
Westerdals

Oslo School of Arts,
Communication and Technology

PGR 101 Objektorientert Programmering 2
Vår 2017

# Forelesning 28.2.17

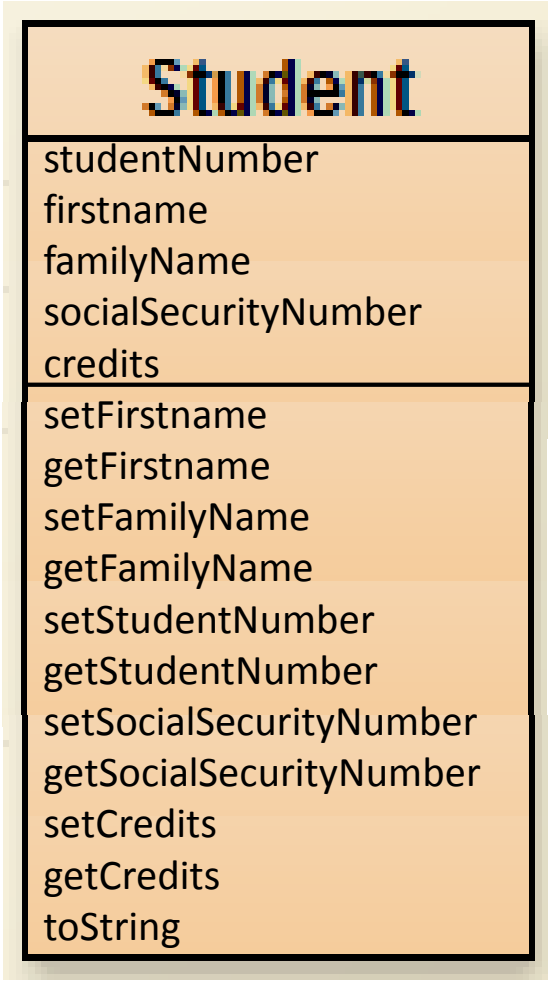
(Stein Marthinsen – marste@westerdals.no)

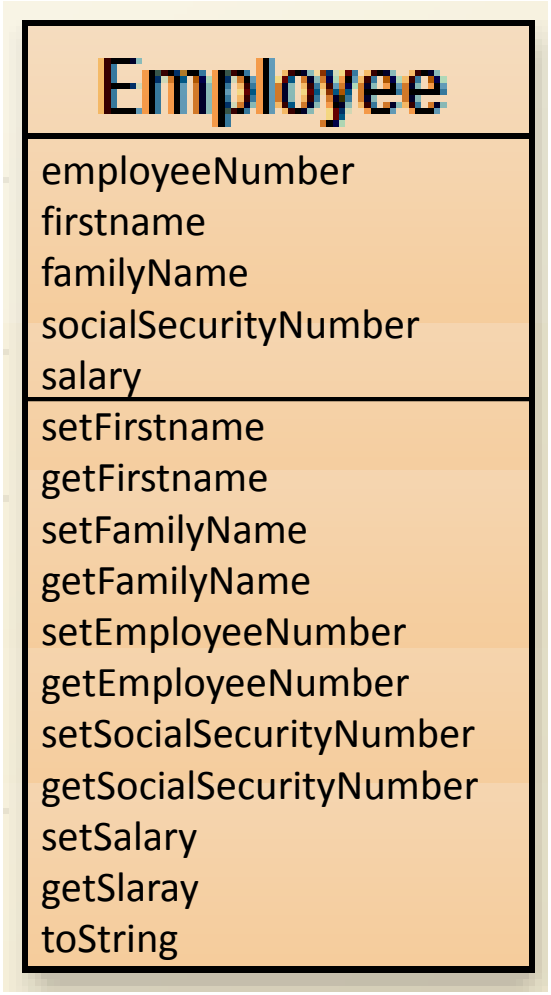
# Dagens tema

## Mer ARV

- Fra forrige gang – project *`westerdals`* med ARV
- Det som kan/må endres:
  - Klassen `Westerdals` – tilpasses nye relasjoner
  - Metoden `toString()` – virker den som forventet?
- Utvidelse med flere typer ansatte.
- Superklasse/subklasse – polymorfi
- Overriding
- Bruk av `super`-kall.

