

Object Oriented Programming

Prepared by Dr. Yasser Abdelhamid

IT223
Object Oriented Programming

Lecture 01



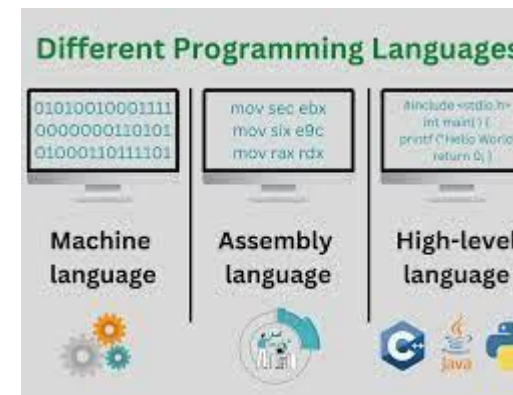


OUTLINE

- ◉ Programming languages
- ◉ Compilers and Interpreters
- ◉ Programming paradigms
- ◉ Introduction to object technology
- ◉ Object Oriented Analysis and Design (OOAD)
- ◉ Java programming language

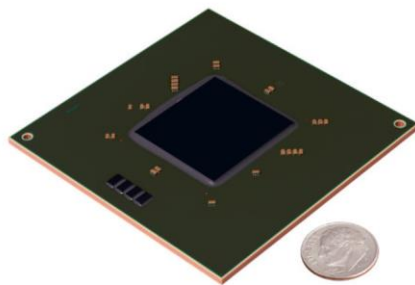
PROGRAMMING LANGUAGES

- ❖ Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate *translation steps*.
- ❖ These may be divided into three general types:
 - Machine languages
 - Assembly languages
 - High-level languages



MACHINE LANGUAGES

- ❖ Any computer can directly understand only its own **machine language**, *defined by its hardware design*.
 - Generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time.
 - *Machine dependent—a particular machine language can be used on only one type of computer.*



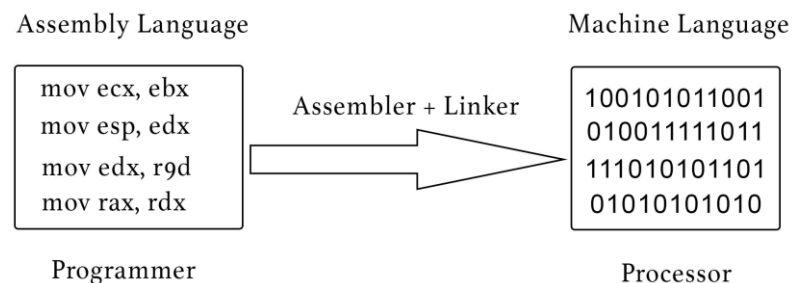
IBM Z15 processor



Intel Core i7 processor

ASSEMBLY LANGUAGES AND ASSEMBLERS

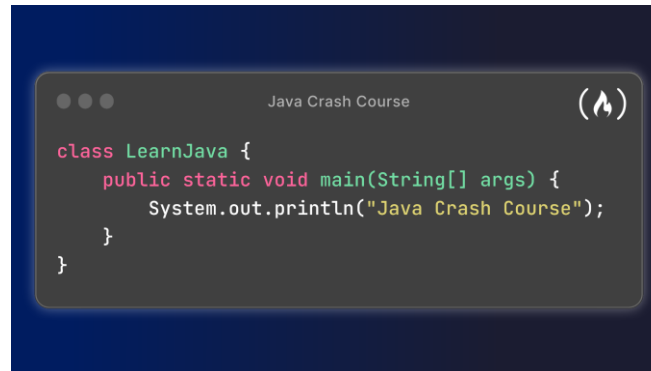
- ❖ English-like abbreviations that represent elementary operations form the basis of **assembly languages**.
- ❖ *Translator programs* called **assemblers** convert assembly-language programs to machine language.



