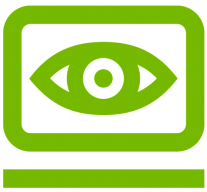

Storyboard - E-Learning Unit Polymorphism

for Informatik 1

HTW University of Applied Sciences Berlin
Faculty 4 - School of Computing, Communication and Business
International Media and Computing M.Sc.

Project: WT3 Didaktics of Media
Authors: Linda Fernsel (555949)
Konrad Ukens (572411)
Lotte Unckell (565918)
Nhu Mong Tran (575044)
Version: 1.1
Date: 08.02.2021

Contents

1

Overview of the E-learning Unit

This document describes an e-learning unit for teaching the programming concept of Polymorphism.

The unit is meant to be taught to first semester CS students within the frame of a 90 minute lesson. Ideally it is taught live with synchronous communication between the teacher and the students.

Familiarity with the following concepts should be given:

- basic syntax and class structure in Java or similar programming language
- inheritance, interfaces
- methods, parameters and field variables
- UML

First, the didaktic foundations for the teaching of the unit are outlined, which are intended for guiding the teaching process. Following this, learning goals relating to the content of the lecture are listed. These are then detailed in the third chapter "The E-Learning Unit", where the content for each lesson segment is described. Furthermore, each segment lists the information base required as well as realization comments that must be considered in preparation of the lesson.

The lesson is constructed to start with a brief warm-up. This consists of a re-cap of concepts and knowledge that should be familiar to students. Following this, the term of polymorphism is introduced from a non-CS background before analogies to programming are drawn. Once these have been explained, detailed examples of polymorphism in Java are shown with a familiar set of java classes.

After these examples, two exercises are given for students to complete in groups of three or four. In between the exercises results are collected and discussed. To complete the learning unit, a homework assignment is given by which students create their own "cheat sheet" overview of the concepts taught. This is to be graded, and notable examples can be made available online for students to compare to their own sheet once grading is done.

1.1 Overview of the Units Phases

Section	Goal	Resources	Minutes
Warm-Up	Recall knowledge about classes, objects, inheritance and interfaces	Example code: `https://github.com/LiFaytheGoblin/Info1-Polymorphism`	15
Intro: Term and Origin	Introduce the term from both a programming and non-programming context	Image of a gene tree to visualize (biological) relationships between species while discussing biological definition	10
Intro: Types and Examples	Exemplify applications while explaining how polymorphism is being applied	Example code: `https://github.com/LiFaytheGoblin/Info1-Polymorphism`	15
Exercise I	Apply knowledge of Polymorphism in a simple exercise, group work and discussion of results, possible challenges	Exercises, which have been cleared of the solution (See: Riddle.java, `https://github.com/LiFaytheGoblin/Info1-Polymorphism`	20
Exercise II	Apply the newly acquired conceptual knowledge to solve a problem and design a simple program, group work and discussion of results, possible challenges	Slides from previous lessons about UML Class Diagrams and/or Unit Testing and Debugging; Example UML Class Diagram for the Exercise (See: Exercise2ClassDiagram.png, `https://github.com/LiFaytheGoblin/Info1-Polymorphism`)	20
Wrap-Up	Repetition and reflection of the key concepts	/	5
Evaluation: Graded Homework	Evaluate knowledge and strengthen understanding	See example cheat sheet in appendix A.	/

Additional resources

See appendix B for an overview of all the tasks. This document helps the student as well as the teacher to follow the unit. Even though this unit is intended as a synchronous e-learning unit, one can use this document to read on what to do in the exercises or to do the unit asynchronously on their own.

2

Didaktical Background

2.1 Learning Theory

This storyboard applies the learning theory of Constructionism, a learning theory described in 1991 by Seymour Papert and Idit Harel in their book "Contructionism" [Papert und Harel 1991]. Constructionism extends Jean Piaget's theory of "Constructivism" - the idea that children learn by building knowledge - with the idea that children learn more effectively when the construction of knowledge is happening actively, like when working on projects in specific learning environments [Papert und Harel 1991]. When designing a constructionist learning experience the design of a learning situation is necessary. In the case of this e-learning unit we chose three different example cases: pets, sports and vending machines. Students will be working on creating their knowledge about Polymorphism within these environments, in micro projects.