# I utgangspunktet

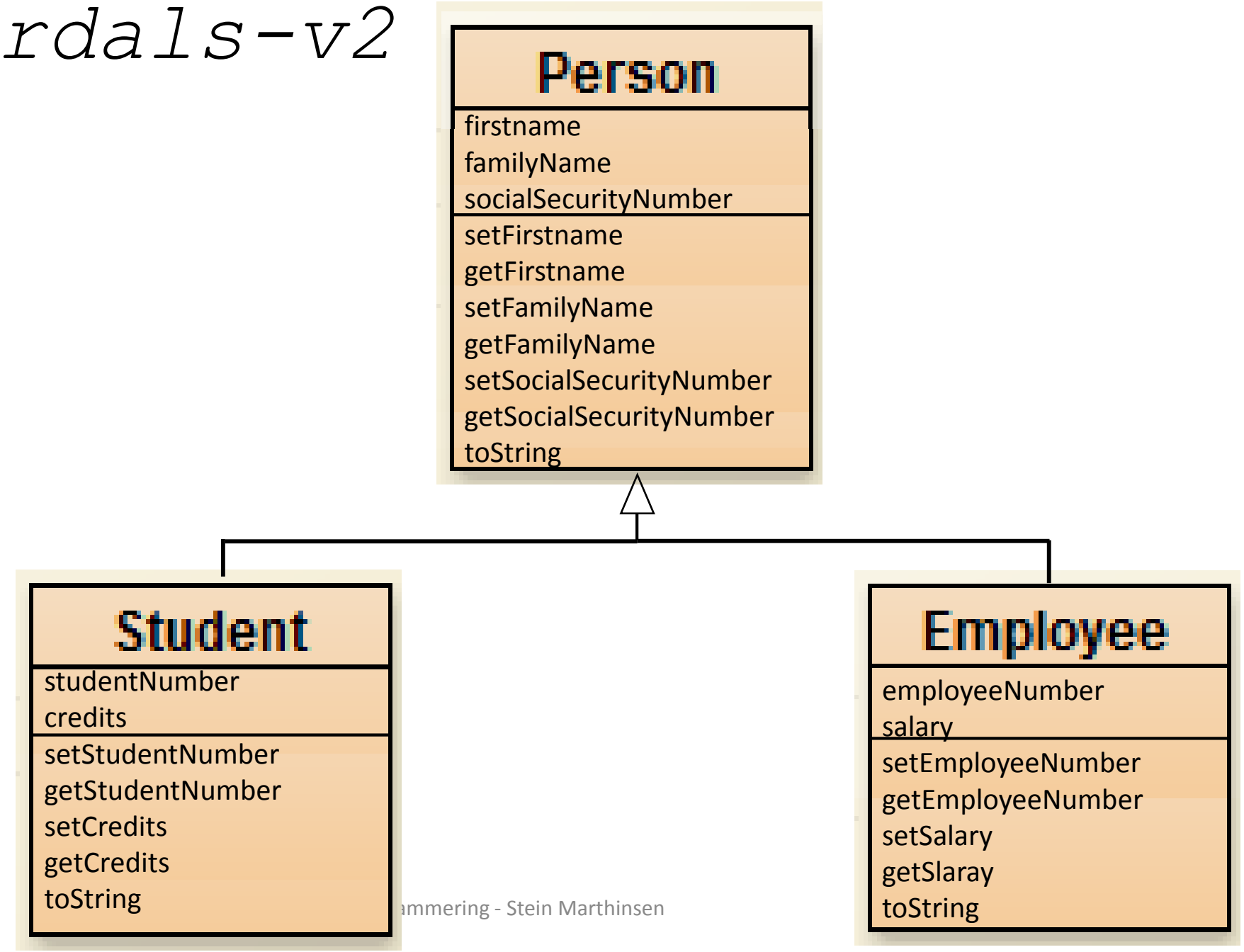


# Project *westerdals-v2*

superklasse

subklasser



# Bruk av ARV

- definerer en **superklasse** : `Person`
- definerer **subklasser** for `Student` og `Employee`
- superklassen definerer *felles fields* (attributter) og *metoder*
- subklassen **arver** superklasse-*fields* og *metoder*
- subklassen legger til *spesielle fields* og *metoder*

# Superklassen (er som en "vanlig" klasse)

```
public class Person {
    private String firstName;
    private String familyName;                    // felles for subklassene
    private String socialSecurityNumber;

    // konstruktør og metoder utelatt.
}
```

# Klassen Student – konstruktøren

```
public class Student extends Person {
   private String studentNumber;
   private int credits;

   public Student(String studentNumber, String socialSecurityNumber,
                  String firstName, String familyName, int credits) {
      super(firstName, familyName, socialSecurityNumber);
      setStudentNumber(studentNumber);
      setCredits(credits);
   }

   // metoder utelatt
}
```

alle data for et Student-objekt!

sender noen av dataene til `Person` sin konstruktør (**super**klassen)!

tar seg av egne data

Tilsvarende for `Employee`-klassen!

# Klassen Person – konstruktøren

```
public class Person {
    private String firstName;
    private String familyName;
    private String socialSecurityNumber;

    public Person(String firstName, String familyName, String socSecNum) {
        setFirstName(firstName);
        setFamilyName(familyName);
        setSocialSecurityNumber(socSecNum);
    }

     // metoder utelatt
}
```

tar imot data til sine fields

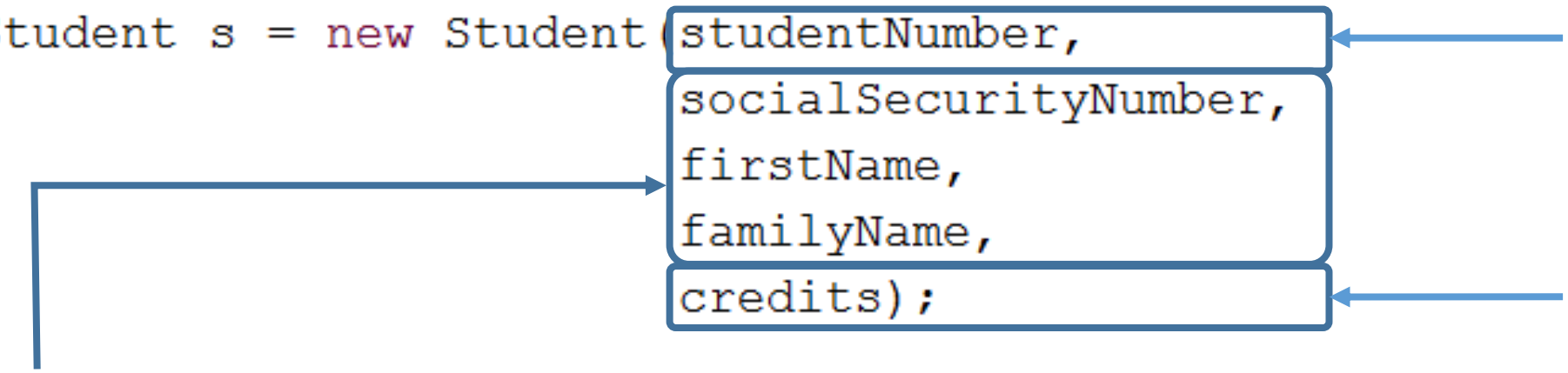
# `Student` sin konstruktør kaller `Person` sin konstruktør

