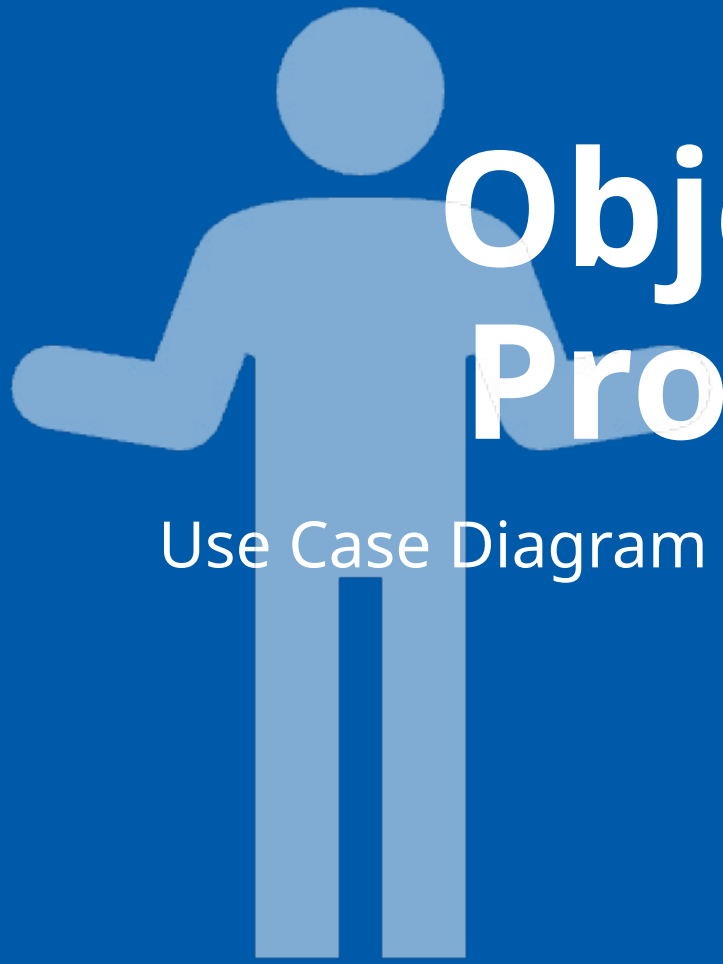


Object-Oriented Programming 2

Use Case Diagram & Inheritance en Abstracte klasse • Week 1

NHL STENDEN
university of
applied sciences

IDOARRT

- **Intention**
- Covering the Use Case Diagram and why we are creating it;
  - Dealing with inheritance and the abstract class.

- **Desired Outcome**
- The student has the knowledge of the Use Case Diagram and can draw up one on their ow;
  - The student can apply inheritance in his projects.

- **Time**
  45 minutes.

NHL STENDEN
university of
applied sciences

IDOARRT • Agenda

Why a Use Case Diagram?

- What does it look like?

What is inheritance?

What are the benefits?

