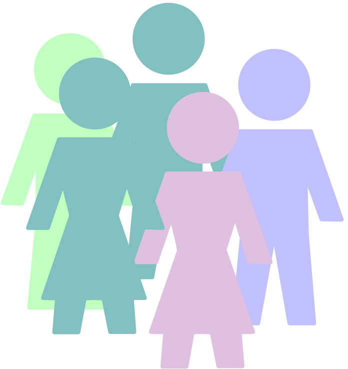
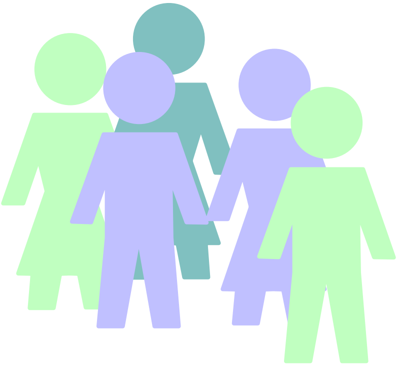
Technische Hochschule **Rosenheim**
Technical University of Applied Sciences

Object-oriented programming
Chapter 10 – Inheritance

Prof. Dr Kai Höfig

Set of similar but different objects

Football fans		Shoe enthusiasts
		
Properties		**Properties**
Name Age	**Similarities**	Name Age
Favourite club	**Differences**	Number of pairs of shoes
Behaviours		**Behaviours**
Sleep Eat	**Similarities**	Sleep Eat
Watch football	**Differences**	Buy shoes

Motivation – Analysis at the meta level

Analysis of the two groups

- Some differences
  → Summarising in one class does not work!
- Many similarities
  → Splitting into two separate classes results in high redundancy

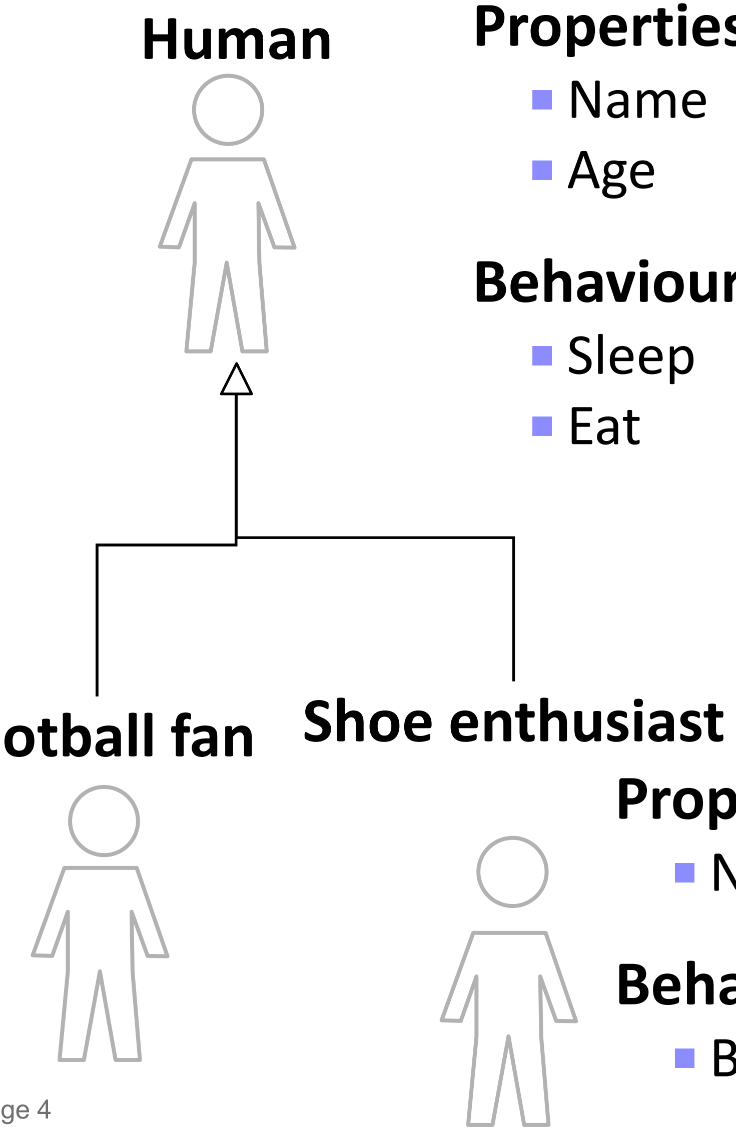
- How are such situations programmed?
  - As little redundancy as possible
  - Make differences clear

Solution idea

- Central definition of the similarities
  - (generalise – general class – base class)
- Specialised class (subclass)
  - Documentation of differences
  - Additional attributes and/or methods
  - Similarities inherited from central definition
  - Methods can be overwritten or redefined