**Må være første setning!**

```
//Konstruktør
public Student(String studentNumber, String socialSecurit
    super(firstName, familyName, socialSecurityNumber);
    setStudentNumber(studentNumber);
    setCredits(credits);
}
```

# Oppretter objekt

```
String studentNumber = "123";
String socialSecurityNumber = "5767";
String firstName = "Stud";
String familyName = "ent";
int credits = 20;
Student s = new Student(studentNumber,
                        socialSecurityNumber,
                        firstName,
                        familyName,
                        credits);
```



*Disse* opplysningene sendes til konstruktøren i klassen `Person`, som setter de inn i passende fields der.

De to andre opplysningene settes inn i passende fields av konstruktøren i klassen `Student`.

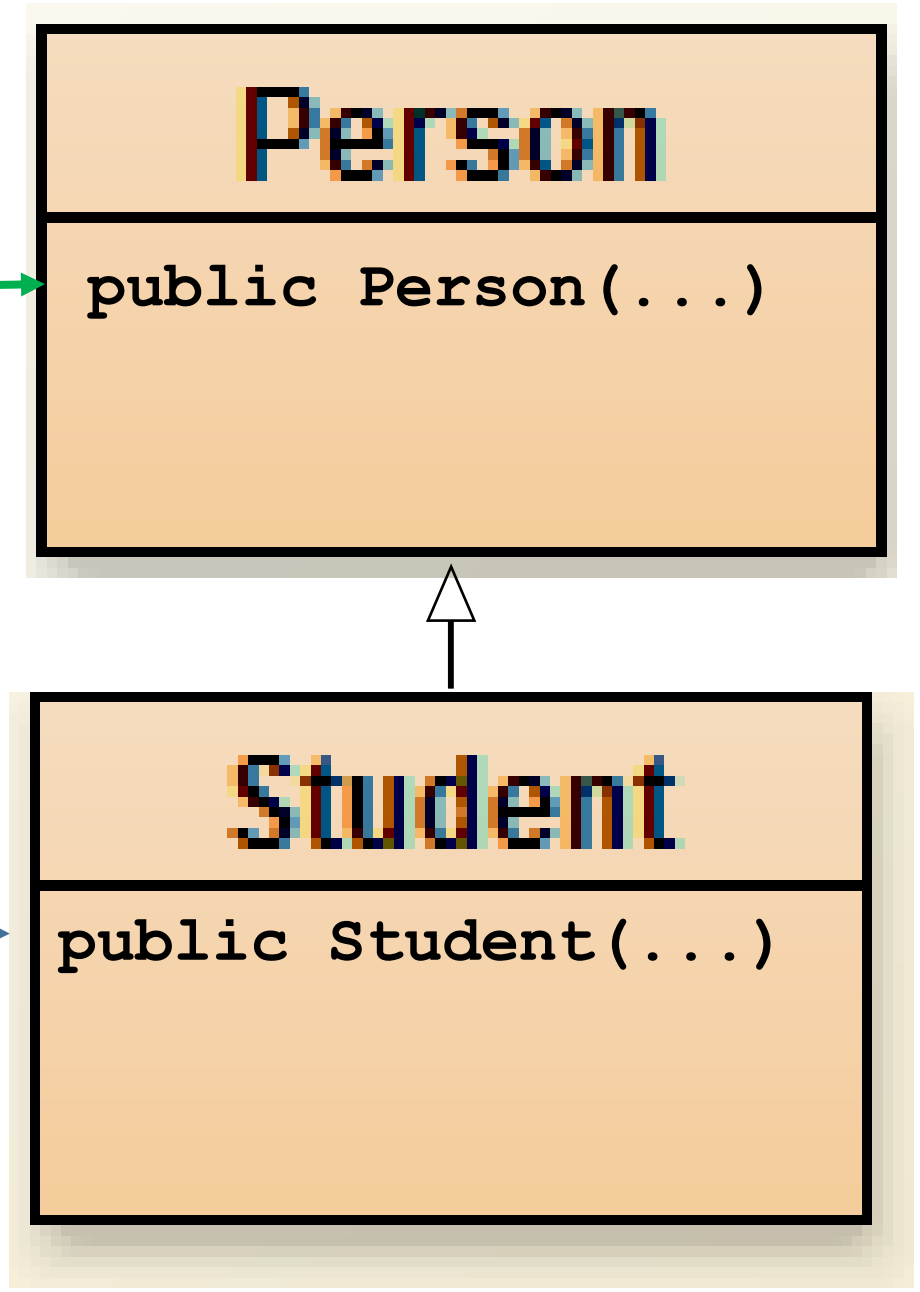
# Konstruktør-kall

Når et `Student`-objekt opprettes, kalles `Student` sin konstruktør.

```
Student s1 = new Student(...);
```

Denne tar imot alle data, og sender noe videre til `Person` sin konstruktør.

```
super(...);
```



