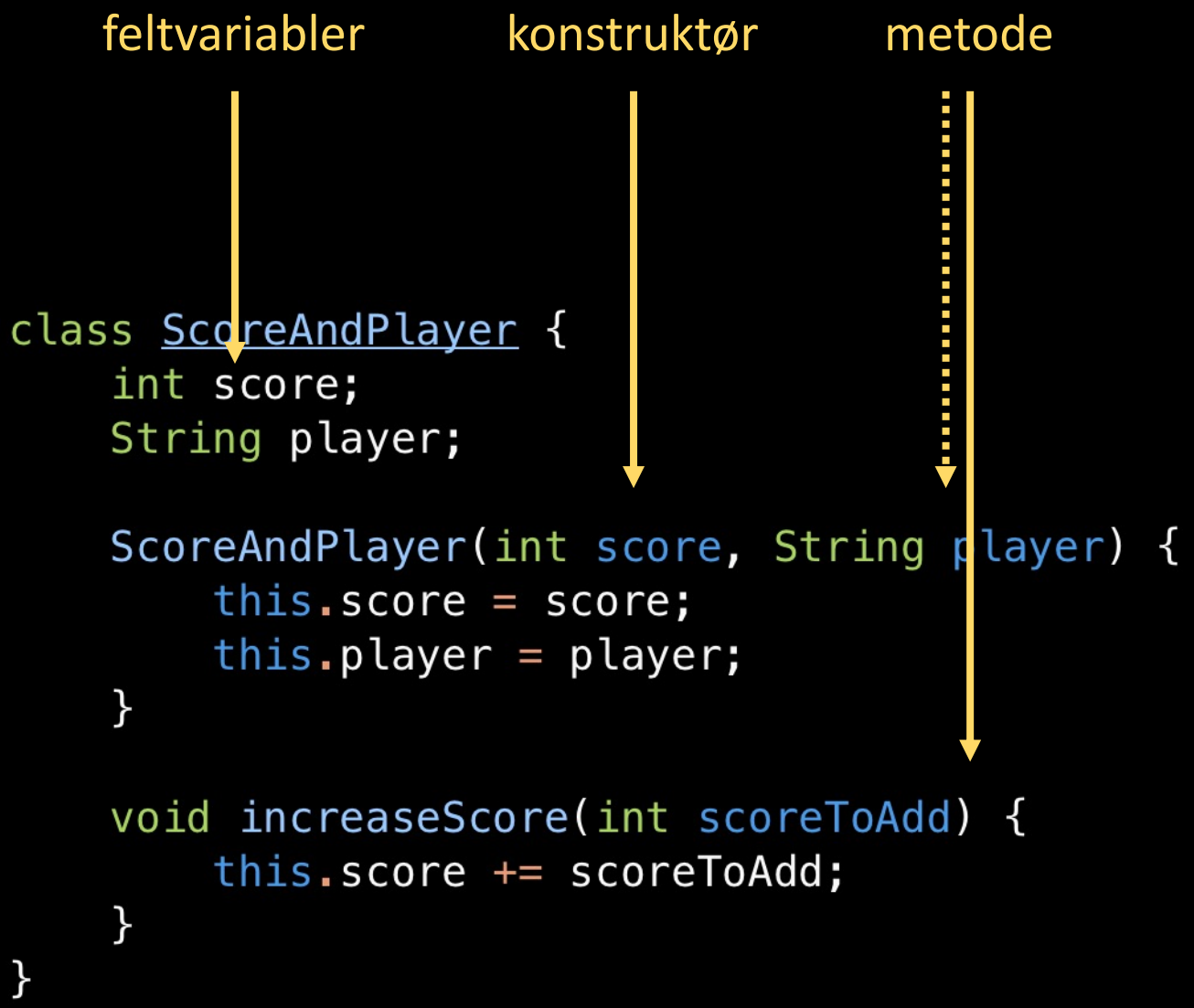
- Beskriver *typen* feltvariabler
- Beskriver oppførselen til metoder

```
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;
    }
}
```



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 har 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