Abstract class

- Demo

NHL STENDEN university of applied sciences

Design

Use Case Diagram

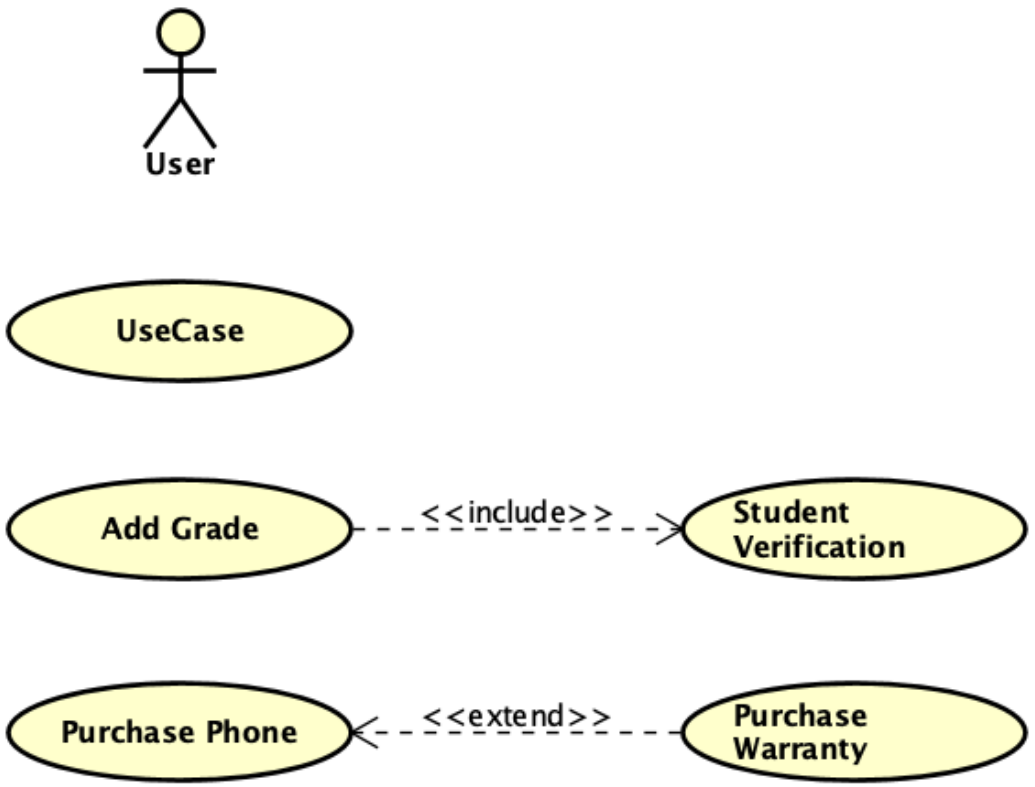
NHL STENDEN
university of
applied sciences

Why a Use Case Diagram?

- Different users in the system
- What Use Cases are in the system
  - What is the system supposed to do?

