Velkommen!

...

Johanna
johannph på mattermost
johannph@uio.no på mail!

Kort: Praktisk informasjon

- Undervisningstilbud
  - https://www.uio.no/studier/emner/matnat/ifi/IN1010/v21/undervisningstilbud/
- Labtime: 10 minutter med en gruppelærer en til en.
  - Her kan du få hjelp med obligen!
  - Mer info om det på emnesiden.
  - Lag drive (G Suite) bruker allerede nå :D
- Oblig 1, frist i kveld 23.59, det er for sent å levere 00.00!
  - Tips: tegn datastrukturtegning
  - Da får dere får bedre forståelse for programmet og dere unngår feil
  - Utsatt frist: se beskjed på emnesiden
  - Husk å skrive: “Jeg ønsker tilbakemelding” i devilry dersom dere vil ha tilbakemelding! Skriv gjerne hva dere vil ha tilbakemelding på.

Repetisjon forrige uke

Sist

Deklarert vs initialisert

Primitive datatyper

Immutable vs mutable

Referanser

Parameteroverføring

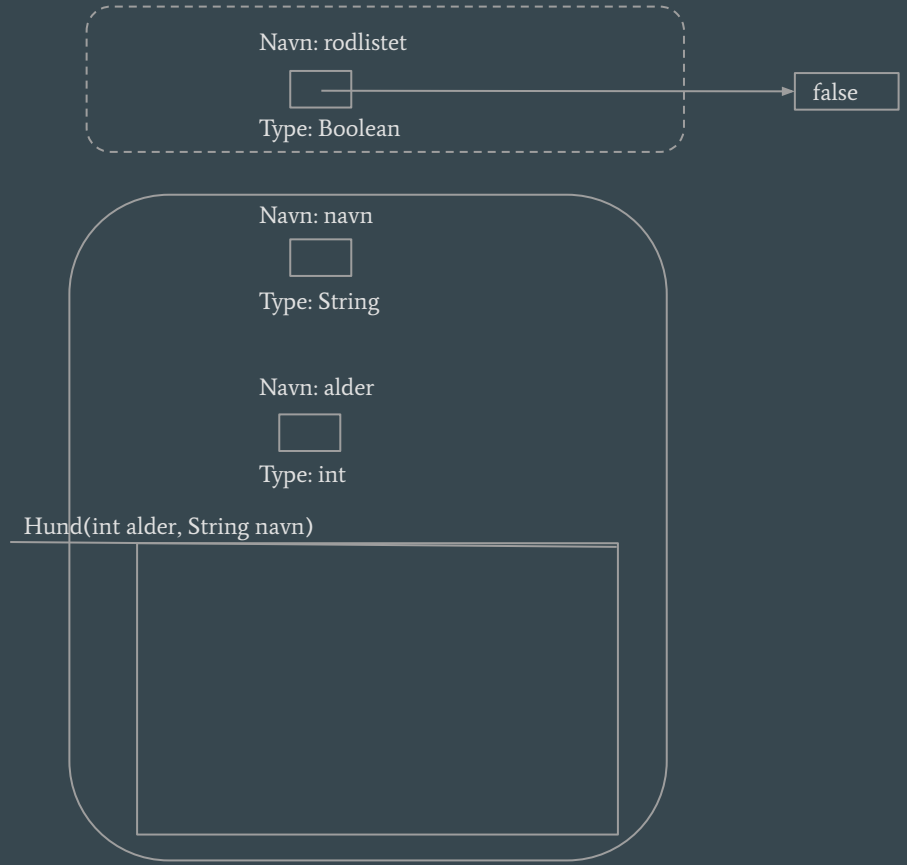
ArrayList

Hashmap

Datastrukturtegning

Error-håndtering

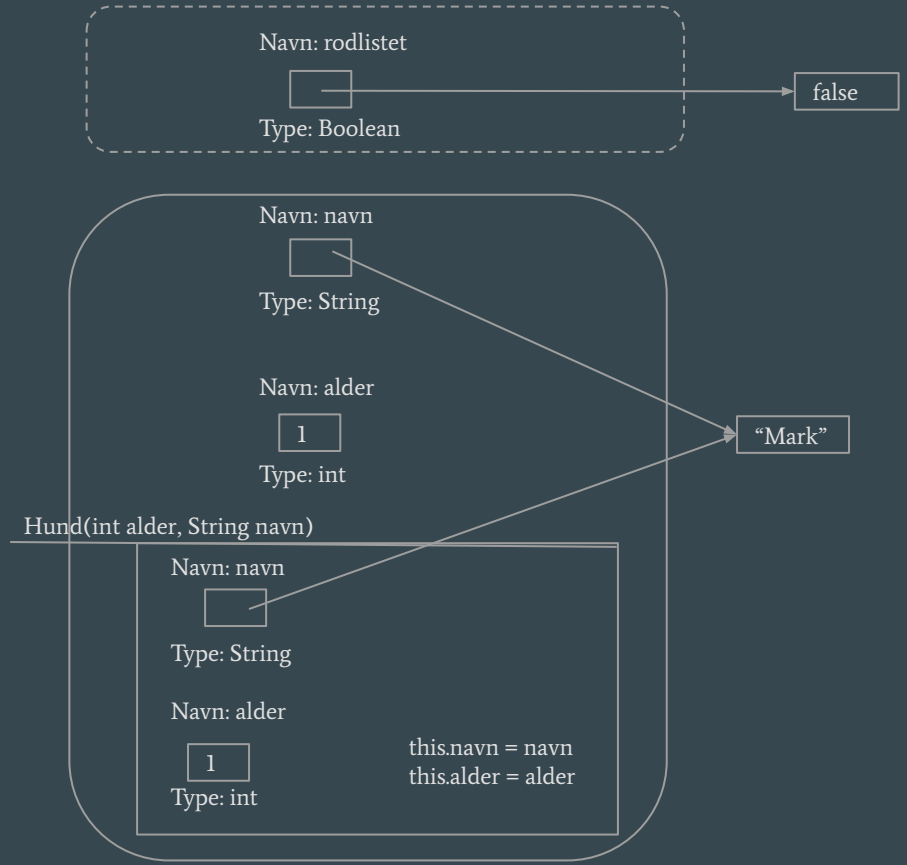
Deklarert vs. initialisert




```java
class Hund{
  private static Boolean rodlistet = false;
  private String navn;
  private int alder;

  public Hund(String navn, int alder){
    this.navn = navn;
    this.alder = alder;
  }
```

Her blir alle variablene deklarert, men det er bare variabelen rodlistet som er initialisert!

Deklarert vs. initialisert




```java
class Hund{
  private static Boolean rodlistet = false;
  private String navn;
  private int alder;

  public Hund(String navn, int alder){
    this.navn = navn;
    this.alder = alder;
  }
```

Først i konstruktøren (init-metoden) blir også navn og alder initialisert!

Legg merke til at to lokale variabler (som ikke er instansvariabler) også blir opprettet.

Primitive data

Primitive data i Java

- int
- char
- float
- boolean (liten b)

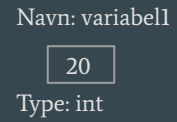
Ikke-primitive data

- String
- Array
- Boolean (stor b)
- Alle klasser

Primitive-data i datastrukturtegning

Primitive data kan vi bare skrive inni boksen i tegningen.

Ikke-primitive data tegnes med pil og en separat boks.

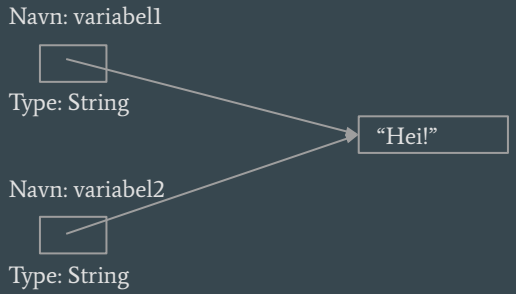


immutble vs. mutable

Strenger og tall er immutable, de kan ikke endres! Kanskje det ser ut som vi endre en string eller et tall, men egentlig lager vi en ny!

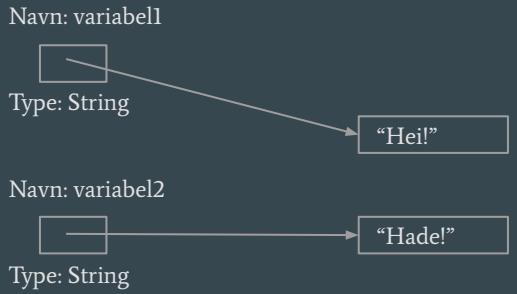
Arrays, dictionaries og mange andre objekter er derimot mutable, altså vi kan endre dem!

