University of Kelaniya

Kesavan Selvarajah

BSc (UOK), MSc (Reading, UCSC)

Temporary Lecturer

Software Engineering Department

Faculty of Computing and Technology

skesa231@kln.ac.lk

Object-Oriented Programming

CSCI 21052 / ETEC 21062

Course Overview

- Course Code: CSCI 21052 / ETEC 21062
- Course Name: Object-Oriented Programming
- Credit Value: 2
- Theory: 30 hours
- Practical: 30 hours
- Independent Learning: 40 hours

Intended Learning Outcomes

- Demonstrate the knowledge of the theory of object–oriented systems.
- Demonstrate how an object-oriented programming language upholds object-oriented concepts.
- Effectively use an industry relevant object-oriented programming language.

Assessment Strategy

- **Continuous Assessment 30%**
  - Mid-semester Exam: 10%
  - Group Project: 20%

- **Final Assessment 70%**
  - Theory: 40%
  - Practical: 30%

References/Reading Materials

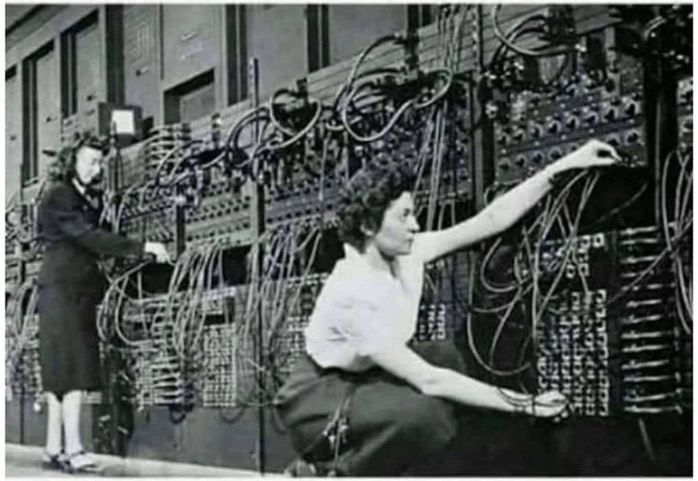
- Bloch, J., (2018), Effective Java, 3rd edition, Addison-Wesley Professional.
- Schildt, H., (2018), Java: A Beginner's Guide, 8th edition, McGraw-Hill Education.
- Schildt, H., (2018), Java: The Complete Reference, 11th Edition, McGraw-Hill Education.
- McLaughlin, B.D., Pollice, G. West, D., (2006), Head First Object-Oriented Analysis and Design, 1st Edition, O'Reilly Mediaence.

Introduction

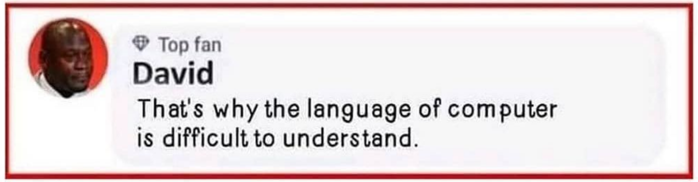
- Top human inventions that have shaped the course of human history
  - Fire
  - Weapons/Tools
  - Language
  - Writing system
  - Agriculture
  - Wheel
  - Currency
  - Printing press
  - Steam engine
  - Telegraph, Telephone
  - Atomic energy/Atom bomb
  - Computer
  - Internet
  - **AI (we are here now)**

What's next?

Did you know?



The first computer programmers were women



Top fan

David

That's why the language of computer is difficult to understand.

Decimal Numbers

- In the decimal number system, we have 10 symbols, and the position values are integral powers of 10. We say that 10 is the base or radix of the decimal number system.

2	4	8	•	7
10^2	10^1	10^0		10^{-1}

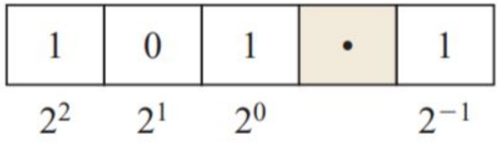
$$= 2 \times 10^2 + 4 \times 10^1 + 8 \times 10^0 + 7 \times 10^{-1}$$

$$= 2 \times 100 + 4 \times 10 + 8 \times 1 + 7 \times 1/10$$

$$= 200 + 40 + 8 + 7/10 = 248.7$$

Binary Numbers

- The binary number system works the same as the decimal number system but uses 2 as its base. The binary number system has two digits (0 and 1) called bits, and position values are integral powers of 2.