NHL STENDEN
university of
applied sciences

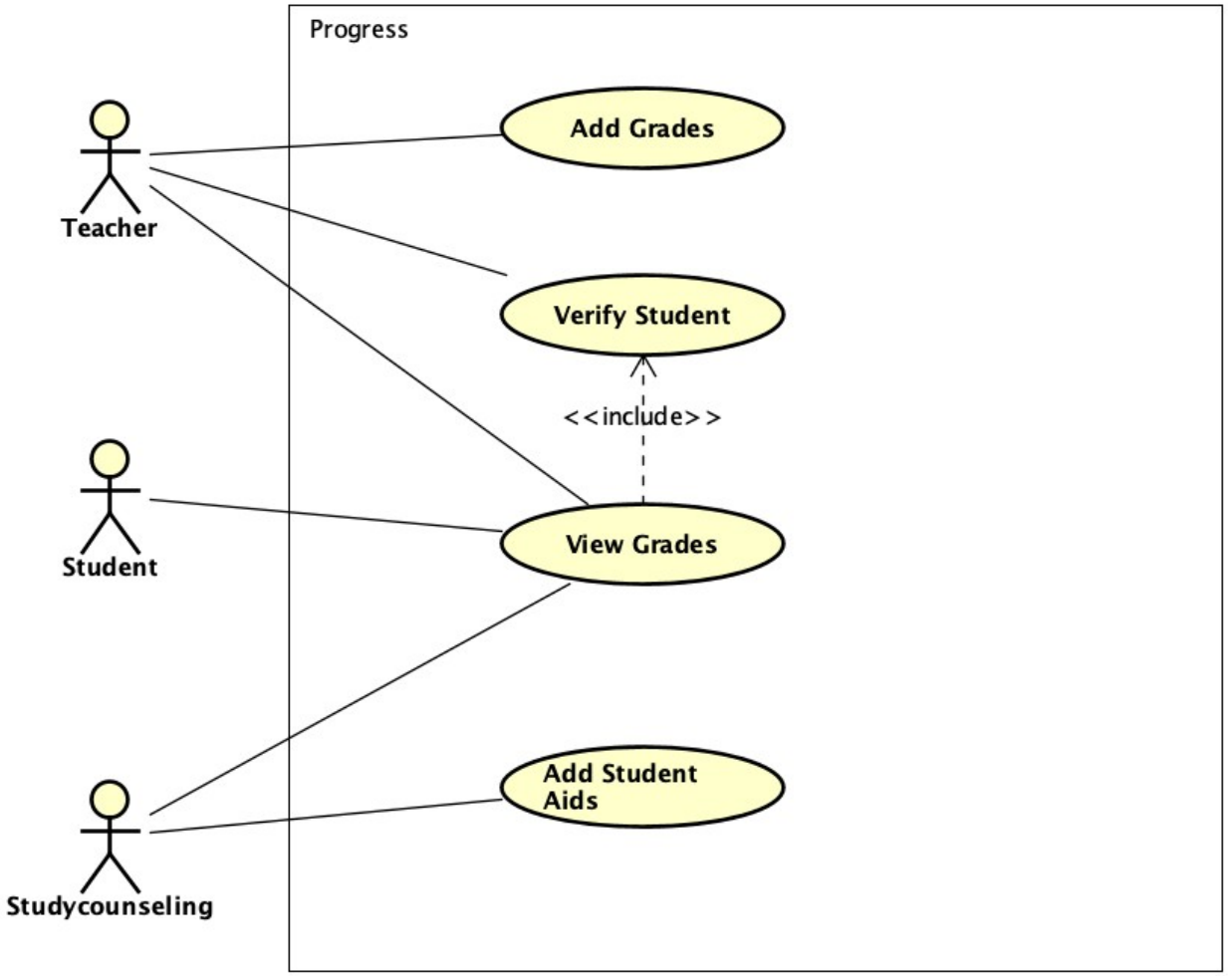
The Components



- **Actor**
  External entities that interact with the system (e.g. customers, administrators, but also other external systems/APIs).
- **Use Case**
  A process in the application.
- **Include**
  Use case that uses a different use case.
- **Exclude**
  Use case that optionally uses another use case.

NHL STENDEN
university of applied sciences

Example



NHL STENDEN
university of applied sciences

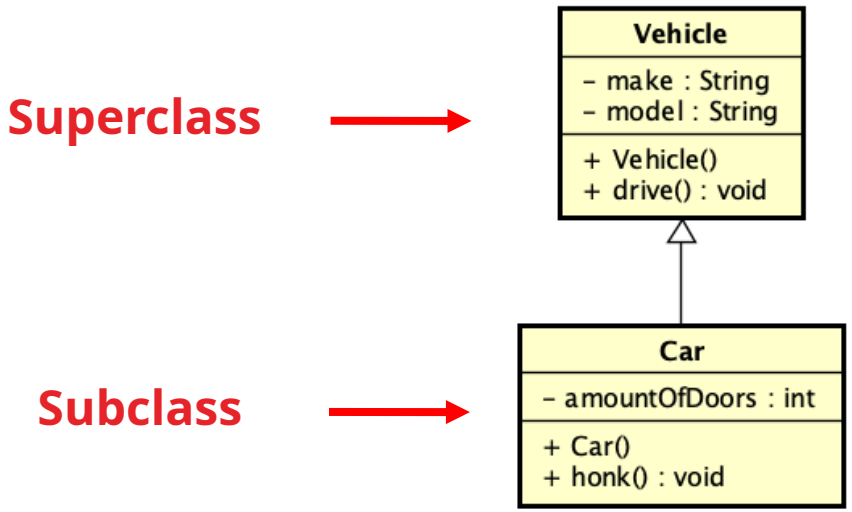
Realise

Inheritance & Abstracte klasse

NHL STENDEN
university of
applied sciences

What is inheritance?

- Inheritance



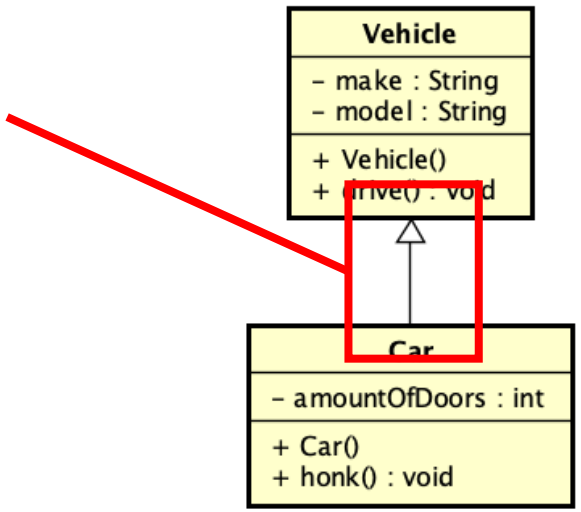
Advantages of inheritance

- Reuse code
- Hierarchy
- Polymorphism
  - Assign subclass instance to a superclass (in a field or parameter)

NHL STENDEN
university of
applied sciences

Inheritance in the class diagram

Generalization
relation.
In the direction of the superclass.