Stringer er ikke-primitiv datatype, men de er immutable!




```
variabel1 = "Hei!"
variabel2 = variabel1
variabel2 = "Hade"
```

Stringer er ikke-primitiv datatype, men de er immutable!




```
variabel1 = "Hei!"
variabel2 = variabel1
variabel2 = "Hade"
```

Parameteroverføring

Når dere lager og kaller på metoder som tar argumenter må dere huske på:

- mutable vs immutable
- primitive type vs objekt

Skal jeg bruke Array eller ArrayList?

Array er raskere, men fixed size.

Hvis du vet antall elementer som skal ligge i listen: Array

Hvis ikke: ArrayList

Error-håndtering

Med error håndtering (try/catch)

Uten error-håndtering, sendes problemet videre.

Da trenger man try/catch når man kaller på denne metoden ellers vil programmet terminere(avslutte).

(Siden dette er main som "Throws exception" vil programmet terminere ved error.)

```
jonbon@jons-macbook-pro uke2 % java FraFil1
Kan ikke lese filen !
jonbon@jons-macbook-pro uke2 % java FraFil2
Exception in thread "main" java.io.FileNotFoundException: tekst.txt (No such file or directory)
        at java.base/java.io.FileInputStream.open0(Native Method)
        at java.base/java.io.FileInputStream.open(FileInputStream.java:219)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
        at java.base/java.util.Scanner.<init>(Scanner.java:639)
        at FraFil2.main(Errorhåndtering.java:21)
```

```java
import java.util.Scanner;
import java.io.File;

// Med error-håndtering
class FraFil1{
  public static void main(String[] args){
    // Åpne datafilen:
    Scanner fil = null;
    try {
      fil = new Scanner(new File("tekst.txt"));
    } catch (Exception e) {
        System.out.println("Kan ikke lese filen !");
        System.exit(1);
    }
}
}

// Uten error-håndtering
class FraFil2{
    public static void main(String[] args) throws Exception {
        Scanner fil = new Scanner(new File("tekst.txt"));

        String linje = "";
        while(fil.hasNextLine()) {
            linje = fil.nextLine();
            System.out.println(linje);
        }
    }
}
```

Error-håndtering

Med error håndtering (try/catch)

Uten error-håndtering, sendes problemet videre.

Da trenger man try/catch når man

```java
import java.util.Scanner;
import java.io.File;

// Med error-håndtering
class FraFil1{
  public static void main(String[] args){
    // Åpne datafilen:
    Scanner fil = null;
    try {
      fil = new Scanner(new File("tekst.txt"));
    } catch (Exception e) {
        System.out.println("Kan ikke lese filen !");
        System.exit(1);
    }
}
}

// Uten error-håndtering
class FraFil2{
```

```
jonbon@jons-macbook-pro uke2 % java FraFil1
Kan ikke lese filen !
jonbon@jons-macbook-pro uke2 % java FraFil2
Exception in thread "main" java.io.FileNotFoundException: tekst.txt (No such file or directory)
        at java.base/java.io.FileInputStream.open0(Native Method)
        at java.base/java.io.FileInputStream.open(FileInputStream.java:219)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
        at java.base/java.util.Scanner.<init>(Scanner.java:639)
        at FraFil2.main(Errorhåndtering.java:21)
```

Hvor skal try/catch? Når skal jeg bruke throws isteden?

Tenk på: 1) Hvilke feil kan komme? 2) Hvor forårsakes problemet?

Der feilen skjer bør det håndteres!

F.eks.

1) Hvilke feil kan vi få?
   > Vi bruker Scanner, den kaster en error dersom en fil med filnavnet ikke finnes.
   > Feilen kommer pga filnavnet! Noen har gitt feil filnavn!
2) Hvor forårsakes problemet (der feilen skjer bør det håndteres!)?
   > Feilen bør håndteres der filnavnet gis!

Error-håndtering

Her burde feilen håndteres i main, fordi filnavnet blir gitt i main!

Så den øverste løsningen er best!

```
jonbon@jons-macbook-pro uke2 % java FraFil1
Kan ikke lese filen !
jonbon@jons-macbook-pro uke2 % java FraFil2
Exception in thread "main" java.io.FileNotFoundException: tekst.txt (No such file or directory)
        at java.base/java.io.FileInputStream.open0(Native Method)
        at java.base/java.io.FileInputStream.open(FileInputStream.java:219)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
        at java.base/java.util.Scanner.<init>(Scanner.java:639)
        at FraFil2.main(Errorhåndtering.java:21)
```

```java
import java.util.Scanner;
import java.io.File;

// Med error-håndtering
class FraFil1{
  public static void main(String[] args){
    // Åpne datafilen:
    Scanner fil = null;
    try {
      fil = new Scanner(new File("tekst.txt"));
    } catch (Exception e) {
        System.out.println("Kan ikke lese filen !");
        System.exit(1);
    }
}
}

// Uten error-håndtering
class FraFil2{
    public static void main(String[] args) throws Exception {
        Scanner fil = new Scanner(new File("tekst.txt"));

        String linje = "";
        while(fil.hasNextLine()) {
            linje = fil.nextLine();
            System.out.println(linje);
        }
    }
}
```

Hvor skal try/catch? Når skal jeg bruke throws isteden?

```java
class Fjellrev{
  private static Boolean rodlistet = true;
  private String navn;
  private int alder;
  private Fjellrev bestevenn;

  public Fjellrev(String navn, int alder){
    this.navn = navn;
    this.alder = alder;
  }
  public static void setRodlistet(Boolean rodlistetParameter){
    rodlistet = rodlistetParameter;
  }
  public void settnavn(String navn) throws Exception{
    if (navn.equals("Stygg")){
      throw new Exception("Du kan ikke kalle fjellreven din stygg!");
    }
    this.navn = navn;
  }
```

```java
import java.util.Scanner;

class TestFjellrev {
  public static void main(String[] args) throws Exception{
    Fjellrev fjellrev1 = new Fjellrev("Bjarne", 5);
    Fjellrev fjellrev2 = new Fjellrev("Alfred", 10);
    Fjellrev.setRodlistet(false);
    Scanner minScanner = new Scanner(System.in);
    String input = minScanner.nextLine();
    try{
      fjellrev1.settnavn(input);
    } catch(Exception error){
      System.out.println("Du kan ikke kalle Fjellreven din stygg!");
    }
    System.out.println(fjellrev1.hentNavn());

  }
}
```

Hvor skal try/catch? Når skal jeg bruke throws isteden?

Tenk på: 1) Hvilke feil kan komme? 2) Hvor forårsakes problemet?

Der feilen skjer bør det håndteres!

F.eks.

1) Hvilke feil kan vi få?
   > En Katt kan ikke ha navnet “Stygg”
   > Noen har gitt katten et ulovlig navn!
2) Hvor forårsakes problemet (der feilen skjer bør det håndteres!)?
   > Feilen bør håndteres der Katten får navn!

Hvor skal try/catch? Når skal jeg bruke throws isteden?

```java
class Fjellrev{
  private static Boolean rodlistet = true;
  private String navn;
  private int alder;
  private Fjellrev bestevenn;

  public Fjellrev(String navn, int alder){
    this.navn = navn;
    this.alder = alder;
  }
  public static void setRodlistet(Boolean rodlistetParameter){
    rodlistet = rodlistetParameter;
  }
  public void settnavn(String navn) throws Exception{
    if (navn.equals("Stygg")){
      throw new Exception("Du kan ikke kalle fjellreven din stygg!");
    }
    this.navn = navn;
  }
```

```java
import java.util.Scanner;

class TestFjellrev {
  public static void main(String[] args) throws Exception{
    Fjellrev fjellrev1 = new Fjellrev("Bjarne", 5);
    Fjellrev fjellrev2 = new Fjellrev("Alfred", 10);
    Fjellrev.setRodlistet(false);
    Scanner minScanner = new Scanner(System.in);
    String input = minScanner.nextLine();
    try{
      fjellrev1.settnavn(input);
    } catch(Exception error){
      System.out.println("Du kan ikke kalle Fjellreven din stygg!");
    }
    System.out.println(fjellrev1.hentNavn());

  }
}
```

Hvor skal try/catch? Når skal jeg bruke throws isteden?

