

Introduction to the course

and Java!

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se



Agenda

Introduction to the course

Course Structure

Hardware

CPU

Memory

I/O units

Software

Operating systems

Digital and binary

Applications

Development

IDE

Compilers

Additional tools

Java

A first program in Java

Software to install

Ett första projekt i IntelliJ

Introduction to the course

- ▶ A warm welcome to a first real course in programming!
- ▶ Teachers this year are:
 - ▶ Tobias Andersson Gidlund
 - ▶ Jonas Lundberg
- ▶ The name of the course is important:
Problem solving and **programming**
- ▶ It is an introduction to how problems can be solved using computers.
- ▶ But, most importantly – it is about **programming**!
 - ▶ We will use Java, but often the problem solving methods are the same no matter what language is used.

Purpose of the course

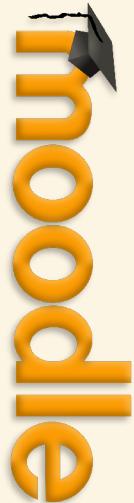
- ▶ All systems need to be *programmed*, that is created using programming code.
- ▶ The purpose of this course is to show how this is done.
- ▶ A system can be created with many different programming languages, but in many cases they have the same methodology.
- ▶ In this course the fundamental concepts in programming will be covered.
- ▶ To make this more understandable, we will be using a programming language called Java.
- ▶ Since it is not possible to know what language you will use in the future, we try to have a focus on the methodology.

Three important concepts

- ▶ This will be repeated again and again and again...
- ▶ For programming, the foundation is:
 - ▶ Sequence
 - ▶ Selection
 - ▶ Iteration
- ▶ On top of that we will put a bit of modularity using object orientation.
- ▶ Object orientation will also give us some more things to handle (and use).
- ▶ But, on top of that not much more makes up a programming language!

Moodle

- ▶ We use Moodle as the learning platform.
 - ▶ Some communication will also be done via Slack...
- ▶ You will find the room at <http://lnu.se/student>
- ▶ We use Moodle for:
 - ▶ Notices
 - ▶ Lecture notes
 - ▶ Assignments
- ▶ We use Moodle to distribute material *to* you, if you would like contact with us, please use e-mail.
- ▶ Always visit Moodle before a lecture to make sure everything is up and running.



Lectures

- ▶ The tentative scheulde is as follows:
 - ▶ Introduction about Java and the course (1)
 - ▶ Fundamental programming, sequence, input and variables (2)
 - ▶ Selection, Character and String (3,4)
 - ▶ Iteration (5)
 - ▶ Methods (6)
 - ▶ Arrays (7, 8)
 - ▶ Objects and Classes (9)
 - ▶ Object oriented thinking (10)
 - ▶ Object oriented workshop
 - ▶ File I/O and error handling (12)
 - ▶ Exam preparations (all)
- ▶ The numbers correspond to chapters in the book.

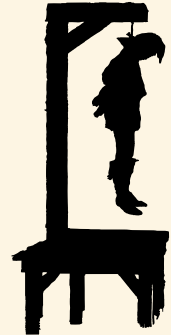
Literature

- ▶ The recommended book is “Introduction to Java Programming, Comprehensive Version”
 - ▶ Also later called “...and Data Structures, Comprehensive Version” with five cereals on the cover.
- ▶ The author is Y. Daniel Liang.
- ▶ Lectures are from 10th edition, but 11th works equally well.
- ▶ It is fairly expensive (800 – 1500 SEK depending on edition) but it will be used in the next course as well.
- ▶ See it as an investment!
- ▶ In addition, it is quite easy to read and has plenty of examples to help you.



Cheating!

- ▶ Cheating is not allowed!
- ▶ You may, and should, discuss *solutions* and *example solutions* with each others.
- ▶ BUT you may not hand in the same exact solution to a problem.
- ▶ Also notice that it just as much cheating to *hand out* your solution as it is to *take* it.
- ▶ Most of you will feel when you go too far, but if anything is unclear just talk to us!
- ▶ We will be watching you, but hope that we will go through this without cheating.



Exam

- ▶ The course ends with an exam.
- ▶ The exam will test your ability to solve problems using Java.
- ▶ It will mainly be about how to think, but there will be course writing as well.
 - ▶ We will not, however, be too hard on the syntax details.
- ▶ Several theoretical concepts will also be tested in the exam.



There will also be a *Java Test* where you during three hours are going to solve a number of practical problems based on the information given in the two first assignments.

HARDWARE

Parts of a computer

- ▶ A computer consists of a number of physical parts that work together in order to perform the user's instructions.
- ▶ It is not necessary, in this course, to understand all parts.
- ▶ What is important, though, is to see the computer as a *tool* that we can solve problems with.
- ▶ That makes it important to briefly understand how the computer works.
- ▶ The computer architecture used today has its roots in the 1950s.
 - ▶ Computers are much older than that, but the “modern” computers appeared at that time.
- ▶ The architecture is the same no matter what producer it is.

The most important parts

- ▶ The hardware can be divided into:
 - ▶ Central Processing Unit – CPU
 - ▶ In- and output units – I/O
 - ▶ Primary memory
 - ▶ Secondary memory
- ▶ For every category there may be several physical units in a computer.
 - ▶ But sometimes just one, most computers only have one primary memory.
- ▶ The different parts are connected on the *motherboard* using *data buses*.
 - ▶ The conduits and wires that connects the parts.

CPU

- ▶ A processor consist of a *control unit* and an *arithmetic unit*.
- ▶ To synchronise the work, a *clock* is used to control the pace.
 - ▶ Measured in *clock frequency* and often in GHz (Gigahertz).
 - ▶ One GHz is one billion ticks per second.
- ▶ The frequency is the *shortest* time unit for the processor, but instructions can take several of these time units to be executed.
- ▶ A higher frequency means that the computer can compute data faster.
 - ▶ It is, though, dependent on all the other part's speed, for example primary and secondary memory.

Control unit

- ▶ The task for the control unit is to collect instructions from the primary memory and then perform them.
- ▶ To do this, it has a number of *registers*.
 - ▶ Tiny, really fast memories.
- ▶ A *Program Counter (PC)* keeps track on what instruction to perform.
- ▶ The content in the instruction has two parts, one *operation part* and one *address part*.
- ▶ The operation part has an address to a *micro program* that performs a very small instruction.
 - ▶ Add, subtract, switch address or similar.
- ▶ The address part holds the memory position in the primary memory or a register where data exist.

