303.1.1
Introduction to Java Basics

Learning Objective

In this presentation, we will explore the program structure of a Java program, and will guide the learner on how to run a Java-based code. By the end of this presentation, the learner will be able to:

- Explore Java history and the Java environment.
- Explain Java fundamentals.
- Demonstrate IDE and Eclipse.
- Explain Java Best Practices such as programming style and documentation.

Outline

- **Java Overview**
  - Why Java?
  - Java History.
  - Java Features.
  - Java Architecture.
  - Java Environment.
  - Java Development Kit.
  - Java Development Kit Editions.
- **IDE and Eclipse**
  - Java Integrated Development Environments.
  - Eclipse Integrated Development Environment.
- **Java Program Overview**
  - Class Name.
  - Entering the main() Method.
  - Executing the Statement.
  - Reserved Words.
  - Special Symbols.
  - Curly Braces and Code Blocks.
  - Parentheses.
  - Inline Comments.
  - Double Quotation Marks.
  - Best Practices - Programming Style and Documentation.
  - Appropriate Comments.
  - Naming Conventions.
  - Proper Indentation and Spacing Lines.
  - Block styles.

Lesson 1

Java Overview

Why Java?

- Java is a general-purpose, computer programming language that is:
  - Concurrent.
  - Class-based.
  - Object-oriented.
  - Designed to have as few implementation dependencies as possible.
- Java is intended to let application developers write once, run anywhere (WORA).
- Compiled Java code can run on all platforms that support Java without the need for recompilation.
- There were five primary goals in the creation of the Java language:
  - Simple, object-oriented, and familiar.
  - Robust and secure.
  - Architecture-neutral and portable.
  - Execute with high performance.
  - Interpreted, threaded, and dynamic.

Why Java? (continued)

- Java enables users to develop and deploy applications on the Internet for servers, desktop computers, and small handheld devices. Today's computing is profoundly influenced by the Internet, and Java promises to remain a big part today and in the future:
  - Java is a general-purpose programming language.
  - Java permeates the Internet, and is the invisible force behind many applications and devices that power our day-to-day lives. From mobile phones to handheld devices and games, and from navigation systems to e-business solutions, Java is everywhere!

Java History

- James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of Sun Microsystems (now a subsidiary of Oracle Corporation) engineers called it **Green Team,** which was originally designed for small, embedded systems in electronic appliances such as set-top boxes.
- As a development of the *Green Project*, the name was changed to "**Oak**."
- In 1994, Oak was renamed **"Java,"** and was released in 1995.
- Java 1.0 released in January 23, 1996.
- The most current version, "**Java 12**," was released in March 2019.

Java Version History

This chart shows the various version and release dates.

For more information on Java History, visit: check on [Wikipedia]

Java Features

- Java is a platform-independent language..
- Java is simple.
- Java is object-oriented.
- Java is distributed.
- Java is JIT-compiled.
- Java is robust.
- Java is secure.
- Java performance.
- Java is multi-threaded.
- Java is dynamic.

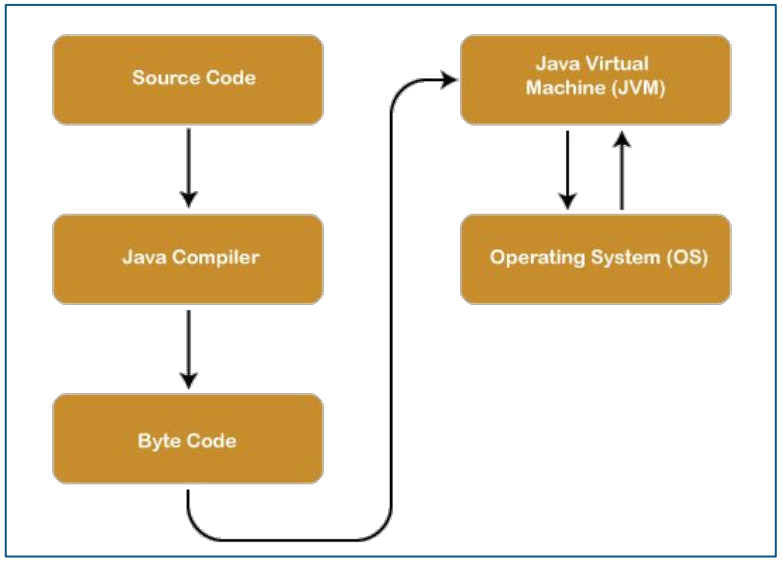
Please view our **Wiki** document to review the features of Java.