HIGH-LEVEL LANGUAGES AND COMPILERS

❖ High-level languages

- Single statements accomplish substantial tasks.
- **Compilers** convert high-level language programs into machine language.
- Allow you to write instructions that look almost like everyday English and contain commonly used mathematical notations.
- A payroll program written in a high-level language might contain a *single* statement such as
 - `grossPay = basePay + overTimePay`



```
class LearnJava {  
    public static void main(String[] args) {  
        System.out.println("Java Crash Course");  
    }  
}
```

-
- ```
1 #include <iostream>
2 using namespace std;
3 int main ()
4 {
5 count <<"Hello World!";
6 return 0;
7 }
8
9
10
```
- Oh, now I understand!
- 01010101010101010101  
101010101010101010101  
10101000100101010101  
101010101010101010101  
1010101010101010101
- Compiler

# PROGRAMMING PARADIGMS

- ❖ Programming paradigms refer to the different approaches or styles of programming that dictate how software is designed, structured, and organized.
- ❖ Each paradigm provides a set of principles, concepts, and techniques for solving problems and expressing computations.
- ❖ Some of the main programming paradigms are:
  - Imperative programming
  - Procedural programming
  - Declarative programming
  - Event-driven programming

# IMPERATIVE PROGRAMMING

- ❖ Imperative programming is the most common paradigm and focuses on describing how a program should execute step by step.
- ❖ It uses statements that change the program's state, typically through assignments and loops.

# PROCEDURAL PROGRAMMING

- ❖ Procedural programming is a subset of imperative programming.
- ❖ It organizes code into procedures or functions.

# DECLARATIVE PROGRAMMING

❖ Declarative programming focuses on describing what a program should achieve rather than how to achieve it. It allows programmers to specify the desired results or properties without explicitly defining control flow.

- Prolog
- SQL

# EVENT-DRIVEN PROGRAMMING

- ❖ Event-driven programming revolves around responding to events or user actions.
- ❖ The program consists of event handlers that are triggered when specific events occur.
- ❖ This paradigm is commonly used in graphical user interfaces (GUIs) and other systems where user interaction is crucial.

# INTRODUCTION TO OBJECT TECHNOLOGY

- ❖ Objects, or more precisely, the *classes* that objects come from, are essentially *reusable* software components.
  - There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc.
  - Almost any *noun* can be reasonably represented as a software object in terms of *attributes* (e.g., name, color and size) and *behaviors* (e.g., calculating, moving and communicating).

# EXAMPLE: THE AUTOMOBILE AS AN OBJECT

- Suppose you want to *drive a car and make it go faster by pressing its accelerator pedal*.
- Before you can drive a car, someone has to *design* it.
- A car typically begins as engineering drawings, similar to the *blueprints* that describe the design of a house.
- Drawings include the design for an accelerator pedal.
- Pedal *hides* from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel “hides” the mechanisms that turn the car.

# EXAMPLE: THE AUTOMOBILE AS AN OBJECT (CONT.)

- ❖ This approach enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.
- ❖ Before you can drive a car, it must be **built** from the engineering drawings that describe it.
- ❖ A completed car has an **actual** accelerator pedal to make it go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must press the pedal to accelerate the car.

# METHODS AND CLASSES

- ❖ Performing a task in a program requires a **method**.
- ❖ The method houses the program statements that actually perform its tasks.
- ❖ Hides these statements from its user, just as the accelerator pedal of a car hides from the driver the mechanisms of making the car go faster.
- ❖ In Java, we create a program unit called a **class** to house the set of methods that perform the class's tasks.
- ❖ A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

# INSTANTIATION

- ❖ Just as someone has to *build* a car from its engineering drawings before you can actually drive a car, you must *build an object* of a class before a program can perform the tasks that the class's methods define.
- ❖ An object is then referred to as an *instance* of its class.

# REUSE

- ❖ Just as a car's engineering drawings can be reused many times to build many cars, you can **reuse a class many times** to build many objects.
- ❖ Reuse of existing classes when building new classes and programs **saves time and effort**.
- ❖ Reuse also helps you build more reliable and effective systems, because existing classes and components often have undergone **extensive testing, debugging and performance tuning**.

# REUSE



## Software Engineering Observation 1.1

Use a building-block approach to creating your programs. Avoid reinventing the wheel—use existing high-quality pieces wherever possible. This software reuse is a key benefit of object-oriented programming.

