IN1010 - Seminar 13

- Design og testing

Laget av: Marlen Jarholt (marleja@ifi.uio.no) og Johanna Haarseth (johannph@ifi.uio.no)

Praktisk

Husk å sjekke emnesiden regelmessig

Undervisningstilbud:

- https://www.uio.no/studier/emner/matnat/ifi/IN1010/v21/undervisningstilbud/
- Jeg har konkrete spørsmål/problemer med min kode -> Labtime!
- Jeg vil ha mer liveprogrammering -> Plenumstime!
- Jeg vil jobbe med andre (og kanskje en kjapp recap av forelesning) -> Gruppetime!
- Jeg vil ha en recap av de vanskeligste konseptene fra forelesning -> Repetisjonsgruppe!

Oblig 6 har frist på mandag, håper dere koser dere med rad og kolonne <3

Send Marlen en direktemelding i chatten

Vil du jobbe sammen med noen andre ? (ja /nei)

Hvis du har noen ønsker på hvem du vil jobbe med, så send det i samme melding

Svar gjerne også om svaret skulle være nei

Repetisjon forrige uke

Hva er rekursjon?

Vi bryter opp komplekse oppgaver i mindre/enklere oppgaver

Rekursjon: samme operasjon flere ganger

Hvordan implementere rekursjon?

1. Basis case
   a. Vi må passe på at vi ikke får uendelig rekursjon!
   b. Basis caset gjør ikke et rekursivt kall!
2. Hvert rekursive kall må gjøre beregningen litt enklere
   a. Slik at vi til slutt treffer basis caset!

TIPS: Hvis du synes rekursjon er litt vanskelig så LES BOKA KAP 13, det gjelder å få litt feelingen for rekursjon. Men som regel er det mye enklere enn man kanskje tror! Et annet tips er å tegne hva vi vil at skal skje!

Løkke vs rekursjon

```java
class EnkelIterasjon{
    public static void main(String[] args) {
        System.out.println("Iterasjon:");
        skrivTallIterasjon(5);
    }
    public static void skrivTallIterasjon(int n){
        for(int i = n; i >= 0; i--){
            System.out.println(i);
        }
    }
```

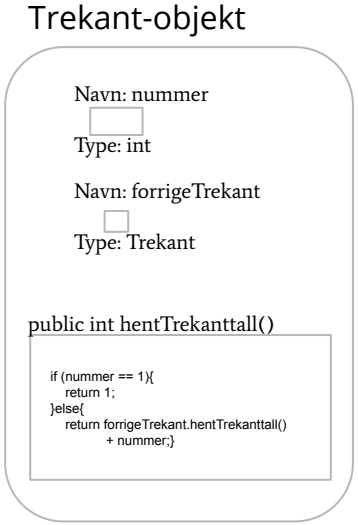
```
jonbon@jons-macbook-pro uke12 % java EnkelIterasjon
Iterasjon:
5
4
3
2
1
0
```

```java
class EnkelRekursjon {
    public static void main(String[] args) {
        System.out.println("Rekursjon:");
        skrivTallRekursjon(5);
    }
    public static void skrivTallRekursjon(int n){
        if (n < 0) {
            return;
        }
        System.out.println(n);
        skrivTallRekursjon(n-1);
```

```
jonbon@jons-macbook-pro uke12 % java EnkelRekursjon
Rekursjon:
5
4
3
2
1
0
```

Rekursjon trekanttall

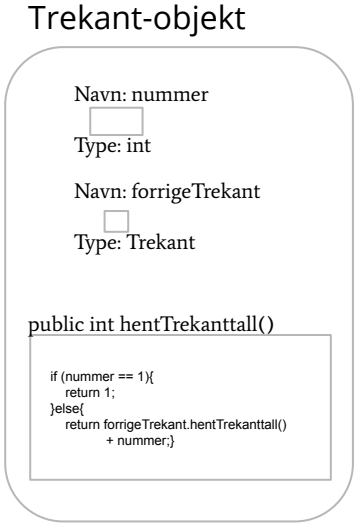
```java
class Trekant{
    private int nummer;
    private Trekant forrigeTrekant;

    public Trekant(int nummer, Trekant forrigeTrekant){
        this.nummer = nummer;
        this.forrigeTrekant = forrigeTrekant;
    }
    public int hentTrekanttall(){
        if (nummer == 1){
            return 1;
        }else{
            return forrigeTrekant.hentTrekanttall() + nummer;
        }
```