Tenk på: 1) Hvilke feil kan komme? 2) Hvor forårsakes problemet?

Der feilen skjer bør det håndteres!

Men feilen må aldri sendes helt ut til terminalen/bruker, slik vi så i FraFil2!! Ok i eksempler, men ikke greit i produksjonskode.

Repetisjon denne uken

I dag

Lese fra fil

Arv

Tegne klassehierarkitegning

Referanser ved Arv

Casting (hva er lov og hva er ikke lov?)

I/O - Lese fra fil

tekstFil.txt

```
1 en
2 to
3 tre
```

- Bruker **Scanner**:
  - Dokummentasjon: https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
  - NB: Må importeres med: import java.util.Scanner
- Sender inn **en fil** i konstruktøren
  - Det forteller Scanner at den skal lese en fil
- Nyttige metoder:
  - hasNextLine(): sjekker om det finnes en ny linje, returnerer en boolean
    - hasNextInt: sjekker om det neste er en int, returnerer en boolean
    - hasNextDouble: sjekker om det neste er en Doble, returnerer en boolean
    - osv....
  - nextLine(): henter den neste Stringen fram til en ny linje
    - nextInt(): henter den neste int-en som er tilgjengelig
    - nextDouble(): henter den neste double-en som er tilgjengelig
    - next(): henter neste ord
    - osv...

```java
import java.util.Scanner;
import java.io.File;
import java.util.HashMap;
import java.io.FileNotFoundException;

class LeseFraFil{

    public static void main(String[] args) {

        File fil = new File("tekstFil.txt");

        Scanner scanner = null;
        try {
            scanner = new Scanner(fil);
        } catch (FileNotFoundException e) {
            System.out.println("Fant ikke filen");
            System.exit(1);
        }

        HashMap<Integer, String> intTilString = new HashMap<>();

        while(scanner.hasNextLine()){
            int tall = scanner.nextInt();
            String tekst = scanner.next();

            intTilString.put(new Integer(tall), tekst);
        }

        System.out.println(intTilString);
    }
}
```

Lese fra fil

```java
class Hund{
  private static Boolean rodlistet = false;
  private String navn;
  private int alder;
  private String favorittgodis = null;

  public Hund(String navn, int alder){
    this.navn = navn;
    this.alder = alder;
  }
  public void settFavorittGodis(String favorittgodis){
    this.favorittgodis = favorittgodis;
  }
  public void skrivUt(){
    System.out.println("Mitt navn er: " + navn + " og jeg er " + alder + " år
    if (favorittgodis != null){
      System.out.println("Min favorittgodis er " + favorittgodis);
    }
    System.out.println("\n");
  }
}
```

```
6
Navn alder favorittgodis
Max 15 eple
Charlie 1 gulrot
Buddy 3 reker
Milo 10 tomat
Oscar 8 ost
Archie 5 Brød
```

Arv

Både Fjellrev og Katt er Dyr, det hadde vært bra å kunne kommunisere det på en måte i koden vår!

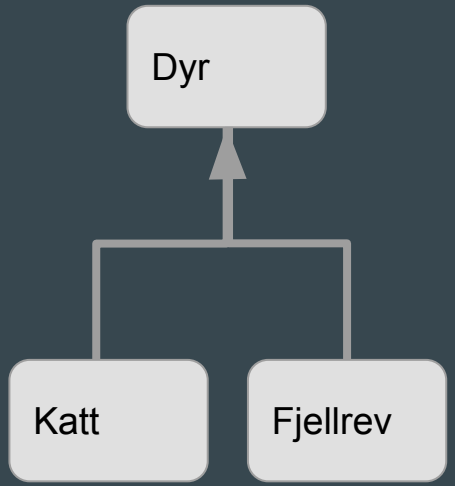
Klassehierarki

Alle Katter er Dyr

Alle Fjellrever er Dyr

Ingen Katter er Fjellrever

Ingen Fjellrever er katter



Klassehierarki

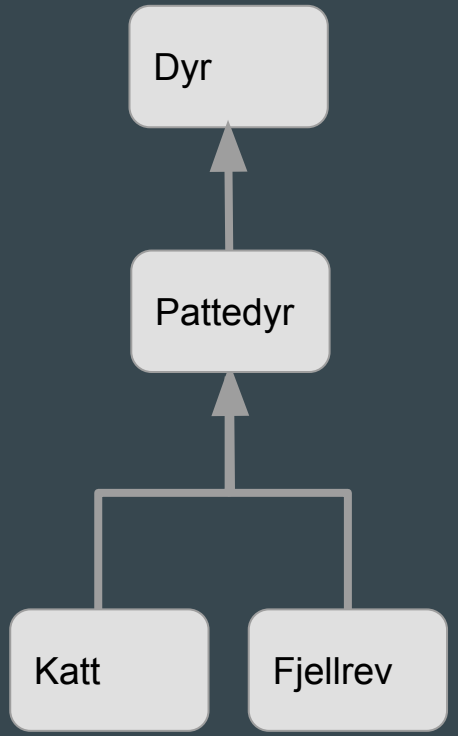
Alle Katter er Dyr

Alle Fjellrever er Dyr

Ingen Katter er Fjellrever

Ingen Fjellrever er katter

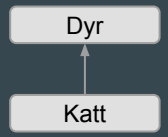
Og man kan ha mange nivåer, f.eks. kan vi legge til pattedyr



Katt og dyr

Her definerer jeg at katter har bostedsadresse, mens andre dyr ikke nødvendigvis har det.

Alle dyr har navn og alder!




```java
class Katt extends Dyr{

  private String bostedsadresse;

  public void settBosted(String bosted){
    bostedsadresse = bosted;
  }
  public String hentBosted(){
    return bostedsadresse;
  }
}
```

```java
class Dyr{
  protected int alder;
  protected String navn;

  public void settAlder(int alder){
    this.alder = alder;
  }
  public  int hentAlder(){
    return alder;
  }
  public  void settNavn(String navn){
    this.navn = navn;
  }
  public  String hentNavn(){
    return navn;
  }
  public void skrivUtInfo(){
    System.out.println("Navn: " + navn);
    System.out.println("Alder: " + alder);
  }
}
```

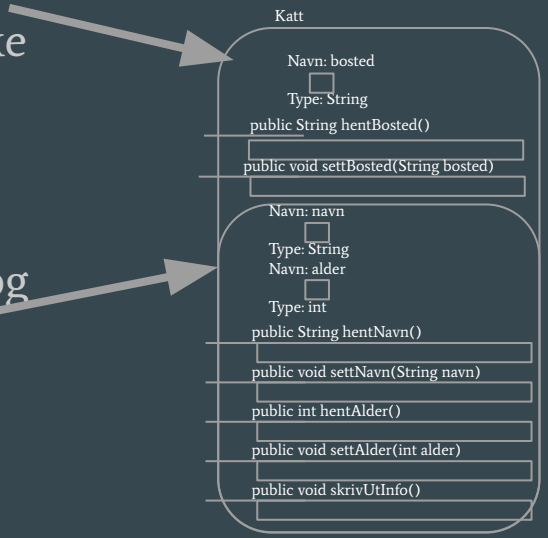
Katt og dyr

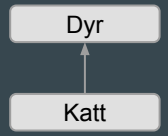
Her definerer jeg at katter har bostedsadresse, mens andre dyr ikke nødvendigvis har det.