# MESSAGES AND METHOD CALLS

- ❖ When you drive a car, pressing its gas pedal sends a *message* to the car to perform a task—that is, to go faster.
- ❖ Similarly, you *send messages to an object*.
- ❖ Each message is implemented as a *method call* that tells a method of the object to perform its task.

# ATTRIBUTES AND INSTANCE VARIABLES

- ❖ A car has **attributes**
- ❖ Color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading).
- ❖ The car's attributes are represented as part of its design in its engineering diagrams.
- ❖ Every car maintains its own attributes.
- ❖ Each car knows how much gas is in its own gas tank, but not how much is in the tanks of other cars.

# ATTRIBUTES AND INSTANCE VARIABLES (CONT.)

- ❖ A bank-account object has a *balance attribute* that represents the amount of money in the account.
- ❖ Each bank-account object knows the balance in the account it represents, but *not* the balances of the *other* accounts in the bank.
- ❖ Attributes are specified by the class's **instance variables**.

# ENCAPSULATION AND INFORMATION HIDING

- ❖ Classes (and their objects) **encapsulate**, i.e., encase, their attributes and methods.
- ❖ Objects may communicate with one another, but they're normally not allowed to know how other objects are implemented—implementation details can be hidden within the objects themselves.
- ❖ **Information hiding**, is crucial to good software engineering.

# INHERITANCE

- ❖ A **new class** of objects can be created conveniently by **inheritance**—the new class (called the **subclass**) starts with the characteristics of an existing class (called the **superclass**), **possibly customizing them and adding unique characteristics of its own**.
- ❖ In our car analogy, an object of class “convertible” certainly *is an* object of the more *general* class “automobile,” but more *specifically*, the roof can be raised or lowered.

# INTERFACES

- ❖ **Interfaces** are collections of related methods that typically enable you to tell objects *what* to do, **but not how to do it**.
- ❖ In the car analogy, a “basic-driving-capabilities” interface consisting of a steering wheel, an accelerator pedal and a brake pedal would enable a driver to tell the car what to do.
- ❖ Once you know how to use this interface for turning, accelerating and braking, you can drive many types of cars, even though manufacturers may implement these systems differently.

## INTERFACES (CONT.)

- ❖ A class **implements** zero or more interfaces, each of which can have one or more methods, just as a car implements separate interfaces for basic driving functions, controlling the radio, controlling the heating and air conditioning systems, and the like.
- ❖ Just as car manufacturers implement capabilities *differently*, classes may implement an interface's methods *differently*.

# OBJECT-ORIENTED ANALYSIS AND DESIGN (OOAD)

- ❖ How will you create the **code** (i.e., the program instructions) for your programs?
- ❖ Follow a detailed **analysis** process for determining your project's **requirements** (i.e., defining *what* the system is supposed to do)
- ❖ Develop a **design** that satisfies them (i.e., specifying *how* the system should do it).
- ❖ Carefully review the design (and have your design reviewed by other software professionals) before writing any code.

# OBJECT-ORIENTED ANALYSIS AND DESIGN (OOAD) (CONT.)

- ❖ Analyzing and designing your system from an object-oriented point of view is called an **object-oriented-analysis-and-design (OOAD) process**.
- ❖ Languages like Java are object oriented.
- ❖ **Object-oriented programming (OOP)** allows you to implement an object-oriented design as a working system.

# THE UML (UNIFIED MODELING LANGUAGE)

❖ The Unified Modeling Language (UML) is the most widely used graphical scheme for modeling object-oriented systems.

# JAVA

- ❖ Using Java, you can write programs that will run on a great variety of computer systems and computer-controlled devices. This is sometimes called “write once, run anywhere.”
- ❖ Now used to develop large-scale enterprise applications, to enhance the functionality of web servers, to provide applications for consumer devices, to develop robotics software and for many other purposes.
- ❖ Also the key language for developing Android smartphone and tablet apps.
- ❖ Java has become the most widely used general-purpose programming language with more than 10 million developers.