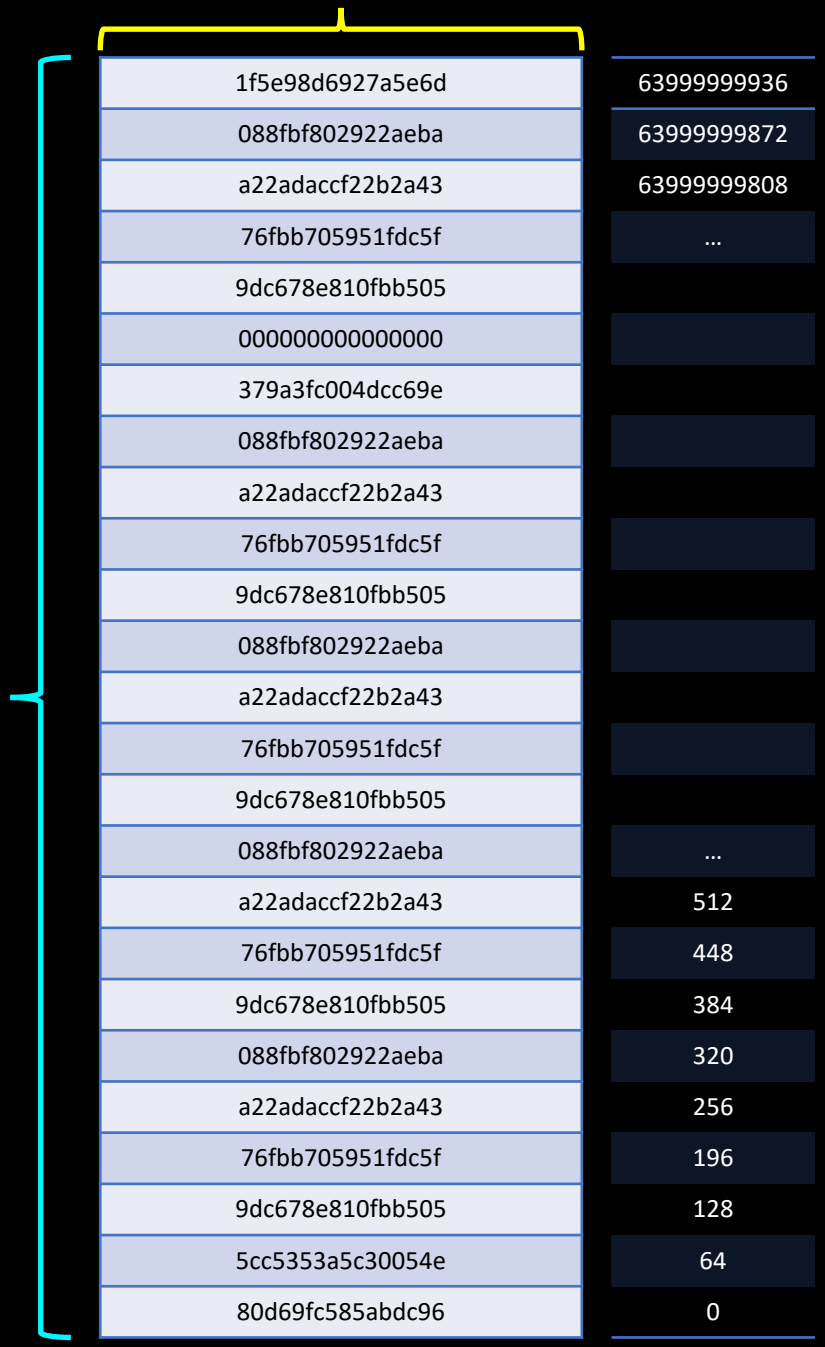
Når en metode kalles

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

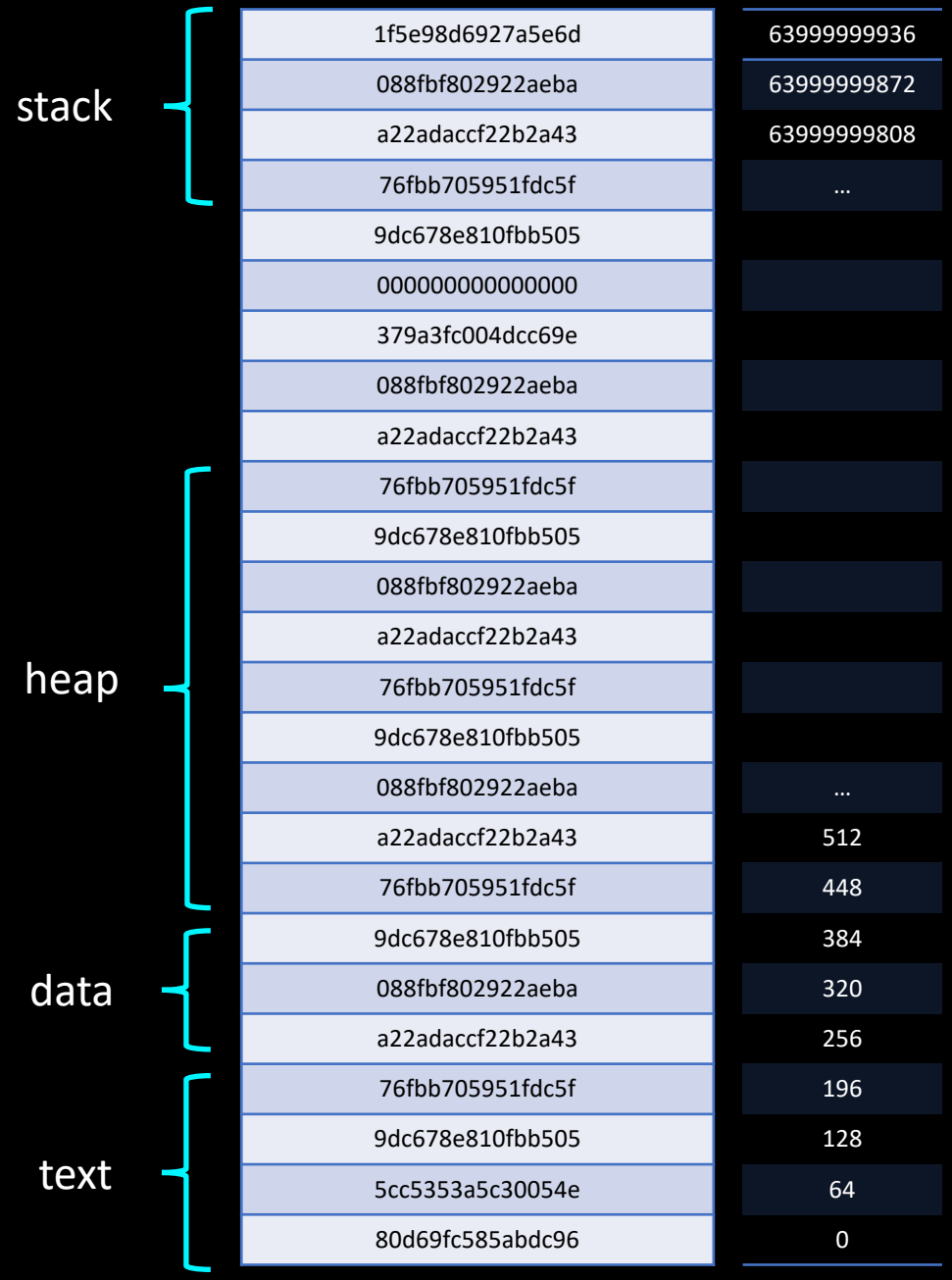
Minne

64-bit arkitektur → 64 bits per celle