Rekursjon trekanttall

```java
class Trekant{
    private int nummer;
    private Trekant forrigeTrekant;

    public Trekant(int nummer, Trekant forrigeTrekant){
        this.nummer = nummer;
        this.forrigeTrekant = forrigeTrekant;
    }
    public int hentTrekanttall(){
        if (nummer == 1){
            return 1;
        }else{
            return forrigeTrekant.hentTrekanttall() + nummer;
        }
```

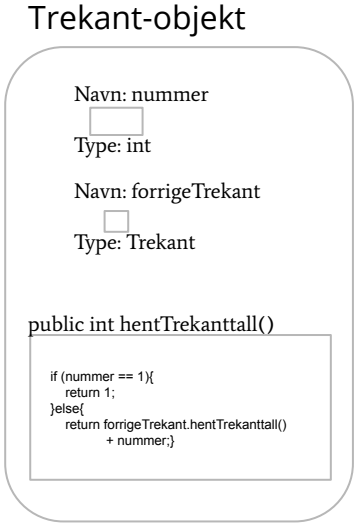
Basis case



Rekursjon trekanttall

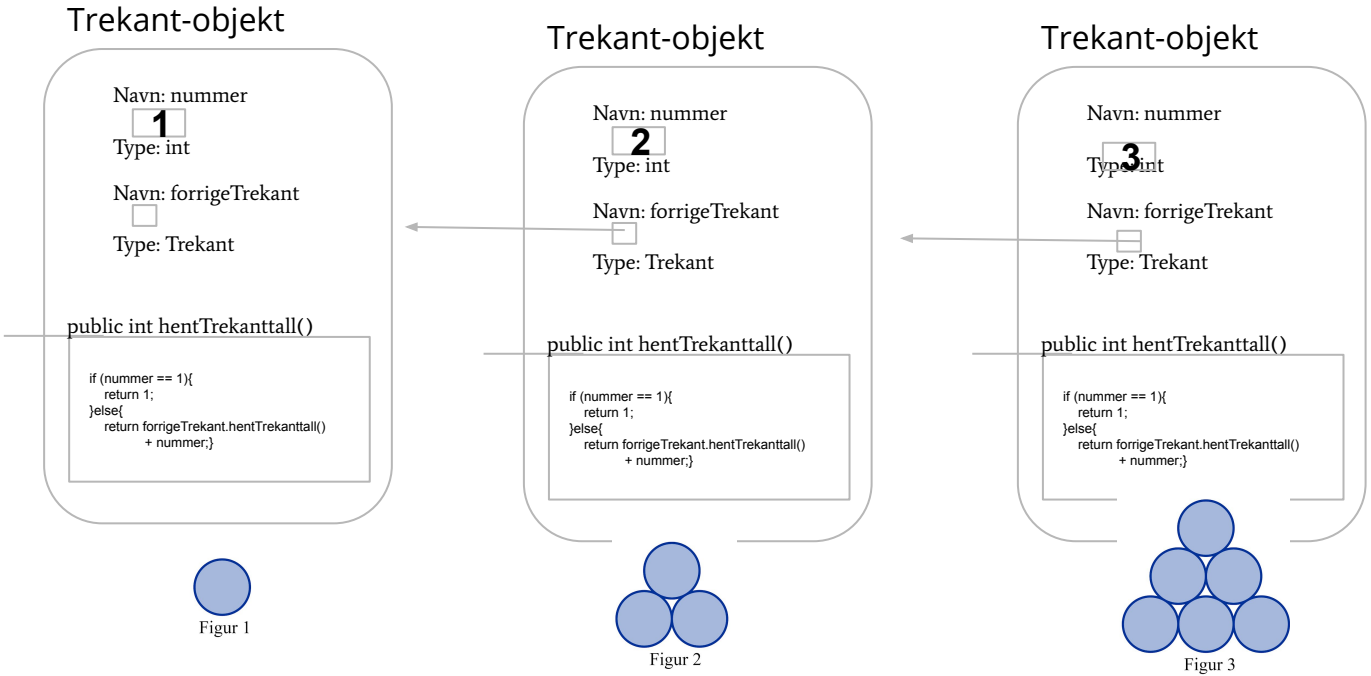
```java
class Trekant{
    private int nummer;
    private Trekant forrigeTrekant;

    public Trekant(int nummer, Trekant forrigeTrekant){
        this.nummer = nummer;
        this.forrigeTrekant = forrigeTrekant;
    }
    public int hentTrekanttall(){
        if (nummer == 1){
            return 1;
        }else{
            return forrigeTrekant.hentTrekanttall() + nummer;
        }
```

