

# CSEN 202 – Introduction to Computer Programming

## Lecture 1: Introduction and overview

Prof. Dr. Slim Abdennadher and  
Dr Mohammed Abdel Megeed Salem  
`slim.abdennadher@guc.edu.eg`

German University Cairo, Faculty of Media Engineering and Technology

February 10/15, 2018

# Structure of this class

- **Lectures.** Divided into 6 groups
  - **Sunday** 1st and Thursday 2nd slots  
**Dr. Mohammed Abdel Megeed**
  - **Sunday** 2nd and Monday 2nd slots.  
**Prof.Dr. Slim Abdennadher**
  - **Thursday** 3rd and 4th slots.  
**Prof.Dr. Slim Abdennadher**
- **Exercises and Homework.**
  - Practical assignments
  - Individual work with feedback and advice from your teaching assistant
- **Labs.**
  - Supervised lab assignments
  - Individual work during the labs

# Grading

**Tentative** weights of the assessments:

■ Final exam	40%
■ Mid-term exam	25%
■ Quizzes	25%
■ Lab Assignments	10%

# Survival guide

How to **succeed** in this course:

## ■ Hang in!

- **Attend** lectures, tutorials, and labs, take **notes**
- Participate in the discussions, be **active**
- **Solve** the assignments, understand the model solutions provided

## ■ Master the infrastructure!

- Learn how to **operate** the **software** (editor, IDE)
- **Understand** the **interaction** (input/output, error messages)
- **Contribute** within the team
- Install **relevant software** on your **own** computer/laptop

Remember that this is an introduction to **computer programming**

# Survival guide

How to **succeed** in this course:

- Do not fall behind!

- Regularly **check** the **course website** for announcements, updates, material, resources

**<http://met.guc.edu.eg>**

- **Ask** your TA (during the tutorial / office hours), professor (lecture)

# Motivation

## Why should you take CSEN 202?

- **Improve** your **problem solving skills** (clarity, precision, logic, *etc.*)
- To use computers for **problem solving**
- Acquire new skills that will allow you to **create** useful and customized computer-based **applications**
- It is in the **curriculum**
- **Acquire** a useful **vocabulary** that will **impress** others in **geeky conversations**

What you will learn (and why)

# Course outline

- Introduction to Java
- Fundamental Data Types
- Decisions
- Iteration
- Methods
- Recursion
- Classes and Objects
- Arrays

# Today's topics and objectives

- Background
  - Problem solving
  - Programming languages
- Java
  - Introduction and history
  - Features and constructs
- A first program
  - Getting started with Java
  - Some aspects of Java



# So what is problem solving?

The **purpose** of writing a program is to **solve a problem**

The general steps of problem solving are

- **Understand** the problem
- **Dissect** the problem into **manageable pieces**
- **Design** a **solution**
- Consider **alternatives** and **refine** it
- **Implement** the solution
- **Test** the solution and **fix any problems** that exist

# “Divide and conquer”

- Many software projects **fail** because the developer did not really understand the problem to be solved
- We must **avoid assumptions** and **clarify ambiguities**
- As problems grow larger, we need to organize the development of a solution in **manageable pieces**:  
“Divide and conquer”
- This technique is **fundamental** to software development. We will see how Java supports this approach.

# Problem solving using a programming language

- A **programming language** specifies the words and symbols that we can use to write a program.
- A programming language employs a **set of rules** that dictate how the words and symbols can be put together to form valid **program statements**.
- **Examples** of Programming Languages:
  - Fortran
  - Cobol
  - C
  - C++, C#
  - Pascal
  - Prolog
  - **JAVA**

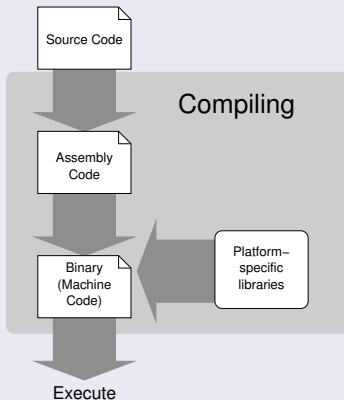
# Levels of programming languages

There are **four** programming languages levels:

- **Fourth-generation language** — Higher order programming language that introduces **complex abstractions** and **concepts**
- **High-level language** — Programming language with **convenient programming constructs** (Somewhat platform independent, compiler-specific)
- **Assembly language** — Somewhat human-readable **mnemonics** that encode machine language (hardware-platform specific)
- **Machine language** — A **byte-stream** that can directly be processed by computing hardware (hardware-platform specific)

# The normal compilation procedure

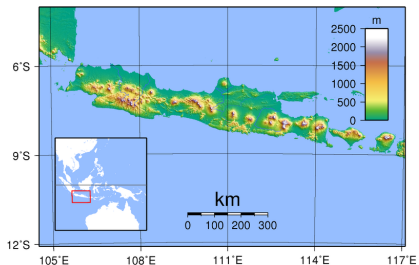
## Translating/compiling a program (simplified)



# What is Java?

# What is Java?

It's an island of **Indonesia**



They grow a lot of **coffee** there.