Arithmetic unit

- ▶ Performs computations in a number of registers.
 - ▶ Arithmetic and logic, therefore sometimes the name *Arithmetic/Logic Unit*.
- ▶ The control unit's micro program performs its instructions using the ALU.
- ▶ Parts of the computation is put in the registers that the unit has.
 - ▶ Quite many, depending on the processor – often tens of them.
- ▶ In modern processors the control and arithmetic units are often put together.

Exemple

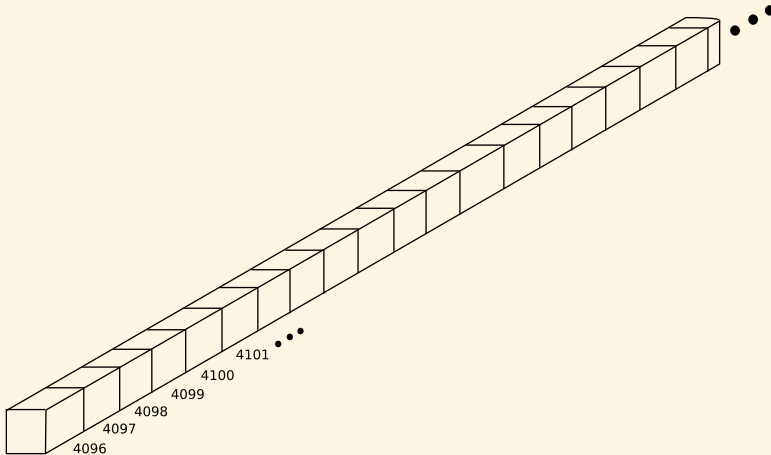
- ▶ Compute $c = a + b$.
 - ▶ This means to sum the values **a** and **b** has and put them in **c**.
- ▶ Can be computed by the following instructions:


```
mov EAX, a ;fetch a's value to the register EAX
add EAX, b ;add to b
mov c, EAX ;copy the result to c
```
- ▶ Very tedious, something you would rather not do...

Primary memory

- ▶ What every is executed at a certain point in time is put in the primary memory.
 - ▶ This includes data that is not yet saved to a secondary memory.
- ▶ The primary memory consists of a number of *cells*.
- ▶ Each cell has a unique *address* and is usually called a *byte*.
- ▶ Every byte consists of eight *bits*.
- ▶ Each bit can have the value 0 or 1.
- ▶ The registers, *memory address register* and *memory data register* is the interface between the processor and the primary memory.

Exemple



Secondary memory

- ▶ In contrast to the primary memory, the secondary memory is non *volatile*.
 - ▶ Information remains even though the program has ended and the computer is shut down.
- ▶ The primary memory is often a few gigabyte, but the secondary memory is often many times bigger.
 - ▶ Hundreds to thousands of gigabytes.
- ▶ The secondary memory is a lot slower than the primary memory.
- ▶ To speed up the experience, frequently used data is stored in a *cache* memory.
 - ▶ A lot smaller, but improves the experience.

Types of secondary memory

- ▶ There are different types of secondary memory and a computer can have several of them at the same time.
- ▶ The most common are:
 - ▶ Hard disk – disc memory
 - ▶ USB memory – semiconductor based
 - ▶ Optic memory like CD/DVD/Blu-ray
- ▶ Hard disks and optical memories save (burn) and read from rotating discs.
 - ▶ They have moving parts.
- ▶ USB memories are more like the primary memory with a small circuit board with memory cells.

von Neumann architecture

- ▶ What has been described is basically the *von Neumann architecture*.
 - ▶ After the Hungarian-American mathematician John von Neumann.
 - ▶ In reality he did not invent the model, but he was the first to describe it.
- ▶ The model is characterised of a *fetch-decode-execute* sequence which means:
 - ▶ Instructions are fetched.
 - ▶ They are decoded to see what instructions to perform.
 - ▶ Operations are executed.
- ▶ A part from that, it says:
 - ▶ Data and program is stored in a joint memory
 - ▶ Functionality is mainly done sequentially
 - ▶ One instruction at a time is executed
 - ▶ One operand (a word or similar) is computed at once

I/O units

- ▶ All units that communicate with the computer are called I/O units.
 - ▶ This includes hard disks and other internal parts as well.
- ▶ Normally, the part parts that the user is using is intended.
 - ▶ Keyboard (in unit)
 - ▶ Mouse (in unit)
 - ▶ Display (out unit)
 - ▶ Loudspeakers (out unit)
 - ▶ And many, many more...
- ▶ Today there are also several units that act as both in and out units, like touch screens.

SOFTWARE

Software

- ▶ The data and instructions that the computer computes are called *software*.
- ▶ Without software, the computer is only a rather expensive dust collector...
- ▶ Software is executable code for:
 - ▶ a specific hardware
 - ▶ a specific operating system
- ▶ It is also possible to run a som programs on a *virtual machine*.
 - ▶ Basically a software computer.
 - ▶ We will discuss this further when we talk about Java.

Categories

- ▶ Software can be categorised in many ways, with many subcategories, here is one list.
- ▶ Operating system
 - ▶ For a specific hardware
- ▶ Applications
 - ▶ Office programs
 - ▶ Word processors
 - ▶ Spreadsheets
 - ▶ Presentation programs
 - ▶ Graphics programs
 - ▶ Bitmap
 - ▶ Vector
 - ▶ 3D
 - ▶ Network
 - ▶ Web browser
 - ▶ E-mail
- ▶ Development
 - ▶ Compilers
 - ▶ Development environments

Purpose

- ▶ An operating system has two purposes:
 - ▶ Allowing for applications to interact with the hardware.
 - ▶ Manage the resources of the system.
- ▶ Before operating systems were introduced, every application had to know *everything* about the hardware.
 - ▶ Know how to put a pixel on the screen.
 - ▶ Know how data should be placed in memory.
 - ▶ Know how data should be stored on disc – and how the disc was organised.
- ▶ The resources of the system is its memory, processor, storage devices but also printers and networks and many, many more things.
- ▶ All of this must be distributed between the different programs that are executed.

What is an OS, physically?

- ▶ An OS is a program, software.
- ▶ This program is the *core* of the system.
 - ▶ Kalled the *kernel*.
- ▶ Outside the kernel layers and layers of functionality is put to make it easier to use the computer.
- ▶ Some of the functionality is managed by the OS directly.
 - ▶ Drivers for different hardware is an example.
- ▶ Some functionality is on application level.
 - ▶ Graphical user interfaces, but the distinction is getting fuzzier.

How and when is an OS executed?

