Inheritance & Polymorphism

What we will learn:
In this session we will look at the OO concept „inheritance“.
We will learn what polymorphism means.
We will also learn the concepts of overloading and overwriting.

Contents

Wrong Programming

1.1 Extending a Social-Network Simulator in the wrong way

In order to understand the benefits of inheritance, we're going program a social network simulator …. and program it in the wrong way first. We will notice how this approach makes maintenance and extensions more difficult and complex.
We want a program with following classes:

`MessagePost`	Class for messages.
`PhotoPost`	Class for photos.
`NewsFeed`	Class has a collection of message and photo posts.

Install the classes from the downloaded source folder.
Each post class has a display-method to print details of the post.

1.1.1 Exercise – Adding an Event Post

Extend the SocialNetwork by adding a new type of post:

`EventPost`	Class for events.

The class has following attributes:

```
private String author;
private long timeStamp;
private int pages;
private int likes;
private ArrayList<String> comments;
```

Implement the constructor and necessary methods accordingly.

1.1.2 Exercise – Extending the NewsFeed class with EventPost

Now we obviously have to add this new post type to our *NewsFeed* class.

```
private ArrayList<EventPost> events;
```

Further, we have to initialize this list also in the constructor. And add the necessary methods.

Analysis:

Obviously, this approach is very tiring and error-prone.

List down the main problems with this approach:

2 Using Inheritance

2.1 Analysis of the wrong solution

One main problem is the duplicated code we are adding. Most attributes in all three post classes are the same.

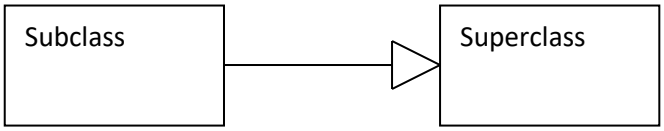
And we have to extend the `NewsFeed` class, with code which is repetitive and cumbersome. Mistakes can happen easily while extending the code. We might even break existing code.

And there's more: if we decide to change the comment attribute from `ArrayList<String>` to `ArrayList<Comment>` we have to change this at several points in the code.

Idea: When we extend our program, we only want to add the new classes. But we don't want to change the remaining classes. For example, when we add a new post class, we don't want to change the `NewsFeed` class.

2.2 Step-by-step to better coding

Inheritance is an important concept in object-orientated programming. Classes can inherit from other classes. This means a class can inherit attributes and methods from another class.



In a first step we want to implement a *Post* class which unifies all shared attributes of the several post classes. From this super class all post classes will inherit attributes and methods. If necessary, a post class can have its own special attributes.

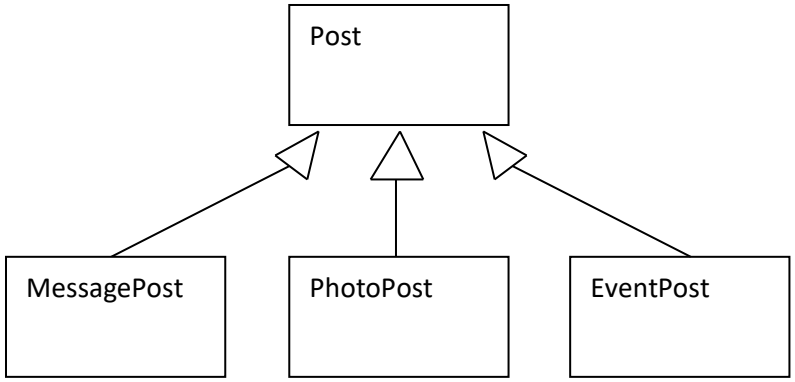
This is how inheritance is programmed in Java:

```
public class MessagePost extends Post {
}
```

2.2.1 Exercise – create a new Post class as superclass

Identify the common attributes of the post classes and add these to a new Post class. This class is the superclass of all other post classes.

The new structure should look like this:

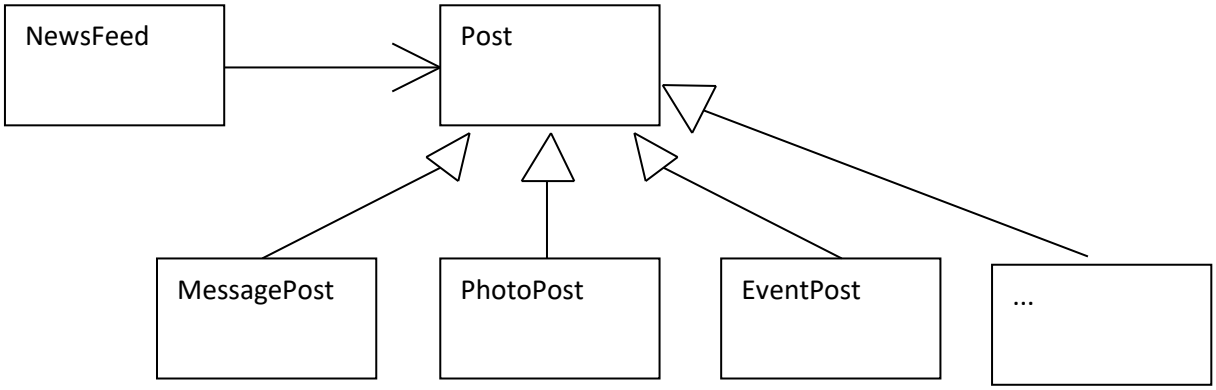


2.2.2 Exercise – Simplifying the NewsFeed Class

Refactor the *NewsFeed* class accordingly. We want one *ArrayList* which deals with all posts.