Alle dyr har navn og alder!

```java
class Katt extends Dyr{

  private String bostedsadresse;

  public void settBosted(String bosted){
    bostedsadresse = bosted;
  }
  public String hentBosted(){
    return bostedsadresse;
  }
}
```

```java
class Dyr{
  protected int alder;
  protected String navn;

  public void settAlder(int alder){
    this.alder = alder;
  }
  public  int hentAlder(){
    return alder;
  }
  public  void settNavn(String navn){
    this.navn = navn;
  }
  public  String hentNavn(){
    return navn;
  }
  public void skrivUtInfo(){
    System.out.println("Navn: " + navn);
    System.out.println("Alder: " + alder);
  }
}
```



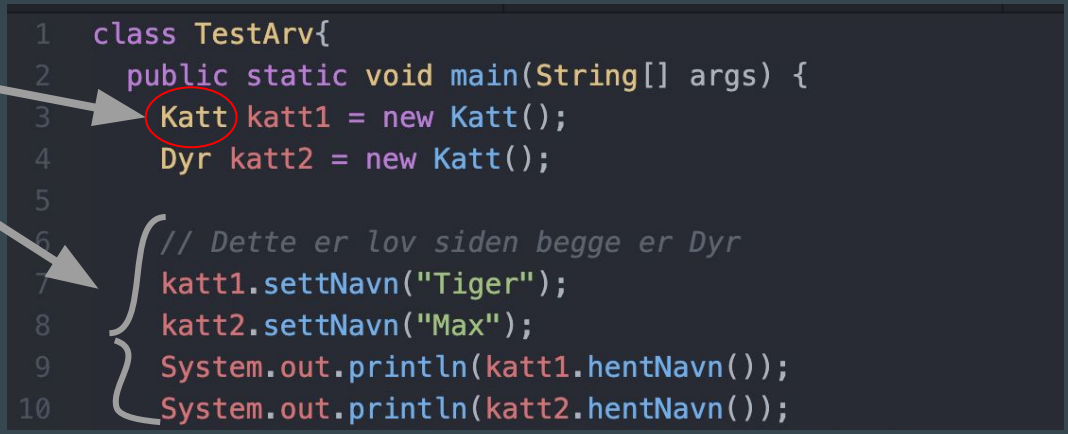


Alle Katter er Dyr

Alle katter får automatisk alle egenskapene til Dyr!

Husk at metoden settNavn() og HentNavn() var metoder i klassen Dyr, og ikke metoder i klassen Katt, siden alle Dyr har navn (i dette programmet).

Det går helt fint å legge en referanse til et objekt av klassen Katt i variabelen katt2, fordi Katt er et Dyr



Arv

En referanse til et Katteobjekt kan ligge i en variabel med type Dyr og Katt

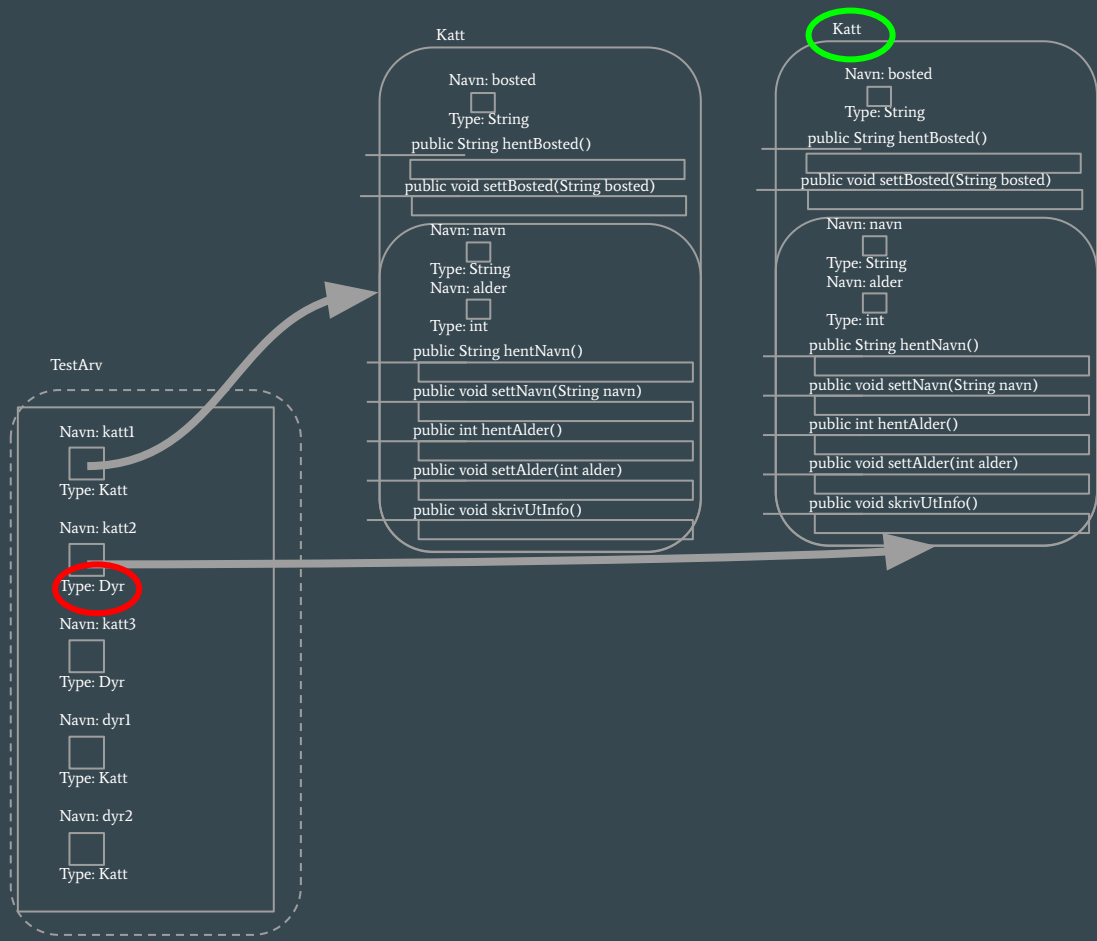
En referanse til et Dyreobjekt kan **ikke** ligge i en variabel med type Katt, heller ikke hvis vi caster.

Generelt: god stil å unngå casting så mye som du kan!

```
1  class TestArv{
2    public static void main(String[] args) {
3      // Lov!
4      Katt katt1 = new Katt();
5      Dyr katt2 = new Katt();
6      // Lov
7      Katt katt3 = (Katt) katt2;
8      // Ikke lov
9      Katt Dyr1 = new Dyr();   Error
10     // heller ikke lov
11     Katt Dyr2 = (Katt) new Dyr();   Error
12
```

Datastrukturtegning

```
1  class TestArv{
2    public static void main(String[] args) {
3      // Lov!
4      Katt katt1 = new Katt();
5      Dyr katt2 = new Katt();
6      // Lov
7      Katt katt3 = (Katt) katt2;
8      // Ikke lov
9      Katt dyr1 = new Dyr();     Error
10     // heller ikke lov
11     Katt dyr2 = (Katt) new Dyr(); Error
12
```

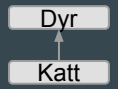


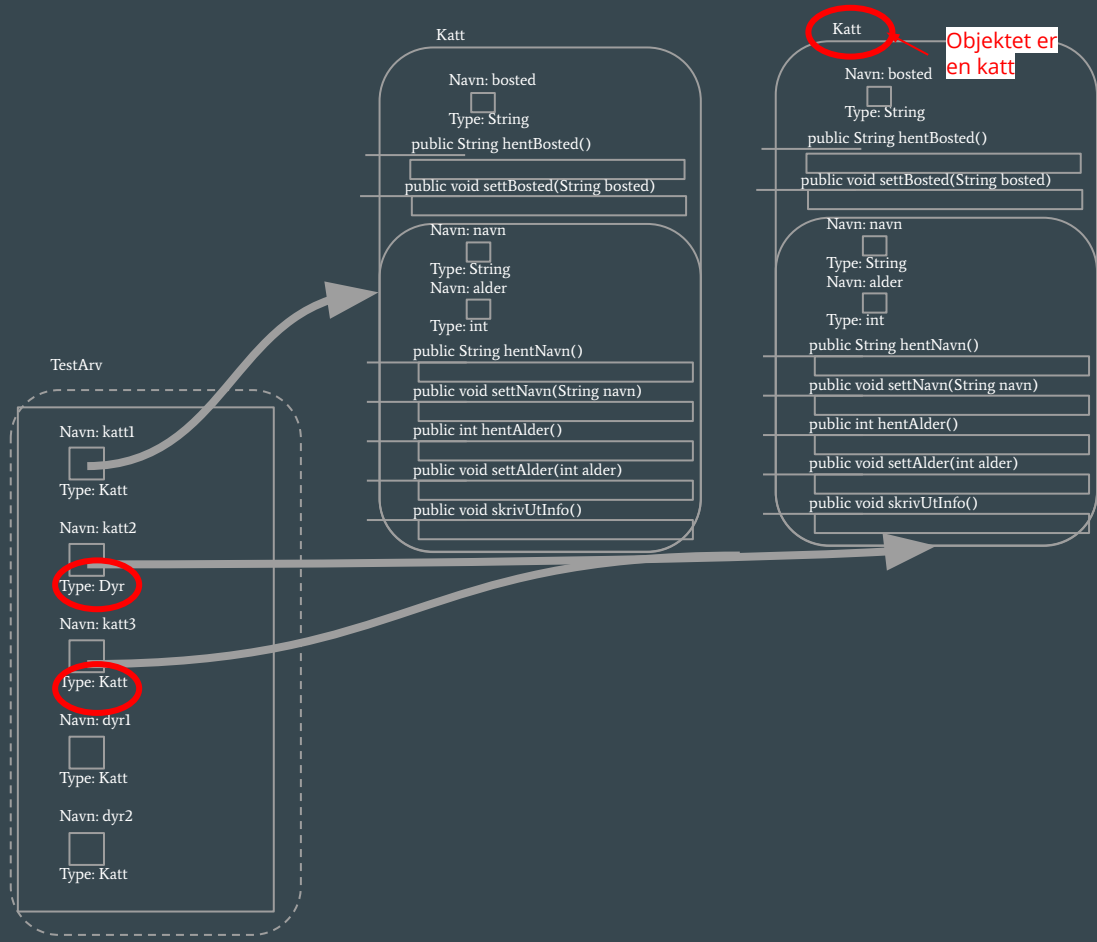
Datastrukturtegning

```
1  class TestArv{
2    public static void main(String[] args) {
3      // Lov!
4      Katt katt1 = new Katt();
5      Dyr katt2 = new Katt();
6      // lov
7      Katt katt3 = (Katt) katt2;
8      // Ikke lov
9      Katt dyr1 = new Dyr();      Error
10     // heller ikke lov
11     Katt dyr2 = (Katt) new Dyr(); Error
12
```

