HOGESCHOOL ROTTERDAM / CMI

Development 3

INFDEV02-3
2016-2017

Number of study points:	4 ects
Course owners:	T. Busker & M. Abbadi

Module description

Module name:	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 materials:	• Book: Think Java; author A. B. Downey (www.greenteapress.com/thinkjava) • Book: Core Java, Volume I - fundamentals; C.S. Horstmann & G. Cornell. • Slides: found on N@tschool and on the GitHub repository github.com/hogeschool/INFDEV02-3 • Exercises and 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)`
Course owners:	T. Busker & M. Abbadi
Date:	February 10, 2017

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 four(4) chapters. The four chapters take place during the six weeks of the course.

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

2.2 Chapter 2 - reuse through polymorphism

Topics

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

2.3 Chapter 3 - architectural considerations

Topics

- Encapsulation
- Input controllers
- State machines

2.4 Chapter 4 - yet more architectural considerations

Topics

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

3 Assessment

The course is tested by means of a written and a practicum exam: Moreover, you have to deliver (on N@tschool) a series of Assignments which will not be graded but are mandatory. The Practicum check is based on the Assignments, wereas the written exam is based on the theory introduced in the course. The final grade is determined as follows:

`if Theoretical exam-grade` $>= 5.5$ `then return Practicum check-grade else return 0`

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 short series of highly structured 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

There are 2 Assignments which are mandatory, and formatively assessed for Feedback.

- All assignments are to be uploaded to N@tschool in the required space (Inlevermap);
- Each assignment is designed to assess the students knowledge related to one or more Learning objectives. If the teacher is unable te assess the students' ability related to the appropriate Learning objective based on his work, then no points will be awarded for that part.
- *The teachers still reserve the right to check the practicums handed in by each student, and to use it for further evaluation.*
- The university rules on fraude and plagiarism (Hogeschoolgids art. 11.10 – 11.12) also apply to code;

Structure of exam INFDEV02-3

The general shape of a theoretical exam for `DEV 3` is made up of only two, highly structured open questions.

3.2.0.1 Question 1:

General shape of the question: *Given the following class definitions, and a piece of code that uses them, fill in the stack, heap, and PC with all steps taken by the program at runtime.*
Concrete example of question:

```
1  interface ICounter {
2    void Incr(int diff);
3  }
4  class Counter : ICounter {
5    private int cnt;
6    public Counter() {
7      this.cnt = 0;
8    }
9    public void Incr(int diff) {
10     this.cnt = (this.cnt + diff);
11   }
12 }
13 ICounter c = new Counter();
14 c.Incr(5);
```

Concrete example of answer:

1. Stack:

PC
1

2. Stack:

PC
13

Heap:

1
cnt=

3. Stack:

PC	...		PC	ret	this
13	...		7	null	ref 1

Heap:

1
cnt=

4. Stack:

PC	...		PC	ret
13	...		7	ref 1

Heap:

1
cnt=0

5. Stack:

PC	c
14	ref 1

Heap:

1
cnt=0

6. Stack:

PC	...		PC	ret	diff	this
14	...		10	null	5	ref 1

Heap:

1
cnt=0

7. Stack:

PC	...		PC	ret
14	...		10	null

Heap:

1
cnt=5

8. Stack:

PC	c
15	ref 1

Heap:

1
cnt=5

Points: *4 (50% of total).*
Grading: *Full points for more than 90% of correct names and values. Three points if at least all names are correctly placed on the stack with at least half the values correct. Half points for more than 40% of correct names and values. Zero points otherwise.*
Associated learning objective: is able to use and **create** interfaces and abstract classes. `(ABS)`

3.2.0.2 Question 2:

General shape of question: *Given the following class definitions, and a piece of code that uses them, fill in the declarations, class definitions, and PC with all steps taken by the compiler while type checking.*
Concrete example of question:

```
1  interface ICounter {
2    void Incr(int diff);
3  }
4  class Counter : ICounter {
5    private int cnt;
6    public Counter() {
7      this.cnt = 0;
8    }
9    public void Incr(int diff) {
10     this.cnt = (this.cnt + diff);
11   }
12 }
13 ICounter c = new Counter();
14 c.Incr(5);
```

Concrete example of answer:

1. Declarations:

PC
1

2. Declarations:

PC
4

Classes:

ICounter
Incr=(ICounter×int) → void

3. Declarations:

PC	this
7	Counter

Classes:

Counter	ICounter
Counter=Counter → Counter Incr=(Counter×int) → void cnt=int	Incr=(ICounter×int)→ void

4. Declarations:

PC	diff	this
9	int	Counter

Classes:

Counter	ICounter
Counter=Counter → Counter Incr=(Counter×int) → void cnt=int	Incr=(ICounter×int) → void

5. Declarations:

PC
13

Classes:

Counter	ICounter
Counter=Counter → Counter Incr=(Counter×int) → void cnt=int	Incr=(ICounter×int) → void

6. Declarations:

PC	c
14	ICounter

Classes:

Counter	ICounter
Counter=Counter → Counter Incr=(Counter×int) → void cnt=int	Incr=(ICounter×int) → void

7. Declarations:

c		PC	ret	arg_1	this
ICounter		14	null	int	Counter

Classes:

Counter	ICounter
Counter=Counter → Counter Incr=(Counter×int) → void cnt=int	Incr=(ICounter×int) → void

8. Declarations:

c		PC	ret	arg_1	this
ICounter		14	void	int	Counter

Classes:

Counter	ICounter
Counter=Counter → Counter Incr=(Counter×int) → void cnt=int	Incr=(ICounter×int) → void

9. Declarations:

PC	c
15	ICounter

Classes:

Counter	ICounter
Counter=Counter → Counter Incr=(Counter×int) → void cnt=int	Incr=(ICounter×int) → void

Points: *4 (50% of total).*
Grading: *Full points for more than 90% of correct names and types. Three points if at least all names are correctly placed on the declarations and classes, with at least half the types correct. Half points for more than 40% of correct names and types. Zero points otherwise.*
Associated learning objective: can apply the concepts of data types. `(TYPE)`

Appendix 1: Assessment matrix

	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