$$= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1}$$

$$= 1 \times 4 \;\; + 0 \times 2 \;\; + 1 \times 1 \;\; + 1 \times 1/2$$

$$= 4 \quad + 0 \quad + 1 \quad + 1/2 \quad = 5.5$$

Programming languages

- Programming languages are broadly classified into three levels,
  - **Machine languages**
    - Machine language is the only programming language the CPU understands. Machine-language instructions are binary-coded and very low level.
  - **Assembly languages**
    - One level above machine language is assembly language, which allows “higher-level” symbolic programming. An assembler translate them into machine-language equivalents.
  - **High-level languages**
    - High-level languages were developed to enable programmers to write programs faster than when using assembly languages. A compiler translate them to assembly language equivalents.

Programming languages

machine code

```
10110011 00011001
01111010 11010001 10010100
10011111 00011001
01011100 11010001 10010000
10111011 11010001 10010110
```

assembly code

```
MV    0,    SUM
MV    NUM,  AC
ADD   SUM,  AC
STO   SUM,  TOT
```

high-level code

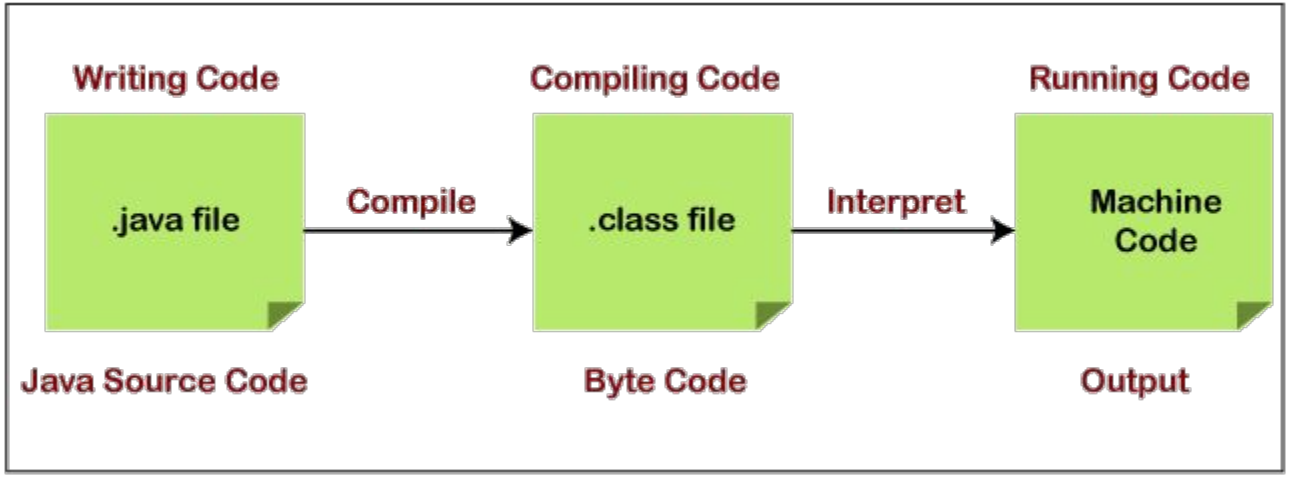
```
X = (Y + Z) / 2
```

Java

- **Java** is a high-level, class-based, object-oriented programming language.
- **Object-oriented programming** is a programming paradigm where everything is represented as an object.
- The language was based on C and C++ and was originally intended for writing programs that control consumer appliances such as toasters, microwave ovens, and others.
- Java is a versatile programming language used to develop various types of software applications.
  - Applets that run within a Web browser (No longer used)
  - Server-side applications
  - Mobile/desktop applications