Another concept of Papert that we designed our storyboard by is the idea of "Low Floor" and "High Ceiling", to which we added "Wide Walls", Resnick's extension of Papert's idea [Resnick 2016]. To ensure a "Low Floor", this e-learning unit starts with a warm up. Here, students are encouraged to start participate and prerequisite knowledge is recapitulated by way of playing around with a code example. The "High Ceiling" is provided by an open ended homework: One could focus on the theory alone but one can also go ahead and build the project the homework is about. Finally, as Resnick says, "[i]t's not enough to provide a single path from low floor to high ceiling; we need to provide wide walls so that kids can explore multiple pathways from floor to ceiling." [Resnick 2016]. Therefore we designed the different example cases mentioned earlier but we also added an abstract riddle for students to solve. And we give students the opportunities to work with the plenum, with peers and on their own. We also provide various methods for constructing knowledge about Polymorphism.

These methods for constructing knowledge - or, phrased differently, the ways students "act, think or feel" [Bloom et al. 1956] - are organized after Bloom's Taxonomy, as defined by Benjamin Bloom et al. in 1956 in "Taxonomy of educational objectives". In the introduction, when Polymorphism is first introduced, we start on the lowest level with "knowledge", defining Polymorphism and identifying elements of Polymorphism in program code. In the first in-depth exercise we go over to "comprehension", interpreting code containing Polymorphism. In the second in-depth exercise students proceed to the "application" of their knowledge about Polymorphism, completing a UML diagram. In

the homework, students are asked to paraphrase and structure their knowledge in form of a cheat sheet, adding examples for all elements of Polymorphism. This leads students to the next two levels of Bloom's taxonomy. First, "Analysis" is defined by Bloom as focusing on "the breakdown of the material into its constituent parts and detection of the relationships of the parts" [Bloom et al. 1956], exactly what students do when figuring out what to put on the Cheat Sheet and where. Second, "Synthesis" Bloom defines to be "the putting together of elements [...] as to form a whole" [Bloom et al. 1956], which is the case when assembling a Cheat Sheet with coherent examples.

2.2 Learning Goals

After finishing this e-learning unit the students will have the skills listed below. These learning goals are categorized in the five levels of Bloom's Taxonomy [Bloom et al. 1956], starting at the lowest level.

2.2.1 List of Learning Goals

Bloom 1 - Remember
As a Recap: - Remember how to make a class in Java inherit from another class - Remember how to make a class implement an interface in Java - Remember what the difference between an interface and a class is - Remember why we use interfaces and not just classes
During the Unit: - Define the term polymorphism - State and define the two types of polymorphism - Define overloading and overriding
Bloom 2 - Understand
- Understand the difference between Overloading and Overriding - Understand the difference between the two types of polymorphism - Give examples for polymorphism - Explain in your own words what Polymorphism is in general and how it works - Explain what polymorphism has to do with inheritance
Bloom 3 - Apply
- Identify where polymorphism might be useful in solving a specific problem - Discuss where/why/when polymorphism is useful in general
Bloom 4 - Analyze, Evaluate, Create
- Evaluate a solution that uses polymorphism on whether polymorphism is useful here and why - Be able to use existing polymorphic structures - Design simple polymorphic structures - Program simple polymorphic structure in Java

3

The E-Learning Unit

3.1 Warm Up

The first phase of the e-learning unit on Polymorphism is the "Warm Up". Principally it functions as a recapitulation of priorly learned topics that are a prerequisite for understanding Polymorphism. The secondary function is to encourage students to start interacting and thinking along.

In order to complete the stated goals, the teacher can ask for examples for or comparisons between concepts. Each concept is then demonstrated using Java. The teacher could ask questions on how to implement certain concepts. Finally the teacher can give a résumé on the prerequisite concepts.

As a follow-up the teacher can comment and upload the code created during this unit.