2.3 Final product

The **`NewsFeed`** class only works with the new **`Post`** class and does not have any knowledge of the subclasses.
This will simplify any extensions we do later.



2.3.1 Simple print-out and a smarter print-out of attributes

Make sure that your superclass *Post* has the *display()* method.

a) We want a simple print-out of the general attributes which every post has.

b) Now also include the special attributes of each extended post subclass. What must you do in the method of the subclass in order to combine both attributes (from superclass and subclass)?

➔ Show your results to the teacher.

2.4 Some Information on Overriding methods

When printing out attributes you have to see that the superclass and subclasses work together. Especially you have to make sure that the print-method of the superclass is overridden in the subclass. You'll find some tips on this on following websites:

http://docs.oracle.com/javase/tutorial/java/IandI/override.html
https://beginnersbook.com/2014/01/method-overriding-in-java-with-example/

Notes

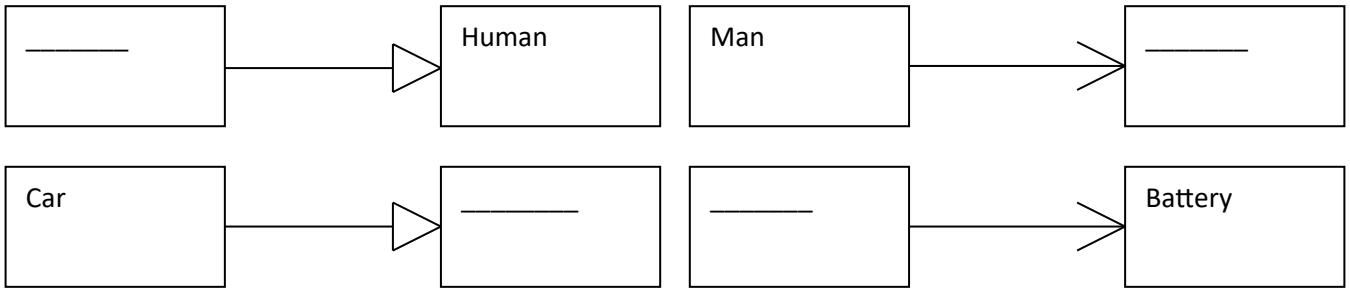
3 Exercises for Competences in the First Column

Do following exercises and use the internet to research for definitions or examples. Exercises 1 – 6 will show that you have understood the basic concept of inheritance and that you can use overriding and overloading of methods. Exercises 7 and 8 show that you can draw a design in detail (with IS and HAS relationships) and that you can implement unit-tests.

3.1 Exercise – Understanding Symbols

Look at following diagrams and use examples from our daily lives (school, work, etc.) to show the different relationships. Fill in the blanks:

IS-Relation HAS-Relation



3.2 Exercise – New Project “Flix-Bus Switzerland”

We want to implement a small system which simulates a national bus service in Switzerland. The company “Flix-Bus” offers services to national but also international destinations. (-> see similar exercise “Airport” in the compendio book).
Our system should have following classes:

Class	Responsibility
BusTerminal	has all information which bus leaves from which platform List of platforms, Name of Terminal
Platform	has a number and information about bus type Platform number, platform size, bus service (national or international), bus type (small or large), occupied or not occupied
Travel	has all specific information about a bus trip Destination, departure time, arrival time, national or international
Bus	has all information of the bus type bus type (double decker, single), passenger capacity, comfort (basic or 1st class)

➔ Before you start coding, do a design (UML classdiagram) of the relationships between classes. This is part of competence 1A.

3.3 Exercise – Including inheritance

Flix-bus wants to make sure that the bus terminal can deal with different types of buses. In order to do this, your system should make a distinction between the vehicles they use. Flix-bus uses double decker coaches for international destinations and a smaller, single-floored bus for national destinations.
Be creative and implement a form of inheritance.

Class	Responsibility
Vehicle	has all basic information about the vehicle

3.4 Alternative Exercise – Your own example

You can also implement your own example to show an interaction between a hierarchy of classes and a managing or data-pool class which can deal with these different types. Maybe you want to start be defining some unit test cases: How could you write a unit test before you have the code?

➔ See also the “Airport” example in the compendio book (see OneDrive folder)

3.5 Exercise – Relations

Implement a main program which instantiates the objects and creates a working object-hierarchy.

Our application should allow the user to do following:
Check times when bus leaves on platform. The user can also generate a new trip with a bus and the system checks what platform is available for that time. Implement a text-based user interface which allows the user to see the schedule. For example: when does the bus leave for Munich and on which platform?

3.6 Exercise – Managing your trips

Make sure that the platforms are correct for the right buses. All platforms can be used for national buses, but only a few platforms are big enough to hold international buses.

3.7 Exercise – Overriding Methods

One common OO feature is *overriding* methods. Use method overriding to implement a *print-out.*

3.8 Exercise –Overloading Methods

The second common feature is *overloading* methods. Do some research on this topic. Then show that you have understood this feature by implementing this in your project.

3.9 Exercise – UML Classdiagram (1A)

You did a design before you started implementation. Now draw a UML-classdiagram which shows the relationships of your project (in particular IS- and HAS relationships). Use a tool to do this.

➔ Show your finished design to your teacher.

3.10 Exercise – Testing (3A)

Note down essential test cases which prove that your application works. Get a colleague to do the testing for you.

Based on a script by Rinaldo Lanza, BBW. Adapted by Julian Käser. Latest version Nov. 2019