# Student *er en* Person

Vi kan skrive:

```
Student stud1 = new Student(....);
```

Og vi kan nå også skrive:

```
Person pers1 = new Student(....);
```

# Regel

Et **Student-objekt**

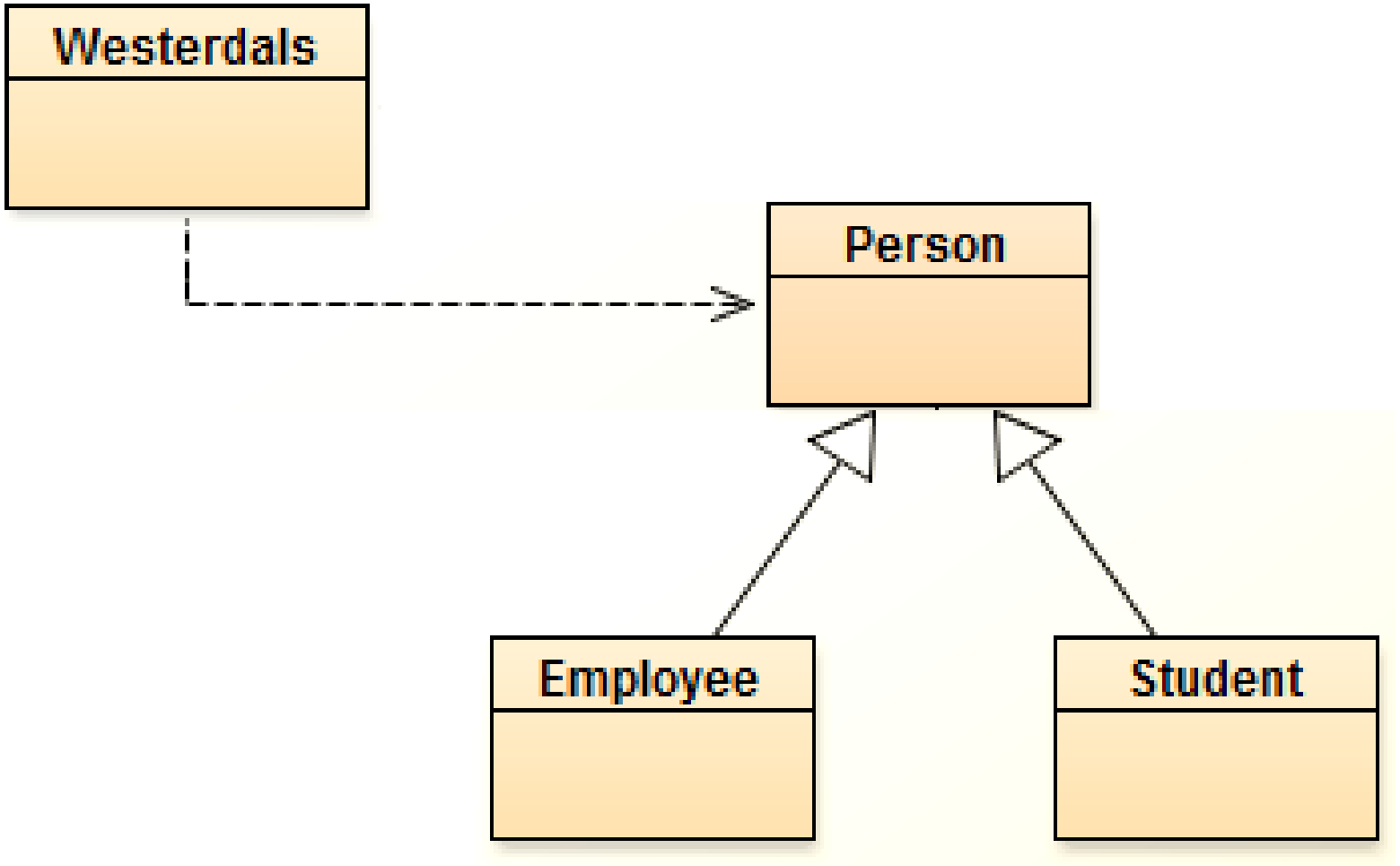

kan refereres av en **Person-referanse**.

# Regel

En **Person-referanse** kan "peke på" et **Student-objekt**

```
Person pers1 = new Student(...);
```

# Project *westerdals*



# Klassen `Westerdals` ("arkiv"-klassen)

```
public class Westerdals {
  private ArrayList<Student> studentList;
  private ArrayList<Employee> employeeList;


  public Westerdals() {
    studentList = new ArrayList<Student>();
    employeeList = new ArrayList<Employee>();
  }

  public void addStudent(Student student) {
    studentList.add(student);
  }

  public void addEmployee(Employee employee) {
    employeeList.add(employee);
  }
```

# Klassen `Westerdals` ("arkiv"-klassen) – med ARV!

Kan nå endres, siden arv er innført:

p

...i stedet for

...i stedet for

```
public void showAll() {
  for (Student s : studentList) {
    System.out.println(s);
  }

  for (Employee e : employeeList) {
    System.out.println(e);
  }
  System.out.println();
}
```

}

# Klassen `Westerdals` ("arkiv"-klassen) – med ARV!

Kan nå endres, siden arv er innført:

```
public class Westerdals {
  private ArrayList<Person> list;

  public Westerdals() {
    list = new ArrayList<Person>();
  }
  public void addPerson(Person person) {
    list.add(person);
  }
  public void showAll() {
    for (Person p : list) System.out.println(p);
    System.out.println();
  }
}
```

# Klassen `Westerdals`

```
Westerdals archive = new Westerdals();
...
archive.addPerson(new Student(....));
archive.addPerson(new Employee(....));
...
archive.showAll();
```