# JAVA (CONT.)

## *Java Class Libraries*

- ❖ Rich collections of existing classes and methods
- ❖ Also known as the **Java APIs (Application Programming Interfaces)**.



### **Performance Tip 1.1**

Using Java API classes and methods instead of writing your own versions can improve program performance, because they're carefully written to perform efficiently. This also shortens program development time.

# A TYPICAL JAVA DEVELOPMENT ENVIRONMENT

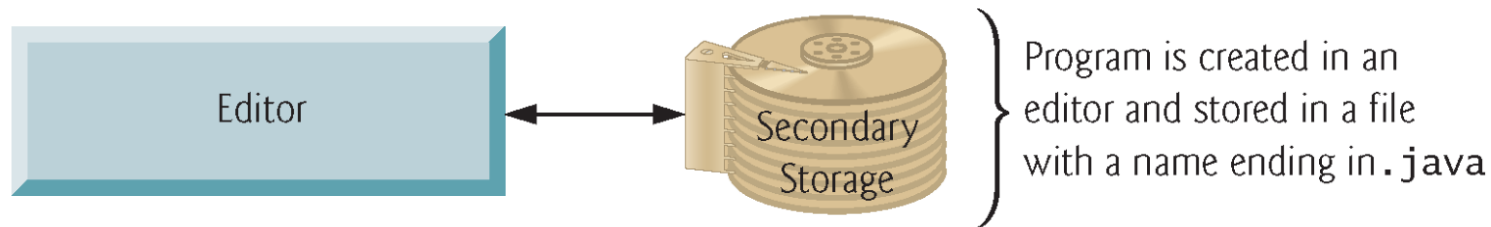
❖ Normally there are five phases

- edit
- compile
- load
- verify
- execute.

# EDITING

❖ Phase 1 consists of editing a file with an *editor program*

- Using the editor, you type a Java program (**source code**).
- Make any necessary corrections.
- Save the program.
- Java source code files are given a name ending with the **.java extension** indicating that the file contains Java source code.



# COMMON EDITORS

- ❖ Windows provides Notepad.
- ❖ macOS provides TextEdit.
- ❖ Many freeware and shareware editors are also available online, including
  - Notepad++ (<http://notepad-plus-plus.org>)
  - EditPlus (<http://www.editplus.com>)
  - TextPad (<http://www.textpad.com>)
  - jEdit (<http://www.jedit.org>) and more.

# INTEGRATED DEVELOPMENT ENVIRONMENTS

❖ **Integrated development environments (IDEs)** provide tools that support the software development process, such as editors, debuggers for locating **logic errors** that cause programs to execute incorrectly and more.

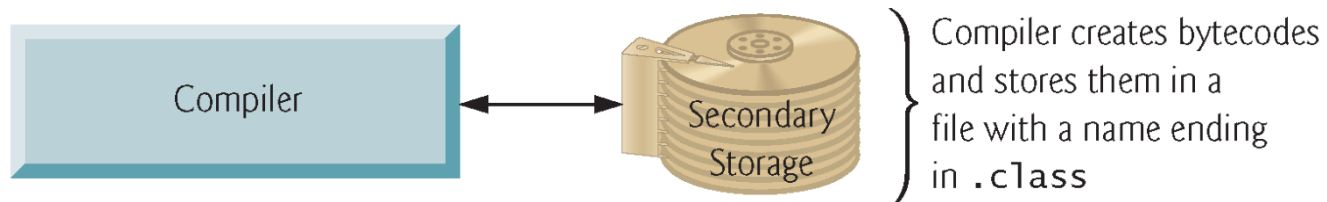
❖ The most popular Java IDEs are:

- Eclipse (<http://www.eclipse.org>)
- IntelliJ IDEA (<http://www.jetbrains.com>)
- NetBeans (<http://www.netbeans.org>)

# COMPILATION

## ❖ Phase 2: Compiling a Java Program into Bytecodes

- Use the command `javac` (the `Java compiler`) to `compile` a program. For example, to compile a program called `Welcome.java`, you'd type
  - `javac Welcome.java`
- If the program compiles, the compiler produces a `.class` file called `Welcome.class` that contains the compiled version.