3.1.1 Description of the page

- Review of examples for prerequisite concepts

3.1.2 Didaktical background

Learning goals and knowledge presentation
Remember definitions, explanations and examples for prerequisite knowledge about the concept of classes, objects, inheritance and interfaces Remember how to work with said concepts using Java

Interaction and/or exercises
"Give examples to explain the difference between a class and an object." "Give examples for an inheritance hierarchy in real life." "How is implementing an interface different from inheriting from a class?"

3.1.3 Information Base

Example Program

An example program that demonstrates objects, classes, inheritance and interfaces can be found at `https://github.com/LiFaytheGoblin/Info1-Polymorphism`. Teachers can fork this project to modify as needed.

The difference between a class and an object

Example: You can have a class "Carnivore" and instantiate an object concerning a specific individual animal.

Demonstration: Prepare a class "Carnivore" with an "eat()" method in advance. For the demonstration, instantiate an object of the "Carnivore" class.

Inheritance hierarchy

Example: A Cat IS s Carnivore. A Cat IS a Pet.

Demonstration: Build upon the sample project from before. Create a new class "Cat" that inherits from "Carnivore" (but does not override any method for now). Instantiate an object of class "Cat" next to the "Carnivore" object. Call "eat()" on both objects to demonstrate that the "Cat" object calls its parent's "eat()" method. Then override the "Cat" class' "eat()" method and demonstrate that the "Cat" object will now call its own "eat()" method.

The difference between implementing an interface and inheriting from a class

An interface can be implemented by a class. It says which methods the class should define. The class won't have default methods that will be used if it doesn't override them - it needs to implement all methods from the interface.

Demonstration: Build upon the sample project from before. Create a new interface "Animal" with a method "eat()", "sleep()" (Question: "What do all animals do?"). Have "Carnivore" implement "Animal". Show that one needs to implement all of "Animal"'s methods. You can additionally have the "Cat" class implement a "Pet" interface and add another class "Stone" that implements "Pet" but is not an "Animal".

Résumé

The "Animal" interface defines what all animals do. The "Carnivore" class gives a default for a method. A "Cat" class that inherits from "Carnivore" can use the provided method or override it. An object is the individual cat, while the class is like a blueprint for it.

3.2 Introduction: Term and Origin

In this phase, the concept of polymorphism is introduced. The term is introduced with a question to the students if anyone has heard the term before or if anyone can imagine what

it might refer to. Following any answers (or absence of any), the greek translation is presented, along with the term when used in biological contexts: "Polymorphism", Greek for "Many forms" “Polymorphism, in biology, [is] a discontinuous genetic variation resulting in the occurrence of several different forms or types of individuals among the members of a single species."

After introducing the term outside of a programming concept, a simple definition from W3schools is given, and it’s implication in programming: “Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.” Following the definition, the teacher explains that this is done by allowing child-classes to overwrite methods of the parent class with their own implementations. Not only does this allow children classes to maintain a method name, but also for different child-classes to have their own implementations of identically named methods.

As a final question in this segment, the teacher may ask where applying something like this may be helpful, or where example analogies in the real world may be observed (e.g. ’eating’, or ’self-protection’ in nature).

3.2.1 Description of the page

- Biological and programming context definitions of polymorphism

3.2.2 Didaktical background

Learning goals and knowledge presentation
Explanation of the concept: Polymorphism (without and within the programming context) Presentation of various definitions

Interactions and/or exercises
"What might Polymorphism mean?" "What might Polymorphism be useful for/used for?"

3.2.3 Information Base

External links and resources

Item	Source/References
Definition Polymorphism (Biology)	`https://www.britannica.com/science/polymorphism-biology`
Definition Polymorphism (Programming)	`https://www.w3schools.com/java/java_polymorphism.asp`

Additional resources

An image of a genetic tree to visualize (biological) relationships between species while discussing biological definition.

3.3 Introduction: Types and examples of Polymorphism

During this phase the teacher will introduce types and examples of polymorphism by giving examples following a known analogy.

The sample code can be found in the following repository folder:
`https://github.com/LiFaytheGoblin/Info1-Polymorphism/tree/main/Polymorphism`

Image: Code excerpt with example of polymorphism.

Question to students: "Can anyone see how polymorphism is being applied here?"