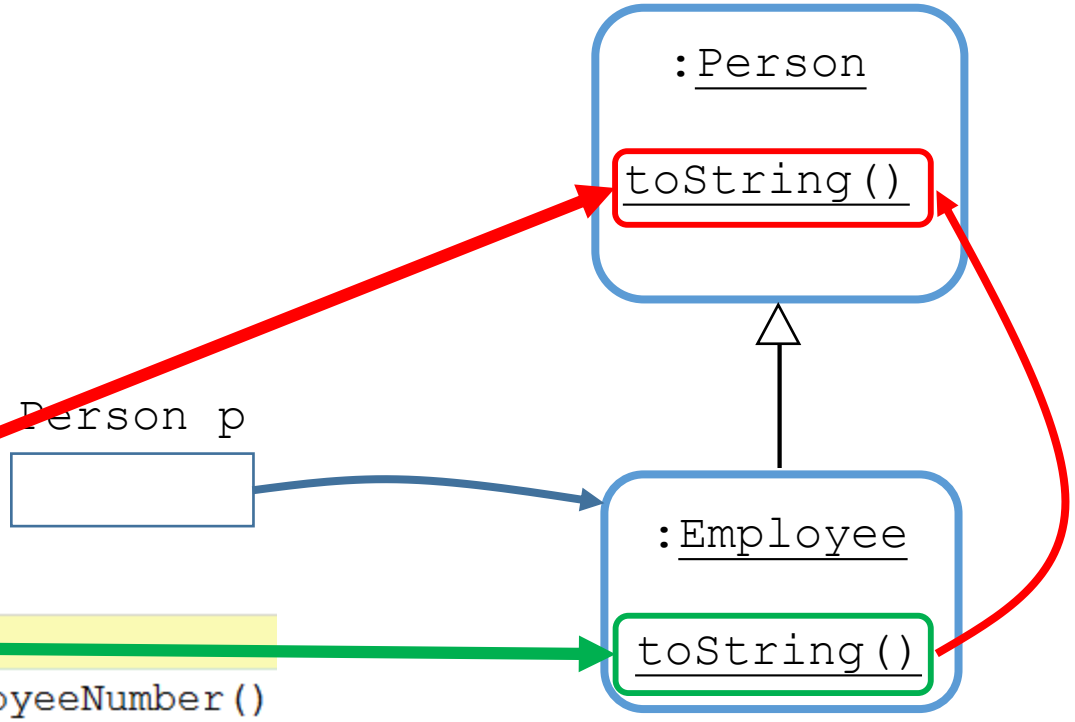
Nå kan vi altså legge inn både **`Student`**– og **`Employee`**–objekter med samme metode!

# Metoden `showAll`

```
public void showAll() {
  for (Person p : list) {
    System.out.println(p.toString());
  }
  System.out.println();
}
```

Husk
`System.out.println(p);`
virker på samme måte!

Here blir `Employee`-versjonen av `toString` kalt på.




```
public String toString() {
    String retur = super.toString() + "(" + getEmployeeNumber()
    return retur;
}
```

# Overriding – `super`-kall

`Employee` sin `toString`-metode skjuler superklassens (`Person` sin) versjon!

Siden det er `Employee` sin `toString`-metode som blir brukt, må denne sørge for at også `Person` sin versjon blir kalt!

Alternative løsninger (i klassen `Employee`):

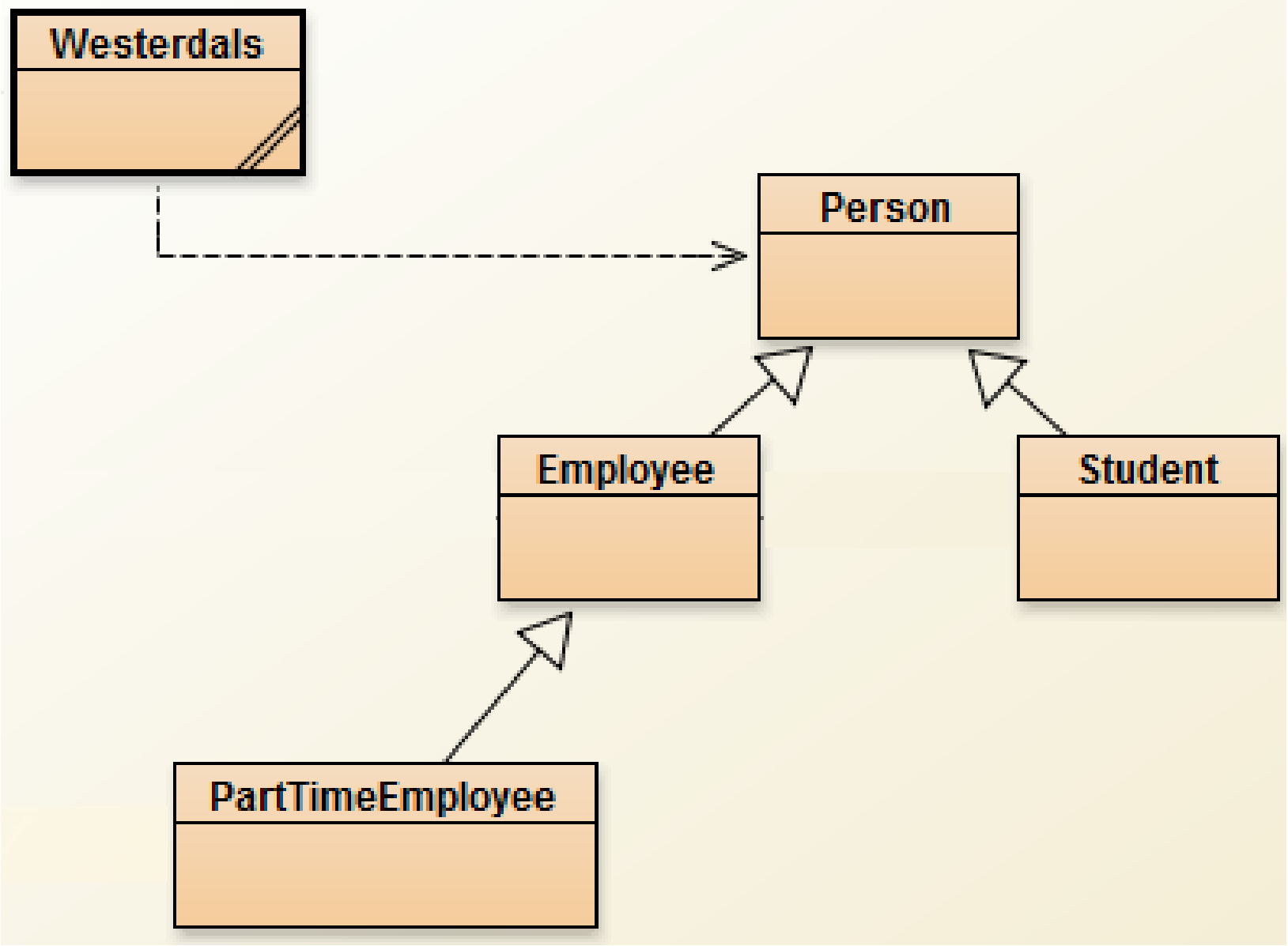
```
public String toString() {
  String retur =
        super.toString() + "(" + getEmployeeNumber() + ")"
         + " Lønn: " + getSalary();
  return retur;
}
```