Casting betyr at vi forteller kompilatoren at vi vet hva vi driver med! Vi vet at objektet som katt2 referer til er en Katt. Hvis vi ikke caster får vi kompileringsfeil

Må caste her fordi typen til variablen katt2 er Dyr.

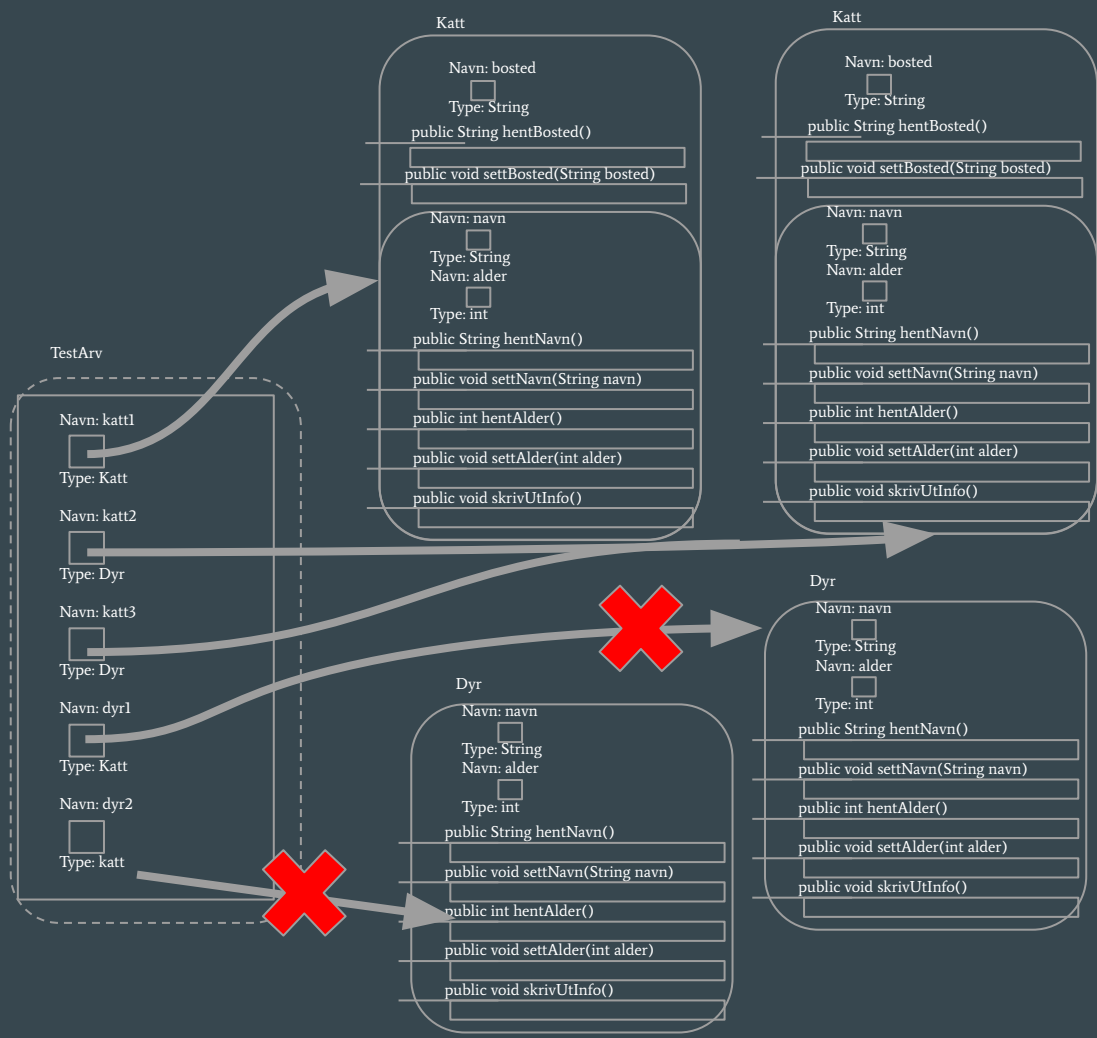




Datastrukturtegning

```java
class TestArv{
  public static void main(String[] args) {
    // Lov!
    Katt katt1 = new Katt();
    Dyr katt2 = new Katt();
    // Lov
    Katt katt3 = (Katt) katt2;
    // Ikke lov
    Katt dyr1 = new Dyr();     Error
    // heller ikke lov
    Katt dyr2 = (Katt) new Dyr(); Error

```

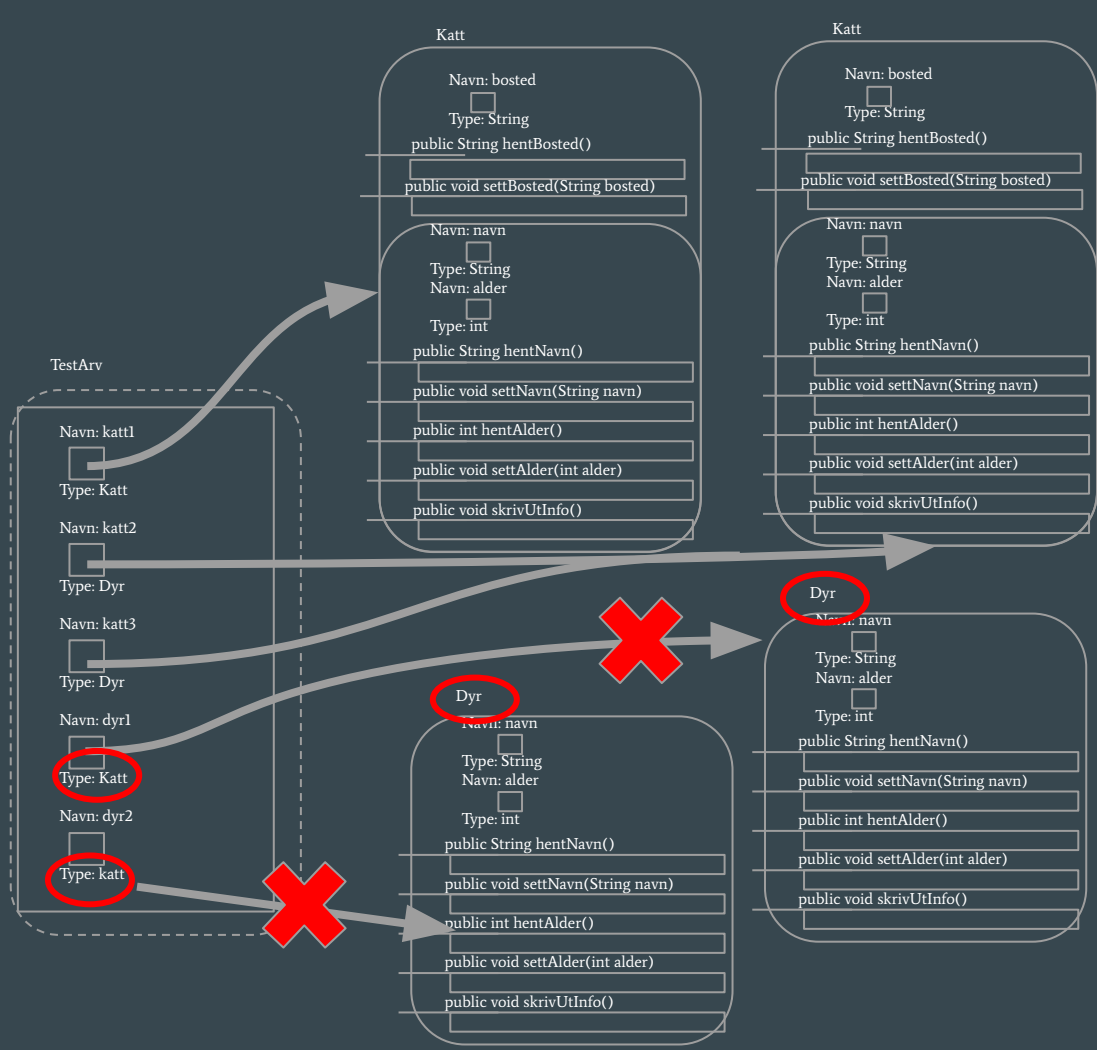
1. Vi får kompileringsfeil
   - ➢ "Løses" med casting, kompilatoren klager ikke fordi vi har sagt at vi vet hva vi driver med
2. Dyr er ikke Katt: run time error
   - ➢ kan ikke løses