Enklere kall hver gang



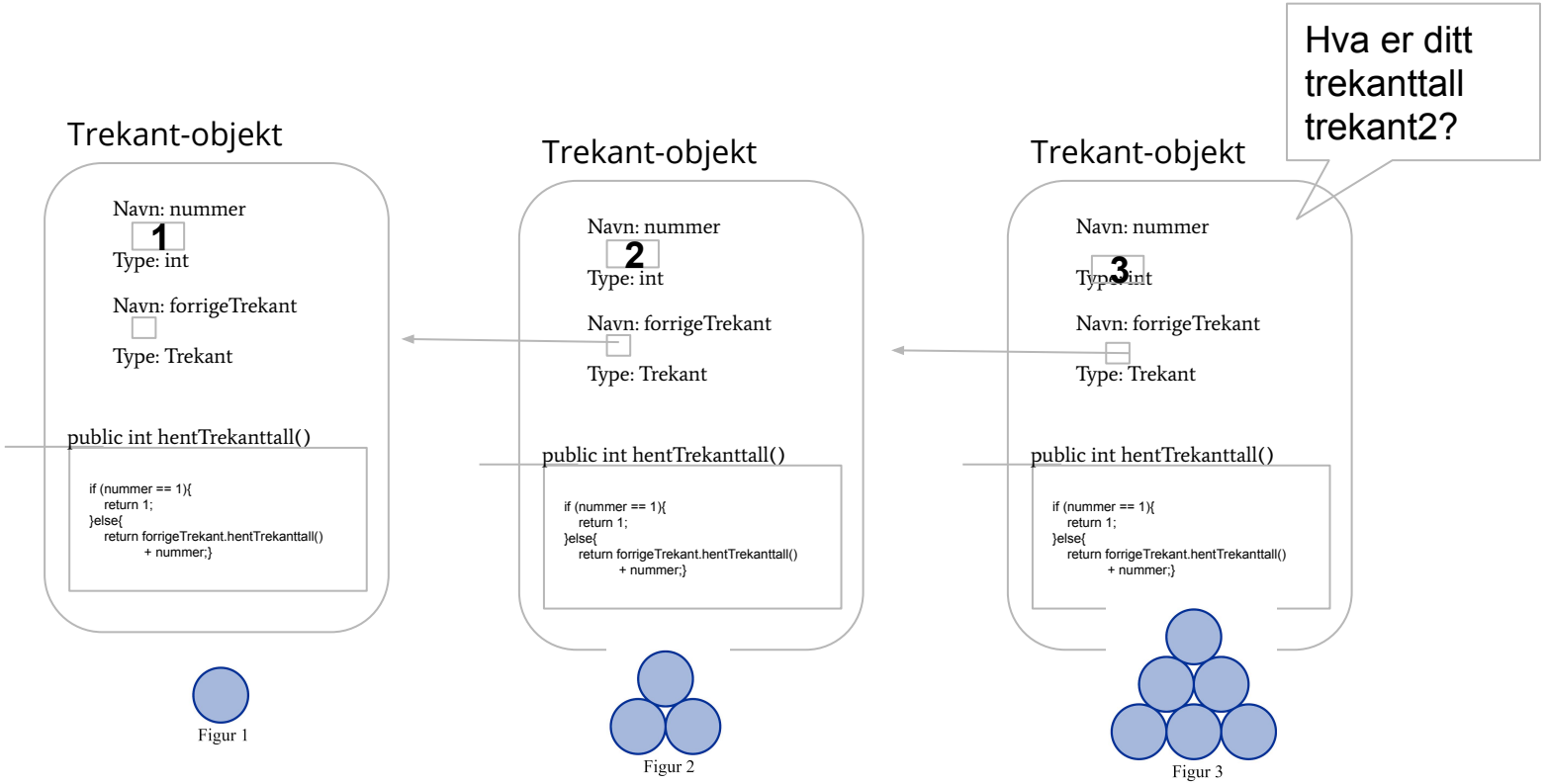
Rekursjon trekanttall

```
public int hentTrekanttall(){
    if (nummer == 1){
        return 1;
    }else{
        return forrigeTrekant.hentTrekanttall() + nummer;
    }
```