Explanation: Following the definitions offered outside of the computer science realm, the two types of polymorphism and their respective subtypes are introduced.

Polymorphism with method referencing: overriding and overloading

In the following example, the method of the parent class (carnivore) is overwritten in the child class. This is called *method overriding*.

```
carnivore.speak(); // Output: "Rawr"
cat.speak(); // Output: "Purr"
```

If the method speak is defined in cat as such (in addition to the no-argument speak()):

```
public void speak (int nrPurrs){
    for(int i = 0; i < nrPurrs; i++) {
        System.out.println("purr");
    }
}

cat.speak(2);
// Output: "Purr"
// Output: "Purr"
```

In this example above, the signature of the method are modified to accept different parameters. This is called *method overloading*.

Polymorphism with object referencing:

Objects of a child class may be used the same way an object of the parent class is used.

```
Carnivore carnivore = new Carnivore();
Cat cat = new Cat();

Carnivore [] hunters = new Carnivore[] {carnivore,cat};

for (Carnivore hunter : hunters){
    hunter.speak();
}
// Output: "Rawr"
// Output: "Purr"
```

In this example, both carnivore and cat are 'treated' as objects of type Carnivore by being added to the array of type Carnivore (hunters), but the method speak() is being called from the type of the object of cat, Cat.

Finally, variables of the type of a parent class may be used to reference objects of a child class. In this case, the object is treated as the type of the parent class, and can not access method implementations or variables specific to child objects. During runtime however, the method executed will be that of the child class

```
Carnivore otherCat = new Cat();
otherCat.speak();
// Output: "Purr"
```

In this example, variable 'otherCat' is used to reference an object of type Carnivore. However, as we are using the constructor of the child class, Cat, we are making an object of type Cat that is though used as a Carnivore type object. It can execute the method speak(), which will print a "Purr", but can not execute a speak(2), a method overload specific to the child class.

3.3.1 Description of the page

- Slides showing the types of Polymorphism as well as code examples

3.3.2 Didaktical background

Learning goals and knowledge presentation
Show and explain cases of Polymorphism

Interactions and/or exercises
"Please point out Polymorphism in the given examples."

3.3.3 Information Base

External links and resources

Item	Source/References
Github Repository with code samples	`https://github.com/LiFaytheGoblin/Info1-Polymorphism/tree/main/Polymorphism`

3.4 In-depth exercise I

For the first of two larger group exercises, students will analyse a program (or excerpt of one, see draft in figure 3.1 on the next page) with the goal of figuring out what will be printed to the console after each of the eight commands. Each group, consisting of students and their neighbors (so groups of three) should gather for around 20 minutes and discuss the code excerpt. Afterwards the results can be discussed for 5-10 minutes. Students should motivate their results, thus proving their understanding of the underlying principles of Polymorphism.

```
public static void main(String []args){
    class A{
        public int getX(){
            return 10;}
    }
    class B extends A{
        public int getX(){
            return 10;
        }
        public int getX(int y ){
            return y+10;
        }
    }
    class C extends B{
        public int getX(){
            return 30;
        }
    }

    int x;

    {
        A a = new A();
        x = a.getX();
        System.out.println(x);
        // returns 10

        B b = new B();
        x = b.getX();
        System.out.println(x);
        // prints 10

        x = b.getX(x);
        System.out.println(x);
        // prints 20
    }
{

    C c = new C();
    x = c.getX();
    System.out.println(x);
    // prints 30

    x = c.getX(c.getX());
    System.out.println(x);
    // prints 40

    A b = new C();
    x = b.getX();
    System.out.println(x);
    // prints 30

    C a = new C();
    x = a.getX();
    System.out.println(x);
    // prints 30

    x = a.getX() + b.getX() + c.getX();
    System.out.println(x);
    // prints 90
}
```

Figure 3.1: Exercise I - See section 3.4.3 "Information Base"

3.4.1 Description of the page

- Code excerpt and task definition of group exercise
- Blank board or slide where results can be collected

3.4.2 Didaktical background

Learning goals and knowledge presentation
Solving of simple Polymorphism cases presented as riddles Application of theory learned

Interactions and/or exercises
Group exercise: Discussion of code excerpt to analyse and understand application of polymorphism Group discussion of results Collection of results

3.4.3 Information Base

External links and resources

Item	Source/References
Github Repository with code sample	`https://github.com/LiFaytheGoblin/Info1-Polymorphism/blob/main/Riddle/Riddle.java`

Realization comments

Remove commented out solutions before presenting the exercise.