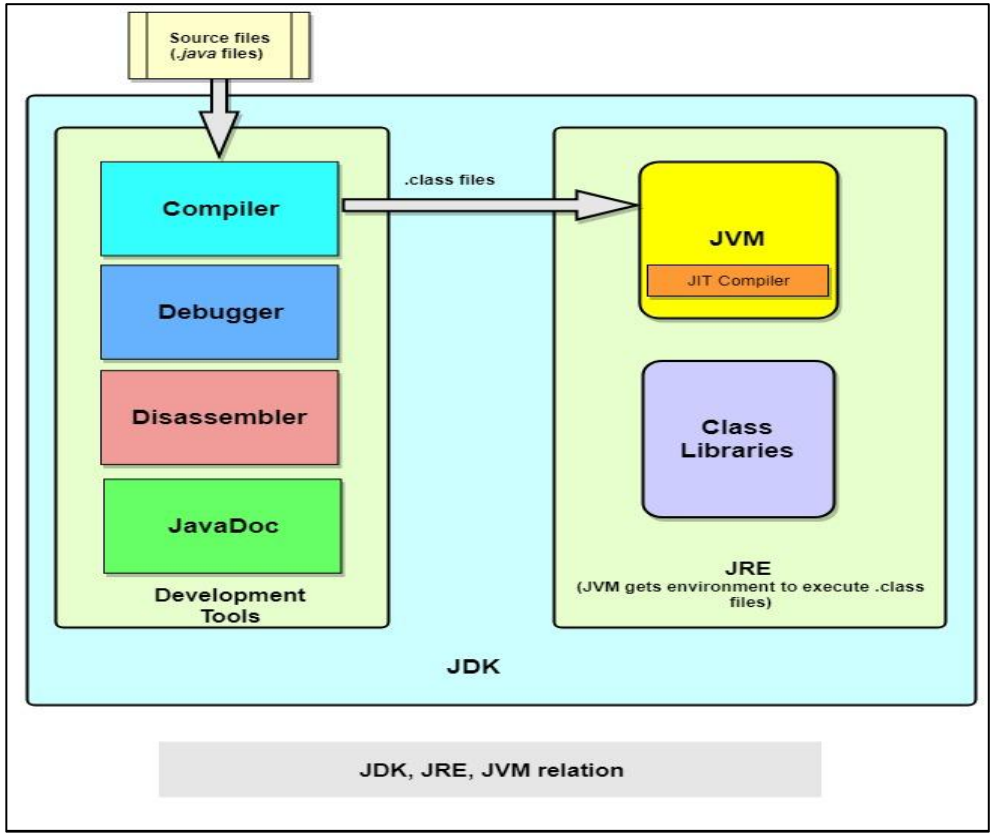
Java Architecture

Java Architecture is **a collection of components** (**e.g., JVM, JRE, and JDK**.) It integrates the process of interpretation and compilation, and defines all of the processes involved in creating a Java program.

Java Architecture can be explained by using the following steps:

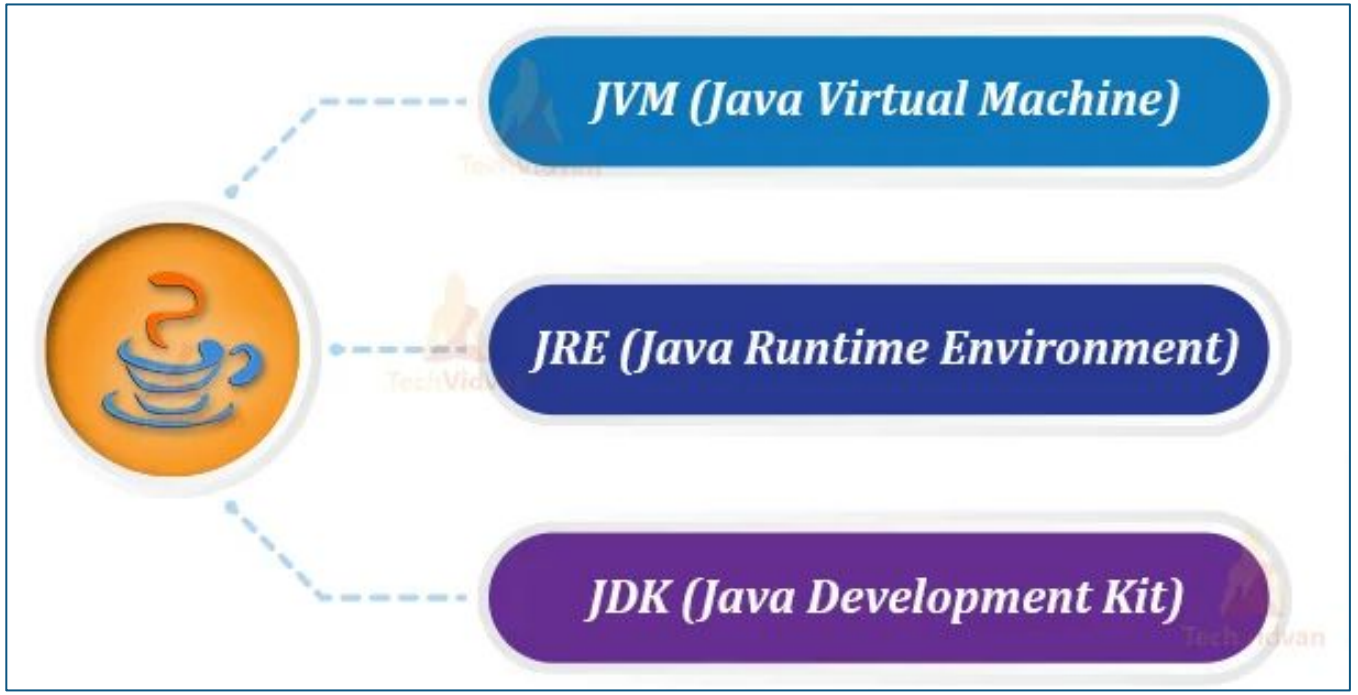
- Java compiler converts the Java code into bytecode.
- The JVM converts the byte code into machine code.
- The machine code is then executed by the machine.





TEKsystems
Own change

Java Environment



Source: techvidvan

Java Environment - Java Development Kit

Java Development Kit–

The Java Development Kit (JDK) is a software development environment used for developing Java applications. It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed in Java development.

Java Environment - Java Runtime Environment

Java Runtime Environment–

Java Runtime Environment (JRE), also written as Java RTE, provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files.

Java Environment -Java Virtual Machine

Java Virtual Machine–

Java Virtual Machine (JVM) manages system memory and provides a portable execution environment for Java-based applications. It includes:

- A **specification** — where requirements of a JVM are specified.
- An **implementation** — a computer program that meets the requirements of the JVM specification.
- A **runtime instance** — an instance of JVM is created whenever a java command is written on the command prompt to run the java class.

Java Development Kit Editions

- Java technology is both a programming language and a platform. The Java programming language is high-level and object-oriented, and has a particular syntax and style. A Java platform is a particular environment on which Java programming language applications run.
- There are four platforms of the Java programming language:
  - **Java Platform, Java Standard Edition (Java SE)**
    - Java SE can be used to develop client-side, stand-alone applications or applets. Throughout this course, we will use Java SE 1.8 for development.
  - **Java Platform, Java Enterprise Edition (Java EE)**
    - Java EE can be used to develop server-side applications such as Java Servlets, JavaServer Pages, and JavaServer Faces.
  - **Java Platform, Java Micro Edition (Java ME)**
    - Java ME can be used to develop applications for handheld devices such as cell phones.
  - **Java FX**
    - JavaFX can be used to create rich Internet applications using a lightweight user-interface API. Its applications may be clients of Java EE platform services.

Lesson 2

IDE and Eclipse

Java Integrated Development Environments

IDE is the **Integrated Development Environment** that provides the user interface for code development, and testing and debugging features. It helps to organize the project artifacts that are relevant to the source code of the software application. The IDEs also have the functionalities to **compile** and **interpret** the program, and IDE has the ability to monitor resources like memory usage and checking hard disk space, etc.