# Fordeler med arv så langt

- Unngår kode-duplisering (gjentatt koding)
- Gjenbruk
- Lettere vedlikehold
- Utvidbarhet

# Project *westerdals* - utvidelse

Utvide med flere typer:



```java
public class PartTimeEmployee extends Employee{
    //Attributter
    private int weeks;

    //Konstruktør
    public PartTimeEmployee(String employeeNumber, String socialSecurityNumber,
                            String firstName, String familyName, int salary, int weeks) {
        super(employeeNumber, socialSecurityNumber, firstName, familyName, salary);
        setWeeks(weeks);
    }

    //tilgangsmetoder
    public void setWeeks(int weeks) {
        this.weeks = weeks;
    }

    public int getWeeks() {
        return weeks;
    }
```

```
    public String toString() {
        String retur = super.toString() + " Weeks: " + getWeeks();
        return retur;
    }
}
```

# Klassen `Westerdals` trenger ikke å endres!

```
public class Westerdals {
    private ArrayList<Person> list;

    public Westerdals() {
        list = new ArrayList<Person>();
    }

    public void addPerson(Person person) {
        list.add(person);
    }

    public void showAll() {
        for (Person p : list) {
            System.out.println(p);
        }
    }
}
```

# Klassen `Westerdals` trenger ikke å endres!

Kan "være" et
**`Student`**/**`Employee`**/**`PartTimeEmployee`**-objekt

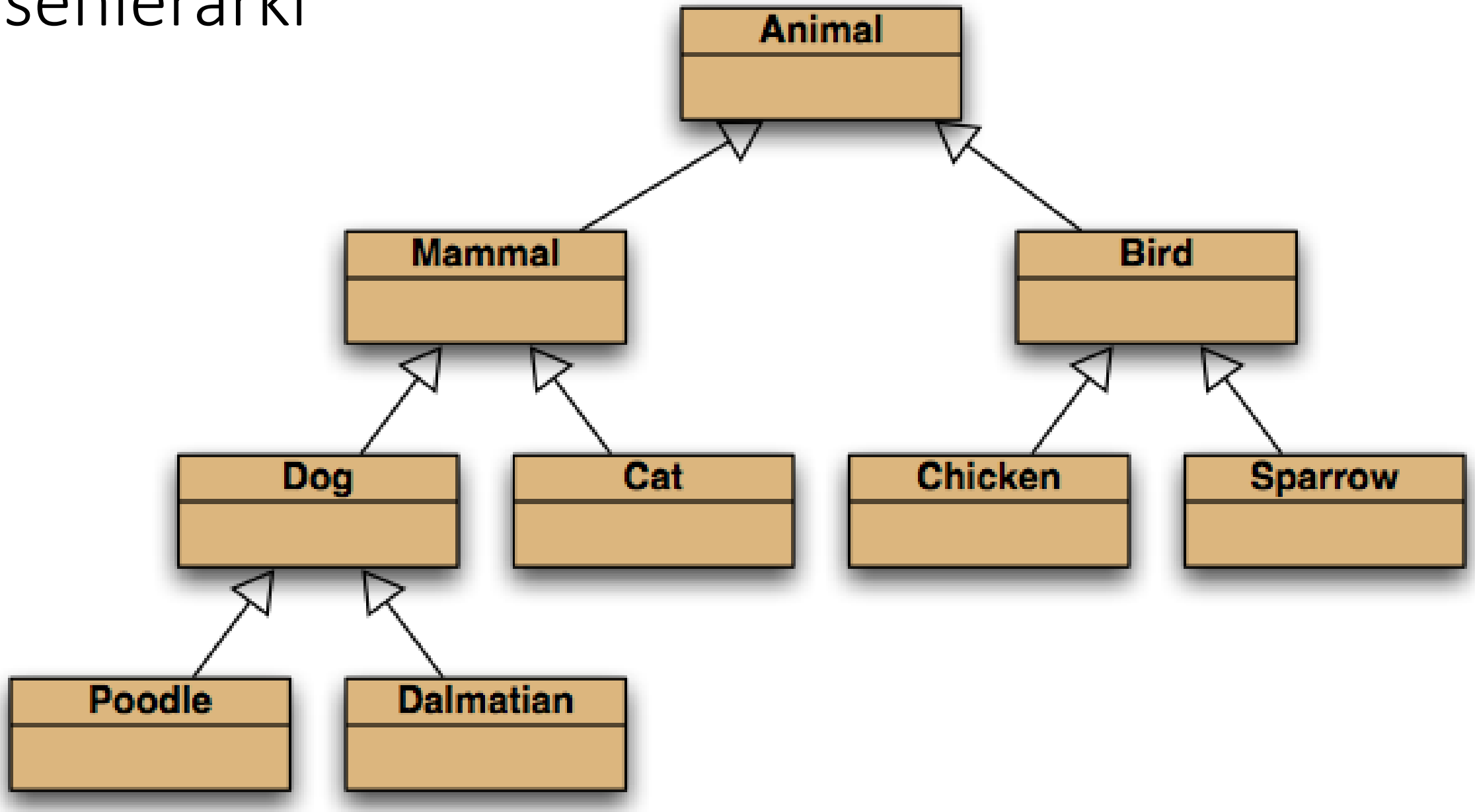
```
public void addPerson(Person person) {
    list.add(person);
}
```

