# Falk-Jonatan Strube

*Liebigstr. 30*
*01187 Dresden*
✉ *s74053@htw-dresden.de*


Mrs B.Arch. B.Sc. Darlene Kilian
*University of Applied Sciences Dresden*
*Office Z 718*
Friedrich-List-Platz 1
01069 Dresden

June 17, 2016

**Written Follow-Up Report**

Dear Mrs Kilian:

This is my submission of the written follow-up report on my seminar "Object oriented programming".

The report provides a conspectus of object oriented programming and its uses. By covering the basics of *generic* as well as *object oriented programming*, it gives an adequate foundation for understanding the concept of object orientation.

The report will be advantageous for students who are studying computer science or are going to work with computer scientists.

Sincerley,

Falk-Jonatan Strube

encl: Report on object oriented programming

# ENGLISH C1 REPORT

## Object oriented programming

submitted to

Mrs B.Arch. B.Sc. Darlene Kilian

Teacher *English as a foreign language*

University of Applied Sciences Dresden

June 17, 2016

by

Falk-Jonatan Strube

This report provides an overview over *object oriented programming* and introduces the advantages of preparing and implementing an object oriented program over generic programming.
The report is concluded by summarizing the benefits of object orientation.

# Contents

# List of Figures

# Abstract

The process of programming an computer application does not start with the first line of code. It starts by thinking of demands of the desired application and how these demands should be implemented.

Object oriented programming (OOP) is an advanced way of structuring a computer program before and during development. It therefore aids in the process of finding and implementing good solutions for various demands of an application.

By modeling dependencies and correlations in classes with the *Unified Modeling Language* (UML), a complex code formation can be made understandable. In addition to helping the programmer with designing his application, it benefits team communication and communication with a client. It also improves scalability since the general overview allows specific changes in the model and the program itself.

The key concepts of object oriented programming are inheritance and polymorphism. With inheritance, similar parts of a program may be reused in *child*- and *parent*-objects. Polymorphism allows the modification of a inherited function. Both features add to comprehensibility and scalability.

Object oriented programming is used in almost every complex computer program. From operating systems such as *Windows* or *Linux* to word processor applications such as *Microsoft Word* – object oriented programs are everywhere. Though generic programming is more commonly known, since it is easier to grasp at first, object oriented programming is more common overall.

# 1 Introduction

When a computer scientist starts working on a computer application, he has to work out the following questions:

- What should the application be able to achieve?
- How should the application be structured to be able to achieve those goals?

For most applications, the first step towards answering the second question is *object orientation*. By creating an object oriented model, the computer scientists has a solid base for implementing the desired features.

But what is object orientation? What is an object in the context of computer programming?

> "An object is a software bundle of related state and behavior." [2]

By thinking object oriented, the computer scientist can break a big problem down into smaller ones by assigning different functions to different objects. By doing this, the computer scientist does not only make the code more understandable, the application becomes more versatile, too. The proper separation of functions and properties into different objects aids in making these objects re-usable for similar problems.

The computer scientist who thinks and programs object oriented keeps himself open towards the growth of an application. Objects can be modified with ease and by having a bigger picture in mind, the application can have potential for more and is not crippled by a first model.

# 2 Overview

Object oriented programming stands in contrast to generic programming. While object oriented programs are separated into objects, generic programs are essentially one big object.

## 2.1 Generic programming

Generic programming is the easy way – at first. The programmer can write whatever he wants without putting some thought into how the application should work in the end. This freedom at the beginning of the development process comes at the expense of the difficulties later on. Unstructured code becomes more and more difficult to maintain and expand.

Imagine programming the behavior of the human being, more precisely the eating behavior. One has to differentiate between different types of humans at first. If one differentiates between age, there are some basic differences. While the adult is more unrestricted in what or how he eats, a toddler is very dependent on the parents and can not eat everything an adult can.

```
if ( adult ) {
  cook_something();
  eat();
} else if ( student ) {
  call_pizza_service();
  wait();
  eat();
} else if ( child ) {
  tell_mom_youre_hungry();
  wait();
  eat_only_dessert();
} else if ( baby ) {
  while ( nothing_happens ) {
    cry();
  }
  be_breastfed();
}
```

Figure 2.1: Pseudo-code of a generic program

Figure 2.1 shows a simplified pseudo-code snippet of a generic program. The Code basically says "If you are an adult, cook something and eat. If you are a student, call a pizza service, wait for the pizza to arrive and eat" and so on. While some code is reused by reusing basic functions (such as *eat()*), most of the code seems very redundant and complicated.