Java

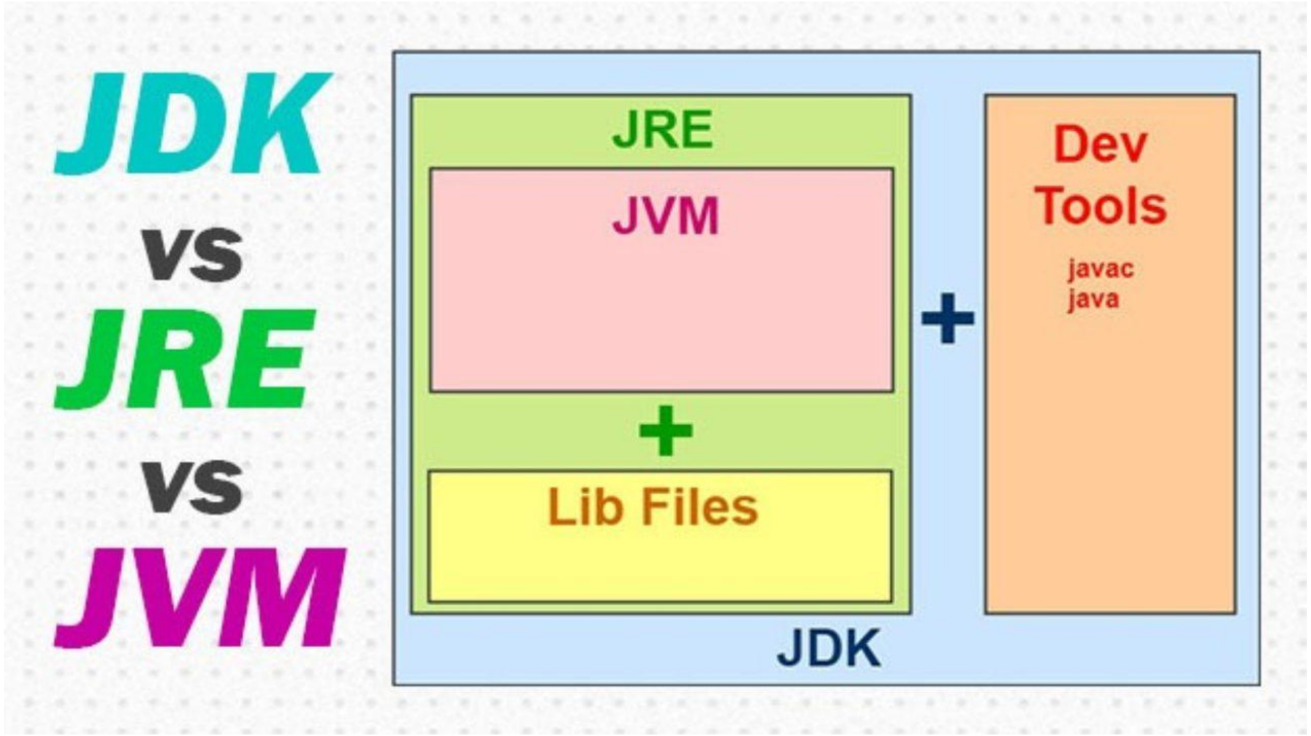
- **Java** is designed to be a **"write once, run anywhere"** language, which means that once a Java program is compiled, the resulting bytecode can be executed on any platform that has a Java Virtual Machine (JVM) installed.



JDK vs JVM vs JRE

- **JDK** is a software development kit that includes tools to develop, debug, and deploy Java applications.
- It includes tools to compile Java source code into bytecode.
- **JRE** is a software package that provides the runtime environment needed to execute Java bytecode.
- **JVM** is the component of the Java platform that executes Java bytecode.

JDK vs JVM vs JRE



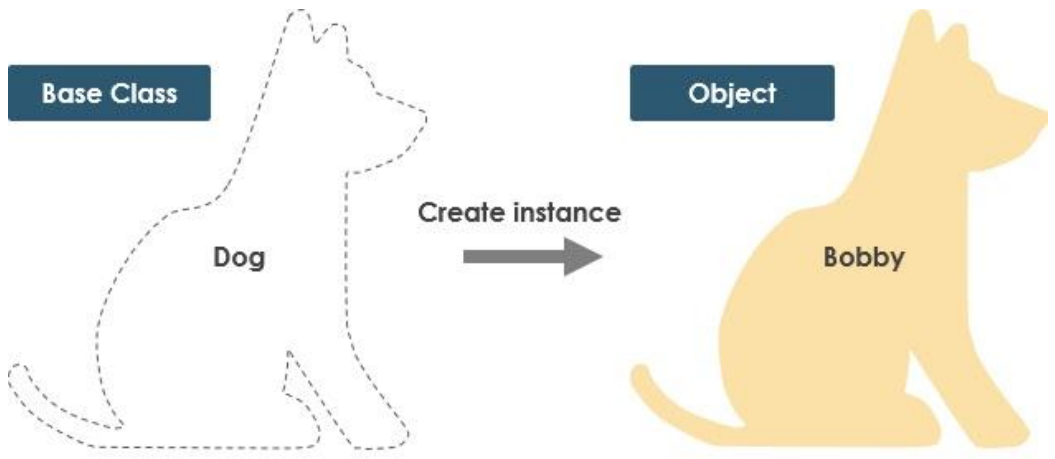
Classes and Object

- An **Object** is a thing, both tangible and intangible, that we can imagine.
- A program written in object-oriented style will consist of interacting objects.
- An object is comprised of **data** and **operations** that manipulate these data.
- Inside a program we write instructions to create objects.
- Objects occupy memory location at run-time.
- For the computer to be able to create an object, we must provide a definition, called a **Class**.

