



Faculty of Computer Science


Programming Language 2
Object oriented design using JAVA

# Introduction

# Unit 1: OO Programming

# Dr. Ayman Ezzat

**Email: ayman@fcih.net**

**Web: www.fcih.net/ayman**


**PhD (Univ. of Tsukuba, Japan)**

*Major* CS

*Specialization* Human Computer Interaction

**MSc (Helwan University , Egypt)**

*Major* CS

*Specialization* Software Engineering

**BSc (Helwan University)**

*Major* CS

*Specialization* CS

# Welcome!

- **In this course, you will learn:**
  - how Java originated and how it is used nowadays.
    - Java is the language most identified with the internet and its friendly face, the web.
    - Java was introduced by **Sun Microsystems**.
  - the language used to build systems that power so much of the web in applications ranging from e-commerce to online games, downloadable music to online banking and mobile applications.

# Before you start

- **Course materials:**
  - **12 units of study text** (units 11 and 12 are optional)
  - We can work with Netbeans/Eclipse
    - https://netbeans.org/downloads/
    - https://eclipse.org/downloads/
    - JDK http://www.oracle.com/technetwork/java/javase/downloads/index.html
  - Project **Phase 1 (40%)**-> Last lecture before mid term.
  - Project **Phase 2 (60%)**-> Last lecture before final exam.

# The course at a glance

- The course has 12 units:
  - Unit 1: Java everywhere
  - Unit 2: Java in the small
  - Unit 3: Java in the large
  - Unit 4: Input, output and exceptions
  - Unit 5: Packages and abstraction
  - Unit 6: Graphical user interfaces
  - Unit 7: Event-driven programming
  - Unit 8: Threads
  - Unit 9: Internet programming
  - Unit 10: let's do Java
  - *Unit 11: Web-based case studies (Optional)*
  - *Unit 12: Web-based case study(Optional)*

# How to pass

OO is an intensive course that requires significant effort from students. Here are some guidelines to help you pass the course:

- **Study**
  - Always study what has been given in a lecture asap.
- **Revise**
  - Always revise what you have studied in previous units before coming to the lecture.
- **Prepare**
  - Before coming to the lecture, skim through the contents of the unit(s) to be given in that lecture. Use the study calendar to help you in that matter.
- **Practice.. Practice.. Practice!!**
  - Don’t just read or memorize, practice what you have learned and watch your skills improve. Practicing also helps you understand and memorize! You may use Java NetBeans for that purpose.

## Introduction

- Welcome!
- Before you start
- Course Assessments
- The course at a glance

## Unit1: Java Everywhere

1. Java background
2. Getting Java running
3. Objects and state changes
4. Classes in Java
5. Inheritance

- Unit1: Java Everywhere

# 1. Java background

- Java runs on servers, PCs, mobile phones and PDAs (personal digital assistants).

- It can be found in
  - TV set-top boxes,
  - in embedded devices (forming part of larger systems such as cars, robots or printers),
  - in smart cards, and
  - wearable computers.

  Java really is everywhere.

# 1. Java background

**The aims of the Java language**

- Simple to learn, & closely based on C++, but reduced in complexity.
- Object-oriented
- Robust: Java programs are strictly checked by software before they run
- **Secure**: Java ensures that programs running **over networks** cannot damage your computer files or introduce viruses.
- **Portable**: Java programs can easily be transferred from one platform (e.g. Windows) to run on another platform (e.g. Linux) with little or no change.
- High performance (fast)
- Interpreted: a key aspect of Java portability (more in Section 3).
- **Threaded**: allow a program to do several things at once.
- Dynamic: Java programs can adapt to **changes in their environment** even while the program is running.