Above all, the code has limited scalabilty. Every additional age (i.e. an elderly) would be inserted at the end of the different cases and contribute to clutter the code. Additionally the program may run into problems when shared functions (such as *eat()*) are changed. If the *eat()*-function would be expanded to chew the food more (respecting the weak-jawed elderly), the whole code from figure 2.1 had to be rewritten. Since code often changes during development [4] this is a major problem with generic programming.

## 2.2 Object oriented programming

Object oriented programming splits a problem into smaller pieces. These pieces, called objects, are set into relation to each other. This braid of objects forms the unit of meaning which is the application itself.

To visualize this braid, a *unified modeling language* (UML) is used [3]. It basically displays an object (or the abstract version of an object, a class) in one box as seen in figure 2.2. The box includes the properties and the functions of the object. These boxes are connected with lines or arrows to symbolize their relation.


Figure 2.2: UML basics

Using this UML visualization, one can start to display the example from 2.1, seen in figure 2.3. This UML diagram shows, that the problem of how different ages handle their eating habits is split between the objects. While every *human being* has the property *hunger* and a way to *satisfy its hunger*, the objects *adult* and *student* have additional properties. The *student* additionally has the function *satisfy his hunger* – the same as the *human being*. Although the *student* is

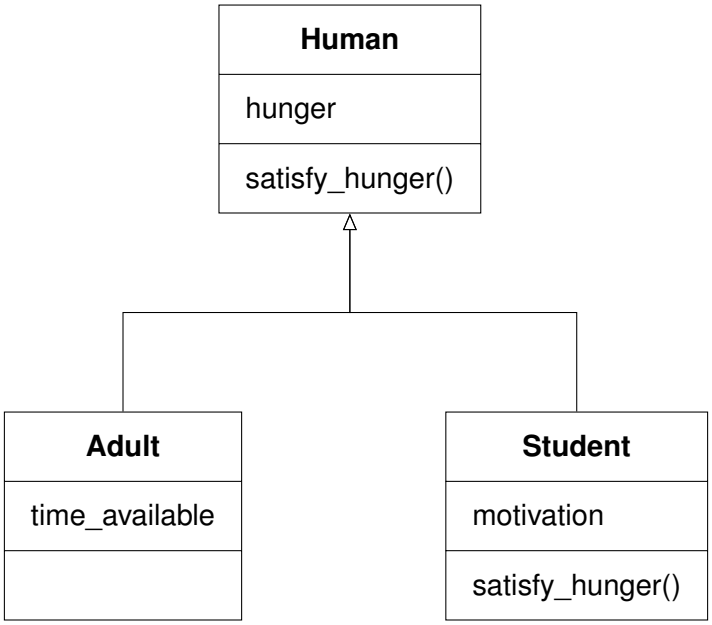


Figure 2.3: UML visualization of an object oriented program

connected to the *human being* and thereby already has the function to *satisfy his hunger*, he redefines it. That way, the *student* may include his motivation to cook something into the function to *satisfy his hunger* as a parameter: If he has motivation to cook something, he cooks. Else he may call a pizza service.

The connection between objects and the redefinition of functions of a connected object are the two main advantages and key features of object orientation. They are called *inheritance* and *polymorphism*.

# 3 Object oriented programming

As indicated before, the main features of object orientation are *inheritance* and *polymorphism*. The following sections are going to explain the concepts in more detail.

## 3.1 Inheritance

The term *inheritance* describes how two objects are connected.

In the example from figure 2.3, the objects *adult* and *student* both inherit their functions from their *parent human being*. That means, that every property or function written in the *parent* object apply to its *child* objects as well. That advantage is, that the *human* object may be expanded with functions such as *breathing()* or *drinking()* which then apply to its *children* as well. The result is an excellent scalability.