Example solution idea

Extract common characteristics



Human

Properties

- Name
- Age

Behaviours

- Sleep
- Eat

Football fan

Properties

- Favourite club

Behaviours

- Watch football

Shoe enthusiast

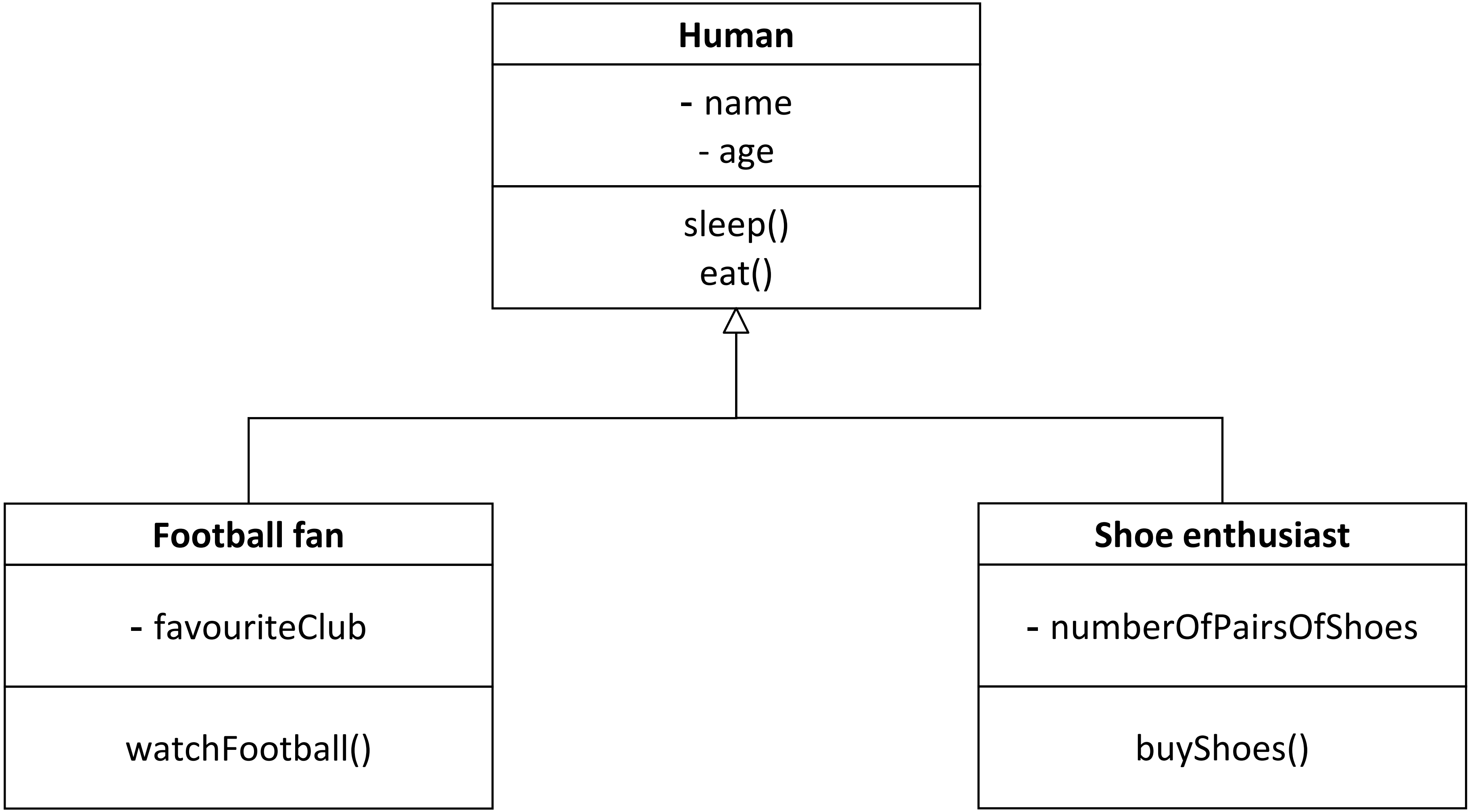
Properties

- Number of pairs of shoes

Behaviours

- Buy shoes

Inheritance in UML class diagram

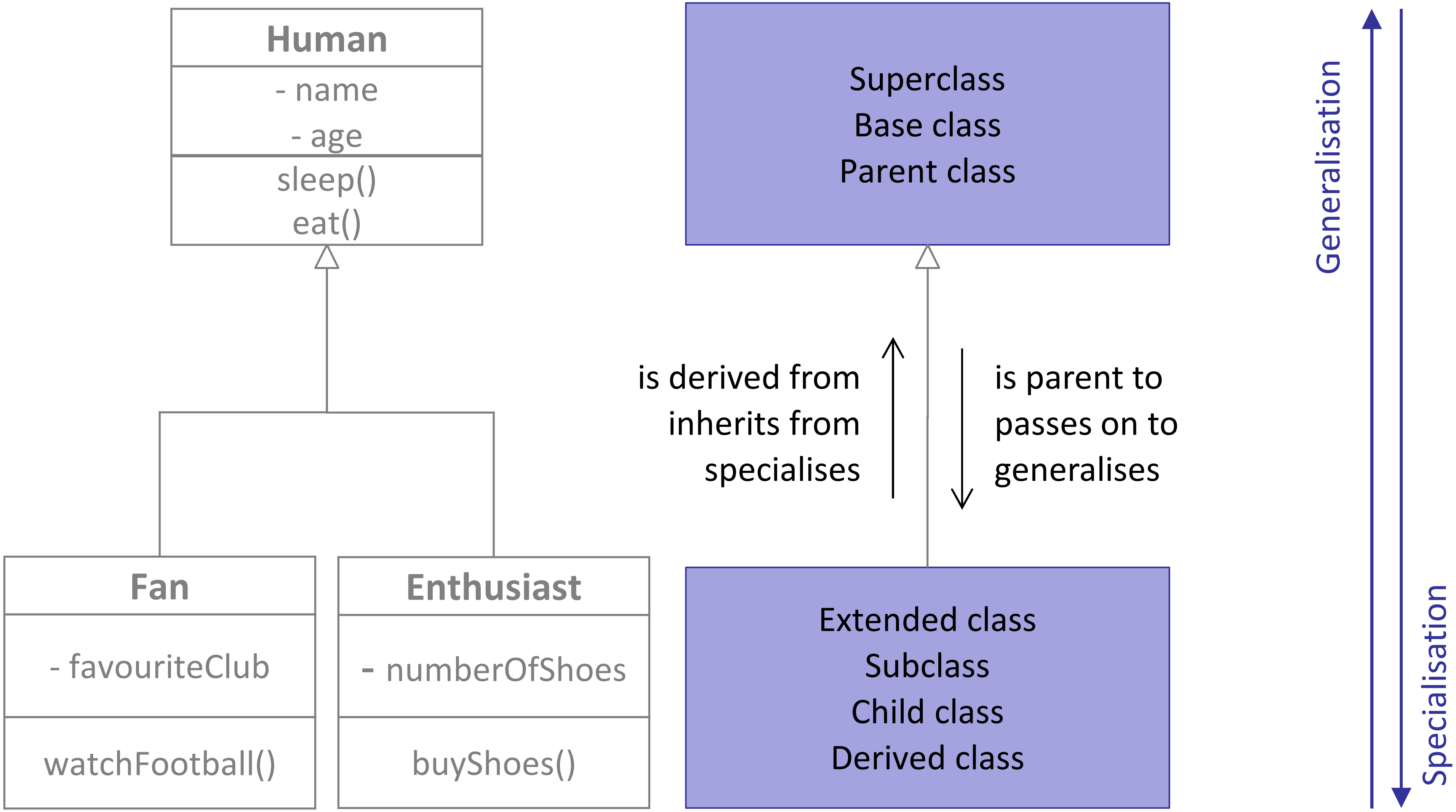


Meaning of inheritance

- **Basic idea**
  - Describes similarity between classes
  - Special case of a relationship between classes
    - Each object of the subclass is an (is a) object of the base class
  - Structures classes in hierarchy of abstraction levels
  - Enables definition of a new class based on existing classes (reuse!)