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

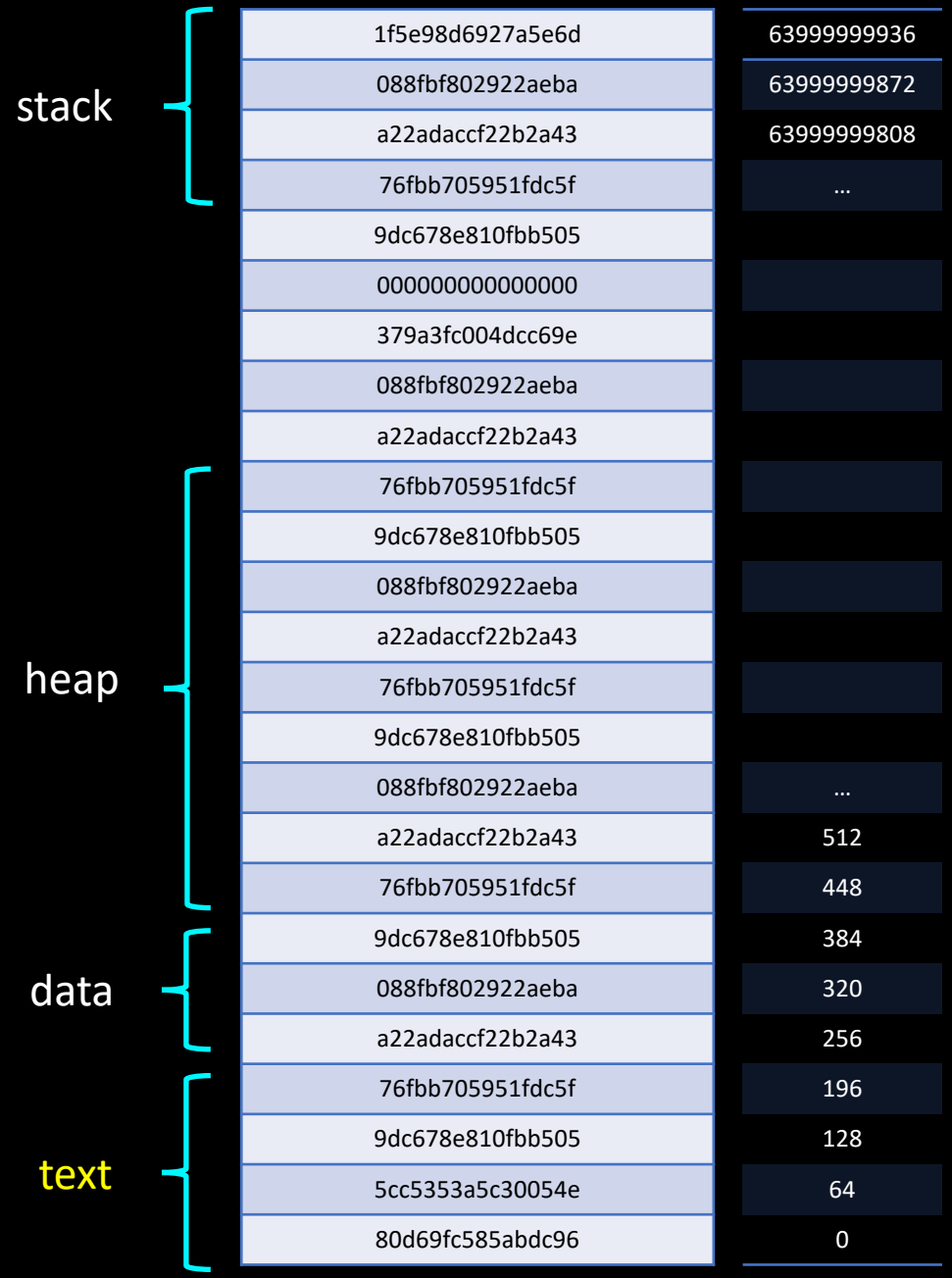


Minne


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

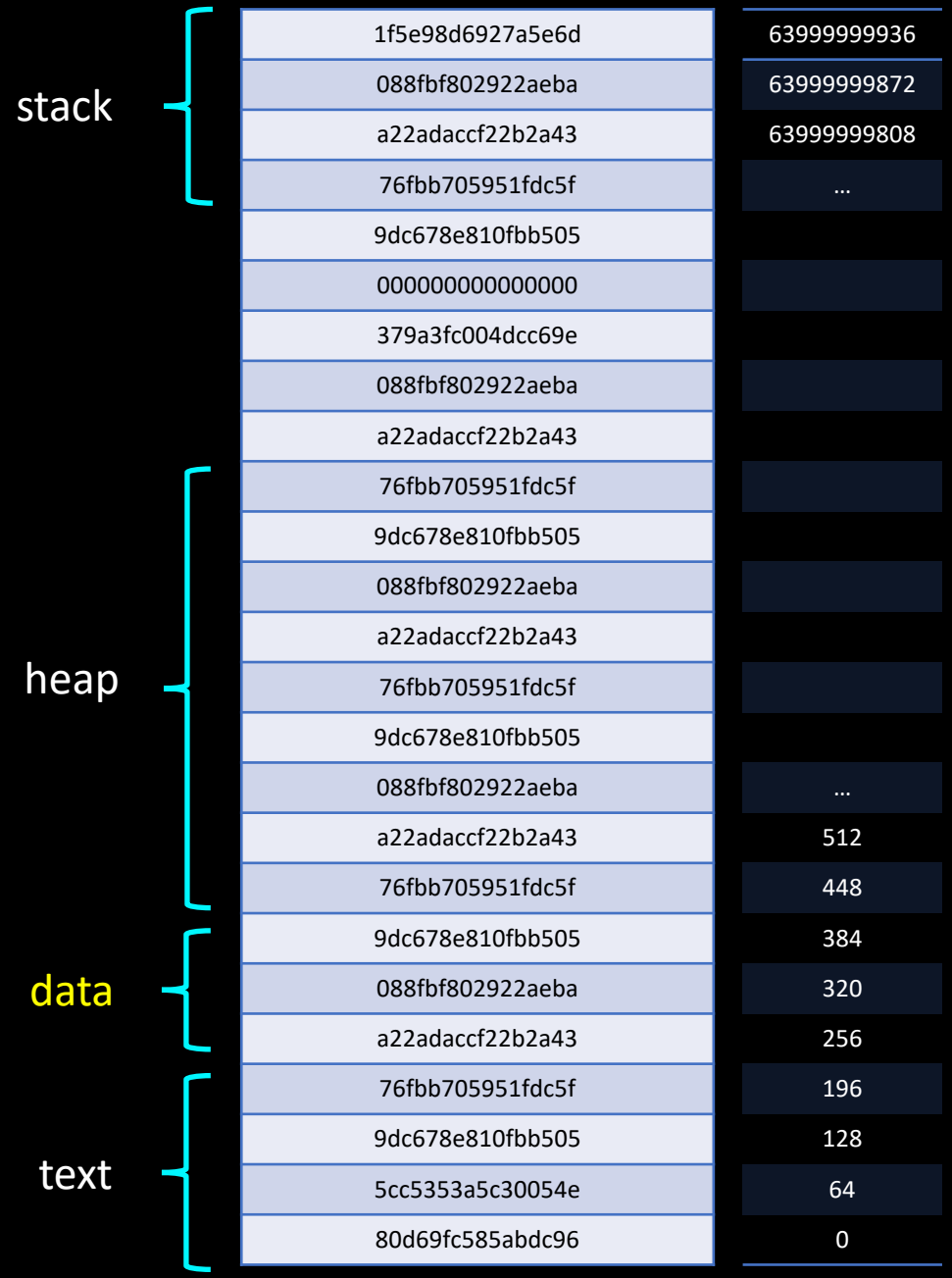