Classes and Object

- A **class** is a kind of blueprint or template that dictates what objects can and cannot do.
- In other words, a class refers to the category or type of objects.
- An object is called an instance of a class.
- An object is an instance of exactly one class.
- An instance of a class belongs to the class.
- Once a class is defined, we can create as many instances of the class as a program requires.

Classes and Object



Properties	Methods
Color	Sit
Eye Color	Lay Down

Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down

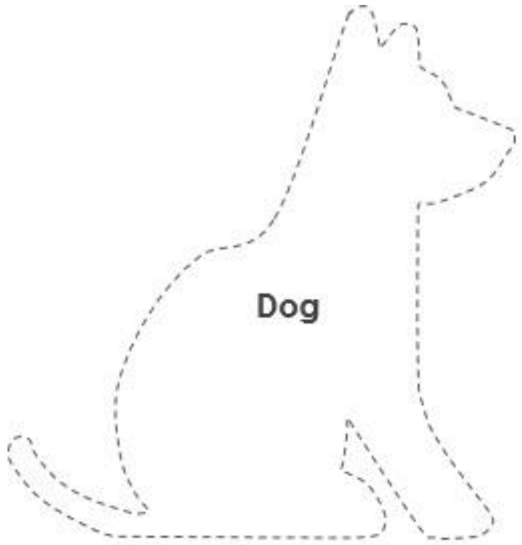
Syntax of the Classes

```
Public Class Dog
{
      color;
      eyeColor;

      sit();
      layDown();
}
```

Name of the class

$\left.\begin{array}{l}\text{color;}\\ \text{eyeColor;}\end{array}\right\}$ **Variables**

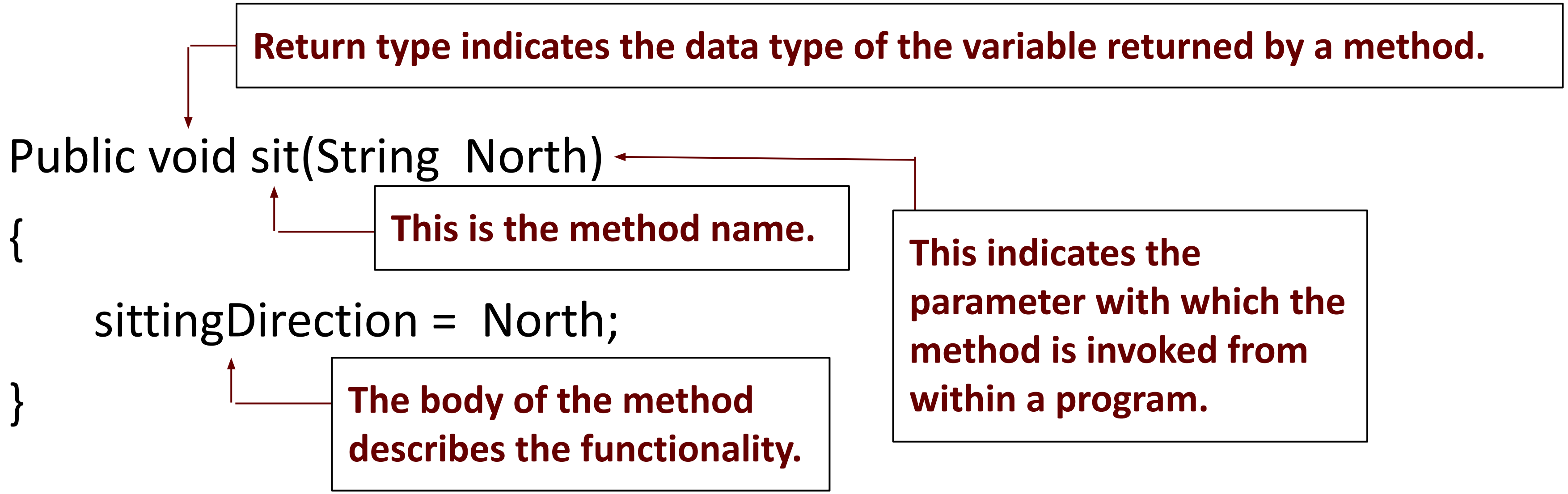
$\left.\begin{array}{l}\text{sit();}\\ \text{layDown();}\end{array}\right\}$ **Methods**