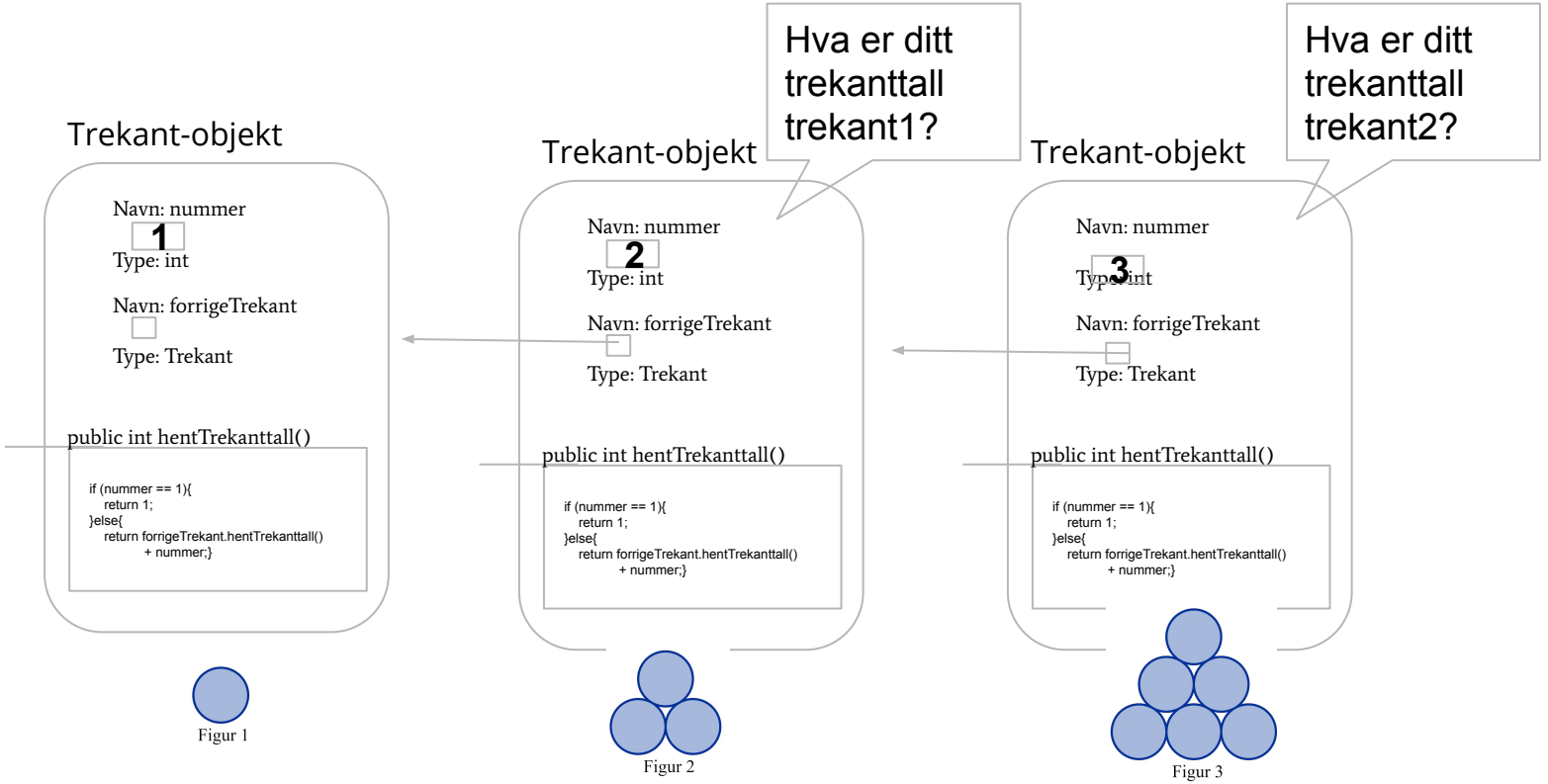
Rekursjon trekanttall

```
public int hentTrekanttall(){
    if (nummer == 1){
        return 1;
    }else{
        return forrigeTrekant.hentTrekanttall() + nummer;
    }
```



