Characteristics of Java

Let's explore the key characteristics that make Java programs unique and powerful. These include its platform independence, robust memory management, and strong security features.

Platform independence (Write Once, Run Anywhere)

Java code is compiled into bytecode that can run on any device with a Java Virtual Machine (JVM), making it **platform-independent**. This means developers don't need to rewrite code for different operating systems, making Java especially popular for cross-platform applications.

Java is simple

Java is partially modelled on C++, but greatly simplified and improved. Some people refer to Java as "C++--" because it is like C++ but with more functionality and fewer negative aspects.

Java is object-oriented

Java is inherently object-oriented. Although many object-oriented languages began strictly as procedural languages, Java was designed from the start to be object-oriented. Object-oriented programming (OOP) is a popular programming approach that is replacing traditional procedural programming techniques.

One of the central issues in software development is how to reuse code. OOP provides great flexibility, modularity, clarity and reusability through encapsulation, inheritance and polymorphism.

Large standard library

Java provides a comprehensive standard library (Java Standard Library) that includes built-in support for data structures, networking, file input and output (I/O), database connections, and more. This library simplifies development by providing pre-built functionality that developers can leverage.

Java is distributed

Distributed computing involves several computers working together on a network. Java is designed to make distributed computing easy. Since networking capability is inherently integrated into Java, writing network programs is like sending and receiving data to and from a file.

Java is interpreted

You need an interpreter to run Java programs. The programs are compiled into the Java Virtual Machine code, which is called bytecode. The bytecode is machine-independent and can run on any machine that has a Java interpreter, which is part of the Java Virtual Machine (JVM).

Java is robust

Java compilers can detect many problems that would first show up later, at execution time, in other languages. Java has eliminated certain types of error-prone programming constructs found in other languages. Java has a runtime exception-handling feature to provide programming support for robustness.

Java is secure

Java implements several security mechanisms to protect your system against harm caused by stray programs.

Java is architecture-neutral

With a Java Virtual Machine (JVM), you can write one program that will run on any platform.

Java is portable

Because Java is architecture-neutral, Java programs are portable. They can be run on any platform without being recompiled.

Java's performance

Although Java is not as fast as C++, the Java just-in-time compiler improves runtime performance.

Java is multithreaded

Multithread programming is smoothly integrated in Java, whereas in other languages you have to call procedures specific to the operating system to enable multithreading.

Java is dynamic

Java was designed to adapt to an evolving environment. New code can be loaded on the fly without recompilation. There is no need for developers to create, and for users to install, major new software versions. New features can be incorporated transparently as needed.

Automatic garbage collection

Java manages memory automatically, freeing unused objects.

Let's now demonstrate a simple Java program, known as 'Hello World'. First we will look at the syntax of the program, then Christos will demonstrate how it works.

Citation:----(This document references content from the following URL--
https://www.coursera.org/learn/learn-java-programming/supplement/Otf0p/characteristics-of-java)

Anatomy of a Java program

Even though we will discuss some aspects of a Java program (in lecturers see notes), it's crucial for you to understand each part in depth, because every component has a specific role that ties into the program's functionality and reflects real-world applications. In this way you can appreciate how the pieces come together to create software that runs efficiently and effectively.

All Java programs have examples of the features below.

- Class name
- Main method
- Statement
- Statement terminator
- Reserved words
- Comments
- Blocks

Let's look at each of these features so that you can identify them more easily in the future.

Class name

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!");
  }
}
```

Main method

In the example below, Line 2 (in bold) defines the main method. In order to run a class, the class must contain a method named main.

public static void main(String[] args){

The program is executed from the main method.

```
public class Welcome {
  public static void main(String[] args) { // Your program always starts here

    System.out.println("Welcome to Java!");

  }
}
```

Statement

A statement represents an action or a sequence of actions.

The statement **System.out.println("Welcome to Java!")** in the code below is a statement to display the greeting "Welcome to Java!"

```
public class Welcome {
  public static void main(String[] args) {

    System.out.println("Welcome to Java!"); // This line is an example of a statement

  }
}
```

Statement terminator

Every statement in Java ends with a semicolon (**;**).

```
public class StatementTerminator {
  public static void main(String[] args) {

    System.out.println("A semicolon is printed after this statement");
```

```
    System.out.println("A semicolon is also printed after this statement");
    System.out.println("Finally, a semicolon is printed at the end of this statement");

  } // No semicolons are needed to close blocks of code
}
```

Reserved words

Reserved words or keywords are words that have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word **class**, it understands that the word after class is the name for the class.

Examples of reserved words are:

abstract	**default**	**goto**	**package**	**this**
assert	**do**	**if**	**private**	**throw**
boolean	**double**	**implements**	**protected**	**throws**
break	**else**	**import**	**public**	**transient**
byte	**enum**	**instanceof**	**return**	**true**
case	**extends**	**int**	**short**	**try**
catch	**false**	**interface**	**static**	**void**
char	**final**	**long**	**strictfp**	**volatile**
class	**finally**	**native**	**super**	**while**
const	**float**	**new**	**switch**	
continue	**for**	**null**	**synchronized**	

A full list of reserved keywords can be found on the Oracle website.

Comments

Comments can be used to explain Java code, and to make it more readable. They can also be used to prevent execution when testing alternative code. We can use the following types.

Single-line comments

Single-line comments start with two forward slashes (//). Any text between // and the end of the line is ignored by Java (will not be executed).

This is an example of a single-line comment:

```
// This is a Single-line comment

System.out.println("Welcome to Java!");

// This comment will be ignored
```

Multi-line comments

Multi-line comments start with **/*** and end with ***/**

Any text between /* and */ will be ignored by Java.

This is an example of a multi-line comment:

```
/* The code below will print the
 Welcome to Java message to the screen*/


System.out.println("Welcome to Java!");
```

Blocks

A pair of braces **{ }** in a program forms a block that groups components of a program.

```
public class Test {              // Start of class block


  public static void main(String[] args){    // Start of method block
      System.out.println("Welcom to Java!");
  }                              // End of method block


}                        // End of class block
```

Class block

- The outermost pair of curly braces {} defines the class block for the Test class.
- Everything inside these braces belongs to the Test class.

Method block

- Inside the class block, there's a method called main. The main method is the entry point of any Java application.
- The curly braces {} following **public static void main(String[] args)** define the method block.
- Everything inside these braces belongs to the main method.