Messages and Methods

- To instruct a class or an object to perform a task, we send a **Message** to it.
- For a class or an object to process the message it receives, it must possess a matching **Method**.
- **Method** is a sequence of instructions that a class or an object follows to perform a task.
- A method defined for a class is called a **class method**, and a method defined for an object is an **instance method**.
- A value we pass to an object is called an **argument** of a message.

Syntax of the Methods



Return type indicates the data type of the variable returned by a method.

```
Public void sit(String  North)
{
      sittingDirection =  North;
}
```

This is the method name.

This indicates the parameter with which the method is invoked from within a program.

The body of the method describes the functionality.

Constructor

- A **Constructor** in Java is a special method that is used to initialize objects.
- The constructor is called when an object of a class is created.

```
Public Class Dog
{
    Dog()
    {
        System.out.println(“Woof! Woof! I am a Dog”);
    }
}
```

Types of Constructors

- **Default Constructors**
  - If a class does not have any constructor, Java compiler inserts a default constructor into the code with empty body.

```
Public Class Dog
{
    Dog()
    {
    }
}
```

Types of Constructors

- **No-argument Constructors**
  - A No-argument constructor is one that takes no arguments or parameters.

```
Public Class Dog
{
    Dog()
    {
    System.out.println("Woof! Woof! I am a Dog");
    }
}
```

Types of Constructors

- **Parameterized Constructors**
  - A constructor with arguments or parameters is known as a Parameterized constructor.

```
Public Class Dog
{
    Dog(String name)
    {
    System.out.println(“Woof! Woof! I am ”+name);
    }
}
```

Class and Instance Data Values

- We can define data values to class and instance.
- An **Instance data value** is used to represent information of that particular instance.
- A **class data value** is used to represent information shared by all instances or to represent collective information about the instances.
- An instance can access the class data values of the class to which it belongs.
- A data value that can change is called a variable, and one that cannot change is a constant.

Once again Syntax of the Classes

```
Public Class Dog
{
    String color;
    String eyeColor;
    Static noOfLegs = 4;

    Dog()
    {
    System.out.println("Woof! Woof! I am a dog");
    }

    Public void sit()
    {
    System.out.println("I am a going to sit now.");
    }
}
```

Name of the class

Variables

Constructor

Method

Creating an object

- `Dog bobby = new Dog();`
- Dog bobby is the declaration of the object bobby of class Dog.
- Objects are created by using the operator **New** which allocates memory for the object.
- The Dog() command invokes the constructor.
- So, when Dog bobby = new Dog() runs, booby an object of class Dog will be created and the following gets printed: **"Woof! Woof! I am a dog"**

Class Vs Object

Class	Object
● A class is a data type. It defines a template or blueprint for an object.	● Object is an instance of a class.
● Does not occupy memory location.	● Occupies memory location at run time.
● Does not really exist as it only provides a template. You can therefore not perform operations on a class.	● You perform operations on objects.

Java syntax #1

- Every line of code that runs in Java must be inside a class.
- A class should always start with an uppercase first letter.
- Java is **case-sensitive**: "MyClass" and "myclass" are not same.
- The filename and class name in Java must match because the JVM needs to know which class to start executing when it loads a Java program.
- There should be a main method in Java because it is the entry point for all Java programs
- The curly braces {} marks the beginning and the end of a block of code.
- Each code statement must end with a semicolon (;).

`Hello World` code breakdown

```
public class Main {
  public static void main(String[] args) {
    System.out.print("Hello World");
  }
}
```

- **`public class Main`**
  - This line declares a public class called Main.
- **`public static void main(String[] args)`**
  - This line declares a public static method called main(). The main() method takes an array of strings as its argument.
- **`System.out.print("Hello World");`**
  - This line prints the text "Hello World" to the console.