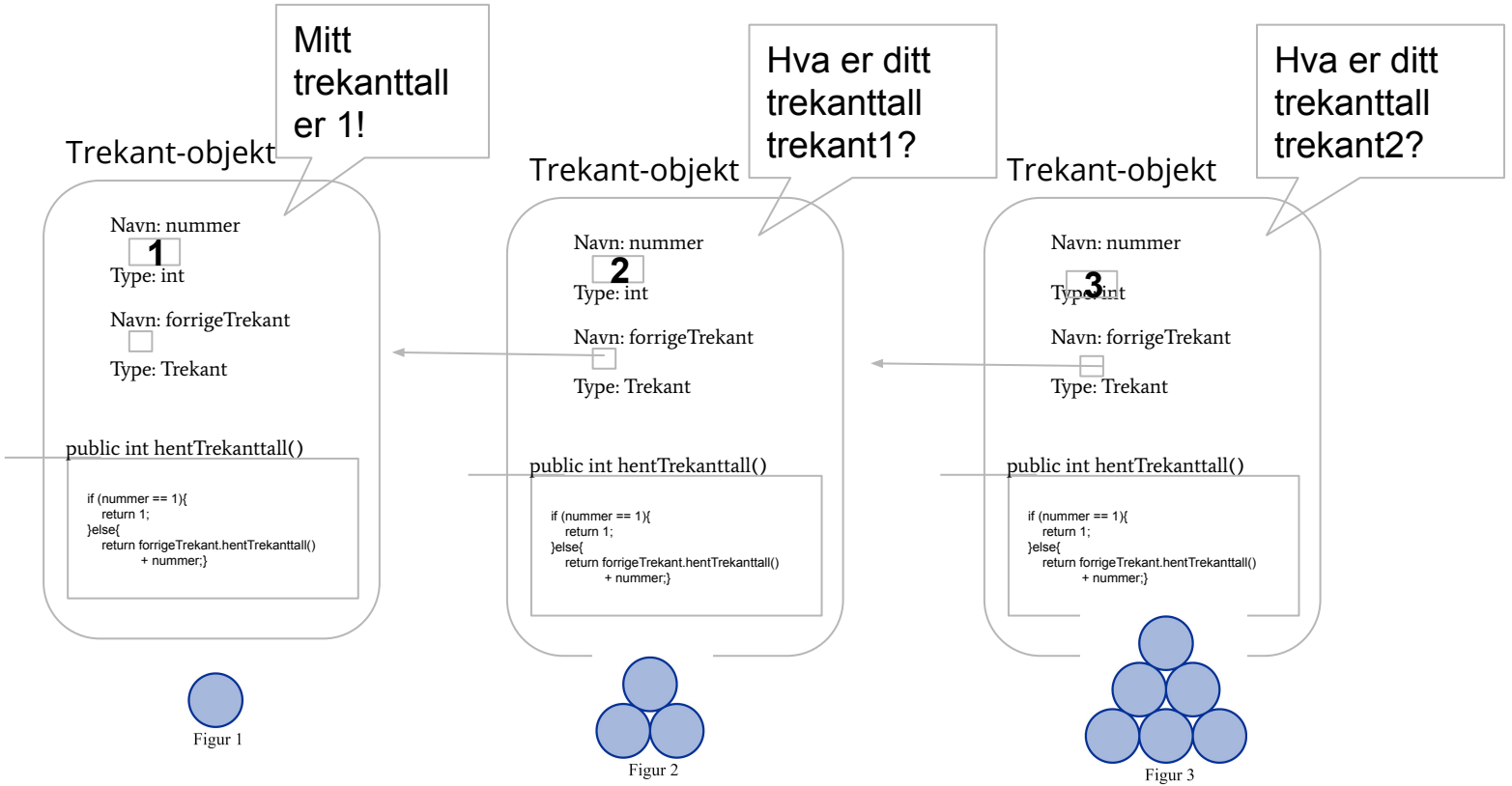
Rekursjon trekanttall

```
public int hentTrekanttall(){
    if (nummer == 1){
        return 1;
    }else{
        return forrigeTrekant.hentTrekanttall() + nummer;
    }
```