# JAVA VIRTUAL MACHINE (JVM)

- ❖ Java compiler translates Java source code into **bytecodes** that represent the tasks to execute.
- ❖ The **Java Virtual Machine (JVM)**—a part of the JDK and the foundation of the Java platform—executes bytecodes.
- ❖ **Virtual machine (VM)**—a software application that simulates a computer
  - Hides the underlying operating system and hardware from the programs that interact with it.
- ❖ If the same VM is implemented on many computer platforms, applications written for that type of VM can be used on all those platforms.

# BYTECODE

- ❖ Bytecode instructions are *platform independent*
- ❖ Bytecodes are **portable**
  - The same bytecode instructions can execute on any platform containing a JVM that understands the version of Java in which the bytecode instructions were compiled.
- ❖ The JVM is invoked by the **java** command. For example, to execute a Java application called `Welcome`, you'd type the command
  - `java Welcome`

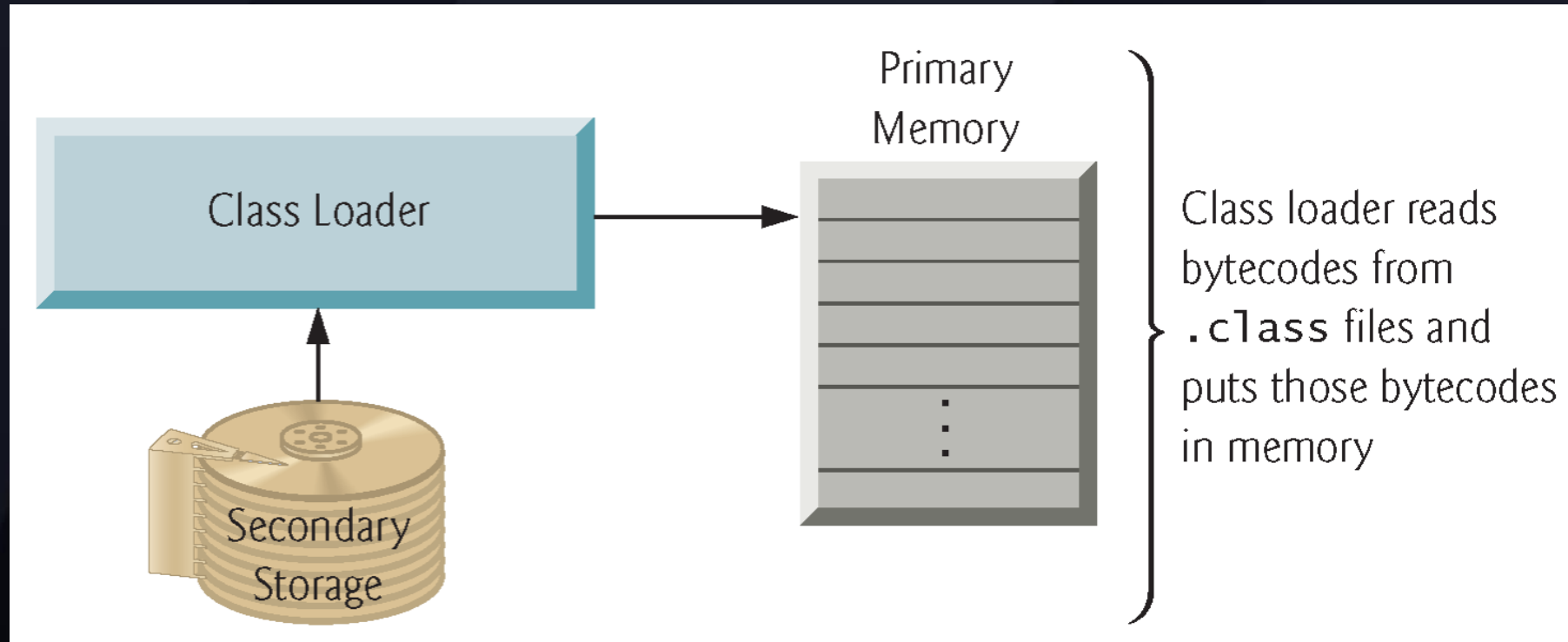
# LOADING

## ❖ Phase 3: Loading a Program into Memory

- The JVM places the program in memory to execute it—this is known as **loading**.
- **Class loader** takes the `.class` files containing the program's bytecodes and transfers them to primary memory.
- Also loads any of the `.class` files provided by Java that your program uses.