# Klassen `Westerdals` trenger ikke å endres!

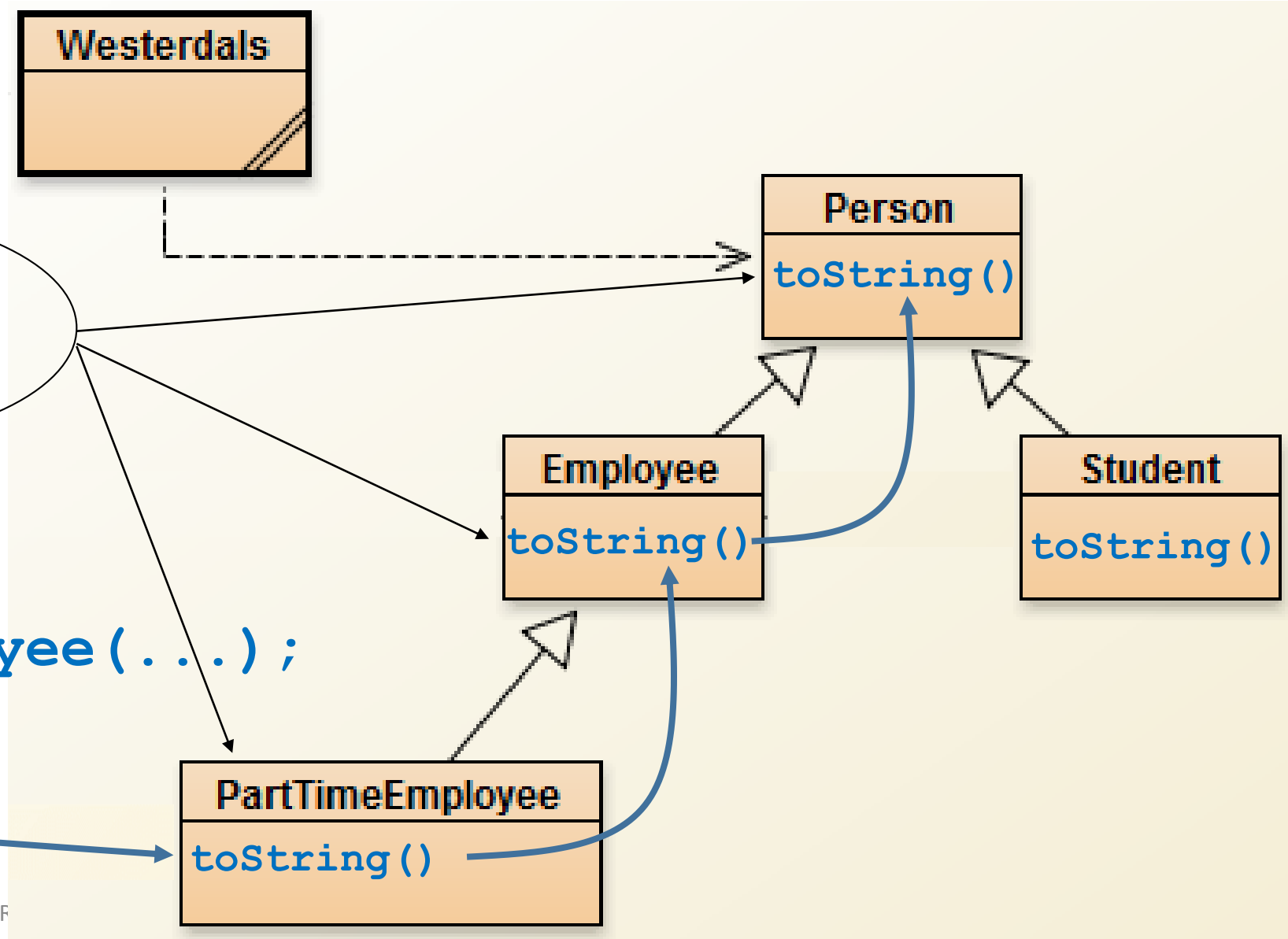
```
Westerdals archive = new Westerdals();
archive.addPerson(new Student("1234", "11111", "Albin", "Hub
archive.addPerson(new Student("2341", "22222", "Eruk", "Juhe
archive.addPerson(new Employee("12345", "111111", "Rerik", "
archive.addPerson(new Student("3412", "33333", "Dreahu", "Un
archive.addPerson(new Employee("23451", "222222", "Waeren",
archive.addPerson(new PartTimeEmployee("123456", "1111111",
archive.addPerson(new Student("4123", "44444", "Milke", "Lok
archive.addPerson(new Employee("34512", "333333", "Drokloe",
archive.addPerson(new Student("4321", "55555", "Per", "Spell
archive.addPerson(new PartTimeEmployee("234561", "2222222",


archive.showAll();
```