Essential mechanism that distinguishes object-oriented languages from functional/procedural languages!

Terms



Human

- name

- age

sleep()

eat()

Fan

- favouriteClub

watchFootball()

Enthusiast

- numberOfShoes

buyShoes()

Superclass
Base class
Parent class

is derived from
inherits from
specialises

is parent to
passes on to
generalises

Extended class
Subclass
Child class
Derived class

Generalisation

Specialisation

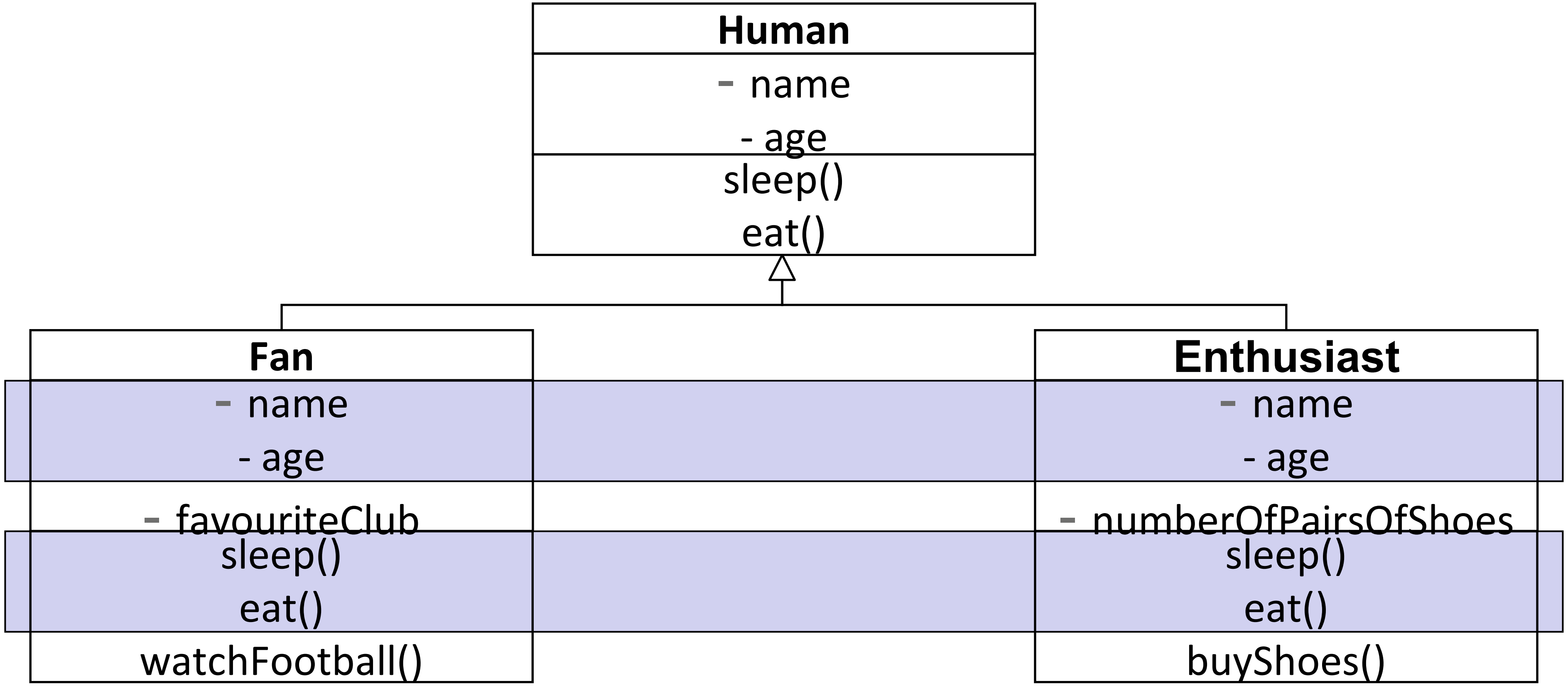
Method

- **Two possible approaches:**
  - Bottom-up: from special to general
  - Top-down: from general to special

- **When do we choose what?**
  - Bottom-up:
    If similarities only become noticeable in a partially complete solution
  - Top-down:
    If we know in advance that there are similarities