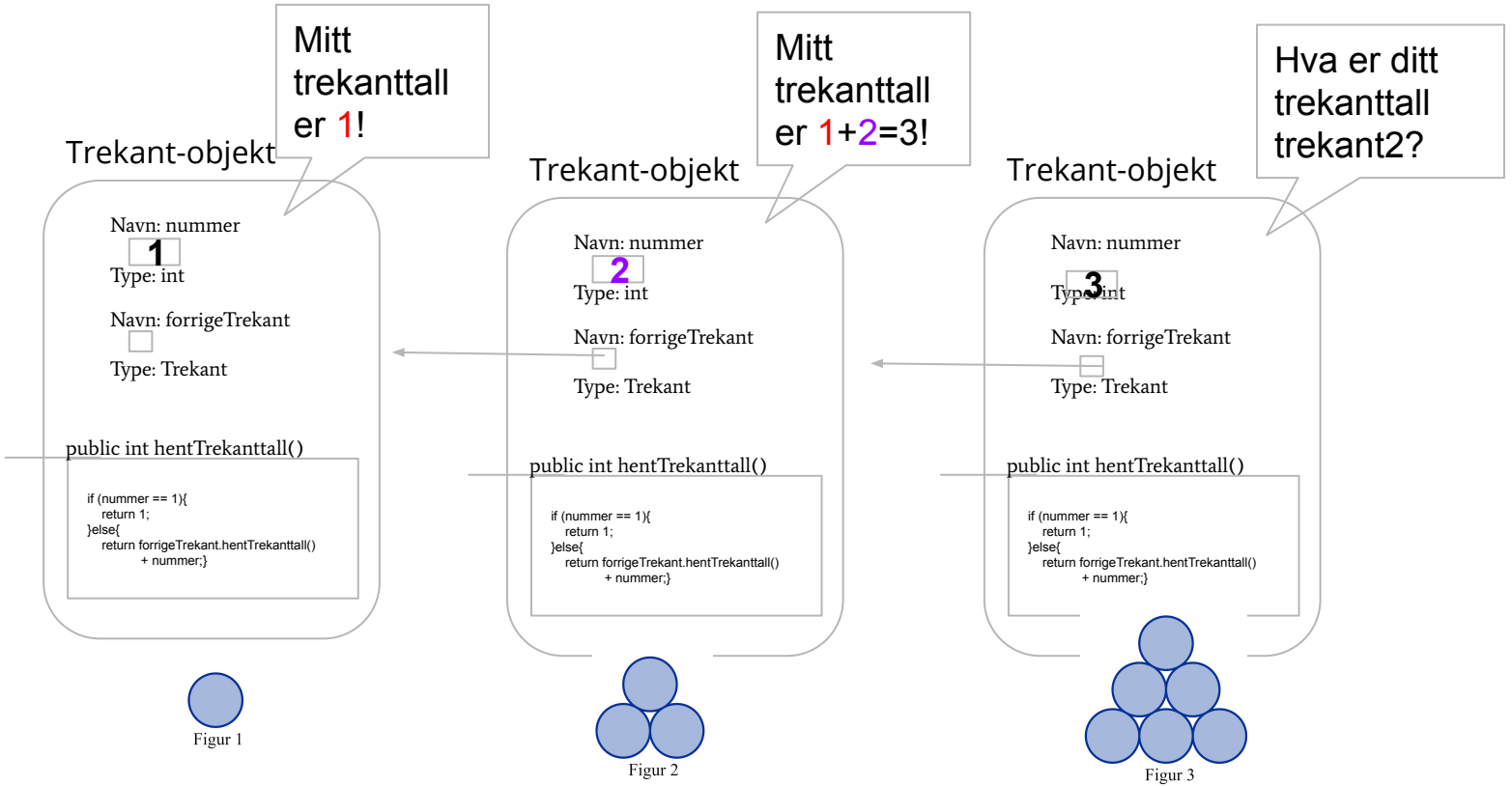
Rekursjon trekanttall

```
public int hentTrekanttall(){
    if (nummer == 1){
        return 1;
    }else{
        return forrigeTrekant.hentTrekanttall() + nummer;
    }
```



