THEORY 1

Programming Concepts

Programming is the way of writing a sequence of instructions to tell a computer to perform a specific task. A **program** is known as the sequence of instructions for a computer, therefore, a set of well-defined notations is used. The set of notations used to write a program is called a **programming language**. The person who writes a program is called a **programmer**. A programmer uses a programming language to write a program. [4]

Components of a Programming Language

Syntax

The syntax of a language defines rules how sentences of this language are built.

For instance, we can define that in a programming language an assignment consists of a variable, an assignment symbol and an expression.

```
1 Assignment = Variable "<-" Expression.
```

This syntax rule consists of a left and a right side, separated by an equal sign. The left side defines the construct of the language. The right side defines how this construct has to be built. Each syntax rule is terminated by a point.

Semantics

The semantics define the meaning of a sentence.

For instance, the rule

```
1 Assignment = Variable "<-" Expression.
```

means: Evaluate the expression and assign it to the variable.

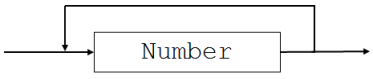
Grammar

The grammar is a set of syntax rules that describe all constructs of the language (e.g. EBNF - Extended Backus-Naur-Form).

For instance, the grammar of decimal numbers is the following:

```
1 Digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
2 Number = Digit {Digit}.
3 Decimalnumber = Number "." Number ["E" ["+"|"-"] Number].
```

Grammar rules can be symbolized in form of syntax diagrams. The rule for `Number` looks, for instance, as follows:



It is important that we can read grammars. Construct of programming languages, e.g. Java. are expressed by grammars. They define how programms have to look like.

Sign	Semantic	Example	
=	separates left and right side of rules	A = x y z .	
.	terminates rules		
|	separates alternatives	x | y	x, y
( )	parenthesizes alternatives	(x | y) z	xz, yz
[ ]	optional occurence	[x] y	xy, y
{ }	0..n-occurence	{x} y	y, xy, xxy, xxxy, ...

Programming Paradigms

The Merriam-Webster Learner's Dictionary defines a paradigm as "a theory or a group of ideas about how something should be done, made, or thought about." [2] (dt. lt. Duden: "Beispiel, Muster; Erzählung mit beispielhaftem Charakter") [1]

In the context of programming the following definition has been made by Robert W. Floyd - Turing Award winner - in 1978:

> "A programming paradigm is a way of conceptualizing what it means to perform computation, and how tasks that are to be carried out on a computer should be structured and organized."

A computer program consists of two parts: data and algorithm. A programming paradigm is used to solve problems by combining data and algorithms in different ways. Mostly used are imperative, procedural, declarative, functional, logic and object-oriented paradigms.

Imperative Paradigm

"... uses statements that change a program's state." [3] The data defines the state of the program at a particular point in time. Sequences of instructions are used to change data and state.

Suppose you want to add 10 to the integer 15. You can write a program by using an imperative language as follows:

```
int num = 15;  // num holds 15 at this point
int counter = 0;  // counter holds 0 at this point
while (counter < 10) {
   num = num + 1; // modifying data in num
   counter = counter + 1;  // modifying data in counter
}
// num holds 25 at this point
```

Examples for imperative programming languages: FORTRAN, COBOL or C.

Procedural Paradigm

It bases on the imperative paradigm, however, extends it by combining multiple commands in a unit called a procedure. The procedure is executed as a unit. The following piece of code illustrates a procedure named addTen:

```
void addTen (int num) {
   int counter = 0;  // counter holds 0 at this point
   while (counter < 10) {
      num = num + 1; // modifying data in num
      counter = counter + 1;  // modifying data in counter
   }
   //num has been incremented by 10
}
```

The procedure uses a parameter named `num` as placeholder, which has to be supplied at the time of its execution.

A procedure has side effects. It modifies the data part of the program as it executes its logic.

Example for procedural programming languages: C, C++, Pascal or Java.