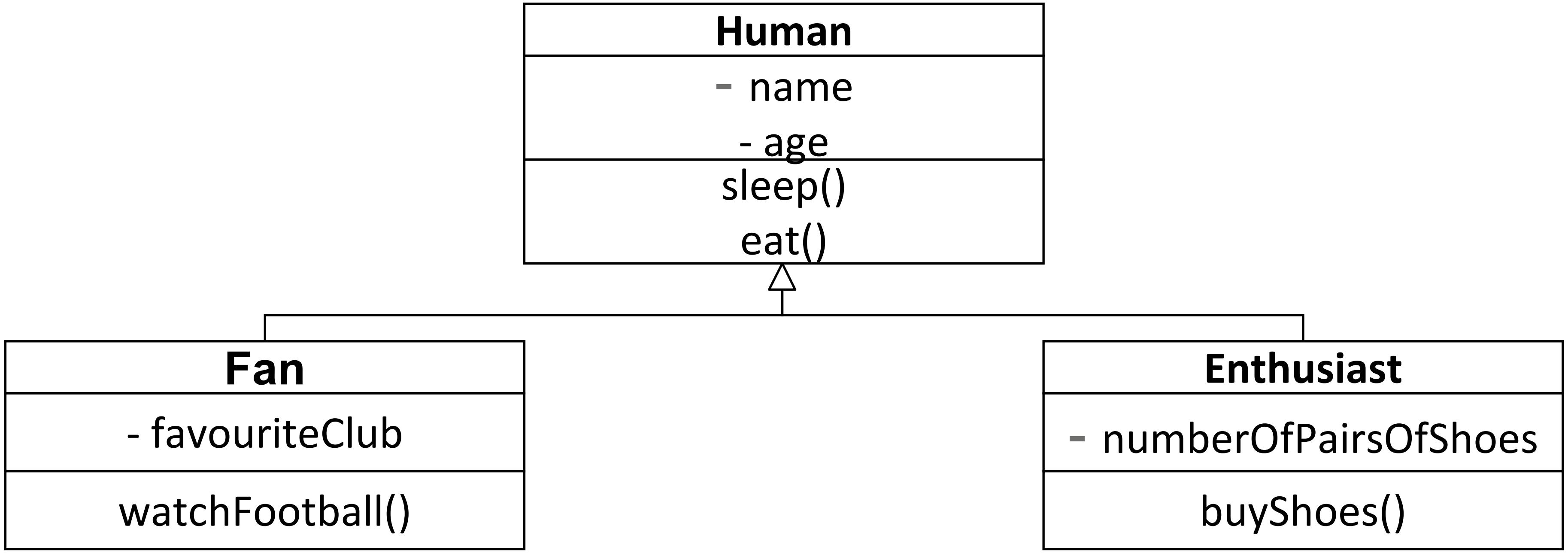
Approach – Bottom-up

1. First model the individual classes
2. Identify redundancies
3. Put similarities into the base class
4. Derive original classes from base class and "declutter"



Approach – Top-down

1. First define the similarities in the central base class
2. Define specialising classes, derive from base class
3. Then define the specifics of the derived classes
4. If necessary, gradually expand the number of derived classes



Keyword "extends"

- Reference to **base class (parent class)** through keyword ***`extends`*** in the header of the **derived class** (subclass)
  - Example:

```
Class Cat extends Pet {…}
```

- Derived class inherits *all* variables and *all* methods of the base class.

- The functionality of the base class can be changed by
  - Adding new elements (attributes, methods, …)
  - Overloading the existing methods
    - For example: `public String getName(String greeting)`
  - Redefining (overwriting) the existing methods

Visibility at a glance

Technische Hochschule **Rosenheim**
Technical University of Applied Sciences

Modifier	Class	Package	Subclass	World
`public`	Yes	Yes	Yes	Yes
`protected`	Yes	Yes	Yes	No
no attribute	Yes	Yes	No	No
`private`	Yes	No	No	No

http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

- Attributes are usually `private`
  - …except if there’s a good reason for `protected` or `public`
- Methods are usually `public`
  - …except if there’s a good reason for `protected` or `private`

Implementation – Definition of the base class

```
public class Person {

 // common properties of all subclasses
 private String name;
 private int age;

 // common functionality of all subclasses
 public String sleep() {
  return "sleep: Chrrrrr.... chrrrr...";
  }
 public String eat() {
        return "eat   : Mmmh, delicious.";
  }
}
```

Implementation – Define subclass (1)

```
public class FootballFan extends Person {

 // new attribute
 private String favouriteClub;

 // new functionality
 public String watchFootballGame() {
  return "play : yes... YES... GOOOOAAAL!!!";
  }
}
```

Implementation – Define subclass (2)

```
public class ShoeEnthusiast extends Person {

 // new attribute
  private int pairsOfShoes;

  // new functionality
 public String buyShoes() {
  pairsOfShoes++;
  return "shop : THOSE look great..., " +
             "Pair number" + pairsOfShoes;
  }
}
```

Implementation – Define main class

```
public class Main {
 public static void processPerson(Person person) {
  person.eat();
   }
 public static void main(String[] args) {
  FootballFan eva = new FootballFan();
  ShoeEnthusiast adam = new ShoeEnthusiast();
  System.out.println("What Eva does:");
  eva.sleep();
  processPerson(eva);
  eva.watchFootballGame();
  System.out.println();
  System.out.println("What Adam does:");
  adam.sleep();
  processPerson(adam);
  adam.buyShoes();
  System.out.println();
   }
}
```

Implementation - Output

- Output of the main programme

```
What Eva does:
sleep: Chrrrrr.... chrrrr...
eat: Mmmh, delicious.
play: Yes... YES... GOOOOAAAL!!!
What Adam does:
sleep: Chrrrrr.... chrrrr...
eat: Mmmh, delicious.
shop: THOSE look great...
```