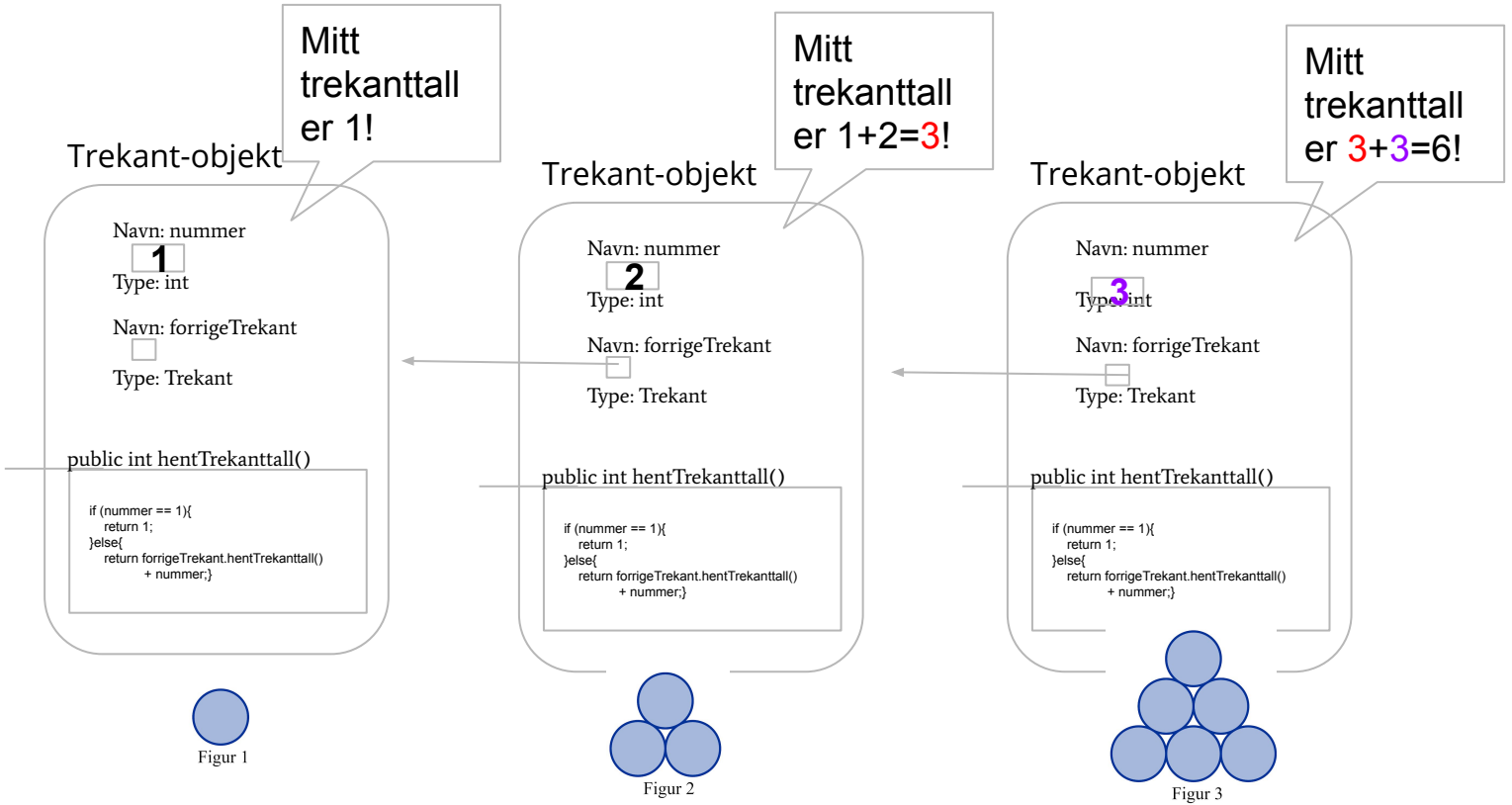
Rekursjon trekanttall

```
public int hentTrekanttall(){
    if (nummer == 1){
        return 1;
    }else{
        return forrigeTrekant.hentTrekanttall() + nummer;
    }
```