Minne: text

- Selve logikken i kildekoden
- Klasser



Minne: data

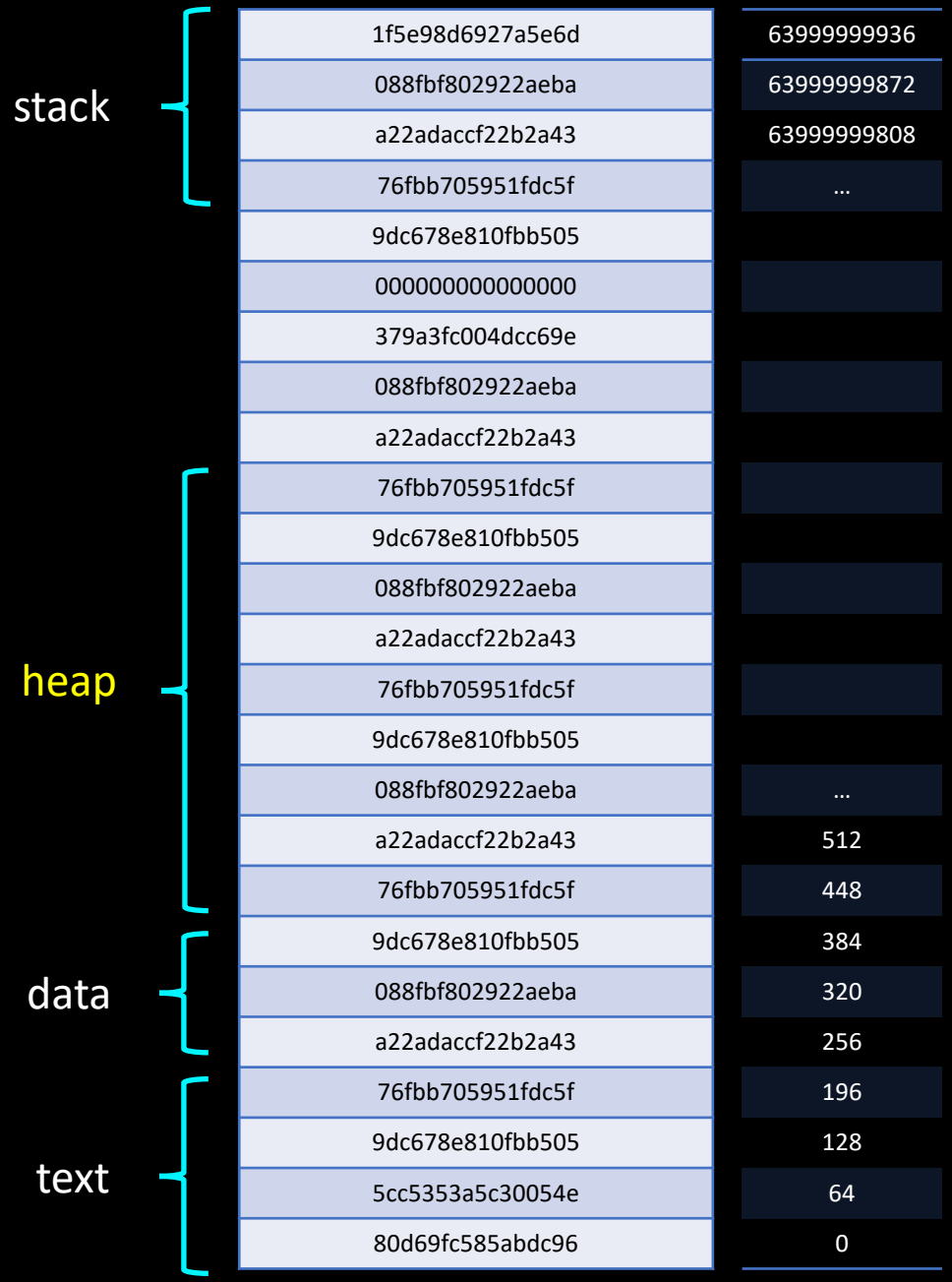
- Konstanter
- Globale/static variabler



Minne: heap

- Objekter
- Arrays