- ▶ Since the OS is an important part of a system, it needs to execute first.
- ▶ Today the OS is not inbuilt in the system, so it is not really the first thing to run.
 - ▶ This was not uncommon in the 80s, though.
- ▶ When the computer is started an initialisation program is run, called *bootstrap program*.
 - ▶ That's way starting a computer is called *booting*...
- ▶ This is a relatively simple program in a memory chip inside of the computer.
 - ▶ Called *BIOS*, Basic Input Output System.
- ▶ The purpose is to initiate the computers, for example the register of the CPU, units and memory.
- ▶ BIOS is hardware dependent and must also know how to start the operating system.

The OS is starting

- ▶ The bootstrap program loads and starts the kernel.
- ▶ After that, the OS starts its first *process*.
 - ▶ Almost like a program, more about it later.
- ▶ The first process is usually *init* (except on Windows) and it in turn starts a number of other processes.
- ▶ When the OS has loaded all its processes it waits for an *event* to occur.
- ▶ An event is initiated by an *interrupt*.
 - ▶ Generated from soft- or hardware.
- ▶ Hardware sends interrupts through the system bus.
- ▶ Software sends via a special operation called *system call* or *monitor call*.

What the interrupt does

- ▶ When the CPU is interrupted, it will end what it is currently doing.
- ▶ Execution is immediately moved to a fixed position with an address to what is about to be executed.
- ▶ When the execution has ended, the CPU resumes the work it did before the interrupt.
- ▶ This is an important part of a computer's architecture and can be differently implemented in different computers.
- ▶ The most important part is that the interrupt must be managed fast, otherwise the computer will be experienced as slow.

The digital computer

- ▶ A computer can store data in *digital* format.
 - ▶ This in contrast to *analogue* format.
- ▶ Data is stored in discrete parts with a value.
 - ▶ Analogue data is stored continuously, there is no sharp border between different values.
- ▶ Quite often the process of converting from analogue to digital (and vice verse) is done.
 - ▶ For example for music on a computer.
- ▶ The number of discrete values in a time frame of the continuous signal that are stored is called the *resolution*.
 - ▶ Or *sample rate* for music.
- ▶ All information, like text, music and images, are stored digitally on a computer.

Decimal numbers

- ▶ Humans most often use the *decimal* system.
- ▶ In a number like 6061, the two “6” have different meaning.
 - ▶ The first means 6000 and the other 60.
- ▶ This kind of system uses the *base* 10.
- ▶ In a decimal system, any number should be interpreted as:

$$d_n \times 10^n + d_{n-1} \times 10^{n-1} + \dots + d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0$$
- ▶ For 6061 this means:

$$6 \times 10^3 + 0 \times 10^2 + 6 \times 10^1 + 1 \times 10^0 = 6 \times 1000 + 0 \times 100 + 6 \times 10 + 1 \times 1$$
- ▶ For several reasons a computer is not constructed for this many discrete values and only uses the *binary* system.

Binary numbers

- ▶ The binary system is much simpler, it uses two numbers: 1 and 0.
 - ▶ Just like the computer.

- ▶ This means:

$$d_n \times 2^n + d_{n-1} \times 2^{n-1} + \dots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$$

- ▶ A number like 11011 can then be converted to decimal like this:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 =$$

$$1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 16 + 8 + 2 + 1 = 27$$

- ▶ As seen, more digits are used to represent a number in binary form than in decimal form.
- ▶ This makes the binary format hard for humans, but ideal for computers.

Length to store

- ▶ A bit can store two different values: 1 or 0.
- ▶ For more information than that, for example to store the binary 27, we need 5 bits.
- ▶ A byte is eight bits and therefore a normal size to store.
- ▶ That gives in total 256 different values to store.
 - ▶ Early tables for characters were based on seven or eight bits.
 - ▶ Called ASCII – American Standard Code for Information Interchange and later Latin-1 (8 bit).
- ▶ For more characters we today use *UTF-8* with about four bytes.
 - ▶ Gives about a million different characters.
 - ▶ It corresponds to 8-bit ASCII for the first 256 characters.

Applications

- ▶ For most parts, an *application* or a *program* is the same thing as any software on the computer except for the OS.
- ▶ In a way, an application is the software that makes it possible to do something “useful” on the computer.
- ▶ It is possible to make as many categories as you like for applications.
 - ▶ Previously we have seen office, graphics and network.
 - ▶ Other possible categories could be games, databases and music.
- ▶ Almost all programs used today are *graphical* and use a *Graphical User Interface*.

DEVELOPMENT

Program development

- ▶ The third type of software is *program development tools*, that in reality are ordinary applications.
 - ▶ But with the purpose of creating new applications.
- ▶ Roughly, they can be divided into:
 - ▶ Programs to write code in, and get help to code – Integrated Development Environment.
 - ▶ Compilers that translate code into machine code.
 - ▶ Project support tools like test tools, communication tools and build servers.
- ▶ In this lecture we talk generally, with some Java at the end.

Text editors

- ▶ When a program is created, programming code is translated into machine code.
 - ▶ More about that later.
- ▶ To write code you only need a simple *text editor*.
 - ▶ Notepad, gEdit, TextEdit and so on.
- ▶ To be more effective, editors especially for programming have been developed.
 - ▶ Code completion, direct help, automatic library inclusion...
- ▶ One of the most advanced text editors for programming is *EMACS*.

Integrated Development Environments (IDE)

- ▶ To help the developer even more, the *integrated development environments* were developed.
- ▶ It helps the developer not only with coding, but also with project management, testing, GUI design and so on.
- ▶ There are IDEs for specific languages or for platforms.
 - ▶ Visual Studio – for .Net languages like C# and Visual Basic.
 - ▶ C++ Builder – for C++ and Windows.
 - ▶ QT Creator – for C++ and several platforms.
- ▶ Others are more general but with a certain focus.
 - ▶ Eclipse – for Java but it is possible to use many other languages as well.
 - ▶ KDevelop – primarily for C++ and KDE but works for other languages too.

Translation

- ▶ The part that translates the code to a language that the computer understands is called a *compiler*.
- ▶ Consists of three different parts.
 - ▶ Translation
 - ▶ Linking
 - ▶ Building
- ▶ Normally, we say that we *compile* when we create an executable file.
 - ▶ Some languages are only *interpreted*, that is – they never become an executable file.
- ▶ Every programming language and platform has its specific compiler.

Machine language

- ▶ Each processor has its own *machine language* – a number of instructions that it understands.
- ▶ Compilation translates from one high level language to the specific computer's machine language.
- ▶ Each instruction in the processor is rather limited and therefore many instructions are needed to create a program.
- ▶ The processor is really fast, so most often it does not take too long to execute a program.
- ▶ For a human, though, it is easy to mix up all the different, but small and simple, instructions.
 - ▶ It is, in other words, rather simple to make mistakes.