# Klassehierarki



# Overriding




```
Person p =
   new PartTimeEmployee(...);
p.toString();
```

# Oppsummering

- `Person`-klassen, `Employee`-klassen og `PartTimeEmployee`-klassen definerer hver sin versjon av metoden

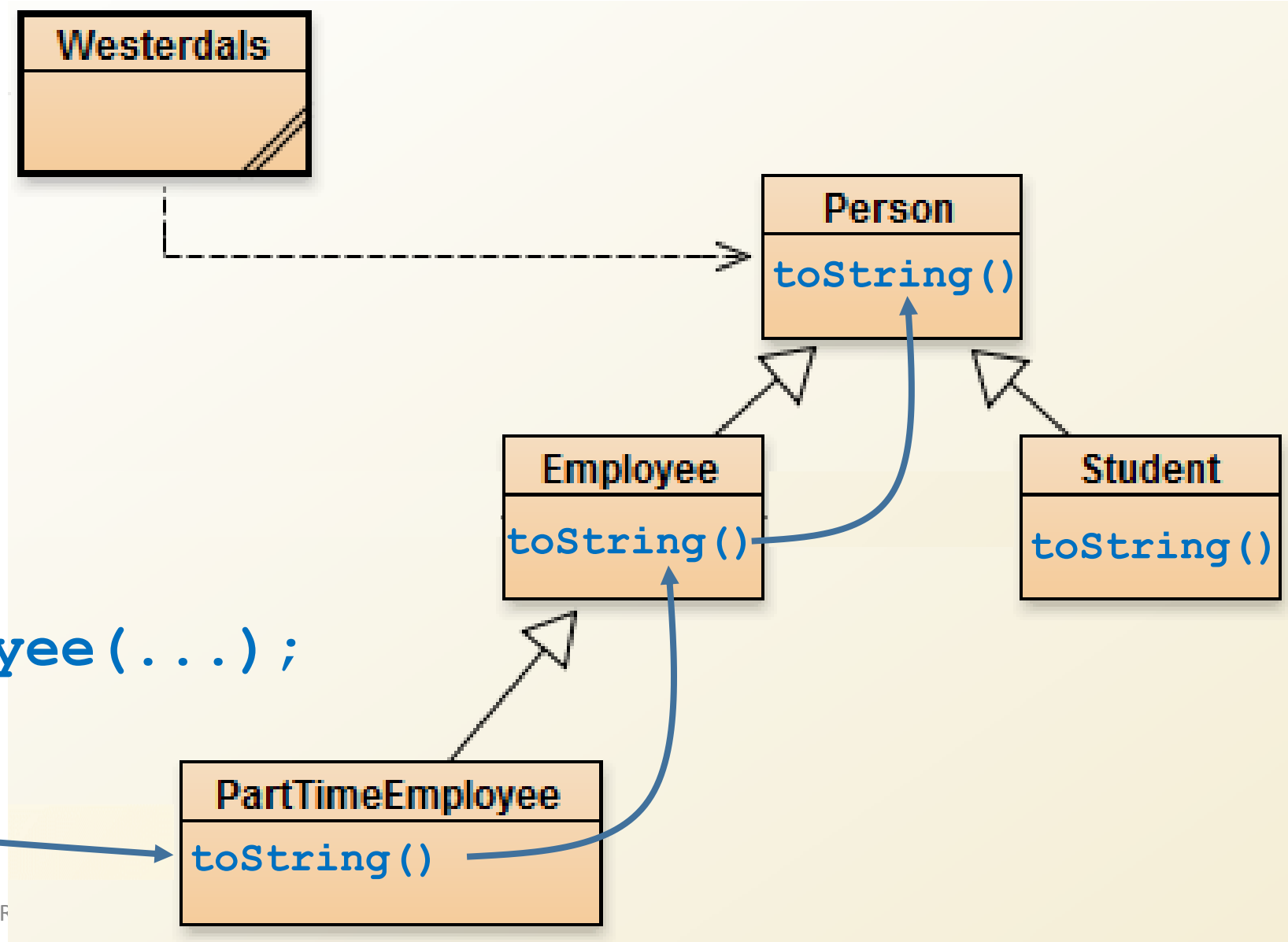
```
public String toString()
```

- Hver versjon har tilgang til klassens fields.

- `PartTimeEmployee`-metoden kalles ved runtime (den *skjuler* `Employee`-versjonen).

- `Employee`-metoden *skjuler* `Person`-versjonen.

# Men de kaller på hverandre!




```
Person p =
   new PartTimeEmployee(...);
p.toString();
```

# Oppgaver på øvingen

## Oppgave 1

**Oppgaver fra boka (6. utgave)**

10.3, 10.9 – 10.13, 10.17 – 10.19

**Oppgaver fra boka (5. utgave)**

8.3, 8.9 – 8.13, 8.17 – 8.19

## Oppgave 2

**Se itsLearning**