HOGESCHOOL ROTTERDAM / CMI

Concept-Development 3

INFDEV02-3
2015-2016

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

Modulebeschrijving

Module name:	Concept-Development 3
Module code:	INFDEV02-3
Study points and hours of effort:	This module gives 4 ects, in correspondence with 112 hours: • 2 X 3 x 6 hours of combined lecture and practical • the rest is self-study
Examination:	Written examination and practicums (with oral check)
Course structure:	Lectures, self-study, and practicums
Prerequisite knowledge:	INFDEV02-1 and INFDEV02-2.
Learning tools:	• Book: Think Java; author A. B. Downey (www.greenteapress.com/thinkjava) • Book: Head First Java (2nd ed.); authors K. Sierra, & B. Bates. (2005). • Presentations: found on N@tschool and on the GitHub repository github.com/hogeschool/INFDEV02-3 • Assignments, to be done at home and during practical part of the lectures (pdf): found on N@tschool and on the GitHub repository github.com/hogeschool/INFDEV02-3
Connected to competences:	realiseren en ontwerpen
Learning objectives:	At the end of the course, the student: • **is able to use** and **create** interfaces and abstract classes. `(ABS)` • **has developed skills** to adopt a new programming language with little support. `(LEARN)` • **is able to apply** the concepts of data encapsulation, inheritance, and polymorphism to software. `(ENC)` • **can apply** the concepts of data types. `(TYPE)` • **understands** basic human factors. `(BHF)`
Course owners:	Youri Tjang & Giuseppe Maggiore
Date:	24 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 analyis 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 Chapter 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
  - Arrays
  - (**Advanced**) Lambda's

Homework [1]

- 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 Chapter 2 - reuse through polymorphism

Topics

- What is code reuse?
- Interfaces and implementation
- Implicit vs explicit conversion
- (**Advanced**) Implicit and explicit conversion type rules
- Runtime type testing

[1] The solution to all homework is published on Natschool or GitHub

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

2.3 Chapter 3 - reuse through generics

Topics

- (**Advanced**) Generic parameters
- (**Advanced**) Interfaces and implementation in the presence of generic parameters
- (**Advanced**) Covariance and contravariance in the presence of generic parameters

Homework

- (**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

2.4 Chapter 4 - architectural considerations

Topics

- Encapsulation
- Abstract classes: between interfaces and implementation
- Inheritance of classes and abstract classes

Homework

- Write an `Event` abstract class or interface with a method `perform`;
- Write a `Timer` class with a method `tick` and a method `reset`; `reset` restarts the timer, while `tick` makes the timer move forward and returns whether or not the target time has been reached; when the timer reaches the target time, then fire the events in the list of timer responses
- Make a `TrafficLight` class which uses timers to implement red, green, and yellow lights;
- (**Advanced**) Rebuild timers, but this time with lambda's instead of our custom `Event`.

2.5 Chapter 5 - yet more architectural considerations

Topics

- (**Advanced**) Composition versus inheritance
- (**Advanced**) Entity/component model

Homework

- (**Advanced**) Make a `Component` interface;
- (**Advanced**) Make an `Entity` abstract class which houses a list of components;
- (**Advanced**) Write a `Car` class that inherits from `Entity` and which implements all the functionality that you would expect from a car, but with the *Entity-Component* model; you will need to build components for the engine, the wheels, etc. and all that the `Car` class does is make correct use of these components.

3 Extra homework

[2] In order to exercise on your own, you can try your hand at a series of extra, smaller assignments.

1. Make a for-loop that sums all numbers between two inputs read from the console
2. Make a (static) function for (1)
3. Make an `Interval` class that:
   - takes two integers, `l` and `u`, as its constructor parameters
   - has a `Sum` method that returns the sum of all numbers between `l` and `u`
   - has a `Product` method that returns the product of all numbers between `l` and `u`
4. Make a `Person` interface with methods (or properties with only a getter):
   - Name
   - Surname
   - Age
5. Make the `Customer`, `Student`, `Teacher` implementations of `Person`, ensuring that they all get at least three additional methods and attributes over those in (4)
6. Make a generic `Number<N>` abstract class, with methods:
   - `Zero` that returns an `N`
   - `One` that returns an `N`
   - abstract methods `Negate`, that takes an `N` and returns an `N` (for example `Negate(1)` return `-1`) - Plus, Times, DividedBy that all take two N's and returns an N
   - The non-abstract method `Minus` that makes use of `Plus` and `Negate`
   - abstract methods `SmallerThan` and `Equal`, that take two `N`'s and return a `boolean`
   - The non-abstract methods `SmallerOrEqual`, `GreaterThan`, `GreaterOrEqual`, `NotEqual`
7. Make a class `IntNumber` that implements `Number<int>`
8. Make a class `FloatNumber` that implements `Number<float>`
9. Try to make a class `StringNumber` that implements `Number<string>`: how far can you come?
10. Make the class of (3) generic with respect to the type of the parameters `l` and `u`; specifically, build a generic class `Interval<N>` which takes as input two `N`'s `l` and `u`, and also an instance of `Number<N>`

[2] The solution to all homework is published on Natschool or GitHub

4 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.

4.1 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.

4.2 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 te 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.

4.2.0.1 Question 1:

Concrete example of question:

Concrete example of answer:

Points: *25%.*

Grading:

Associated learning objective: `ABS`

4.2.0.2 Question 2:

Concrete example of question:

Concrete example of answer:

Points: *25%.*

Grading:

Associated learning objective: `ENC`

4.2.0.3 Question 3:

Concrete example of question:

Concrete example of answer:

Points: *25%.*

Grading:

Associated learning objective: `TYP`

4.2.0.4 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