Levels of languages

- ▶ To make it easier for the programmer, languages are created in many *levels*.
 - ▶ Low level: Machine code and Assembler.
 - ▶ High level: Java, Ada, C/C++, C# and many, many more ...
- ▶ Assembler is relatively close to machine code and very little work is needed to translate it into machine code.
- ▶ Intel machine code for storing the value 01010101 (eight bits) in register BL is:
1011000001010101
- ▶ The same instruction in assembler, with the value in hexadecimal format is:
MOV BL, 055H

High level languages

- ▶ Even if assembler was easier to use than machine code it still needed the programmer to know about the inner workings of the computer.
- ▶ This means that every processor needs its own assembler.
- ▶ High level languages were invented to make it even easier to program.
- ▶ It also meant that it was easier to move a program from one processor to another.
- ▶ Most high level languages use the English language as a foundation, which makes them rather easy to understand.
- ▶ Several *paradigms*, that is essential concepts, are available for high level languages.
 - ▶ Imperative, object oriented, logical and functional.

Syntax

- ▶ The words and phrases that can be used are called the *syntax* of the language.
 - ▶ They also decide the order in which the words may be put.
- ▶ Many “modern” languages has inherited its syntax more or less from Algol via C.
 - ▶ Algol is short for *ALGO*rithmic *L*anguage.
- ▶ Java is also a so called *C-inspired* syntax.

Example of “Hello, World”

► Algol68:

```
begin
  printf(($gl$, "Hello, world!"))
end
```

► C:

```
main()
{
    printf("hello, world\n");
}
```

► Java:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

More examples

► Smalltalk:

```
Transcript show: 'Hello, world!'.

```

► Simula:

```
Begin
  OutText ("Hello World!");
  Outimage;
End;

```

► Ada:

```
with Ada.Text_IO;

procedure Hello_World is
begin
  Ada.Text_IO.Put_Line("Hello World!");
end;

```

Additional tools

- ▶ System development is a large and complex task.
- ▶ A large amount of tools exist to support the process to develop new programs.
- ▶ There are also many different processes to lead and manage the project.
 - ▶ Object oriented like UML with RUP or OpenUP.
 - ▶ Agile like Scrum and XP.
 - ▶ Traditional like the waterfall model.
- ▶ For each of these processes, there are a number of tools for supporting cooperation and modelling.

Git – decentralised code repositories

- ▶ Git is a distributed version control system widely used in industry.
 - ▶ Originally created by Linus Torvalds of Linux fame for kernel development.
- ▶ In real life, it is important to keep track of your code and *any changes* made to it.
- ▶ This is even more true when the project is shared between many developers.
- ▶ To reduce complexity, we will *not* introduce Git in this course, but rather wait till the next Java course.
- ▶ From next year, we will have an instance of GitLab running on campus for you to use.
- ▶ Anyone interested could start experiment with GitLab for this course, but it is by no means mandatory.

JAVA

History of Java

- ▶ Java was developed by Sun Microsystems.
- ▶ The person responsible for the developement was James Gosling.
- ▶ To begin with, the language was called *Oak* and was meant for embedded systems.
- ▶ In 1995 the language changed name to Java and was directed to the internet.
 - ▶ At the time the internet was fairly new and Sun wanted to build on that.
- ▶ To begin with Java was mostly associated with so called *applets*.
 - ▶ Small programmes embedded in web pages.
- ▶ This made Java famous and made it used in many other situations as well.

Further developement

- ▶ The first version was released in 1995 and spread fast.
- ▶ Java 2, or as it was called J2SE, was a modernisation that was released in 1998.
 - ▶ At the same time Java was expanded to fit several other use cases.
 - ▶ J2EE (Enterprise Edition) had a focus on web and business applications.
 - ▶ J2ME (Micro Edition) focused on embedded systems.
- ▶ Sun developed several versions of Java but as several other parts of Sun worked poorly, Sun Microsystems were eventually sold to Oracle Corporation in 2010.
- ▶ This led to a long pause in the development of Java from version 6 in 2006 to Java 7 in 2011.
 - ▶ Microsoft with .Net and C# grew significantly during this period.

Present day

- ▶ Even if Java 7 wasn't a large update, Oracle showed that they wanted to continue develop Java.
- ▶ In the spring of 2014 Java 8 was released with many large and modern updates:
 - ▶ Declarative programming using lambda (functional programming)
 - ▶ New library for GUI with JavaFX
 - ▶ Many minor updates to date handling, file handling and so on
- ▶ Java continues to dominate, according to TIOBE Programming Index the language is number one with 16,25%!
 - ▶ Number two is C with 16.0%
 - ▶ On top of that, Java has never been worse than in second place since TIOBE started in 2001.

What is Java used for?

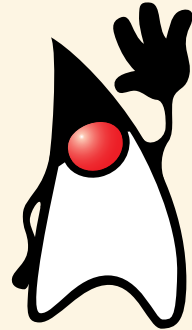
- ▶ Java is a “general purpose language” and can be used for almost anything.
- ▶ What it *can't* or should not be used for is low level programming like drivers for graphics cards and so on.
 - ▶ It might be possible, but not realistic and would probably not give any pros.
- ▶ On a higher level, Java is represented in most areas:
 - ▶ Web applications (back-end) like Netflix
 - ▶ Desktop applications
 - ▶ Apps for Android
 - ▶ Games like Minecraft
- ▶ Naturally it is still used in embedded systems like Blu-ray players and TV sets.

What are we going to do?

- ▶ What we are going to do is more on the lines of:
 How many apples do you want? 5
 How much does each apple cost? 2
 If every apple costs 2 SEK the total will be 10 SEK.
- ▶ You have to start somewhere...
- ▶ Next course will have you creating graphical user interfaces.
- ▶ But, we will start like this!

What is Java?

- ▶ Java is really two things:
 - ▶ A programming language
 - ▶ A virtual machine
- ▶ Both things are called “Java” so it is easy to mix them up...
- ▶ In this course we are mostly interested in Java as a programming language, but it is still important to understand the virtual machine.
- ▶ To the right is “Duke”, the Java mascot!



The Java virtual machine

- ▶ A *virtual machine* is kind of an abstract computer.
 - ▶ An *emulation* of a real computer.
- ▶ Instead of a Java program being compiled to a specific hardware and operating system, it is compiled to run on the Java Virtual Machine (JVM).
- ▶ A Java program is therefore compiled into *bytecode* which is run on the JVM and compiled to run on the specific platform.
- ▶ Microsoft has a similar setup with C# and .Net, where .Net is the virtual machine and C# is the language.
 - ▶ Java was, however, quite a lot earlier to use a virtual machine...

What does it mean to run on a JVM?