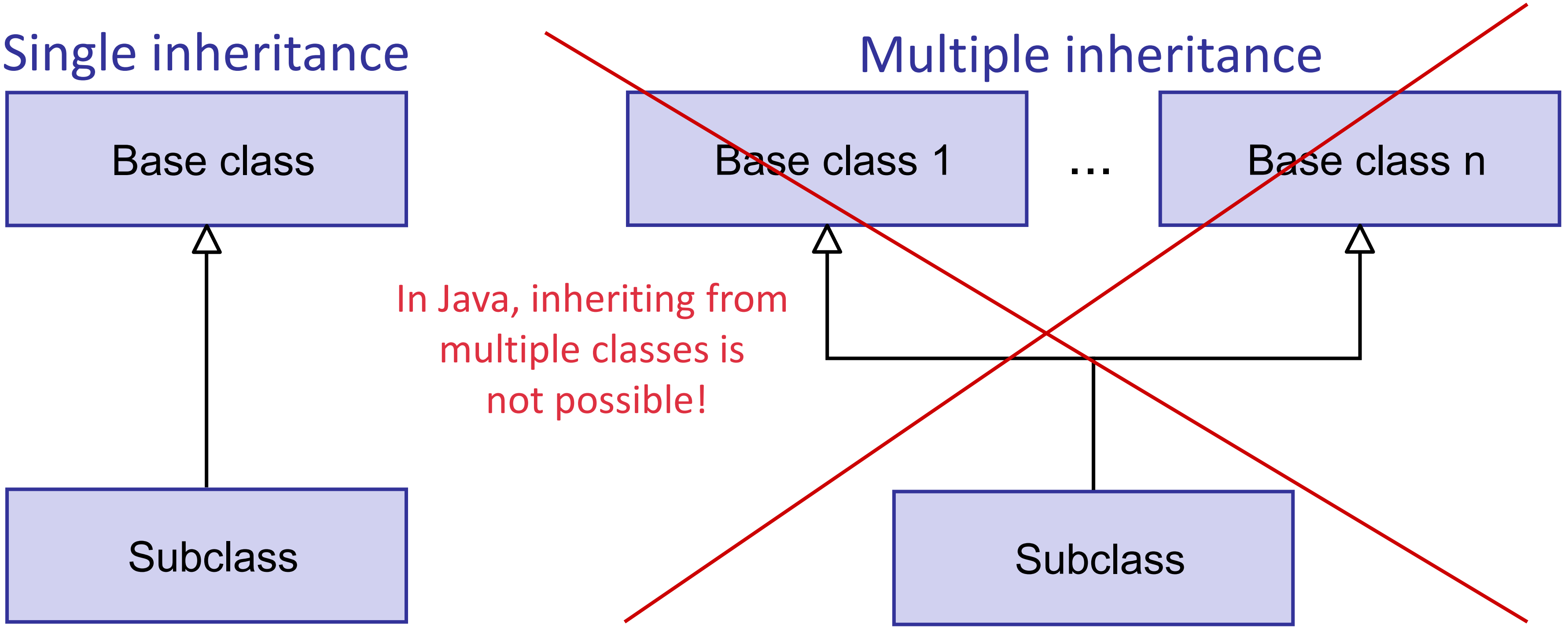
Types of inheritance

Single inheritance

- Subclass inherits from exactly one base class

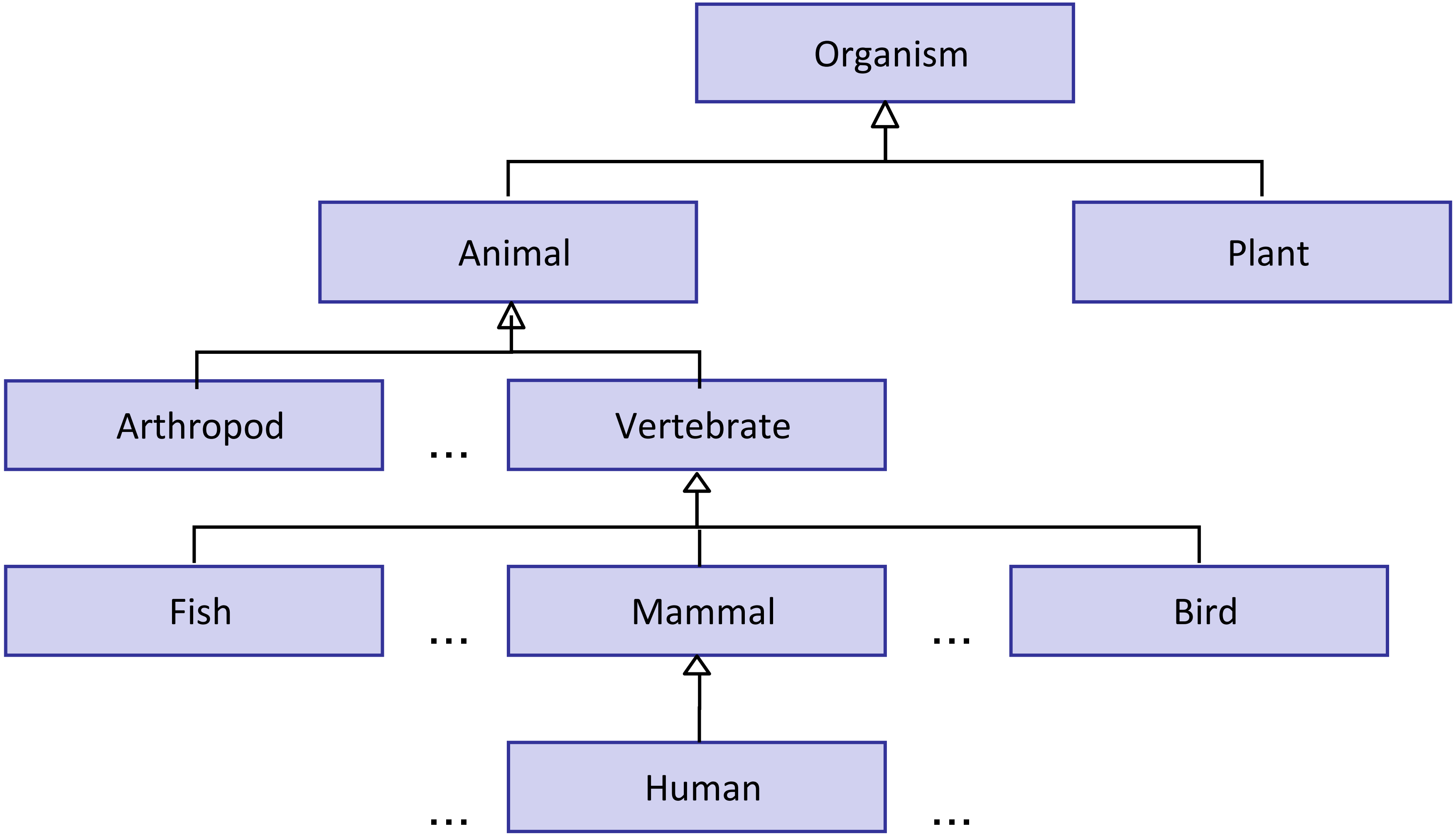