- Popular IDEs for Java:
  - IntelliJ IDEA - Best IDE for Java (commercial version).
  - Eclipse - Most popular Java IDE.
  - JDeveloper - Java IDE for everything Oracle.
  - Android Studio - Free Java IDE for Android apps.
  - Visual Studio Code.
  - NetBeans - Out of date and no longer used Java IDE (Free Tier).

Eclipse Integrated Development Environment

- The Eclipse IDE is defined as a platform for developing the computer-based applications using various programming language such as Java, Python, C/C++, Ruby, and many more. The Eclipse IDE Java-based programming is done in this platform. There are several plugins that can be installed in the platform. The advanced client applications can be developed.
- The Eclipse IDE provides the **workspace** in which a user can bundle all of the projects in a single workspace. In that single workspace, the ***source files, artifacts, and images*** can all be stored in the workspace. The user has the complete functionality to select the name of the workspace and manage the projects in a single workspace.

Click here for more details about Eclipse IDE.

Click here to review and download the Eclipse installation guide.

Lesson 3

Java Program Overview

Anatomy of a Java Program

Let's explore the anatomy of a Java Program.

- Class name.
- Main method.
- Statements.
- Statement terminator.
- Reserved words.
- Curly Braces
- Parentheses
- Inline Comments
- Double Quotation Marks

Class Name

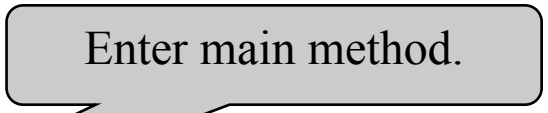
- **What is Class in Java?**
  - Any real life object which is a Noun is a **class**, for example Car, Shape, Bank etc.
- Every Java program must have at least **one *class***.
- Each class has a **name**.
  - By convention, class names start with an **uppercase** letter.
- In this example, the class name is ***Welcome***.

```
// This program prints "Welcome to Java!"
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Note: Upcoming lectures will cover Java Classes in greater detail.

Entering the **main()** Method

- In order to run a class, the class must contain a method named ***main()***.
- In the below code, line 3 defines the ***main()*** *method*.
- The program execution starts with the ***main()*** method. This is the ***entry point*** for the JVM.