- ▶ The JVM takes care of everything that has to do with the specific platform.
 - ▶ How characters are shown, what a window is, how the file system works and so on.
- ▶ This means that *any* program that has been compiled to bytecode can run on all computers that have a JVM.
 - ▶ A program written and compiled on Linux can be run on a Mac.
- ▶ This is what gives Java the property of being *platform independent*.
- ▶ In the same way, any programming language that can be compiled to Java bytecode can be run on the JVM.
 - ▶ Examples of this are Kotlin, Scala and Groovy.

Safe and robust

- ▶ As everything is run in a virtual machine, Java is safer than many other programming environments.
 - ▶ The program is *sandboxed* which limits the contact it has with the “real” computer.
- ▶ This also means that Java seldom crashes the entire computer, that is – it is more robust.
 - ▶ It can, however not often, crash the virtual machine.
- ▶ There are also many features in the language itself and functionality in the virtual machine that makes it secure and robust.

The programming language Java

- ▶ Java is an object oriented language that has inherited much of its syntax from (C and) C++.
 - ▶ In contrast to C++, Java is exclusively an object oriented language – all programs are classes.
- ▶ As most programming languages today Java is *multi paradigm*, which means that it takes inspiration also from imperative programming, generic, a bit declarative and parallel programming among others.
- ▶ In this course we are going to look closer on the fundamental object oriented programming.



About versions and editions

- ▶ Java is available in several editions and in this course the Java SE edition is used.
 - ▶ SE stands for Standard Edition.
- ▶ All Java versions has an external and an internal version number.
- ▶ The latest version to date is 13.
 - ▶ Internally it is called “13.0.0”.
- ▶ Anyone who just wants to *run* Java can download the virtual machine, called the JRE.
 - ▶ Java Runtime Environment
- ▶ As we are also going to compile code, we need the developement version, called JDK.
 - ▶ Java Development Kit

Program library – Java API

- ▶ API stands for *Application Program Interface*.
- ▶ Java in itself consists of relatively few primitive functions, to do something more you need external libraries.
 - ▶ For example reading from keyboard.
 - ▶ This is *external* to Java as a programming language, but *internal* to the JVM.
- ▶ With Java comes a *standard library* from which many of the most usually used classes are collected.
 - ▶ What parts that are needed depends on the application.
- ▶ It is important to know and be able to use the API to be a good programmer.
- ▶ The Java API is available at:
<https://docs.oracle.com/en/java/javase/12/docs/api/index.html>

API

The screenshot shows a web browser window displaying the 'Overview (Java SE 12 & JDK 12)' page. The address bar shows the URL 'docs.oracle.com/en/java/javase/12/docs/api/index.html'. The page has a navigation bar with tabs for 'OVERVIEW', 'MODULE', 'PACKAGE', 'CLASS', 'USE', 'TREE', 'DEPRECATED', 'INDEX', and 'HELP'. The 'OVERVIEW' tab is selected. Below the navigation bar is a search bar with the text 'SEARCH: Search'. The main content area has the title 'Java® Platform, Standard Edition & Java Development Kit Version 12 API Specification'. Below the title, it states 'This document is divided into two sections:'. There are two sections: 'Java SE' and 'JDK'. Under 'Java SE', it says 'The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with java.' Under 'JDK', it says 'The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with jdk.' At the bottom, there is a table with four tabs: 'All Modules', 'Java SE', 'JDK', and 'Other Modules'. The 'All Modules' tab is selected. The table has two columns: 'Module' and 'Description'. The first four rows are visible:

Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.

Content of the Java API

- ▶ The API contains everything that *is* Java but not in the language itself, which is quite a lot.
- ▶ Some examples are:
 - ▶ In- and output on all devices
 - ▶ Graphical User Interfaces
 - ▶ Database management with SQL
 - ▶ XML management
 - ▶ Network communication
- ▶ The functionality in the API is available also in the JRE and can be used by all Java programs.
 - ▶ And any other program written in a language compiled to the JVM.

A FIRST PROGRAM IN JAVA

hello, world

- ▶ It is custom to create a program printing “hello, world” as first example in any language.
 - ▶ The text has later been replaced with “Hello World!”.
- ▶ This kind of program shows most of the basic constructs in a language.
- ▶ And is usually one of the smallest programs that can be constructed.
- ▶ The example with “hello, world” comes from the language C and an internal document used where it was used.
- ▶ We will show a similar example, but with another text.

“Hello World!”

- This is “Hello World!” in Java:

```

1  package lecture1;
2
3  /* A first example
4     of a Java program
5     printing a quote */
6
7  public class Quote {
8
9      // Main method
10     public static void main(String[] args) {
11         System.out.print("Violence is the diplomacy");
12         System.out.println(" of the incompetent.");
13     }
14 }

```

- The numbers to the left are *not* part of the program – we use them to be able to refer back to them in this explanation.

Executing

- ▶ Later we will discuss how to execute it.
 - ▶ In this course, we will let the IDE do it for us.
- ▶ When the program runs, it will display the following on the screen:
`Violence is the diplomacy of the incompetent`
- ▶ The program is strictly text based and no window is shown (except from the IDE).
 - ▶ In- and output is handled by the IDE in this course.
- ▶ Also not that everything is printed on the same line, more on that later.

Packages, projects and classes

- ▶ All Java programs are classes but even more different parts exist.
- ▶ This program is part of the *package* `lecture1` as shown on line 1.
- ▶ If no package is used, everything will belong the package *default*.
 - ▶ This isn't good in the long run and should be avoided.
 - ▶ Most IDE:s give you the possibility to choose (define) a package when the project is created.
- ▶ Speaking of which, each package belongs to a *project*, which is part of the IDE, not Java itself.
 - ▶ No code shows that a class or package is part of a project.
- ▶ Line 7 shows that the class is called `Quote`
 - ▶ In Java this also means that the file itself must be called `Quote.java` to be able to compile.

Block

- ▶ Each *logic* part in a program is marked as a *block*.
 - ▶ This could be a class, a method or other statements that will be discussed later.
- ▶ A block is shown with curly brackets (braces) with a { for start and } for end of the block.
- ▶ In the example the class Quote starts on line 7 (where the last character is a {).
- ▶ The class continues to line 14.
- ▶ Java code praxis says that the beginning curly bracket should be put last on the line where the block begins.
 - ▶ This is, however, only praxis and the compiler does not care at all.

Comments

- ▶ To make the code easier to understand it is a good recommendation to add *comments*.
- ▶ These comments are also important to remember what a specific part of the program does.
- ▶ Even more important is to show for others that might read the code what it does.
- ▶ In Java there are two types of comments:
 - ▶ Multi line: first line is marked with `/*` and the last line ends with `*/`
 - ▶ Single line: the line starts with `//`
- ▶ In the example there are comments on line 3 to 5 (multi line) and line 9 (single line).
- ▶ Make it a habit to comment your code!