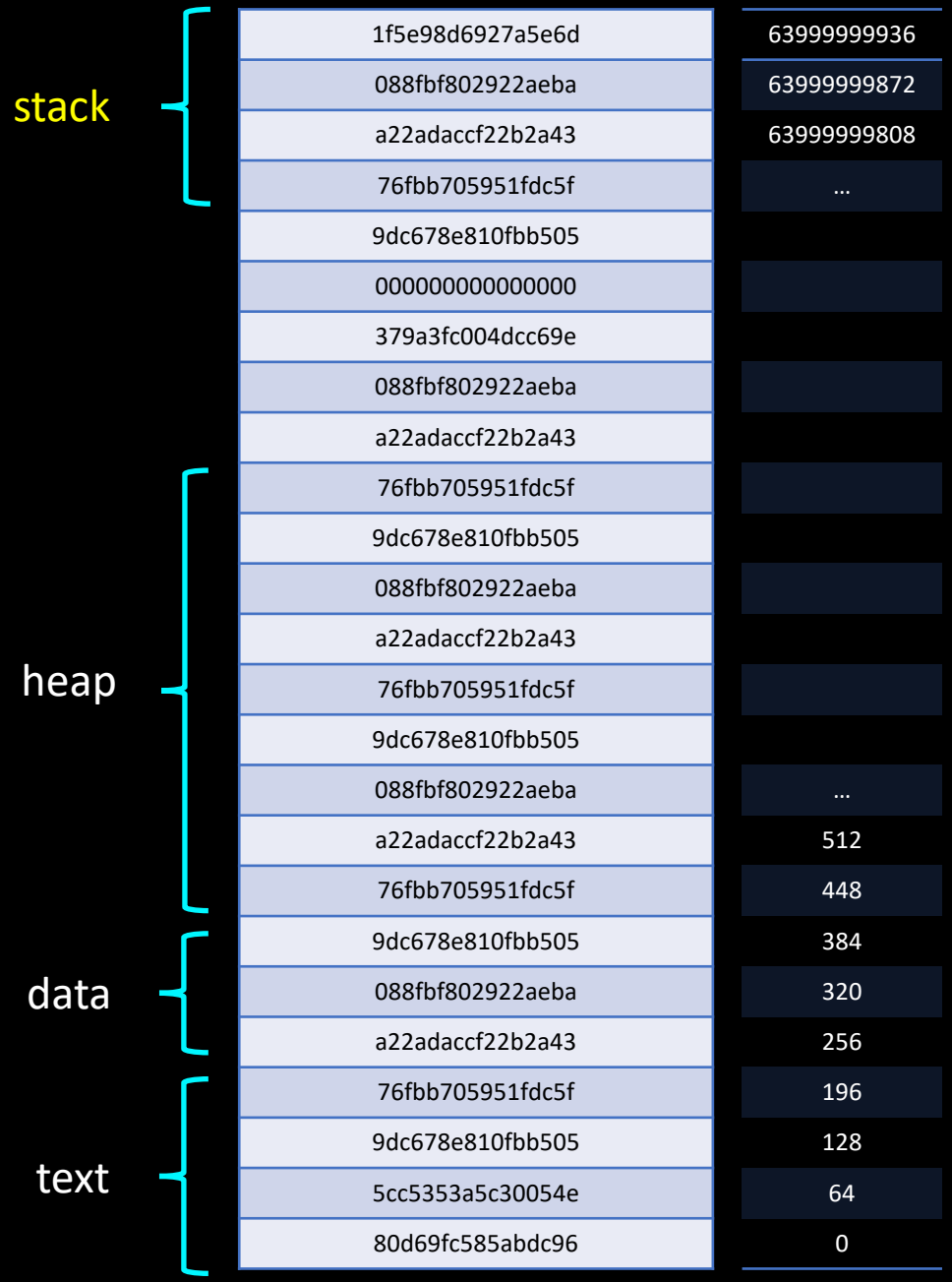
- Dynamisk



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



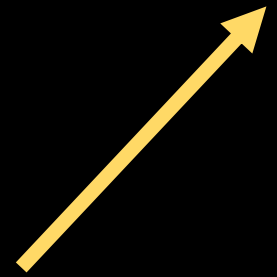
Primitive typer

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

på stacken

Refererte typer

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



i heap
(på stacken lagres
kun en referanse)

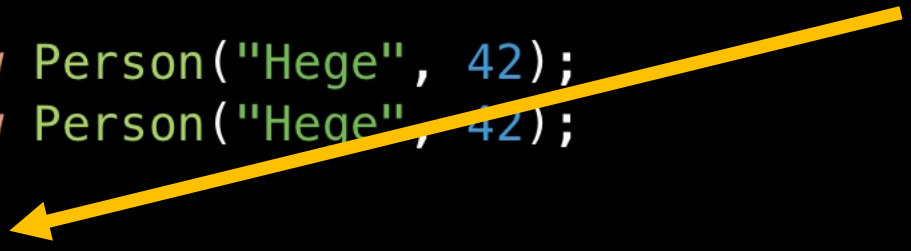
Likhet for primitive typer

```
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)     →     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

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