```
1.  // This program prints Welcome to Java!
2.  public class Welcome {
3.    public static void main(String[] args) {
4.      System.out.println("Welcome to Java!");
5.    }
6.  }
```

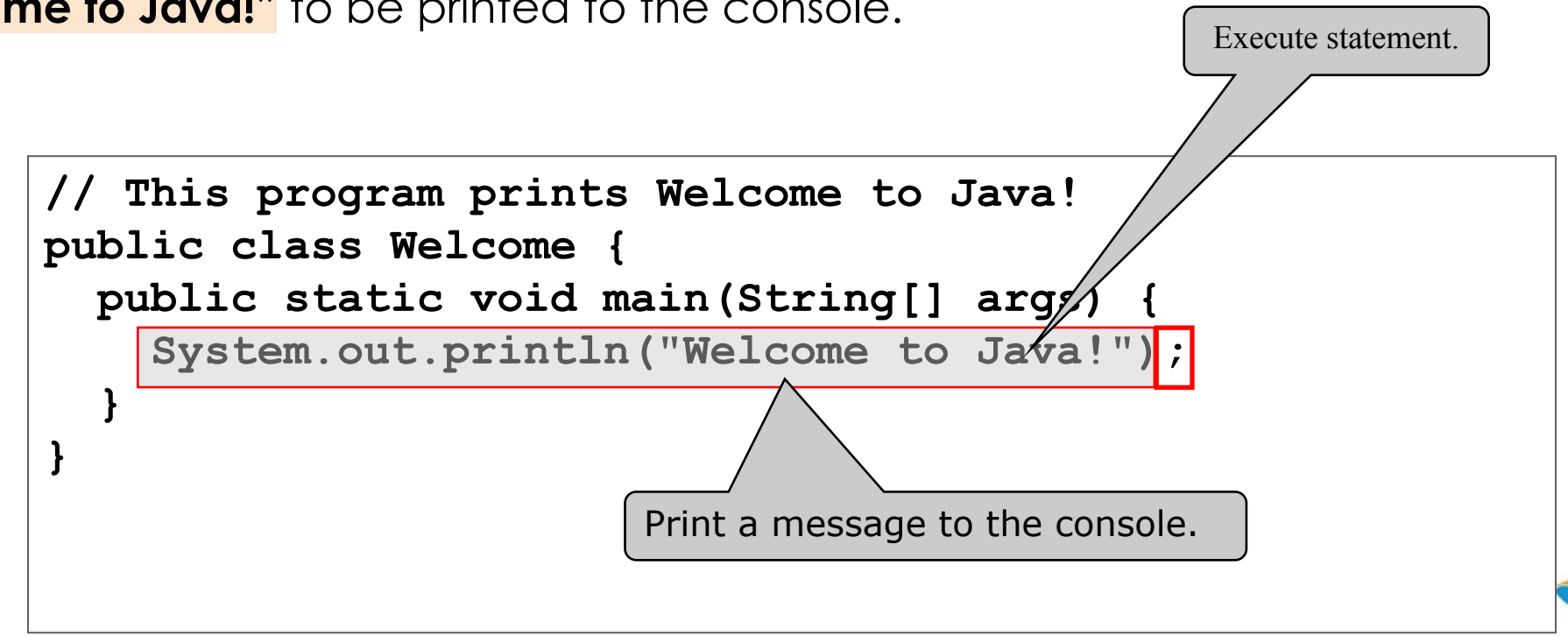
Note: Upcoming lectures will cover methods in greater detail.

continue…

Executing the **Statement**

- A *statement* represents an action or a sequence of actions. Every statement in Java ends with a semicolon **(;)**.

- The statement `System.out.println("Welcome to Java!");` in this program causes **"Welcome to Java!"** to be printed to the console.



Reserved Words

- Reserved words are words that cannot be used as variable names in a Java program because they are part of the syntax of the Java programming language.
  - Examples of Reserved Keywords: public, class, and static.
  - The complete list of reserved words (Java Language Keywords) can be found here.

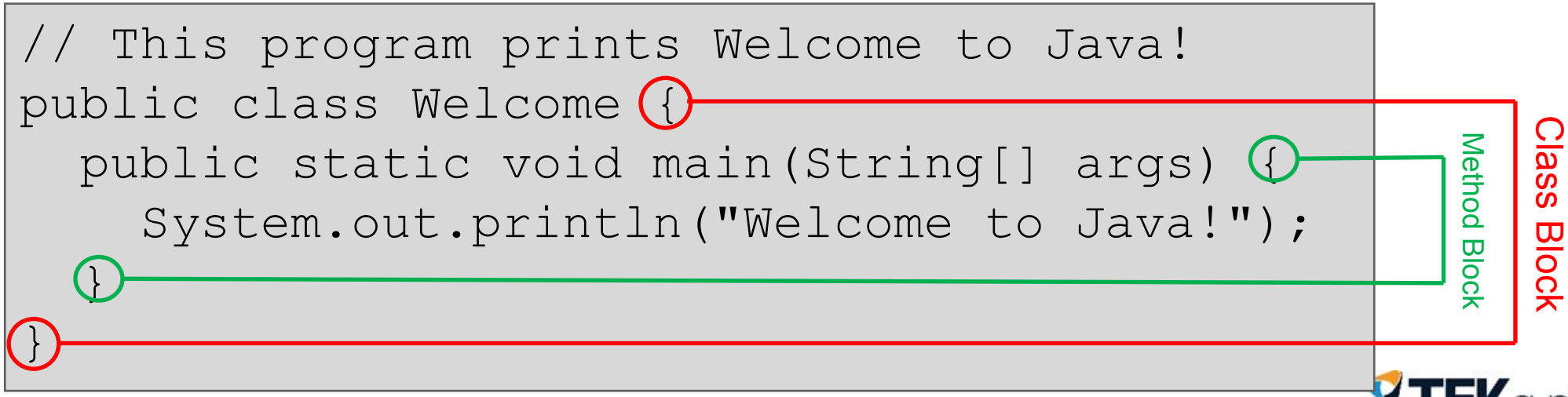
```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Special Symbols

Character	Name	Description
{}	Opening and closing braces	Denotes a block to enclose statements.
()	Opening and closing parentheses	Used with methods.
[]	Opening and closing brackets	Denotes an array.
//	Double slashes	Precedes a comment line.
" "	Opening and closing quotation marks	Enclosing a string (i.e., sequence of characters).
;	Semicolon	Marks the end of a statement.