Comment, but...

90% of all code comments:



Methods

- ▶ All action for a program is done in methods.
- ▶ All Java program that should be compiled into an executable program must have a method called `main`.
 - ▶ Projects that are larger than the first few assignments use several files, not all of them will have a `main` method.
- ▶ The line with `main` (line 10) is a *method signature* which defines what is sent to it and what might be returned from it when it is called.
 - ▶ The part just before the name is returned and the things in the parenthesis is sent to it.
 - ▶ More on this later.
- ▶ The method begins on line 10 and ends on line 13.

Statements

- ▶ In the methods there are two *statements* that actually perform some kind of action.
- ▶ A statement can be a built in command or a call to an object.
- ▶ The built in commands are usually called *reserved words* for the language itself.
 - ▶ This means that they cannot be used for anything else than what the language dictates.
- ▶ The statements used in the example (lines 11 and 12) comes from the class System that has an attribute called out that has the methods print and println.
- ▶ These methods are used for printing something to whatever is set as standard.
- ▶ Note that System isn't a part of the language, but is always available.
- ▶ Also notice that every statement is ended by a semicolon.

What the program does

- ▶ The example will print the two parts of the sentence on one line.
- ▶ With the method `println` the content is printed and then the program “jumps” to the next line.
 - ▶ The content that is printed is what is inside the quotation marks inside the parenthesis...
- ▶ The method `print` will print the content without moving to the next line at the end.
- ▶ This means that the formatting of the text isn't decided by how it is formatted in the code, but rather by which methods it is printed.
 - ▶ There are a lot of *parameters* that could be used to further decide how to format the output.

More reserved words

- ▶ Java consists of a, relatively small, set of reserved words:


<code>abstract</code>	<code>continue</code>	<code>for</code>	<code>new</code>	<code>switch</code>
<code>assert</code>	<code>default</code>	<code>goto</code>	<code>package</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>if</code>	<code>private</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>protected</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>public</code>	<code>throws</code>
<code>case</code>	<code>enum</code>	<code>instanceof</code>	<code>return</code>	<code>transient</code>
<code>catch</code>	<code>extends</code>	<code>int</code>	<code>short</code>	<code>try</code>
<code>char</code>	<code>final</code>	<code>interface</code>	<code>static</code>	<code>void</code>
<code>class</code>	<code>finally</code>	<code>long</code>	<code>strictfp</code>	<code>volatile</code>
<code>const</code>	<code>float</code>	<code>native</code>	<code>super</code>	<code>while</code>

- ▶ `const` and `goto` are reserved but not used.
- ▶ `true`, `false` and `null` are not real reserved words, but are used as such.

White space

- ▶ The space between words, identifiers, brackets and so on is called *white space*.
 - ▶ This since this space is white on a piece of paper.
- ▶ It is important that there exists *one* space between different words.
- ▶ More than that is not necessary for the compiler.
- ▶ The formatting of the code is more for the humans writing and reading it.
- ▶ “Correct” formatting does make the code a lot easier to read!

Praxis

- ▶ There are a lot of recommendations available for how to properly format Java code.
- ▶ In the book most of the most used are shown, read it and learn from that.
- ▶ It is not important to know everything, but during the course we will look at some of it.
- ▶ A mandatory part, however, is *indentation*.
- ▶ This is when each new block or logic part is “sent in” on the line with the tabulator key.
 - ▶ One press on the tab-key for each new part.
 - ▶ The key looking something like 

SOFTWARE TO INSTALL

Two things to install

- ▶ Two pieces of software need to be installed for this course:
 - ▶ A JDK
 - ▶ An IDE
- ▶ There are several alternatives for both, but we will give you some recommendations.
- ▶ The most important thing is that you need to install them as soon as possible!

Java Development Kit

- ▶ As stated previously, Java is developed by Oracle, but there are several *distributions* of the JDK.
- ▶ All of them originate from *OpenJDK* now, but we **recommend** that you use Oracle's distribution.
 - ▶ Available at: <https://www.oracle.com/technetwork/java/javase/downloads/index.html>
- ▶ Other alternatives are “pure” OpenJDK or one of the following:
 - ▶ Adopt OpenJDK
 - ▶ RedHat OpenJDK
 - ▶ Corretto (from Amazon)
 - ▶ Zulu
- ▶ In practice it doesn't matter which one you choose as long as it is at least version 11.

Compiling Java

- ▶ In the JDK only the terminal based compiler (javac) is included.
- ▶ This means that the code needs to be written in a text editor and compiled in a terminal (command prompt).


```
tanmsi@vader:~/Filer/src$ javac Quote.java
tanmsi@vader:~/Filer/src$ java Quote
Violence is the diplomacy of the incompetent.
```

- ▶ For larger projects this will be hard to keep track of, so some sort of *Integrated Development Environment* is necessary.
 - ▶ Or, as we have seen, IDE for short.

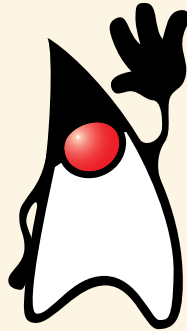
IDEs

- ▶ Two options are recommended for IDE:
 - ▶ IntelliJ IDEA (<https://www.jetbrains.com/idea/download>)
 - ▶ Eclipse (<https://www.eclipse.org/downloads/>)
- ▶ Both are competent and stable, IntelliJ has stronger commercial support while Eclipse is entirely open source and free.
 - ▶ As student you have access to IntelliJ IDEA Ultimate, the full version of IDEA.
- ▶ For Eclipse, use the standard edition (called IDE) and not Enterprise.
- ▶ Other alternatives are NetBeans and VS Code, but today we recommend one of the other two.

Getting stated

- ▶ To get started with either IDE follow a tutorial.
- ▶ IntelliJ:
 - ▶ Follow from “Create a new project” to “Build and run the application”.
 - ▶ <https://www.jetbrains.com/help/idea/creating-and-running-your-first-java-application.html>
- ▶ Eclipse:
 - ▶ <https://www.wikihow.com/Run-Java-Program-in-Eclipse>
- ▶ There are also numerous tutorials on  for how to use both IDEs.
- ▶ You will also be able to discuss details during the lab sessions.
- ▶ After this lecture, you will find step-by-step instructions for IntelliJ (only in the slides).


Now let's go!