3.5 In-depth Practise II

Since the students should be able to identify the elements now, during this phase they will further deepen their understanding of polymorphism. There are several exercises possible. They will analyze, combine and construct elements of polymorphism. They can either complete a given UML diagram with the correct presentation of the class relationships, design a simple program with the use of polymorphism or look for errors in a given code and suggest a solution on how to fix it.

Each group, consisting of students and their neighbors (so groups of three) should gather for around 20 minutes and discuss the given problem. Afterwards the results can be discussed for 5-10 minutes. Students should motivate their results, thus proving their understanding of the underlying principles of polymorphism.

3.5.1 Description of the page

- Description of individual task and/or group exercise
- Short overview about UML Class Diagrams (depending one the chosen exercise)

- Short overview about unit testing and debugging (depending one the chosen exercise)

3.5.2 Information base

- Slides from previous lesson about UML Class Diagrams
- Slides from previous lesson about unit testing and debugging

3.5.3 Exercises Task & Code Example

Interaction and/or exercises

Possible Individual Exercises:

- Complete an UML Class Diagram - Provide an incomplete UML class diagram and classes, which are needed to fill the diagram. Tell the students to represent their relationship in the incomplete diagram and let them describe where polymorphism is made use of. Give them around 10-15 min to solve individually. Evaluate by solving it together. *Task description example:* Have look into polymorphism by completing this following UML class diagram (see figure 3.2 on the next page and the following URL `https://github.com/LiFaytheGoblin/Info1-Polymorphism/blob/main/Exercise2ClassDiagram.png`). This diagram represents the relationship between different types of cups and liquids. How do you represent the following relationships in the incomplete design: GlassCup, Tumbler, Liquid, Water, Coffee, Soy Milk. What kind of relation do the classes have. Describe where polymorphism is made use of.
- Create an own UML Class Diagram - Present a use case in which polymorphism can be used. It can be anything from an abstract example such as geometrical Shapes to a paint program. A proper UML Class Diagram is not required. It could also be a simple sketch. *An possible task description could be:* Design a simple program for a drink vending machine. Let your creative self flow. Possible classes could be: VendingMachine, Cups, Liquids, Recipes, etc. Use the concept of polymorphism and describe where it can be made us of.

Group Exercise: Debugging Strategies: Give simple classes with errors (maybe even different task to every group). Split them into groups and put them in Breakout-Rooms. The students need to find the errors and suggest a solution on how to fix them. Evaluate by letting each group present their solution.

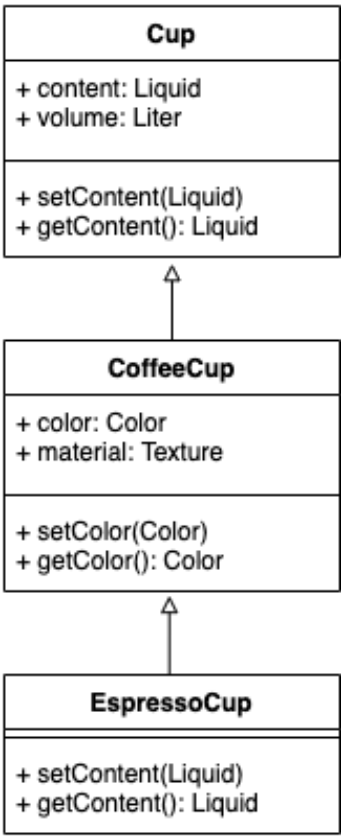

Figure 3.2: Exercise 2 - Example of an incomplete UML Class Diagram

3.5.4 Didaktical background

Learning goals and knowledge presentation
The students are able to apply the newly acquired conceptual knowledge of polymorphism to solve a problem. Present exercise with the use of slides or similar and evaluate the results interactively by using a virtual whiteboard or something similar.