Curly Braces

- A pair of curly braces in a program forms a code block that groups statements together.
- Curly braces have many uses in Java. In general, they define discrete blocks of code.
- Curly braces are always paired.



Parentheses

Parentheses are used to define methods and make calls to methods.

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Inline Comments

- Two forward slashes (//) begin an inline comment.
- Code comments provide information to the programmer.
- The compiler ignores them.

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Double Quotation Marks

Double quotation marks ("...") are used around string literals.

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Lab - Set Up a Project in Eclipse

Please visit the "Lab - 303.1.0 - How to set up a project in Eclipse and run this program."

You can find this lab on Canvas, under the **Guided Lab section**. If you have technical questions while performing the lab activity, ask your instructors for assistance.

Lesson 4

Best Practices - Programming Style and Documentation

Programming Style and Documentation

- Appropriate comments.
- Naming conventions.
- Proper indentation and spacing lines.
- Block styles.
- Programming/Java errors.

Appropriate Comments

- Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.
- At the beginning of the program, include your name, class section, instructor, date, and a brief description of the program:
  - Single-line comment

```
// This would be a single-line comment.
```

  - Multi-line comment

```
/*
This would be a multi-line comment; it could span many lines.
*/
```

Naming Conventions: Classes

- Every Java program has at least **one *class***. Larger Java programs include **many classes**.
- All executable code in a Java program is written inside of a method, inside of a class.
- Class names should be in Upper ***CamelCase***. Try to use nouns. A class is normally representative of something in the real world:
  - class CustomerAccount.
  - class HttpClient.
  - class BrowserConfiguration.

Naming Conventions: Packages

- The Java classes and other program resources (e.g., images and audio used by the program) can be bundled into Packages. Java classes are assigned to Packages, based on the **`package`** keyword at the top of the Java file and by their location in a folder structure.
- Package names should be written in lowercase. With small projects that only have a few packages, give them simple and meaningful names: For example:
  - **`package pokeranalyzer.`**
  - **`package mycalculator.`**
- Normally, the package names are broken-down. For example, for large projects, the package name may start with the company domain before being split into layers or features:
  - **`package com.mycompany.utilities.`**
  - **`package org.teksystem.training.javadeveloper.`**

Naming Conventions: Methods and Variables

- **Methods:** Names should be written in Lower ***CamelCase***, and include verbs that describe what the method does:
  - `void calculateTax(), string getSurname()`
- **Variables:** Names should be written in Lower ***CamelCase*** and listed in alphabetical character:
  - The names should represent what the value of the variable represents:
    - `string firstName, int orderNumber`
  - Only use very short names when the variables are short-lived, such as for loops:
    - `for (int i=0; i<20;i++) { //i is only visible in this code block}`
- **Constants:** By convention, constant names should be written in Uppercase:
  - `static final int DEFAULT_WIDTH.`
  - `static final int MAX_HEIGHT.`

Note: Upcoming lectures will cover this material in greater detail.

Naming Conventions: Interfaces

- **Interfaces:** Names should be written in Upper ***CamelCase***.
- **Interfaces** tend to have a name that describes an operation that a class can perform:
  - `interface Comparable;`
  - `interface Enumerable;`
- Note that some programmers like to distinguish interfaces by beginning the name with an "I":
  - `interface IComparable;`
  - `interface IEnumerable`

Proper Indentation and Spacing

- Indentation:
  - Indentation is critical to a well-crafted code; it makes code *readable*.
  - Be aware of whether you are using tabs or spaces to indent. Tabs give programmers more flexibility in how their code appears in their preferred editor; but in some shops, use of spaces is the norm.

- Spacing:
  - Use a blank line to separate segments of the code.

Block Styles

End-of-line style is the norm in many Java shops; although, the extra white space of next-line style is more readable to many developers.

Next-Line Style

```
Public class Test
{
    public static void main(String[] args)
    {
        System.out.println("Block Styles");
    }
}
```

```
Public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

End-of-Line Style

Summary

In this presentation, we took a brief look at the basics of Java programming language. We learned the basic concepts of Java that you should know to start programming in Java, including:

- Java history and the Java environment.
- Java fundamentals.
- IDE and Eclipse.
- Best Practices.

.

Questions?

End of Module