Multiple inheritance

- Subclass inherits from more than one base class

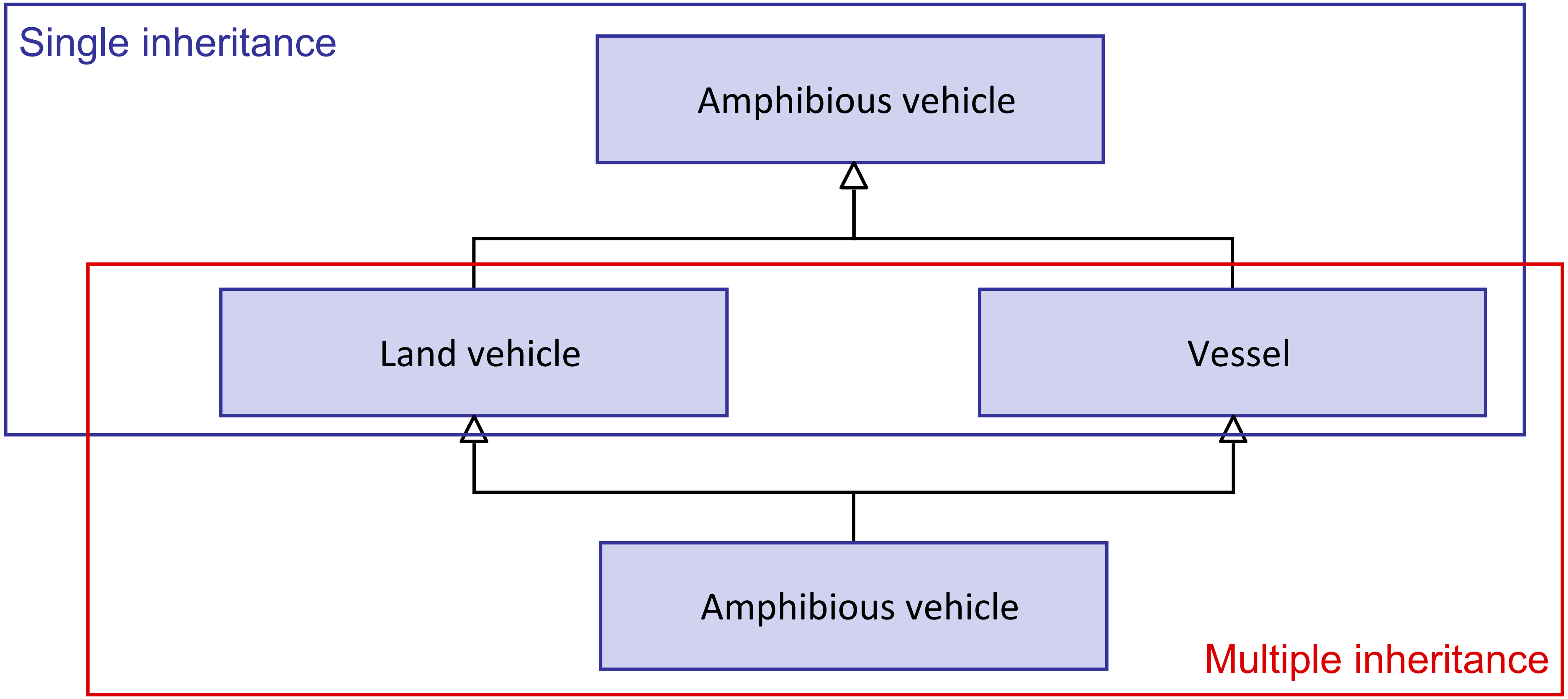


In Java, inheriting from multiple classes is not possible!