Datastrukturtegning

```
1  class TestArv{
2    public static void main(String[] args) {
3      // Lov!
4      Katt katt1 = new Katt();
5      Dyr katt2 = new Katt();
6      // Lov
7      Katt katt3 = (Katt) katt2;
8      // Ikke lov
9      Katt dyr1 = new Dyr();      Error
10     // heller ikke lov
11     Katt dyr2 = (Katt) new Dyr(); Error
12
```

1. Vi får kompileringsfeil
   - ➢ "Løses" med casting, kompilatoren klager ikke fordi vi har sagt at vi vet hva vi driver med
2. Dyr er ikke Katt: run time error
   - ➢ kan ikke løses



Arv

En referanse til et Katteobjekt kan ligge i en variabel med type Dyr og Katt

En referanse til et Dyreobjekt kan **ikke** ligge i en variabel med type Katt, heller ikke hvis vi caster.

Generelt: god stil å unngå casting så mye som du kan!

```
1  class TestArv{
2    public static void main(String[] args) {
3      // Lov!
4      Katt katt1 = new Katt();
5      Dyr katt2 = new Katt();
6      // Lov
7      Katt katt3 = (Katt) katt2;
8      // Ikke lov
9      Katt Dyr1 = new Dyr();   Error
10     // heller ikke lov
11     Katt Dyr2 = (Katt) new Dyr();   Error
12
```

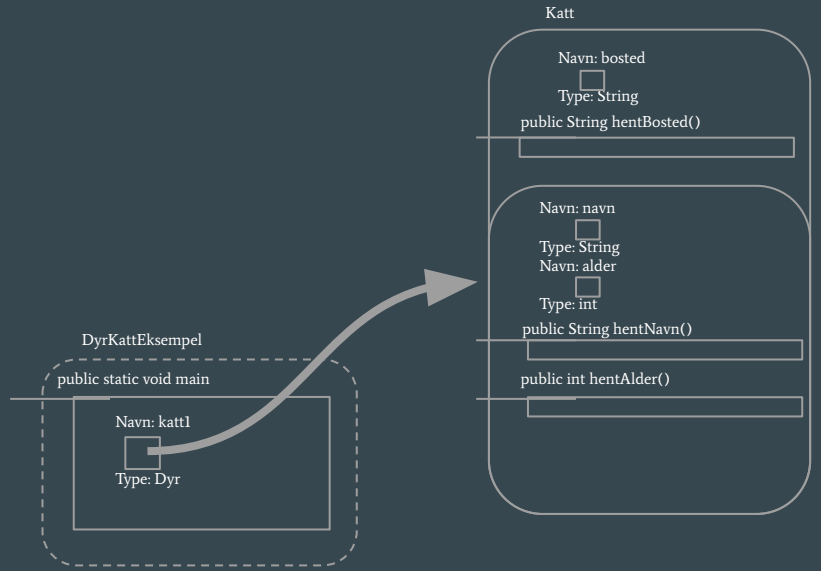
Casting eksempel

```java
class Katt extends Dyr{

    private String bosted;

    public Katt(String navn, int alder, String bosted){
        super(navn, alder);
        this.bosted = bosted;
    }

    public String hentBosted(){
        return bosted;
    }
}
```

```java
abstract class Dyr {

    protected String navn;
    protected int alder;

    public Dyr(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }

    public String hentNavn(){
        return navn;
    }

    public int hentAlder(){
        return alder;
    }
}
```

```java
class Katt extends Dyr{

    private String bosted;

    public Katt(String navn, int alder, String bosted){
        super(navn, alder);
        this.bosted = bosted;
    }

    public String hentBosted(){
        return bosted;
    }
}
```

```java
abstract class Dyr {

    protected String navn;
    protected int alder;

    public Dyr(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }

    public String hentNavn(){
        return navn;
    }

    public int hentAlder(){
        return alder;
    }
}
```

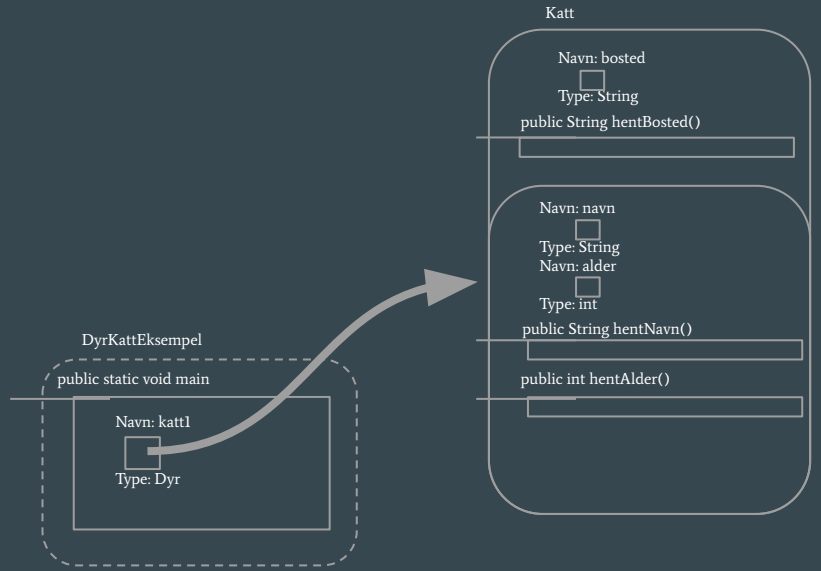



```java
class DyrKattEksempel{

    public static void main(String[] args) {

        Dyr katt1 = new Katt("Pus", 1, "Oslo");

        System.out.println(katt1.hentNavn())
        System.out.println(katt1.hentAlder())
        System.out.println(katt1.hentBosted())
    }
}
```

```java
class Katt extends Dyr{

    private String bosted;

    public Katt(String navn, int alder, String bosted){
        super(navn, alder);
        this.bosted = bosted;
    }

    public String hentBosted(){
        return bosted;
    }
}
```

```java
abstract class Dyr {

    protected String navn;
    protected int alder;

    public Dyr(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }

    public String hentNavn(){
        return navn;
    }

    public int hentAlder(){
        return alder;
    }
}
```