An other use case is, when the *parent* has to be expanded with more *children*. By taking the example from figure 2.1, we can easily add the two *child* objects *child* and *baby*. Both inherit the *hunger* property and the function to *satisfy their hunger*, as seen in figure 3.1.

All *children* are guaranteed to have at least all properties and functions of their *parent*. But they may redefine them to their own needs. This redefining is called overwriting and is a key part of *polymorphism*.

## 3.2 Polymorphism

The word polymorphism consists of the word *poly*, meaning *many*, and *morph*, meaning *form*. It describes when children have many different variations of their parents function.

In the example from figure 3.1 *student*, *child* and *baby* have polymorphic functions from their parent, the *human being*. The function of *satisfying their hunger* has different forms: One may depend on *motivation*, the other on *defiance*. The

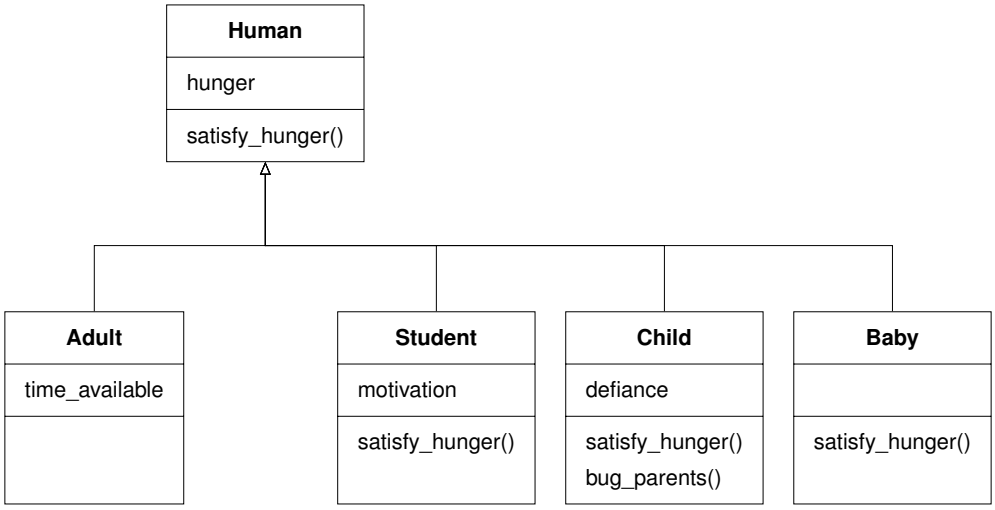


Figure 3.1: Expanded inheritance

*baby* may even have a totally new implementation of the function, as it is totally different in its behavior.

Figure 3.2 shows both polymorphism and inheritance in the example of different groups at the university. The inheritance is applied through *childrens children*. This illustrates the advantage of object oriented models: The complexity may rise, but it is still possible to see the connections. A *beginner* is a *person* as well as a *student*, in fact every object is a *person*.

With both *inheritance* and *polymorphism* these problems have been solved through object orientation. With the aid of UML, an easy to understand diagram can be designed and help to make the application future-proof.