Declarative Paradigm

The imperative paradigm deals with "how" a problem can be solved. In contrast to, the declarative paradigm cares about the "what" part of a problem. So, a program following the declartive paradigm consists of the description of a problem and the computer finds the solution. The functional and logic paradigm are subtypes of the declarative paradigm. SQL - the query language for databases - falls under programming based on the declarative paradigm.

```
CREATE TABLE myperson (name VARCHAR(32), address VARCHAR(32), PRIMARY KEY(name));
SELECT name, address FROM myperson;
```

Examples for declaritive programming languages: SQL (Structured Query Language)

Functional Paradigm

The functional paradigm is based on the concept of mathematical functions. An algorithm computes a value from some given inputs. Unlike a procedure in procedural programming, a function does not have a side effect. In functional programming, values are immutable *(dt. unveränderbar)*. In functional programming, a repeated task is performed using recursion.

```
int add(x, n) {
   if (n == 0) {
      return x;
   }
   else {
      return 1 + add(x, n-1); // Apply add function recursively
   }
}
```

The `add` function does not use any variable and does not modify any data but uses recursion.

Java 8 added a new language construct called a lambda expression, which can be used to perform functional programming in Java.

Examples for functional programming languages: LISP (second-oldest high-level programming language), R (statistical computing and graphics)

Logic Paradigm

In the logic paradigm, a program consists of a set of axioms and a goal statement. The set of axioms is the collection of facts and inference rules that make up a theory. The goal statement is a theorem. The program uses deductions to prove the theorem within the theory. Logic programming bases on mathematical set theory.

Suppose there are two sets `Person` and `Nationality`.

```
Person = {Roman, Anna, Celine}
Nationality = {American, Austrian, French}
```

The Cartesian product - a rule in mathematical set theory - of the two sets, such as `Person` x `Nationality`, would be another set, as shown:

```
Person x Nationality = {{Roman, American},{Anna, American}, {Celine, American},
   {Roman, Austrian}, {Anna, Austrian}, {Celine, Austrian},
   {Roman, French}, {Anna, French}, {Celine, French}}
```

Let `PersonNationality` be a relation defined as follows:

```
PersonNationality = {{Roman, Austrian}, {Anna, American}, {Celine, French}}
```

You can state the goal statement (or the problem) like

```
PersonNationality(?, Austrian)
```

which means “give me all names of people who are Austrian.”

Examples for logic programming languages: Prolog

Object-Oriented Paradigm

In the object-oriented paradigm, a program consists of objects, which are interacting.

An **object** encapsulates data (state of an object) and algorithms (behaviour of an object). Objects are communicating by sending **messages** to each other. In the object-oriented paradigm, data and algorithms are not separate, whereas , in the imperative and functional paradigms, data and algorithms are separated.

Classes are the basic units of programming in the object-oriented paradigm. Similar objects are grouped into one definition called a class. A class' definition is used to create an object. An object is also known as an instance of the class. A class consists of **instance variables** and **methods**. The values of instance variables of an object define the state of the object. Methods in a class define the behavior of its objects. A method is like a procedure in the procedural paradigm. Methods can access or modify the state of the object. A message is sent to an object by invoking one of its methods.

Suppose we want to represent real-world persons in a program. We will create a `Person` class and its instances will represent people in your program.

```java
public class Person {
   private String name;
   private String address;

   public Person(String initialName, String initialAddress) {
      name = initialName;
      address = initialAddress;
   }

   public String getName() { return name; }
   public void setName(String newName) { name = newName; }
   public String getAddress() { return address; }
}
```

The `Person` class includes three things:

- Two instance variables: `name` and `address`.
- One constructor: `Person(String initialName, String initialAddress)`
- Three methods: `getName()`, `setName(String newName)`, and `getAddress()`

A class contains the definition (or blueprint) of objects. There needs to be a way to construct (to create or to instantiate) objects of a class. Therefore, we use constructor(s).

```java
Person lydia = new Person("Lydia Corbett", "Devonshire, England");
Person leonardo = new Person("Leonardo da Vinci", "Florence, Italy");
```

You can send a message, say `getName`, to a `Person` object and it will respond by returning its name. It is the same as asking "What is your name?" and having the person respond by telling you his name.

```java
String leoName = leonardo.getName();
```

The `setName` message to the `Person` object asks him to change his current name to a new name.

```java
lydia.setName("Sylvette David");
```