Single inheritance over multiple stages



Single and multiple inheritance



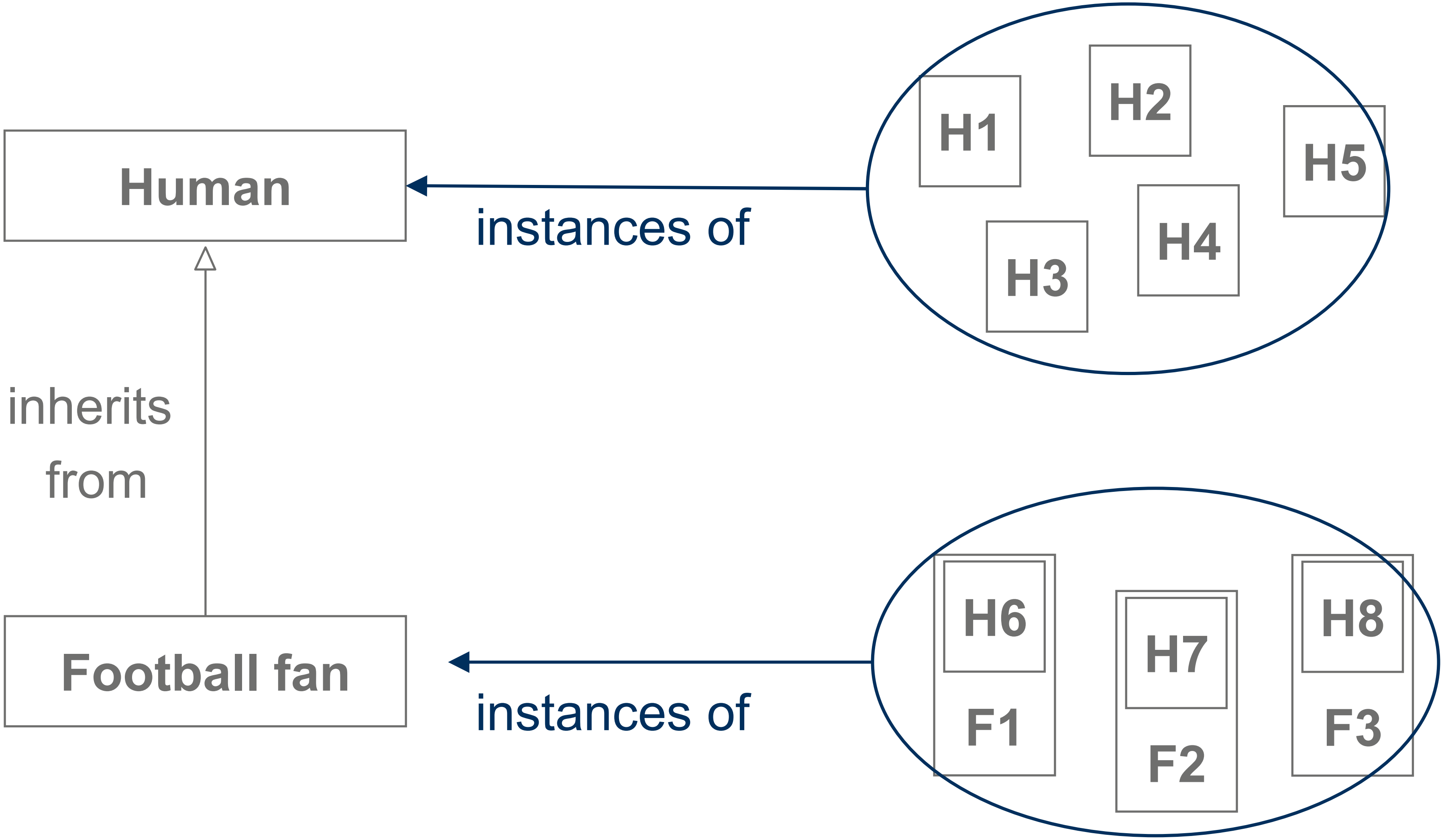
What is inherited?

- Subclass inherits from base class …
  - the operations (the behaviour)
  - the attributes (the possible states)
  - the semantics!
    (i.e. instead of an object of the base class, an object of any subclass can also always be used!
    => Substitution principle)