Hello World code

```
public class Main {
  public static void main(String[] args) {
    System.out.print("Hello World");
    System.out.println("Hello World");
  }
}
```

- Try println method instead of print, and find the difference.
  - `System.out.println("Hello World");`

Simple Window code

```
/* Sample Program: Displaying a Window
File: SampleWindow.java */
import javax.swing.*;
class SampleWindow {
    public static void main(String[] args) {
        JFrame myWindow;
        myWindow = new JFrame();
        myWindow.setSize(300, 200);
        myWindow.setTitle("My Window");
        myWindow.setVisible(true);
    }
}
```

`Simple Window` code breakdown

- **`import javax.swing.*;`**
  - This line imports the javax.swing package, which contains all of the classes and interfaces that are used to create Swing GUIs.
- **`class SampleWindow {`**
  - This line defines a new class called sampleWindow.
- **`public static void main(String[] args) {`**
  - This line defines the main() method, which is the entry point for all Java programs.
- **`JFrame myWindow;`**
  - This line declares a variable called myWindow of type JFrame. The JFrame class is the base class for all Swing windows. It provides a number of methods for setting the size, title, and other properties of a window.

`Simple Window` code breakdown

- **`myWindow = new JFrame();`**
  - This line creates a new JFrame object and assigns it to the myWindow variable.
- **`myWindow.setSize(300, 200);`**
  - This line sets the size of the myWindow object to 300 pixels wide and 200 pixels high.
- **`myWindow.setTitle("My Window");`**
  - This line sets the title of the myWindow object to "My Window".
- **`myWindow.setVisible(true);`**
  - This line makes the myWindow object visible.

Java syntax #2

- A Java identifier is a sequence of letters, digits, underscores (_), and dollar signs ($) with the first one being a non digit.
- We use an identifier to name a class, object, method, and others.
- The following words are all valid identifiers:
  - **MyFirstApplication**
  - **_age**
  - **Hello$World**
  - **x123**
  - **velocity**
  - **DEFAULT_VALUE**
  - **$count**
- The following words are all invalid identifiers:
  - **123x**
  - **my application**

Standard naming convention

- Java standard naming convention advises using an uppercase letter for the first letter of the class names and a lowercase letter for the first letter of the object names.
  - `dog = new Dog();`
- Since Java is case-sensitive, we can use dog as the name for an object of the class Dog.
- In Java naming convention, multiple-word identifiers capitalize the first letter of each word except for the first word in object names.
  - **myFirstApplication = new MyFirstApplication();**
- Programs that follow the standard naming convention are easier to read than those that do not.

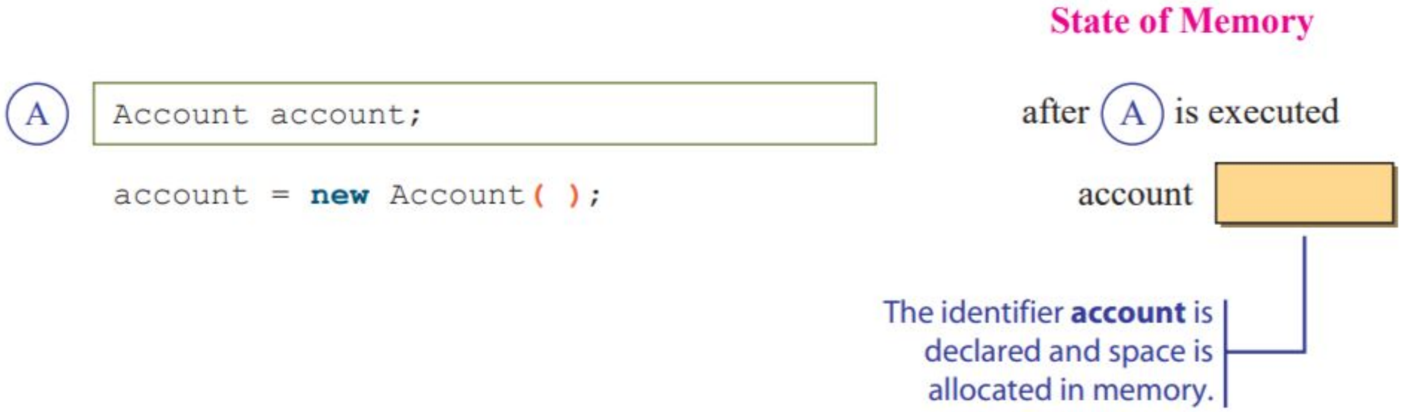
Object Declaration

- Every object we use in a program must be declared.
- An object declaration designates the name of an object and the class to which the object belongs.
  - `Dog bobby;`
  - `Dog scooby,max;`
- The first declaration declares a dog object named bobby, and the second declaration declares three dog objects.