A first project, step-by-step

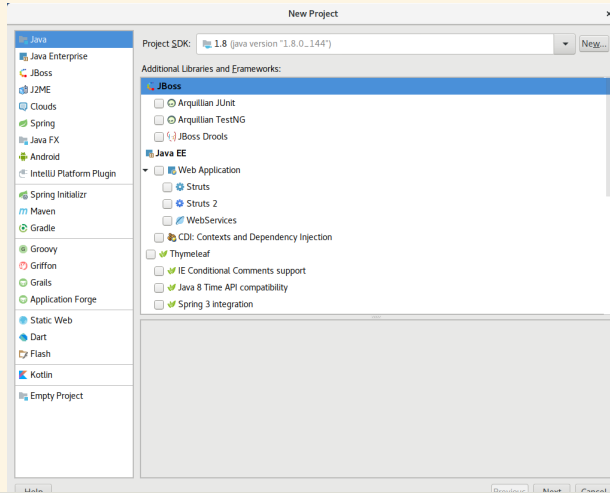
- ▶ Select File ▶ New ▶ Project or alternatively *Create New Project* on you are on the “front page”.
- ▶ On the left side, see to that *Java* is selected, nothing on the right side needs to be selected. Press *Next*
- ▶ In the dialogue *Template*, do not select anything – just press *Next*
- ▶ Name the project something good (more on this later) and press *Finish*
- ▶ In the hierarchy on the left there is now a project name that can be selected.
 - ▶ Beneath it are two folders, one called *.idea* and one called *src*
- ▶ Select *src* and right click.

A first project, cont.

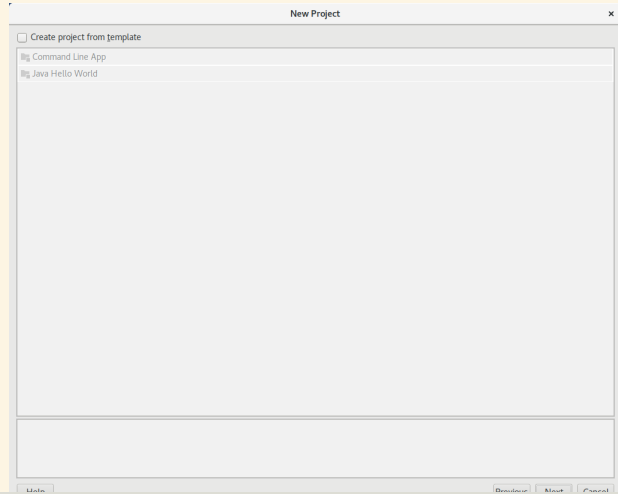
- ▶ Select New ▶ Package and type a name (more later).
- ▶ Select the package and right click for New ▶ Java Class and give it a name.
- ▶ Create some space after the first curly brace and type `pswm` followed by tab ().
 - ▶ A new method called *main* will be created.
- ▶ Where the cursor stands, write:


```
System.out.println("Hello World");
```
- ▶ To run the program, select it on the right side and right click – select *Run ProjectName.Main()*.
- ▶ Once this has been done one time, you can use the green arrow up to the right.