Siden variabelen er av type dyr må vi ta på oss dyre brillene

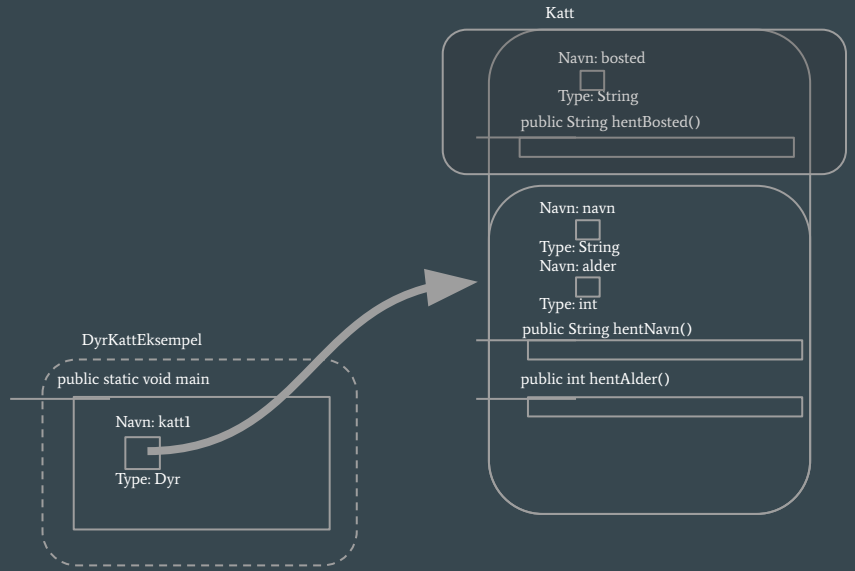



```java
class DyrKattEksempel{

    public static void main(String[] args) {

        Dyr katt1 = new Katt("Pus", 1, "Oslo");

        System.out.println(katt1.hentNavn())
        System.out.println(katt1.hentAlder())
        System.out.println(katt1.hentBosted())
    }
}
```

```java
class Katt extends Dyr{

    private String bosted;

    public Katt(String navn, int alder, String bosted){
        super(navn, alder);
        this.bosted = bosted;
    }

    public String hentBosted(){
        return bosted;
    }
}
```

```java
abstract class Dyr {

    protected String navn;
    protected int alder;

    public Dyr(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }

    public String hentNavn(){
        return navn;
    }

    public int hentAlder(){
        return alder;
    }
}
```

Katt1 ser nå kun egenskapen til klassen Dyr

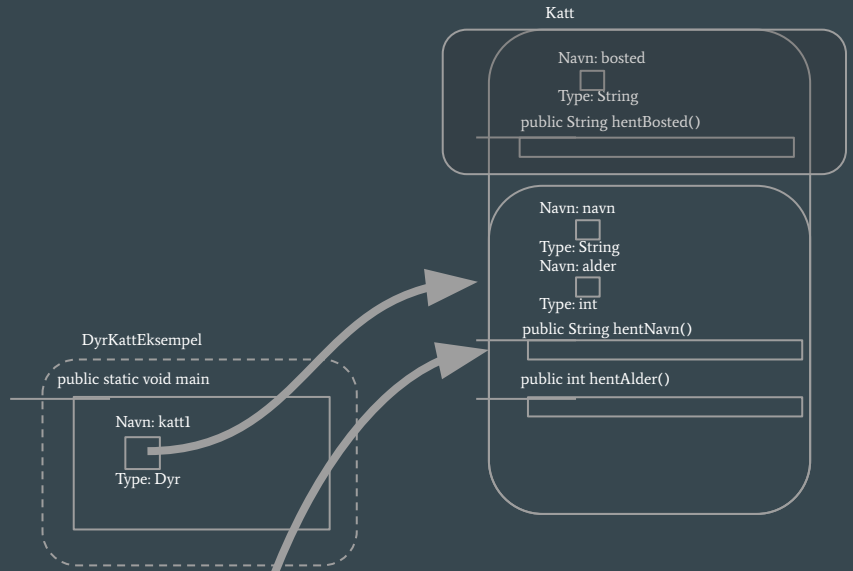



```java
class DyrKattEksempel{

    public static void main(String[] args) {

        Dyr katt1 = new Katt("Pus", 1, "Oslo");

        System.out.println(katt1.hentNavn())
        System.out.println(katt1.hentAlder())
        System.out.println(katt1.hentBosted())
    }
}
```

```java
class Katt extends Dyr{

    private String bosted;

    public Katt(String navn, int alder, String bosted){
        super(navn, alder);
        this.bosted = bosted;
    }

    public String hentBosted(){
        return bosted;
    }
}
```

```java
abstract class Dyr {

    protected String navn;
    protected int alder;

    public Dyr(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }

    public String hentNavn(){
        return navn;
    }

    public int hentAlder(){
        return alder;
    }
}
```

Output: Pus

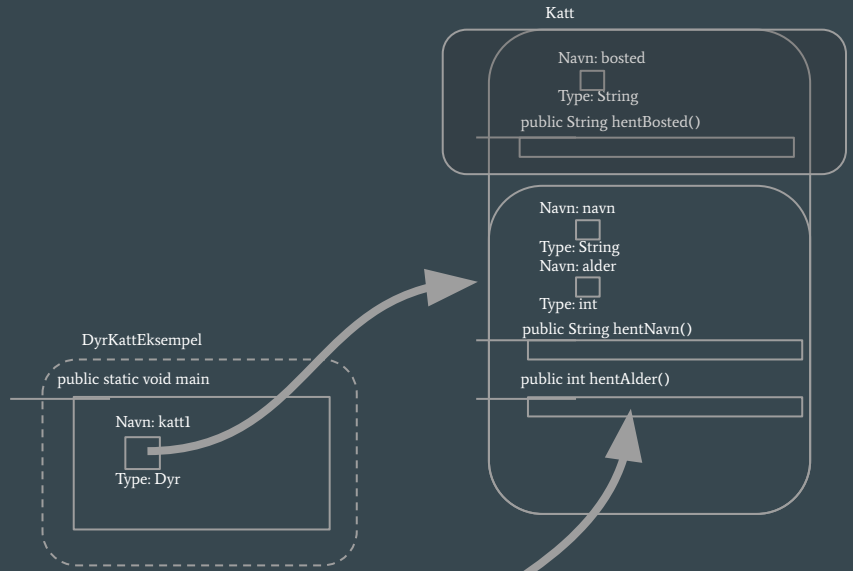



```java
class DyrKattEksempel{

    public static void main(String[] args) {

        Dyr katt1 = new Katt("Pus", 1, "Oslo");

        System.out.println(katt1.hentNavn())
        System.out.println(katt1.hentAlder())
        System.out.println(katt1.hentBosted())
    }
}
```

```java
class Katt extends Dyr{

    private String bosted;

    public Katt(String navn, int alder, String bosted){
        super(navn, alder);
        this.bosted = bosted;
    }

    public String hentBosted(){
        return bosted;
    }
}
```

```java
abstract class Dyr {

    protected String navn;
    protected int alder;

    public Dyr(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }

    public String hentNavn(){
        return navn;
    }

    public int hentAlder(){
        return alder;
    }
}
```

Output: 1

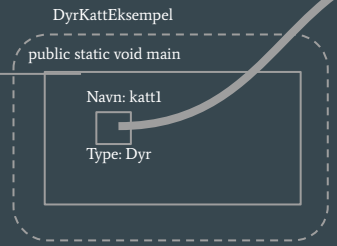


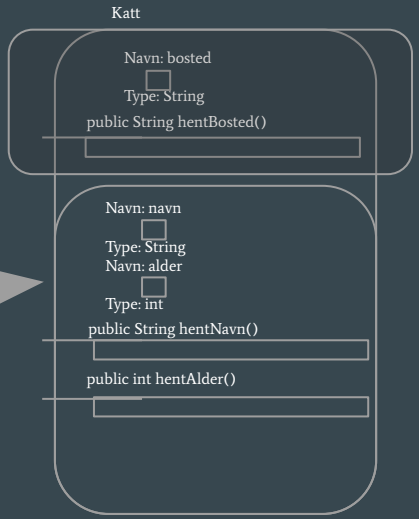

```java
class DyrKattEksempel{

    public static void main(String[] args) {

        Dyr katt1 = new Katt("Pus", 1, "Oslo");

        System.out.println(katt1.hentNavn())
        System.out.println(katt1.hentAlder())
        System.out.println(katt1.hentBosted())
    }
}
```

```java
class Katt extends Dyr{

    private String bosted;

    public Katt(String navn, int alder, String bosted){
        super(navn, alder);
        this.bosted = bosted;
    }

    public String hentBosted(){
        return bosted;
    }
}
```

```java
abstract class Dyr {

    protected String navn;
    protected int alder;

    public Dyr(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }

    public String hentNavn(){
        return navn;
    }

    public int hentAlder(){
        return alder;
    }
}
```

Feilmelding. Siden katt1 ikke ser egenskapene til Katt objektet og dermed heller ikke metoden hentBosted, derfor må vi caste!