Object Creation

- No objects are actually created by the declaration.
- An object declaration simply declares the name (identifier) that we use to refer to an object.
- We create an object by invoking the new operator.
  - `bobby = new Dog();`
- Instead of writing statements for object declaration and creation separately, we can combine them into one statement.
  - We can write,
    - `Dog bobby = new Dog();`
  - Instead of,
    - `Dog bobby;`
    - `bobby = new Dog();`

Distinction between object declaration and object creation



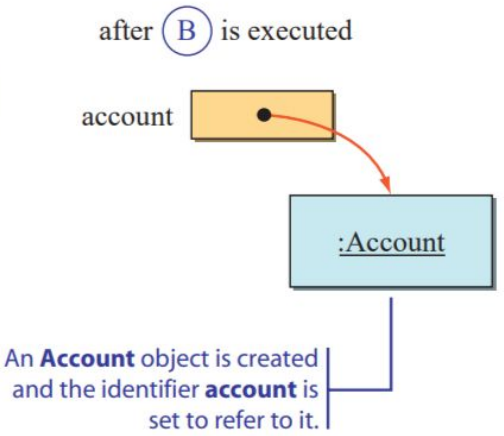
```
Account account;

account = new Account( );
```

A
State of Memory
after A is executed
account
The identifier **account** is declared and space is allocated in memory.

Distinction between object declaration and object creation

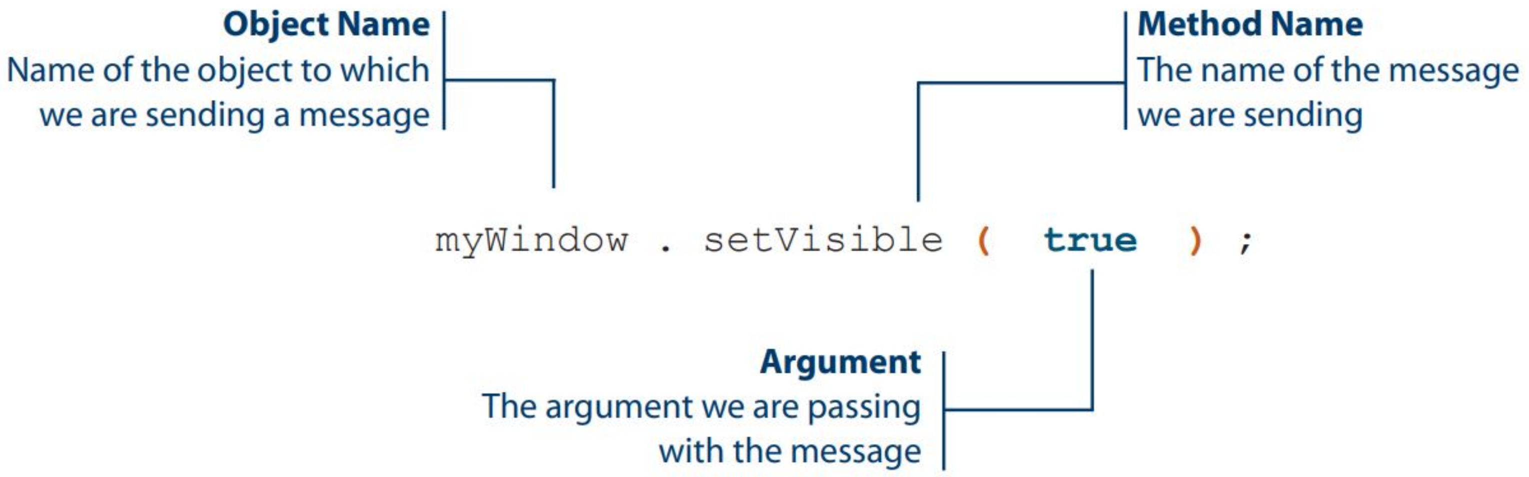
```
Account account;
```

B
```
account = new Account( );
```



An **Account** object is created and the identifier **account** is set to refer to it.

Message Sending


Object Name
Name of the object to which we are sending a message
Method Name
The name of the message we are sending
myWindow . setVisible (true) ;
Argument
The argument we are passing with the message

Program components

- A Java program is composed of,
  - **Comments**
  - **import statements**
  - **Class declarations.**
- We can write a Java program that includes only a single class declaration, but that is not the norm.
- In any nontrivial program, we will see these three components.

