



**HOGESCHOOL ROTTERDAM / CMI**

---

# **Concept-Development 3**

**INFDEV02-3**  
2015-2016

---

Number of study points: 4 ects  
Course owners: Youri Tjang & Giuseppe Maggiore



## Modulebeschrijving

<b>Module name:</b>	Concept-Development 3
<b>Module code:</b>	INFDEV02-3
<b>Study points and hours of effort:</b>	<p>This module gives 4 ects, in correspondence with 112 hours:</p> <ul style="list-style-type: none"> <li>• 2 X 3 x 6 hours of combined lecture and practical</li> <li>• the rest is self-study</li> </ul>
<b>Examination:</b>	Written examination and practicums (with oral check)
<b>Course structure:</b>	Lectures, self-study, and practicums
<b>Prerequisite knowledge:</b>	INFDEV02-1 and INFDEV02-2.
<b>Learning tools:</b>	<ul style="list-style-type: none"> <li>• Book: Think Java; author A. B. Downey (<a href="http://www.greenteapress.com/thinkjava">www.greenteapress.com/thinkjava</a>)</li> <li>• Book: Head First Java (2nd ed.); authors K. Sierra, &amp; B. Bates. (2005).</li> <li>• Presentations: found on N@tschool and on the GitHub repository <a href="https://github.com/hogeschool/INFDEV02-3">github.com/hogeschool/INFDEV02-3</a></li> <li>• Assignments, to be done at home and during practical part of the lectures (pdf): found on N@tschool and on the GitHub repository <a href="https://github.com/hogeschool/INFDEV02-3">github.com/hogeschool/INFDEV02-3</a></li> </ul>
<b>Connected to competences:</b>	realiseren en ontwerpen
<b>Learning objectives:</b>	<p>At the end of the course, the student:</p> <ul style="list-style-type: none"> <li>• <b>is able to use and create</b> interfaces and abstract classes. (ABS)</li> <li>• <b>has developed skills</b> to adopt a new programming language with little support. (LEARN)</li> <li>• <b>is able to apply</b> the concepts of data encapsulation, inheritance, and polymorphism to software. (ENC)</li> <li>• <b>can apply</b> the concepts of data types. (TYPE)</li> <li>• <b>understands</b> basic human factors. (BHF)</li> </ul>
<b>Course owners:</b>	Youri Tjang & Giuseppe Maggiore
<b>Date:</b>	14 januari 2016



# 1 General description

Programming is one of the most ubiquitous activities within the field of ICT. Many business needs are centered around the gathering, elaboration, simulation, etc. of data through programs.

## 1.1 Relationship with other teaching units

Subsequent programming courses build upon the knowledge learned during this course.

The course analysis 3 covers UML, which is often used to demonstrate concepts in this course. Knowledge acquired through the programming courses is also useful for the projects. A word of warning though: projects and development courses are largely independent, so some things that a student learns during the development courses are not used in the projects, some things that a student learns during the development courses are indeed used in the projects, but some things done in the projects are learned within the context of the project and not within the development courses.



## 2 Course program

The course is structured into six lectures. The six lectures take place during the six weeks of the course, but are not necessarily in a one-to-one correspondance with the course weeks. For example, lectures one and two are fairly short and can take place during a single week.

### 2.1 Week 1 - statically typed programming languages

#### Topics

- What are types?
- (**Advanced**) Typing and semantic rules: how do we read them?
- Introduction to Java and C# (**advanced**) with type rules and semantics
  - Classes
  - Fields/attributes
  - Constructor(s), methods, and static methods
  - Statements, expressions, and primitive types
  - (**Advanced**) Lambda's

#### Homework

- Write an example of Python code that would cause a type error in Java/C#
- Given the following semantic and typing rules, write down how we read them; make an example code that uses them
- Write a Java/C# program featuring
  - A **Counter** class;
  - With a **count** integer attribute;
  - With an empty (parameterless) constructor;
  - With a method **Reset**;
  - With a method **Tick**;
  - (**Advanced**) With a static method/overloaded operator **Plus** which adds two counters into one;
  - (**Advanced**) With a method **OnTarget** that takes as input a lambda function which will be fired when the counter reaches a given count.