- Unit1: Java Everywhere

# 1. Java background

**Versions of Java**

- The first publicly available version was Java 1.0.
- Now, Java 2 is the standard language
  - and there have been a number of further versions (Java 2 version 1.3, Java 2 version 1.4, Java 2 version 1.5 and so on), each introducing relatively minor improvements

**Editions of Java**

- Why Java editions was introduced?
  - to cater for the different needs of, say, large international business systems to software running on mobile phones with very limited hardware resources.
- Curent editions:
  - **Java 2 Standard Edition (J2SE)** → focus of our study!
  - Java 2 Enterprise Edition (J2EE) → for large-scale systems
  - Java 2 Micro Edition (J2ME) → for mobile phones (unit 10)

# 1. Java background

**A simple Java program**

```
public class HelloWalrus
{
    public static void main (String[] args)
    {
        System.out.println("The time has come");
        System.out.println("the walrus said");
    }
}
```

**The output:**

```
The time has come
the walrus said
```

# 1. Java background

- The first line of the program introduces a **class**, called HelloWalrus.
  - When the program runs, the class gives rise to an object, which executes the code of the class.
  - This class is defined as **public**, which means that objects of this class can be **freely accessed** by other objects in a larger system.
  - The **curly bracket '{'** on the line following HelloWalrus matches the closing bracket '**}**' at the end of the example, as these enclose the whole class definition.

- The code in this class is in the form of a **method**, in this case a method called **main**.
  - The first line of the method is called the **method header**.
  - We defer explaining the words **static** and **void** until later, but they are **always used with main methods** like this one.
  - The method header defines the name of the method (in this case, main) and is followed by the **method body**, which is enclosed in curly brackets.

- The **System.out.println** method displays, in a screen window, the exact text between the quotes.
  - Each of these two lines of code constitutes **a statement**, **a single command to be carried out**.

- Each statement is terminated by a **semicolon ';'**.

# 1. Java background

**Using comments in Java programs**

- A comment is ignored by the computer system
  - **Form 1: Block Com**

```
/* A simple Java program                          d */
   Author: The Course Team
   Date of creation: 01/01/01 */
public class HelloWalrus
...
```

  - **Form 2: Line Comment** *or* **In-line Comment** (using //)
    - Line comment:

```
// display a poetic message
```

    - In-line comment:

```
} // end of class HelloWalrus
```

## Introduction

- Welcome!
- Before you start
- Course Assessments
- The course at a glance

## Unit1: Java Everywhere

1. Java background
2. **Getting Java running**
3. Objects and state changes
4. Classes in Java
5. Inheritance

• Unit1: Java Everywhere

# 2. Getting Java running

Here we discuss **the portability** issue-> the way that programs written in Java can be run on many different platforms.

### The conventional way

- **source code** → is translated by a **compiler** to → **native code** (the base language of the computer) → which is then executed.
- The native code that has been generated can be executed only by the particular type of computer that recognizes it.

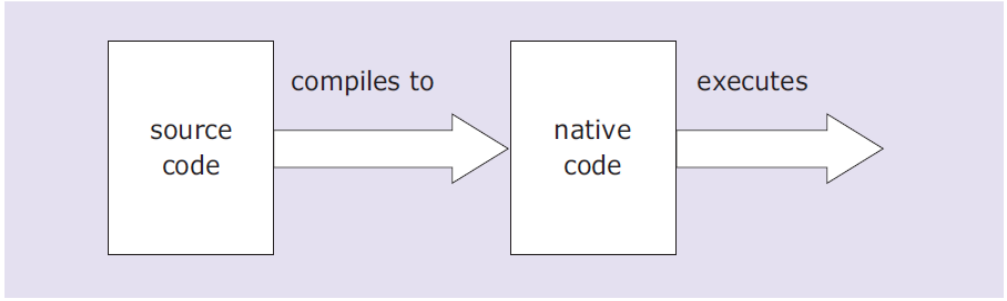



Figure 1 The action of a standard compiler

### In the case of Java

- **source code** → is translated by a **compiler** to → **bytecode** (intermediate form) → which is translated by an **interpreter** into the **native code** of the computer it runs on → which is then executed.
- The interpreter is a program that translates bytecode into the native code of the computer it runs on
- Java interpreters are available for many different platforms.