Comments

- In addition to the instructions for computers to follow, programs contain comments in which,
  - We state the purpose of the program
  - Explain the meaning of code
  - Provide any other descriptions to help programmers understand the program.
- Here’s the comment in the sample SampleWindow program:

```
/* Sample Program: Displaying a Window
File: SampleWindow.java */
```

- Although not required to run the program, comments are indispensable in writing easy-to-understand code.

Comments

- A comment is any sequence of text that begins with the **multi-line comment marker** `/*` and terminates with another marker `*/`.
- Another marker for a comment is double slashes `//`.
- This marker is used for a **single-line comment marker**.

  -
    ```
    /*
    This is a comment with
    three lines of
    text.
    */
    ```
  - `// This is a single-line comment`

Comments

- Comments are intended for the programmers only and are ignored by the computer.
- Comments are really not necessary in making a program executable.
- They are an important aspect of documenting the program.
- Programs needs changes almost every time. There maybe,
  - **Some requirements changes**
  - **Bugs/Errors**
- For a programmer to modify his own or someone else's program, the programmer must first understand the program.
- Comments are useful in disabling a portion of a program.

Import Statement

- We develop object-oriented programs by using predefined classes, both system and programmer-defined.
- In Java, classes are grouped into packages, and the Java system comes with numerous packages.
- To use a class from a package, we refer to the class in our program by using the following format:
  - **javax.swing.JFrame**
- We are calling the JFrame class from the swing package which is inside the javax package.

Import Statement

- If the import statement is not used, we need to refer to a class using its fully qualified name.

  -
    ```
    javax.swing.JFrame myWindow;
    myWindow = new javax.swing.JFrame();
    ```

    Instead of

  -
    ```
    JFrame myWindow;
    myWindow = new JFrame();
    ```

Import Statement

- The import statement of one class from swing package.
  - `import javax.swing.JFrame;`
- The import statement of two classes from swing package.
  - `import javax.swing.JFrame;`
  - `import javax.swing.JButton;`
- The import statement of all classes from swing package.
  - `import javax.swing.*;`

Class Declaration

- A Java program is composed of one or more classes; some are predefined classes, while others are defined by us.
- In the sample program, there are two classes,
  - **JFrame:** one of the standard classes
  - **SampleWindow**: class we defined ourselves.
- To define a new class, we must declare it in the program.
- The syntax for declaring the class is,

```
class <class name> {
    <class member declarations>
}
```

Class Declaration

- One of the classes in a program must be designated as the main class.
- we can designate a class as the main class, by defining a method called main.
- Because when a Java program is executed, the main method of a main class is executed first.
- To define a method, we must declare it in a class.

Method Declaration

- The syntax for method declaration is

```
<modifiers> <return type> <method name>
(<parameters>) {
   <method body>
}
```

Edit-Compile-Run Cycle

- **Step 1**
  - Type in the program, using an editor, and save the program to a file.
  - Use the name of the main class and the suffix .java for the filename. **(Eg: sampleWindow.java)**
  - This file, in which the program is in a human-readable form, is called a **Source file**.

Edit-Compile-Run Cycle

- **Step 2**
  - Compile the source file.
  - Many compilers require us to create a project file and then place the source file in the project file in order to compile the source file.
  - When the compilation is successful, the compiled version of the source file is created.
  - This compiled version is called **Bytecode**, and the file that contains bytecode is called a bytecode file.
  - The name of the compiler-generated bytecode file will have the suffix .class while its prefix is the same as the one for the source file. **(Eg: sampleWindow.class)**

Edit-Compile-Run Cycle

- **Step 2**
  - When any error occurs in a program, an error message will be displayed.
  - If the sample program contains no errors in syntax, then instead of an error message, we will get nothing or a message stating something like “Compiled successfully.”
  - Errors detected by the compiler are called **compilation errors**.
  - Compilation errors are actually the easiest type of errors to correct.
  - Most compilation errors are due to the violation of syntax rules.

`Simple Window` code with errors

```
import javax.swing.*;
class SampleWindow {
    public static void main(String[] args) {
        myWindow = new JFrame();
        myWindow.setSize();
        myWindow.setTitle("My Window");
        myWindow.setVisible(true);
    }
}
```

Edit-Compile-Run Cycle

- **Step 3**
  - Execute the bytecode file.
  - A Java interpreter will go through the bytecode file and execute the instructions in it.
  - If our program is error-free, a window will appear on the screen.
  - If an error occurs in running the program, the interpreter will catch it and stop its execution.
  - Errors detected by the interpreter are called **execution errors.**

Thank you