NHL STENDEN
university of
applied sciences

Inheritance in code

```
1  public class Vehicle
2  {
3    protected String make;
4    protected String model;
5
6    public Vehicle(String make, String model)
7    {
8      this.make = make;
9      this.model = model;
10   }
11
12   public void drive()
13   {
14     // Do something to drive the vehicle
15   }
16 }
```

```
1  Vehicle vehicle = new Vehicle("Volkwagen", "Golf");
2  vehicle.drive();
3  vehicle.honk(); // Not available
```

```
1  public class Car extends Vehicle
2  {
3    private int amountOfDoors;
4
5    public Car(String make, String model, int
amountOfDoors)
6    {
7      super(make, model);
8      this.amountOfDoors = amountOfDoors;
9    }
10
11   public void honk()
12   {
13     // Do something to honk
14   }
15 }
```

```
1  Car car = new Car("Volkwagen", "Golf", 4);
2  car.drive(); // Available from Vehicle
3  car.honk()
```

Polymorfisme

```
1  Vehicle vehicle = new Car("Volkwagen", "Golf", 4);
2  vehicle.drive(); // Available from Vehicle
3  vehicle.honk(); // Not available
4
5  Car car = new Vehicle("Volkswagen", "Golf")
   // Not possible
```

NHL STENDEN university of applied sciences

Inheritance override

```
1  public class Car extends Vehicle
2  {
3    private int amountOfDoors;
4
5    public Car(String make, String model, int amountOfDoors)
6    {
7      super(make, model);
8      this.amountOfDoors = amountOfDoors;
9    }
10
11   @Override
12   public void drive()
13   {
14     super.drive(); // Call functionality from Vehicle, not mandatory
15
16     // Override functionality from Vehicle
17   }
18
19   public void honk()
20   {
21     // Do something to honk
22   }
23 }
```

NHL STENDEN
university of
applied sciences

Abstract class

- Cannot be initialized

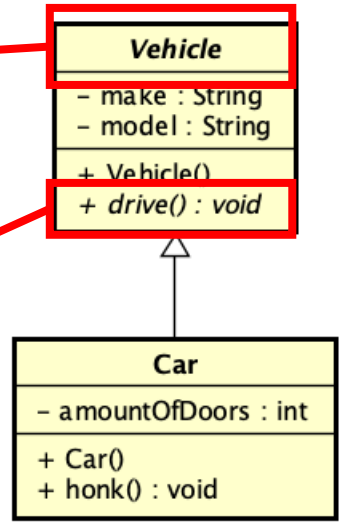
- Can contain an abstract method
- This one has no body!
  - Forces subclasses to implement this method

NHL STENDEN
university of
applied sciences

Abstract class in the class diagram

Class name in *italic.*

Abstract methods in *italic.*



NHL STENDEN
university of
applied sciences

Abstract class in code

```java
public abstract class Vehicle
{
  protected String make;
  protected String model;

  public Vehicle(String make, String model)
  {
    this.make = make;
    this.model = model;
  }

  public abstract void drive();
}
```

```java
Vehicle vehicle = new Vehicle("Volkwagen", "Golf");
// Not possible, because Vehicle is abstract
```

```java
public class Car extends Vehicle
{
  private int amountOfDoors;

  public Car(String make, String model, int amountOfDoors)
  {
    super(make, model);
    this.amountOfDoors = amountOfDoors;
  }

  @Override
  public void drive()
  {
    // Do something to make it drive
  }

  public void honk()
  {
    // Do something to honk
  }
}
```

```java
Car car = new Car("Volkwagen", "Golf", 4);
car.drive();
car.honk()
```

Polymorfisme

```java
Vehicle vehicle = new Car("Volkwagen", "Golf", 4);
vehicle.drive(); // Available from Vehicle
```

NHL STENDEN university of applied sciences

Demo

Questions