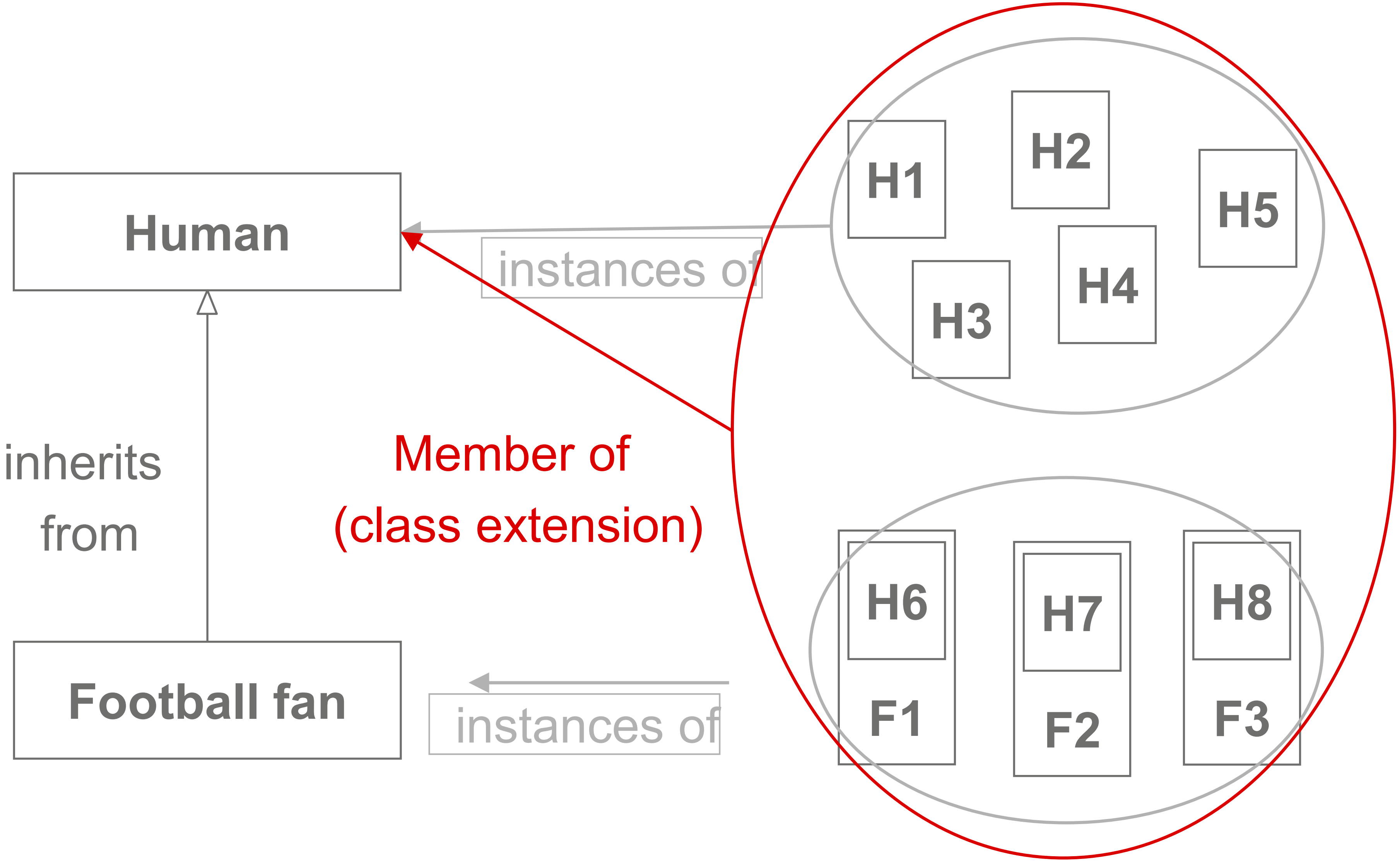
- Examples in Java:

  -
    ```java
    Person p = new Man();
    p = new Woman();
    ```

Syntax inheritance



Semantic inheritance



Constructors in inheritance

- Each constructor of a derived class should call a base class constructor.
  - Otherwise, attributes of the base class might never be initialised.

- Explicit call of the default constructor of the base class:

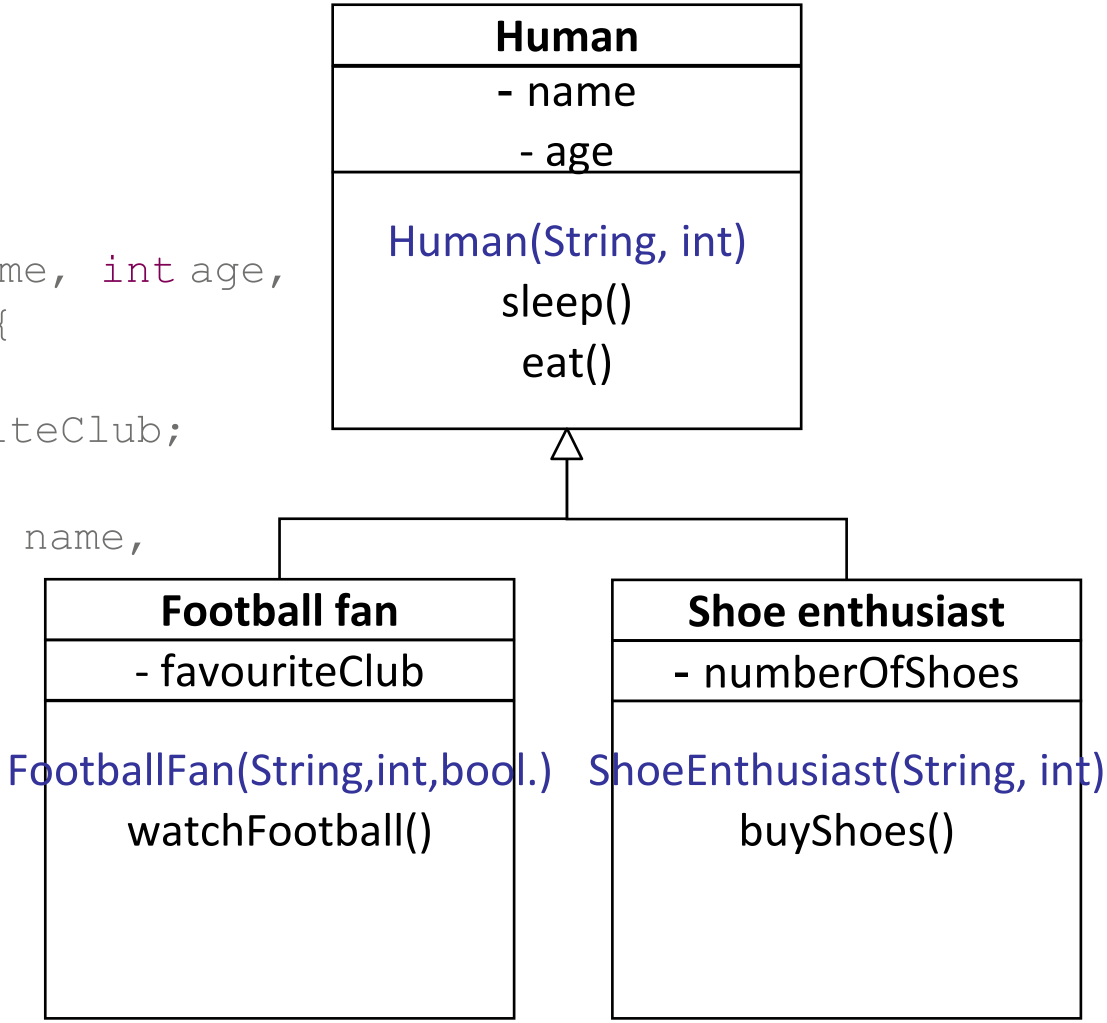
  -
    ```
    super();
    ```
- Explicit call of a value constructor of the base class:

  -
    ```
    super(name,…);
    ```
- If there is no explicit call:
  - Implicit call of the default constructor of the base class. This must be specified explicitly, otherwise an error will occur.

- Rule: a constructor call **must** always be the *first* statement in the constructor of the subclass

Constructors with `super()`

```
public Person (String name,
              int age) {
  this.name = name;
  this.age = age;
}
public FootballFan (String name, int age,
            String favouriteClub) {
    super (name, age);
  this.favouriteClub = favouriteClub;
}
public ShoeEnthusiast (String name,
             int age) {
            (name, age);
  pairsOfShoes = 0;
}
```



Constructors with `this()`

- Reminder
  - Calls another constructor of the same class: `this()`
  - Must be the first statement in the constructor body
  - Useful to avoid redundancies in the constructors

- Example:

```
public ShoeEnthusiast (String name, int pairsOfShoes) {
  this.name = name;
 this.pairsOfShoes = pairsOfShoes;
}

public ShoeEnthusiast (String name) {
 this (name, 0);
}
```

Summary

- Inheritance
- Generalisation and specialisation
- Bottom-up and top-down approach to design
- Visibilities
- Multiple inheritance
- Syntax and semantic inheritance
- Constructors with `super` and `this`