Interaction and/or exercises
Reinforce problem-solving skills and debugging strategies Working in groups and/or individually

3.5.5 Information Base

External links and resources

Item	Source/References
Github Repository with UML class diagram	`https://github.com/LiFaytheGoblin/Info1-Polymorphism/blob/main/Exercise2ClassDiagram.png`

3.5.6 Notes on implementation

Knowledge Requirements
UML Class Diagrams - Knowledge alignment with the students needed. Possibly recap and reference to past lessons about unit tests and JUnit to brush up on the debugging strategies.

3.6 Wrap-Up

In the last phase of the e-learning unit, the "Wrap Up", the key concepts are repeated. This gives the students an opportunity to reflect on what they have learned in this unit. It is also an opportunity for the instructor to check in with the students and see, whether the most important concepts of the e-learning unit came across and to clear up any last misconceptions.

To reach these goals the lecturer asks the students questions and has the students gather what they remember from the unit. These questions could be: "What do you now know about polymorphism?/What is polymorphism?" or "When is polymorphism useful?". After the students are done gathering what they have retained, the lecturer reflects on the gathered points regarding what was said, what was missing and how it fits with the actual concepts of polymorphism. They also tie in these points with the examples from the previous sections (Animal-Example and Vending-Machine-Example) and show what these seemingly very different examples have in common. This is also the last chance during the unit for the lecturer to correct any misconceptions that came up during the gathering of student answers.

3.6.1 Didaktical background

Learning goals and knowledge presentation
The student are able to remember and understand the definition, uses and benefits of Polymorphism and are able generate these in their own words. Use the answers the students give to visually build up the definition, uses and benefits of Polymorphism again. Then highlight which aspects are correct, which are misconceptions and add those that are missing.

Interaction and/or exercises
Verbal interaction with visual aid: collect what the students remember of the definition, uses and benefits of Polymorphism and write it down somewhere. Then add what was missing, point out which are misconceptions and remove them.

3.6.2 Notes on implementation

Keep the repetition brief unless there are major misconceptions.

3.7 Evaluation: Graded Homework

After the synchronous part of the e-learning unit is done, the students get to work on a task by themselves. To gain in depth understanding and to test their knowledge they will create a solution to a problem using Polymorphism.

Students can work on their own or in groups, but they each hand in their results which should also contain any questions that came up while doing the task. The students will create a cheat sheet containing at the minimum the definitions of polymorph variables, dynamic binding, method overriding and method overloading and a code example for each definition. The students can come up wit their own scenario for these examples or use one provided by the teacher. But because they are supposed to apply their newly acquired knowledge to another problem they are not allowed to use one of the scenarios covered in the e-learning unit.

An examplary cheat sheet is attached in appendix A.

The feedback is given in a using one through four stars (four stars being the best and one star the worst). This scale is used instead of the usual grading scale to keep the students motivation high. It also includes a short written feedback and answers to the questions the students had while doing the task.

Reading through, giving feedback on and grading the reports is also an opportunity for the instructor to check, whether all the important points came across. Grading them before the next unit would be beneficial for the students learning and also, because then any remaining misconceptions, reoccurring questions and gaps in the students knowledge can be addressed by the instructor in the next unit.

Some of the best hand-ins will be shared with all of the students in the class anonymously to showcase different ways of approaching the task and to provide the students with more studying material.

3.7.1 Information base

Item	Source/References
Scenario description	to be written by the instructor (e.g. a player which is able to play several different ball games)
E-learning unit on Polymorphism and it's materials	any materials, examples and slides used to conduct this unit can be used by the students to write their cheat sheet

3.7.2 Didactical background

Learning goals and knowledge presentation
The students are able to apply their newly acquired knowledge about polymorphism to a problem. The students get a written exercise (PDF/Webpage/...) and hand in a document containing the cheatsheet (PDF).

Interaction and/or exercises
Interaction in groups if the students desire. Interaction with teacher over written feedback on cheatsheet.