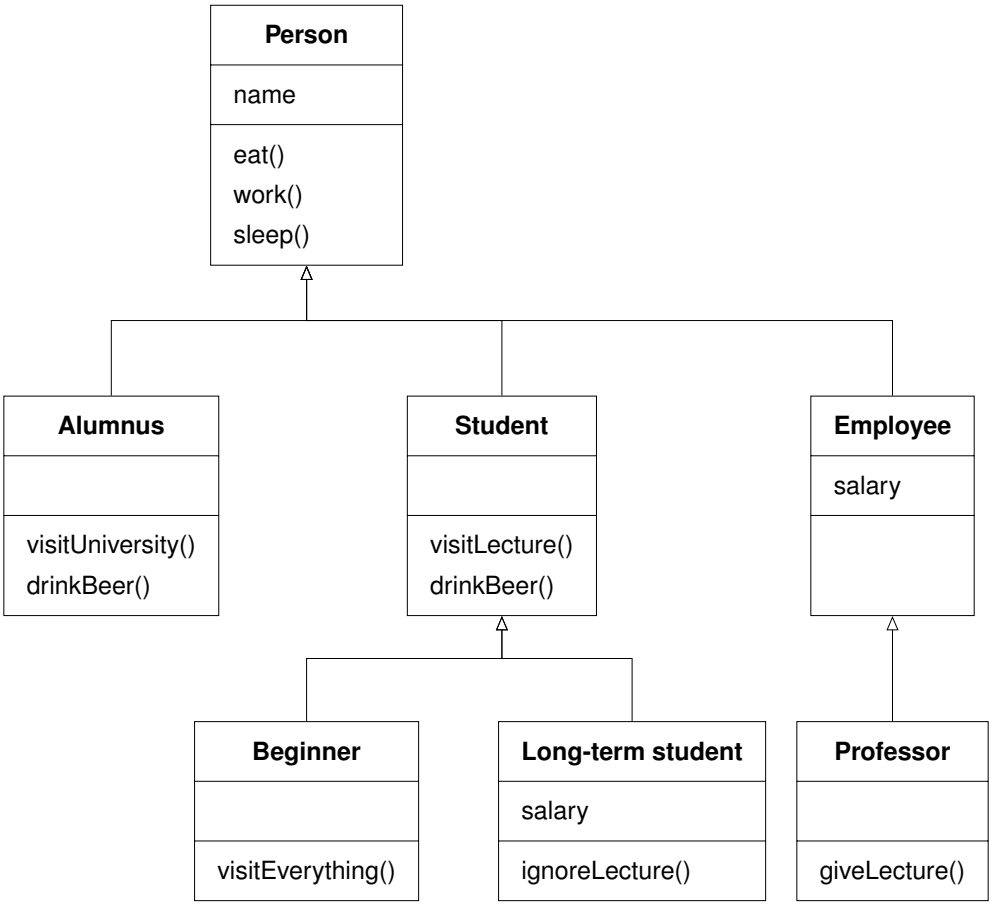


Figure 3.2: Deeper inheritance and polymorphism

# 4 Conclusion

Object oriented programs are everywhere. No major program is written without object orientation [5]. The report shows why this is: A program without object orientation is more difficult to change and to expand – to develop all together. As software becomes more and more complex throughout development, these aspects become more and more important. When applications become more complex, there are more possibilities for errors. With object orientation these errors can be found and dealt with more easily.

But before a program can become more complex, it has to be coded at first. And before this happens, it has to be designed. Object orientation and its visualization model UML help in this progress unlike generic programming.

Nowadays software is a huge part of life. Almost every person carries a mobile phone with him everywhere he goes. Most people use personal computers almost every day. Software development is a huge part of the modern society [1]. As a computer scientist it is imperative to learn object oriented programming to be a good software developer.

Although object oriented programming itself may not be important for many fields of study, the concept of object orientation – that is, splitting problems into smaller ones and set them in relation to each other – can be abstractly used in a lot of different ways.

Altogether, object orientation is a good concept for computer scientists and people who interact with them.

# Bibliography


[1] Birgit Demuth. *Softwaretechnologie für Einsteiger*. 2. geänd. Aufl. München: Pearson, 2014. ISBN: 9781784340230.

[2] Oracle. *Object-Oriented Programming Concepts*. [Online; accessed 20-April-2016]. URL: https://docs.oracle.com/javase/tutorial/java/concepts/.

[3] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The unified modeling language reference manual*. 2. ed. Boston: Addison-Wesley, 2005. ISBN: 0321245628.

[4] Al Sweigart. "Why is Object-Oriented Programming Useful? (With a Role Playing Game Example)". In: *The "Invent with Python" Blog* (Dec. 2014). [Online; accessed 20-April-2016]. URL: http://inventwithpython.com/blog/2014/12/02/why-is-object-oriented-programming-useful-with-an-role-playing-game-example/.

[5] Matt Weisfeld. "The Importance of Object-Oriented Programming in the Era of Mobile Development". In: *informIT.com* (Apr. 2013). [Online; accessed 20-April-2016]. URL: http://www.informit.com/articles/article.aspx?p=2036576.