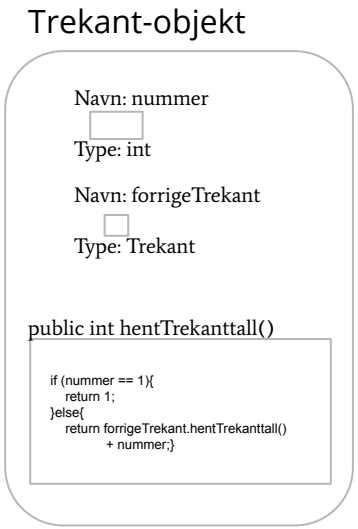


Rekursjon trekanttall

```
public int hentTrekanttall(){
    if (nummer == 1){
        return 1;
    }else{
        return forrigeTrekant.hentTrekanttall() + nummer;
    }
```



Rekursjon trekanttall




```java
class Trekant{
  private int nummer;
  private Trekant forrigeTrekant;

  public Trekant(int nummer, Trekant forrigeTrekant){
    this.nummer = nummer;
    this.forrigeTrekant = forrigeTrekant;
  }
  public int hentTrekanttall(){
    if (nummer == 1){
      return 1;
    }else{
      return forrigeTrekant.hentTrekanttall() + nummer;
    }
```

```java
class TestTrekant{
  public static void main(String[] args) {
    Trekant trekant1 = new Trekant(1, null);
    Trekant trekant2 = new Trekant(2, trekant1);
    Trekant trekant3 = new Trekant(3, trekant2);
    Trekant trekant4 = new Trekant(4, trekant3);
    Trekant trekant5 = new Trekant(5, trekant4);
    Trekant trekant6 = new Trekant(6, trekant5);

    System.out.println(trekant6.hentTrekanttall());
  }
```

Repetisjon denne uka

S O L I D

- **Single responsibility principle**
  En klasse bør bare ha ett ansvarsområde, og dette bør være veldefinert
- **Open/closed principle**
  Komponenter bør være åpne for utvidelser, men lukket for modifikasjoner. Eksempel: klasser kontrollerer hva subklasser kan endre (private, protected, final, etc).
- **Liskov substitution principle**
  Objekter i et program bør kunne byttes ut med instanser av subtyper uten at dette endrer programmets korrekthet.
- **Interface segregation principle**
  Mange små grensesnitt er bedre enn store, monolittiske "super-grensesnitt"
- **Dependency inversion principle**
  Komponenter bør avhenge av abstraksjoner, og ikke konkrete implementasjoner

Enhetstesting

Teste komponenter av koden hver for seg, f.eks. en metode eller en klasse.

Etter enhetstesting kan vi sette komponentene sammen og teste hvordan det fungerer sammen (integrasjontest) og etterhvert hele systemet (systemtesting). Men i in1010 lærer vi bare litt enhetstesting.

Poenget er å finne feil så tidlig som mulig. Dere har sikkert selv erfart at det er vanskeligere å finne feil dess mer kode det er.

Eksempel enhetstesting

Ris, ros, forslag ?

https://nettskjema.no/a/180345

Diskusjon breakout rooms

Repetisjon

- Hvorfor er det lurt med tråder?
- Repeter countdownlatch, cyclic barrier og join join.
- Hva er monitor-design-prinsippet?

Denne uken

- Hva er rekursjon?
- Ukesoppgaver

Breakoutrooms

1. Slå på kamera og ha en presentasjonsrunde
2. Jobb sammen med ukesoppgavene, de ligger på emnesiden -> grupper
    a. Enten ved at én deler skjerm eller med codecollab.io/
    b. OBS: codecollab er gratistjeneseter som UiO ikke har avtale med, sannsynligvis vil de samle data om dere. Dere kan fint løse oppgavene uten å bruke den tjenesten!
3. Bruk “ask for help”-knappen for å få hjelp 😊
4. Vi møtes her igjen for å gå gjennom oppgavene til slutt (dere bestemmer hvilke)

Jobbe med oppgaver