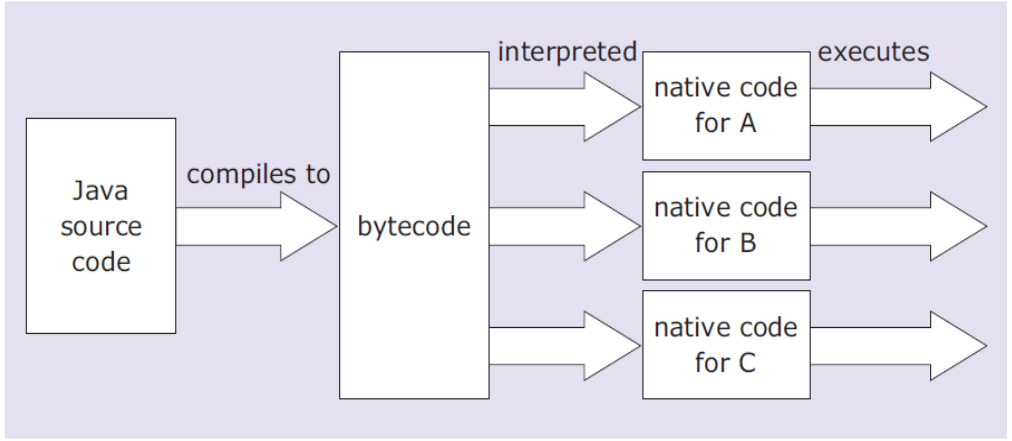



Figure 2 The Java compilation process showing interpretation of bytecode on platforms A, B and C

# 2. Getting Java running

**The Java Software Development Kit (**abbreviated as **SDK** or **JDK)**

- SDK tools are used for compiling and running Java programs on a variety of platforms
- SDK tools are freely available from Sun Microsystems
- SDK tools include:
  - **The Java compiler** (javac) translates Java source into bytecode.
  - **The Java interpreter** (java) translates and executes Java bytecode. Java interpreters are available for many different computer systems.
  - **The document generator (javadoc)** processes Java source files to produce useful documentation, such as standardized descriptions of components of Java code.
  - **The Java debugger (jdb)** helps to look for errors in programs.
  - **The Java disassembler (javap)** reads bytecode files created using the Java compiler and displays the corresponding source code.
  - **The applet viewer (appletviewer)** allows you to run and debug Java applets without a web browser (Java applets are explained in the next section)

# 2. Getting Java running

- Java programmers can produce two types of software:
  1. **Applications** are stand-alone programs, which can run independently.
  2. **Applets** are programs that can be included in web documents and are run using a browser.

- Two ways to develop and run Java software:
  - To use **Java SDK** along with a **text editor** (mostly run from command line)
  - To use an **integrated development environment (IDE)**,
    - such as the tool supplied with this course.
    - IDE provides a sophisticated GUI and editor, with integrated facilities for compiling, running and debugging programs.
    - There are many IDEs available for Java, and they typically build on top of the free software from Sun.