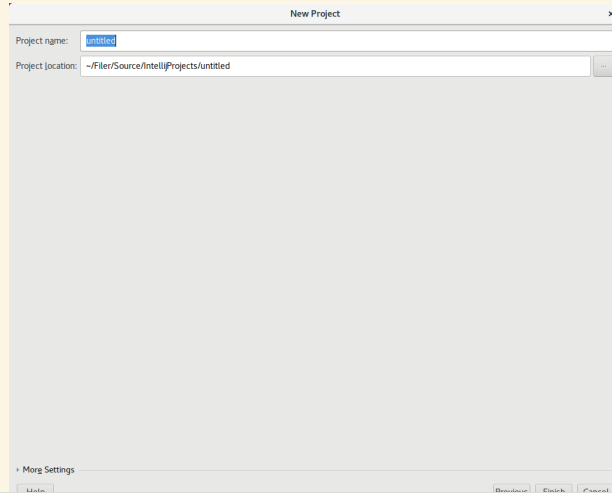
New Java Project



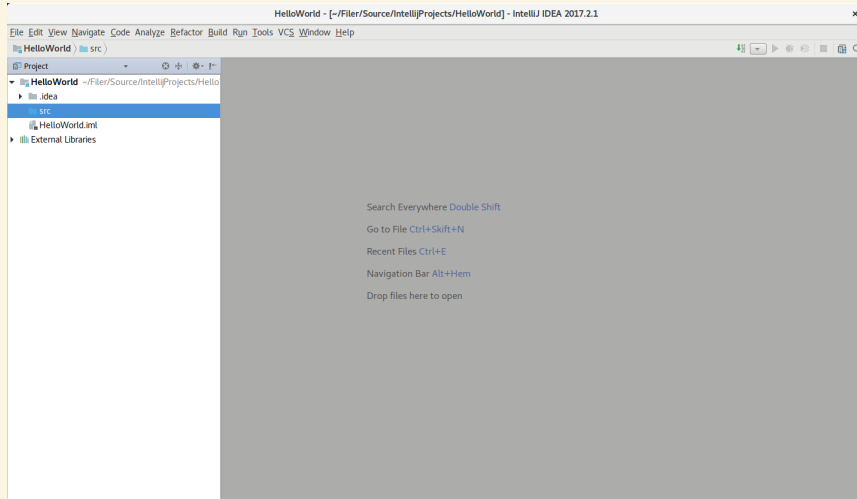
No template



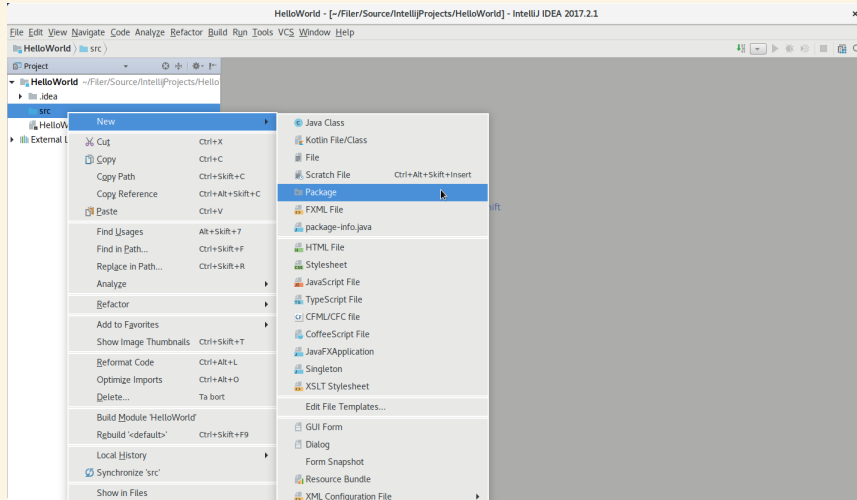
Project name



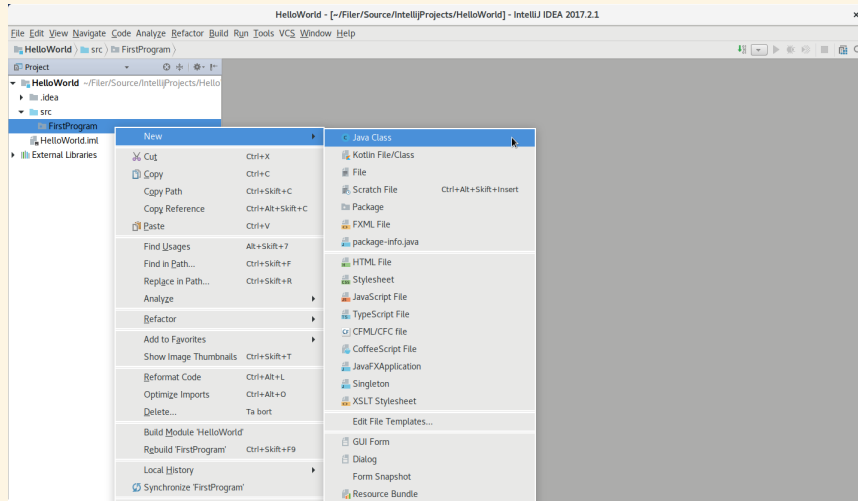
Project structure



Package



A Java class



An empty Java class

IntelliJ IDEA 2017.2.1

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

Project: HelloWorld - [~/Filer/Source/IntelliJProjects/HelloWorld] - [HelloWorld] - ~/Filer/Source/IntelliJProjects/HelloWorld/src/FirstProgram/HelloWorld.java

Project Structure:

- HelloWorld
 - .idea
 - src
 - FirstProgram
 - HelloWorld.iml
 - External Libraries

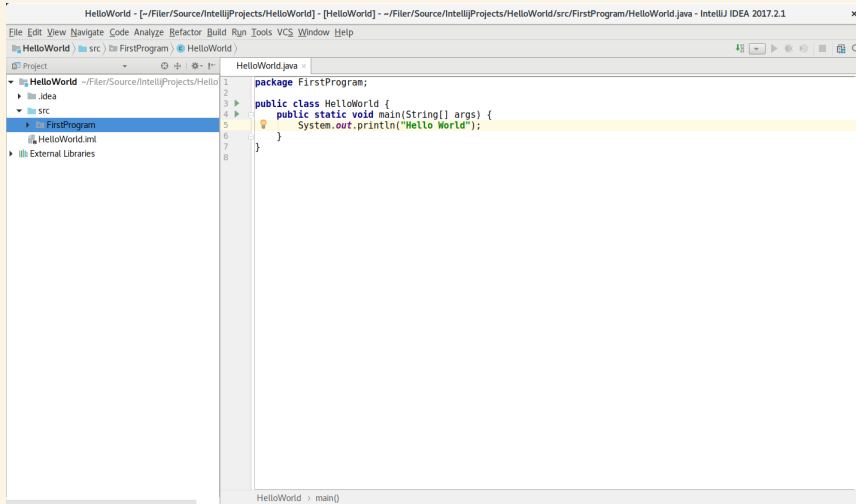
Code in HelloWorld.java:

```

1 package FirstProgram;
2
3 public class HelloWorld {
4 }
5

```

The finished class



IntelliJ IDEA 2017.2.1 window showing the 'HelloWorld.java' file. The project structure on the left shows 'HelloWorld' with subfolders '.idea', 'src', and 'FirstProgram'. The 'HelloWorld.java' file is open, showing the following code:

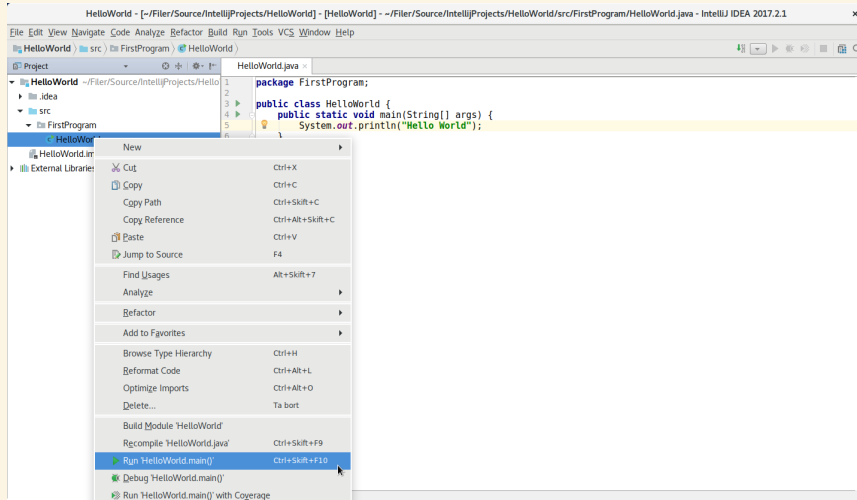
```

1 package FirstProgram;
2
3 public class HelloWorld {
4     public static void main(String[] args) {
5         System.out.println("Hello World!");
6     }
7 }
8

```

The status bar at the bottom indicates 'HelloWorld > main()'.

Compiling and running



The program is running

The screenshot shows the IntelliJ IDEA 2017.2.1 interface. The top toolbar includes buttons for Run (a green play icon), Debug (a blue bug icon), and other development tools. The main editor displays the following Java code:

```

1 package FirstProgram;
2
3 public class HelloWorld {
4     public static void main(String[] args) {
5         System.out.println("Hello World");
6     }
7 }
8

```

The 'HelloWorld' class is selected in the Project view on the left. The Run console at the bottom shows the output of the program:

```

Run HelloWorld
/usr/lib/jvm/java-8-oracle/bin/java ...
Hello World
Process finished with exit code 0

```

Naming

- ▶ The project name is on the highest level and could be named after this course.
 - ▶ That is, possibly 1DV506
- ▶ A project can contain multiple packages, for example one for assignment 1, one for assignment 2 and so on.
 - ▶ Also create packages for your own exercises.
- ▶ The classes are the individual parts of your program.
 - ▶ This could be task 1, task 2, task 3 and so on assignment 1 and likewise for assignment 2 and so on.
- ▶ You cannot use space in any of the names.
- ▶ Also, do not use foreign characters like åäö in any names.