### 2.2 Week 2 - reuse through polymorphism and generics

#### Topics

- What is code reuse?
- Interfaces and implementation
- Implicit vs explicit conversion
- (**Advanced**) Implicit and explicit conversion type rules
- Runtime type testing
- (**Advanced**) Generic parameters
- (**Advanced**) Interfaces and implementation in the presence of generic parameters
- (**Advanced**) Covariance and contravariance in the presence of generic parameters



## Homework

- Write a **Vehicle** interface with a method **move** and a method **loadFuel**; **loadFuel** accepts a **Fuel** instance, where **Fuel** is an interface of your writing; **move** returns a boolean which is **true** if there is enough fuel, and **false** otherwise
- Write a concrete class **Car** and a concrete class **Gasoline** that implement, respectively, **Vehicle** and **Fuel**; the **Car** checks that the given fuel is indeed **Gasoline**
- Write a concrete class **Truck** and a concrete class **Diesel** that implement, respectively, **Vehicle** and **Fuel**; the **Truck** checks that the given fuel is indeed **Diesel**
- Write a concrete class **Enterprise** and a concrete class **Dilithium** that implement, respectively, **Vehicle** and **Fuel**; the **Enterprise** checks that the given fuel is indeed **Dilithium**
- Make a program that receives three vehicles, without knowing their concrete type, and moves them (without resorting to conversions) until their fuel is up
- (**Advanced**) Make a **List<T>** interface with methods **Length**, **Iterate**, **Map**, and **Filter**
- (**Advanced**) Define the concrete classes **Node<T>** and **Empty<T>** both implementing **List<T>**
- (**Advanced**) Make a **List<Vehicle>**, fill it with a series of concrete vehicles, and make them all move ten times



### 3 Assessment

The course is tested with two exams: a series of practical assignments, and a written exam. The final grade is determined as follows:

```
if practicumCheckOK then return writtenExamGrade else return 0
```

The written exam will include questions about the practical assignments as well as theoretical topics.

**Motivation for grade** A professional software developer is required to be able to program code which is, at the very least, *correct*.

In order to produce correct code, we expect students to show: *i*) a foundation of knowledge about how a programming language actually works in connection with a simplified concrete model of a computer; *ii*) fluency when actually writing the code.

The quality of the programmer is ultimately determined by his actual code-writing skills, therefore the written exam will contain require you to write code, this ensures that each student is able to show that his work is his own and that he has adequate understanding of its mechanisms.

#### 3.1 Theoretical examination INFDEV02-3

The general shape of a theoretical exam for INFDEV02-3 is made up of a series of highly structured open questions. In each exam the content of the questions will change, but the structure of the questions will remain the same. For the structure (and an example) of the theoretical exam, see the appendix.

#### 3.2 Practical examination INFDEV02-3

Each week there is a mandatory assignment. The assignments of week 4, 5 and 6 will be graded. Each assignment is due the following week. The sum of the grades will be the *practicumGrade*. If the course is over and *practicumGrade* is lower than 5,5 then you can retry (herkansing) the practicum with one assignment which will test all learning objectives and will replace the whole *practicumGrade*. If the *practicumGrade* is 5,5 or above then *practicumCheckOK*. The following rules apply to the assignment:

- All assignments are to be uploaded to N@tschool of Classroom in the required space (Inlevermap or assignment);
- Each assignment is designed to assess the students knowledge related to one or more learning objectives. The relevant learning objective will be stated above the assignment. If the teacher is unable to assess the student's ability based on his work, then no points will be awarded for that part.
- The university rules on fraude and plagiarism (Hogeschoolgids art. 11.10 – 11.12) also apply to code;

*The teachers still reserve the right to check the practicums handed in by each student, and to use it for further evaluation.*



## Theoretical examination INFDEV02-3

The general shape of a theoretical exam for DEV 3 is made up of a series of highly structured open questions.

### 3.2.0.1 Question 1:

Concrete example of question:

Concrete example of answer:

Points: 25%.

Grading:

Associated learning objective: ABS

### 3.2.0.2 Question 2:

Concrete example of question:

Concrete example of answer:

Points: 25%.

Grading:

Associated learning objective: ENC

### 3.2.0.3 Question 3:

Concrete example of question:

Concrete example of answer:

Points: 25%.

Grading:

Associated learning objective: TYP

**3.2.0.4 Question 4:**

**Concrete example of question:**

**Concrete example of answer:**

**Points:** 25%.

**Grading:**

**Associated learning objective:** BHF





## Bijlage 1: Toetsmatrijs

Learning goals	Dublin descriptors
ABS	1, 2, 4
LEARN	1, 4, 5
ENC	1, 2, 4
TYPE	1, 2, 4
BHF	1, 2, 4

Dublin-descriptors:

1. Knowledge and understanding
2. Applying knowledge and understanding
3. Making judgments
4. Communication
5. Learning skills