## Introduction

- Welcome!
- Before you start
- Course Assessments
- The course at a glance

## Unit1: Java Everywhere

1. Java background
2. Getting Java running
3. **Objects**
4. Classes in Java

# Thinking in Objects

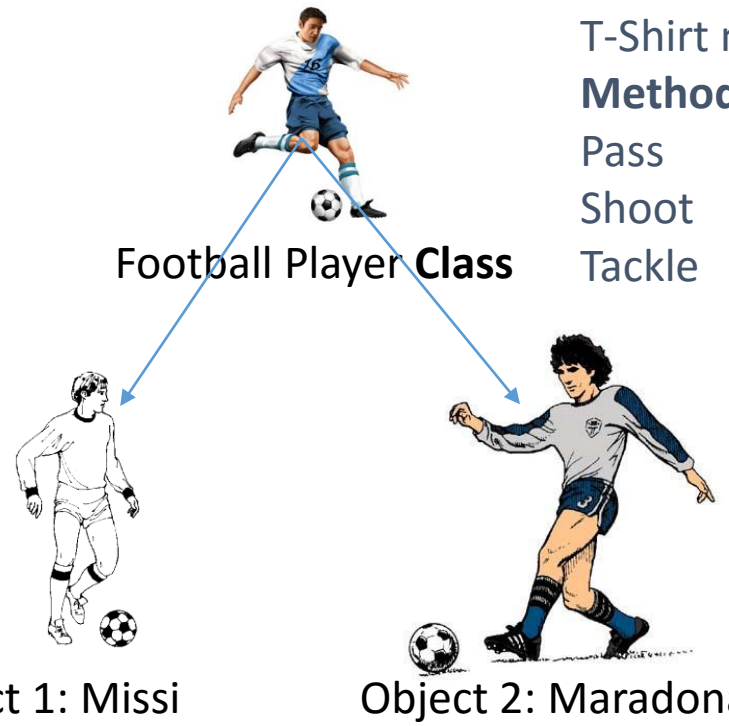
- Object oriented starts on several concepts
  - Never repeat your code.
  - Encapsulation.
  - Data Hiding.

- Data hidden inside capsules, you only know effective chemicals inside (Data types) and you know  when/how to use it (functions).

- Any object can be represented by knowing it’s characteristics and it’s usage (properties and methods)

# Classes and struct

- Any object must have a global big category that can instantiate several objects from this category, they are called Classes
- Struct in Ansi C and classes have many several **common** thing.
- Both share the concept of Encapsulation.
- Struct and classes can embed inside **functions**.
- Struct has no constructors
- Struct has no relationships except assisting
- Struct has no data hiding.



# Struct basics

Declaring Struct (Category for Students)

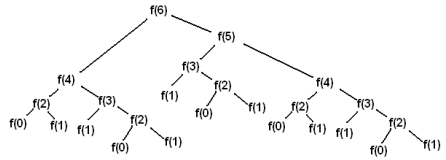
```
struct student
{
    int id;
    int age;
    string name;
    void initage()
    {
        age = 20;

    }
    student *x;
};
```

Pointer names x going to point to some address in memory for object of type student

- Encapsulate different data types inside new data type called struct.
- C now understands that whenever you write student x, it means create a space in memory to store (id, age, name and pointer x)
- Building the category class/struct is called Library development.

Library developers



algorithms

Core Developers

scenarios

Interface Developers

Unity, directx

# Consuming struct/class

- Whenever you create object from your class it is called consuming the library and hence you start writing your scenario (main program)
- You can fill the object with data.
- You can use pointers to objects
- Use “.” if you call objects directly
- Use “->” if you call object from pointer.
- Pointer must point to address of object before usage.

```
void main()
{
    student Ali;
    student Philip;
    Ali.x = &Philip;

    Ali.id = 2014046;
    Ali.age = 61;
    cout << Ali.age;
    Ali.initage();
    cout << Ali.age;
    student *Amr;
    Amr = &Ali;
    cout <<"\n Amr pointer age is "<< Amr->age;
    Amr = &Philip;
    cout << Amr->age;
```

# Static Versus dynamic binding

- Now object Ali, Philip are called static objects.
- They can never removed from memory unless the program terminated.
- Place for objects calculated in compilation time

```
void main()
{
    student Ali;
    student Philip;
```

- New operator introduce dynamic object, it was replacement for Malloc
- Work on the run time
- Return null if no space in memory
- Now object can be removed by
  - Ending program
  - Delete operator
  - Garbage collector (if you forget delete)

```
student *p = new student();
p->id = 5;
p->age = 50;
delete(p);
p = &Ali;
```

# Introducing Java

- No pointer anymore
- Dynamic binding always used with new operator.
- Always use classes as it was written in object oriented
- Very similar syntax of c++