3.7.3 Information Base

External links and resources

Item	Source/References
Github repository with cheat sheet	`https://github.com/LiFaytheGoblin/Info1-Polymorphism/blob/main/PolymorphismCheatSheet.pdf`

3.7.4 Notes on implementation

Knowledge Requirements
This e-learning unit on polymorphism

Hints
Do grade the cheatsheet as close to the hand in as possible for maximum impact on the students learning and to be able to address any remaining misconceptions, reoccurring questions and gaps in the students knowledge as soon as possible.

Bibliography

[Bloom et al. 1956] Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., und Krathwohl, D. R., *axonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain.* Longmans, Green and Company.

[Papert und Harel 1991] Papert, S. und Harel, I., *Constructionism*. Ablex Publishing Corporation.

[Resnick 2016] Resnick, M., Designing for wide walls.

List of Figures

Appendix A

Cheat Sheet

See the following URL to download the cheat sheet:

`https://github.com/LiFaytheGoblin/Info1-Polymorphism/blob/main/PolymorphismCheatSheet.pdf`

The cheat sheet is attached on the next page as well.

Polymorphism

Cheat Sheet - Informatik 1

Source

Merker E. (2004) Objektorientierung mit Java. In: Grundkurs Java-Technologien. Vieweg+Teubner Verlag.
https://doi.org/10.1007/978-3-322-80261-3_2

Polymorph Variables

The type of a variable (eg. "Game") is set when the variable is declared:

```
Game game;
```

The variable "game" can be assigned any object that is of type "Game", it can also be assigned an object that is of a class inheriting from "Game" (eg. "Soccer"):

```
game = new Soccer();
```

An array of type Game could hold objects of any class that inherits from Game.

Dynamic Binding

A class "Game" might implement a method "play()". The classes "Basketball" and "Soccer" inherit from "Game" and override the available "play()"-method.

Dynamic Binding means that it is only decided during runtime, which of the "play()"-methods will be called - depending on which type the object is, on which the method is called.

```
private Game decideOnGame() {
  if(this.lovesSoccer) {
    return new Soccer();
  }
  else {
    return new Basketball();
  }
}
// ...
Game game = decideOnGame();
game.play();
```

Method Overriding

A class (eg. "Game") implements a public, non-static method (eg. "play()"). Several classes (eg. "Soccer", "Basketball") that inherit from class "Game" can override the parent's method "play()" (it needs to have the same signature).

If the parent's method is overridden in the child class, the child's method will be called whenever "play()" is called on an object of the child class.

```
public class Soccer extends Game {
  public void play() {
    // overrides parent's method
    runToBall(); // and so on...
    super.play(); // call parent's
                  method
  }
}
```

Method Overloading

A class may have multiple versions of a method. They have an identical name and return type but differ in their parameters.

Depending on the parameters passed in a method call, the according method is chosen.

```
public void kickBall(){
  System.out.println("a player kicks
  the ball");
}

public void kickBall(Direction dir){
  System out.println("a player kicks
  the ball in direction " +
  dir.toString());
}
```

Appendix B

Overview of the tasks

See the following URL to see the README.md file consisting of all the task mentioned in this storyboard:

`https://github.com/LiFaytheGoblin/Info1-Polymorphism/blob/main/README.md`

The content can be read on the next page.

A Unit on Polymorphism

This page helps you to follow the unit on Polymorphism. It is a synchronous e-learning unit, but you can come here to read on what to do in the exercises or to do the unit asynchronously on your own. If you do the e-learning unit synchronously, you have the advantages of being able to work with your commilitones, to ask questions and get immediate answers and to participate in the plenum discussions of the exercise solutions, to get immediate feedback on your understanding.

1. Warm Up (15 Min)

Download the sample code of the folder https://github.com/LiFaytheGoblin/Info1-Polymorphism/tree/main/ObjectsClassesInheritanceInterfaces and open it in BlueJ.

Answer the following questions: * Give examples to explain the difference between a class and an object. * Give examples for an inheritance hierarchy in real life. * How is implementing an interface different from inheriting from a class?

Explore the code and if needed adjust your understanding of classes, objects, inheritance and interfaces.

2. Introduction (30 Min)

I. Definition of Polymorphism and Origin

Answer the following question: * Have you heard about Polymorphism? What might it refer to?