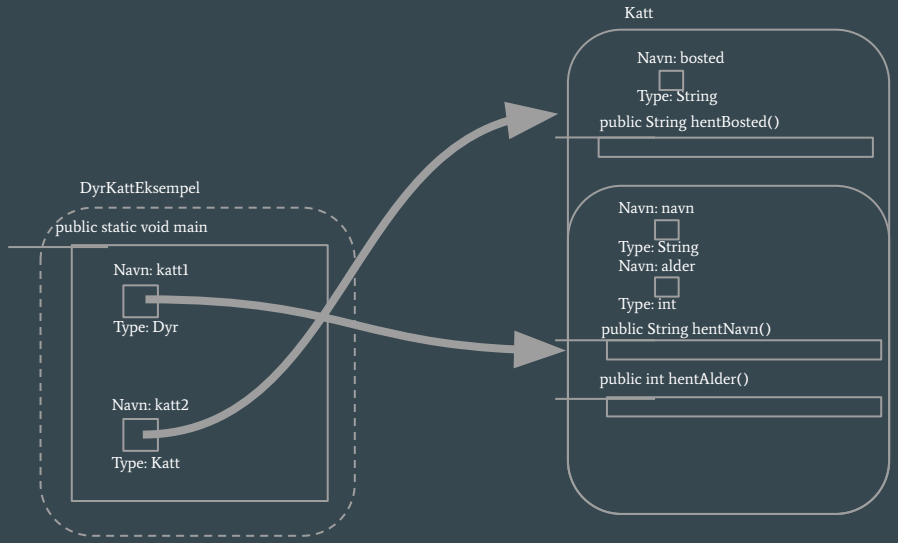

```java
class DyrKattEksempel{

    public static void main(String[] args) {

        Dyr katt1 = new Katt("Pus", 1, "Oslo");

        System.out.println(katt1.hentNavn())
        System.out.println(katt1.hentAlder())
        System.out.println(katt1.hentBosted())
    }
}
```

?? X

```java
class Katt extends Dyr{

    private String bosted;

    public Katt(String navn, int alder, String bosted){
        super(navn, alder);
        this.bosted = bosted;
    }

    public String hentBosted(){
        return bosted;
    }
}
```

```java
abstract class Dyr {

    protected String navn;
    protected int alder;

    public Dyr(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }

    public String hentNavn(){
        return navn;
    }

    public int hentAlder(){
        return alder;
    }
}
```

```java
class DyrKattEksempel{

    public static void main(String[] args) {

        Dyr katt1 = new Katt("Pus", 1, "Oslo");

        System.out.println(katt1.hentNavn())
        System.out.println(katt1.hentAlder())
        //System.out.println(katt1.hentBosted())

        Katt katt2 = (Katt) katt1;
        System.out.println(katt2.hentBosted());
        System.out.println(katt2.hentNavn());

    }
}
```

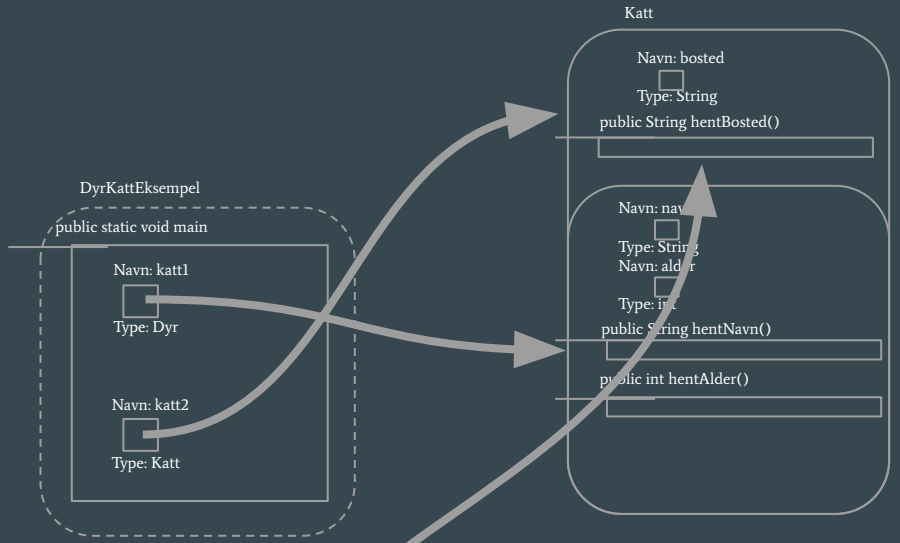


Når vi nå ser på katte objektet men katt2 sine briller får vi også alle egenskapene til en Katt

PS: katt1 kan fortsatt bare se dyreegenskapene

```java
class Katt extends Dyr{

    private String bosted;

    public Katt(String navn, int alder, String bosted){
        super(navn, alder);
        this.bosted = bosted;
    }

    public String hentBosted(){
        return bosted;
    }
}
```

```java
abstract class Dyr {

    protected String navn;
    protected int alder;

    public Dyr(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }

    public String hentNavn(){
        return navn;
    }

    public int hentAlder(){
        return alder;
    }
}
```



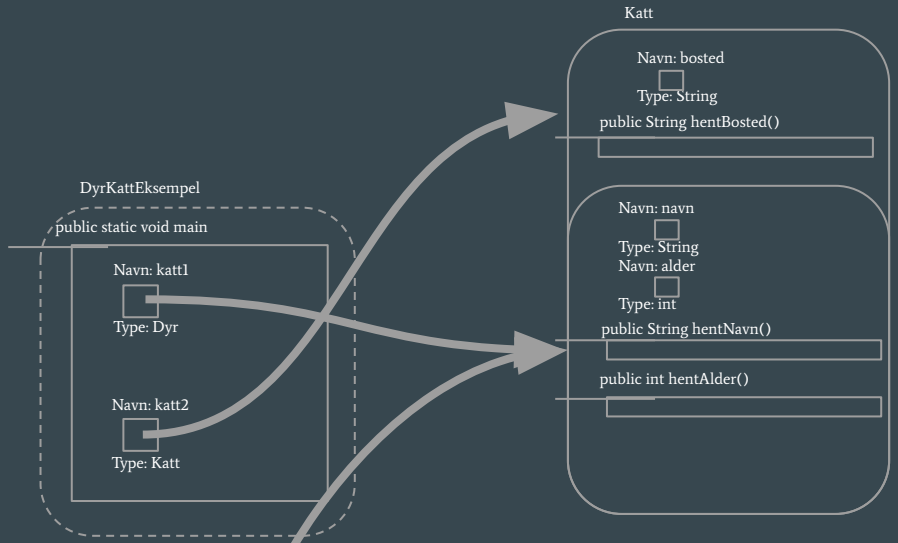

```java
class DyrKattEksempel{

    public static void main(String[] args) {

        Dyr katt1 = new Katt("Pus", 1, "Oslo");

        System.out.println(katt1.hentNavn());
        System.out.println(katt1.hentAlder());
        //System.out.println(katt1.hentBosted())

        Katt katt2 = (Katt) katt1;
        System.out.println(katt2.hentBosted());
        System.out.println(katt2.hentNavn());

    }
}
```

Output: Oslo

```java
class Katt extends Dyr{

    private String bosted;

    public Katt(String navn, int alder, String bosted){
        super(navn, alder);
        this.bosted = bosted;
    }

    public String hentBosted(){
        return bosted;
    }
}
```

```java
abstract class Dyr {

    protected String navn;
    protected int alder;

    public Dyr(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }

    public String hentNavn(){
        return navn;
    }

    public int hentAlder(){
        return alder;
    }
}
```

```java
class DyrKattEksempel{

    public static void main(String[] args) {

        Dyr katt1 = new Katt("Pus", 1, "Oslo");

        System.out.println(katt1.hentNavn())
        System.out.println(katt1.hentAlder())
        //System.out.println(katt1.hentBosted())

        Katt katt2 = (Katt) katt1;
        System.out.println(katt2.hentBosted());
        System.out.println(katt2.hentNavn());

    }
}
```



Output: Pus

```
class A {}
class B extends A {}
class C extends A {}
class D extends C {}

class TrueOrFalse {
        public static void main(String[] args) {
                A a = new A();
                B b = new B();
                C c = new C();
                D d = new D();
                A e = new D();
                A f = new B();

                System.out.println("a) " + (b instanceof A));
                System.out.println("b) " + (c instanceof D));
                System.out.println("c) " + (d instanceof A));
                System.out.println("d) " + (e instanceof C));
                System.out.println("e) " + (f instanceof C));
        }
}
```