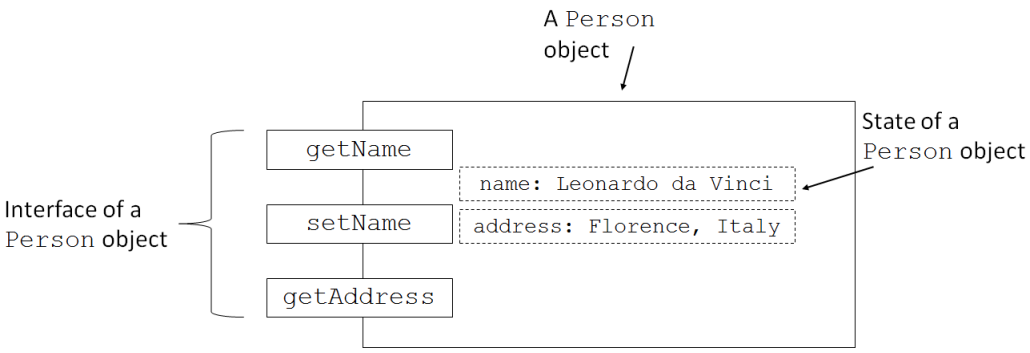


Figure 0.1: The state and the interface of a Person object

Abstraction, encapsulation, polymorphism, and inheritance are some of the important features of the object-oriented paradigm.

Examples for object-oriented programming languages: Smalltalk (is considered being a purely object-oriented programming language), Java, C#

What is Java?

Java is a multi-paradigm programming lanuage. It supports programming features on the object-oriented, procedural and functional paradigms.

Java was released by Sun Microsystems (part of Oracle Corporation since 2010) in 1995. Soon after its release, Java became a very popular programming language. One of the most important features for its popularity was its "Write Once, Run Anywhere" (WORA) feature.

Why was/is Java so popular and acceptated in software industry? Because of

- **Simplicity**. Java bases on the syntax of C++; this made it easy for learning Java for C++ programmers; however, Java excluded some of the most confusing and hard-to-use-correctly features of C++.
- **Wide variety of usage environments**. Java can be used to develop programs that can be used in different environments. Java can be used to develop applets, distributed applications, concurrent applications or mobile applications.
- **Robustness**. Robustness of a program refers to its ability to handle unexpected situations reasonably. An unexpected situation is called as error. Java provides robustness by providing many features for error checking at different points during a program's lifetime, such as comile-time error, runtime error or logic error.

The Object-Oriented Paradigm and Java

The object-oriented paradigm supports four major principles: abstraction, encapsulation, inheritance, and polymorphism. They are also known as four pillars of the object-oriented paradigm:

- **Abstraction** is the process of exposing the essential details of an entity, while ignoring the irrelevant details, to reduce the complexity for the users.
- **Encapsulation** is the process of bundling data and operations on the data together in an entity.
- **Inheritance** is used to derive a new type from an existing type, thereby establishing a parent-child relationship.
- **Polymorphism** lets an entity take on different meanings in different contexts.

Java is a programming language that supports procedural, functional, and object-oriented programming paradigms.

Do It

1. *Explain.*

   Where is the difference between a *variable* and a *value*.

2. *Explain.*

   Why do we define data types for variables. Discuss advantages and disadvantages.

3. *Grammars.*

   Given are the tokens (symbols) *x*, *y* and *z*. Which symbol sequences can be created by the following grammar rules:

   Sequence1 = x (y | z) x. xyx, xzx

   Sequence2 = [x | y] z {z}. z, zz, zzz...., xz, xzz, xzzz..., yz, yzz, yzzz...

   Sequence3 = x {y z | [x] y } z. xz, xyzz, xyzyz..z, xyz, xyy...z, xxyz, xxyxy....z

4. *Grammars.*

   Define a grammar, which describes common date formats, such as

   1. March 2016

   1. 3. 16

   2016-03-01

5. *Syntax Diagram.*

   Sketch a syntax diagram for the grammar rule as follows:

   A = x {y z | [x] y} z.

References

[1] *Duden*, 2016.

[2] *Mirriam-Webster Learner Dictionary*, 2016.

[3] *Wikipedia*, 2016.

[4] Kishori Sharan. *Beginning Java 8 Fundamentals*. Apress, 2014.