Read the following definitions for Polymorphism: * "Polymorphism, in biology, a discontinuous genetic variation resulting in the occurrence of several different forms or types of individuals among the members of a single species." (Source: https://www.britannica.com/science/polymorphism-biology) * "Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways." (Source: https://www.w3schools.com/java/java_polymorphism.asp)

Answer the following question: * Name real world analogies for Polymorphism.

II. Concepts and Examples

Download the sample code of the folder https://github.com/LiFaytheGoblin/Info1-Polymorphism/tree/main/Polymorphism and open it in BlueJ.

Look at the following method of the Carnivore class:

```
public void speak (int howOften){
    for(int i = 0; i < howOften; i++) {
        System.out.println("Rawr");
    }
}
```

Look at the following two methods of the Cat class, that inherits from Carnivore:

```
public void speak (int nrPurrs){
    for(int i = 0; i < nrPurrs; i++) {
        System.out.println("Purr");
    }
}

public void speak (String sound, int nrPurrs){
    for(int i = 0; i < nrPurrs; i++) {
        System.out.println(sound);
    }
}
```

Look at the following code in the main()-method:

```
Carnivore carnivore = new Carnivore();
Cat cat = new Cat();

Carnivore[] hunters = new Carnivore[] { carnivore, cat };

for (Carnivore hunter : hunters){
    hunter.speak();
}

// ...

Carnivore otherCat = new Cat();
otherCat.speak();
```

Answer the following questions (you can put the code above into the sample code to see whether you guessed correctly): * How do the two methods in the Cat class differ from each other? * What happens if I call `carnivore.speak(2);` on an object of type Carnivore? * What happens if I call `cat.speak(2);` on an object of type Cat? * What happens if I call `cat.speak('meow', 2);` on an object of type Cat? * What happens if I call the `main()` -method? * What do you think, why can I put objects of type Carnivore and Cat into the hunters array in the `main()` -method?

Think about the following explanations: * When Cat implements a method with the same signature as the parent class, that is called **method overriding**. * When Cat implements different methods with almost the same signature but different parameters, that is called **method overloading**. * Objects of a child class may be used the same way an object of the parent class is used. In this example, both `carnivore` and `cat` are treated as objects of type Carnivore by being added to the array of type Carnivore (`hunters`), but the method `speak()` is being called from the type of the object of `cat` , Cat. When it is only decided at run time which method exactly is called (of class Carnivore or Cat), that principle is called **dynamic binding**.

3. In-depth Practise

I. Riddle (20 Min)

Download the sample code of the folder https://github.com/LiFaytheGoblin/Info1-Polymorphism/blob/main/Riddle and open it in BlueJ. Do not run the program yet.

Answer the following question: * What will be printed after each of the printing commands and why?

Run the code to see whether you were right.

II. Design (20 Min)

Download the UML class diagram https://github.com/LiFaytheGoblin/Info1-Polymorphism/blob/main/Exercise2ClassDiagram.png and either upload it into a miro board for collaborative editing or put it into an image editing software of your choice. This diagram represents the relationship between different types of cups and liquids.

Answer the following questions: * Take the following elements: Glass Cup, Tumbler, Liquid, Water, Coffee, Soy Milk. If those were classes, what relationships would they have with each other? * Add the elements as classes to the given UML. * After adding the classes to the UML, describe where polymorphism is made use of.

Wrap Up (5 Min)

Reflect on the following questions: * "What do you now know about polymorphism?" * "When is polymorphism useful?"

Graded Homework (60 Min)

Create a cheat sheet containing at the minimum the definitions of polymorph variables, dynamic binding, method overriding and method overloading and a code example for each definition. You can come up with your own scenario for these examples or make one around the theme of ball sports games. You are not allowed to use one of the scenarios covered in this unit. To see whether your code examples work the way you expect it makes sense to test them!

Hand in: * The cheat sheet * Any questions that might have come up

You will be graded with a maximum of 4 stars and be given a feedback. If your cheat sheet is especially good, it will be uploaded into the honorable gallery of great cheat sheets, and you get an extra star for it.