# What is Java?

It is the US-American **slang** term for **coffee**



Programmers run on **coffee**.



# What is Java?

It is the **name** of a **programming language**, because programmers **love** coffee.



**Seriously.**

# Origin of Java

- Began in 1991 with **Green Team** at **Sun Microsystems** in Menlo Park, CA
- Initial title was **OAK** (Object Application Kernel)
- The initial goal was the development of a programming language for **embedded devices** (e. g., toaster, coffee machine, VHS recoder, ...)
- **Java** created in 1992 by **James Gosling**, **Patrick Naughton**, and **Mike Sheridan**.
- Digital TV applications failed to generate business
- Focus turned to the **Internet**
- **New goal** was a **general purpose language with an emphasis on portability and interpretation**

# History of Java

Java was released in 1995

- C functionality
- Object Oriented (OO) capabilities
- Other nice features (*e. g.*, garbage collection)
- **Advantages:**
  - Simple for an OO language
  - Secure and reliable
  - Platform independent: will work on any processor that has a Java interpreter—Java Virtual Machine
  - Extensive libraries (esp. graphics & WWW)
- **Disadvantages:**
  - Slower than C (more overhead)
  - Limits user ability

# Hello world

## The first Java program

```
public class Hello {  
    public static void main (String[] args) {  
        // display a greeting in the console window  
        System.out.println ("Hello, World!");  
    }  
}
```

- This code defines a **class** named `Hello`.
- The definition must be in a file `Hello.java`.
- The **method** `main` is the code that runs when you execute the program

# Building and executing Java code

- Source file name must end in “.java”
- Source file name must match the name of the public class
- A **Java Development Kit (JDK)** must be installed to compile and run the programs
- **Compiling** to produce `.class` file
  - `$ javac Hello.java`
- **Running** in the JVM environment
  - `$ java Hello`
- Notice the lack of `.class` extension

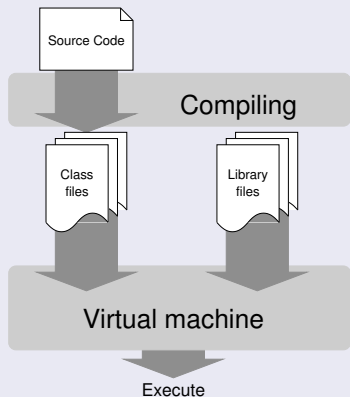
# The Java compilation and execution

## Note:

- The **Java compiler** does **not** produce **machine code**.
- The **Java compiler** produces byte code for the **Java Virtual Machine (JVM)**.
- The **JVM** for a platform reads **byte code** and executes it on that platform at run time.

# The Java compilation and execution

## Translating/compiling a Java program (simplified)



# Concepts that we will handle later

```
public class Hello {  
    public static void main (String[] args) {  
        // display a greeting in the console window  
        System.out.println ("Hello, _World!");  
    }  
}
```

- **public class ClassName**: `public` denotes that the class is usable by the “public”.
- **public static void** `main(String[] args)`: defines a method called `main`.
- The parameter `String[] args` contains the command line arguments



# Concepts that we will handle later

```
public class Hello {  
    public static void main (String[] args) {  
        // display a greeting in the console window  
        System.out.println ("Hello, _World!");  
    }  
}
```

- The keyword **static** means that `main` does not inspect or change objects of the `Hello` class.
- The terminal window is represented in Java by an object called `out`.
- The `System` class contains useful objects and methods to access system resources.

# Concepts that we will handle later

```
public class Hello {  
    public static void main (String[] args) {  
        // display a greeting in the console window  
        System.out.println ("Hello, World!");  
    }  
}
```

- To use the `out` object in the `System` class, we must refer to it as `System.out`.
- The `println` method will print a line of text.

# Identifiers

- Names in programs are called **identifiers**.
- Identifiers
  - Always start with a letter.
  - Can include, digits, underscore and the dollar sign symbol.
  - Must be different from any Java reserved words (or keywords).  
Keywords that we have seen so far include: **public**, **static**, **class**, and **void**.
  - Are case-sensitive, for example `foobar`, `FooBar`, and `FOOBAR` are all different.
- We should try to use **descriptive names**.

# Comments

To make our code understandable, we **comment** sections whose purpose is not immediately obvious.

- First kind of comments:

```
/* This is one kind of comment  
that can span several lines. Don't  
forget to put the closing  
characters at the end. */
```

- Second kind of comments:

```
// This is the other type of comment.  
// It covers the entire line  
// and requires a new set  
// of slashes for each new line.
```

# Comments

To make our code understandable, we **comment** sections whose purpose is not immediately obvious.

- Third kind of comments:

```
/** This is a javadoc comment. It  
    also spans a several lines. */
```

# Errors

- **Syntax errors:** Detected by the compiler
  - `System.ouch.print("Hello");`
  - `System.out.print("Hello);`
- **Logic errors:** Detected **hopefully** through testing
  - `System.out.print("Hell");`
- **Runtime errors:** Detected by the JVM when it is **too late**
  - `System.out.print(1/0);`