❖ The `.class` files can be loaded from a disk on your system or over a network.

# JAVA CLASS LOADING

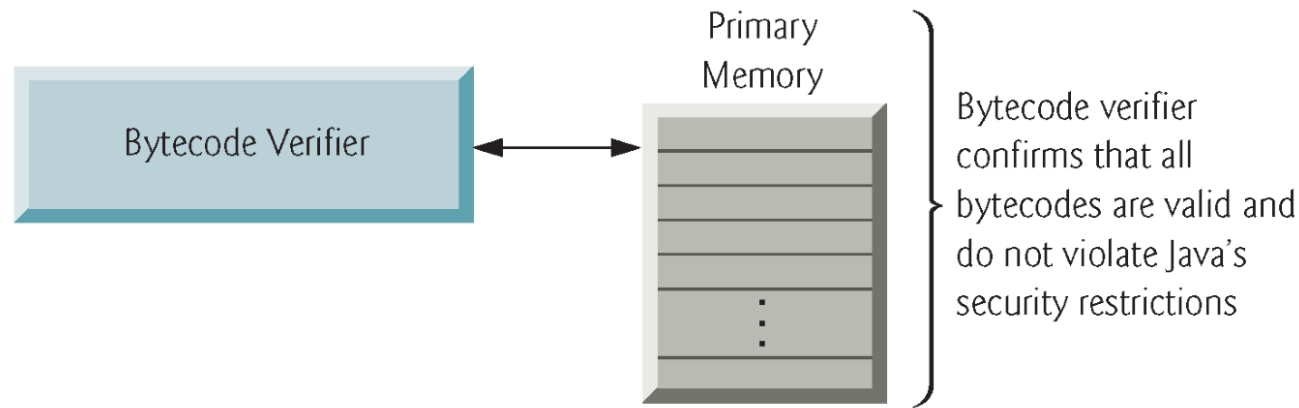


# BYTECODE VERIFICATION

## ❖ Phase 4: Bytecode Verification

- As the classes are loaded, the **bytecode verifier** examines their bytecodes
- Ensures that they're valid and do not violate Java's security restrictions.

❖ Java enforces strong security to make sure that Java programs arriving over the network do not damage your files or your system (as computer viruses and worms might).



# EXECUTION

## ❖ Phase 5: Execution

- The JVM **executes** the program's bytecodes.
- JVMs typically execute bytecodes using a so-called **just-in-time (JIT) compilation**.
- Analyzes the bytecodes as they're interpreted
- A **just-in-time (JIT) compiler**—such as Oracle's **Java HotSpot™ compiler**—translates the bytecodes into the underlying computer's machine language.

## EXECUTION (CONT.)

- ❖ Java programs go through *two* compilation phases
- ❖ One in which source code is translated into bytecodes (for portability across JVMs on different computer platforms) and
- ❖ A second in which, during execution, the bytecodes are translated into *machine language* for the actual computer on which the program executes.

# GETTING YOUR QUESTIONS ANSWERED

- ❖ There are many online forums in which you can get your Java questions answered and interact with other Java programmers.
- ❖ Some popular Java and general programming forums include:
  - StackOverflow.com
  - Coderanch.com
  - The Oracle Java Forum—  
<https://community.oracle.com/community/java>
  - </dream.in.code>—  
<http://www.dreamincode.net/forums/forum/32-java/>

# SUMMARY

- ◉ Programming languages
- ◉ Compilers and Interpreters
- ◉ Programming paradigms
- ◉ Introduction to object technology
- ◉ Object Oriented Analysis and Design (OOAD)
- ◉ Java programming language



مجموعة جامعة  
الجميع الذكية  
للتعليم القابضة

MIDOCEAN  
UNIVERSITY  
BY EVERYONE'S SMART UNIVERSITY



Thank You

[www.midocean.ae](http://www.midocean.ae)

