Unit I

History of Java

Java programming language was originally developed by Sun Microsystems.
Java programming development process initiated by James Gosling.
Java first release Java 1.0 [J2SE] in 1995.
The latest release of the Java Standard Edition is Java SE 8.

Java is guaranteed to be **Write Once, Run Anywhere**.

Features of java.

Object Oriented: In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

Platform Independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

Architecture-neutral: Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

Portable: Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

Robust: Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Multithreaded: With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

High Performance: With the use of Just-In-Time compilers, Java enables high performance.

Distributed: Java is designed for the distributed environment of the internet.

Dynamic: Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts:

1. Polymorphism
2. Inheritance
3. Encapsulation
4. Abstraction

Encapsulation

Encapsulation means putting together all the variables (instance variables) and the methods into a single unit called Class. It also means hiding data and methods within an Object.

Encapsulation provides the security that keeps data and methods safe from inadvertent changes. Programmers sometimes refer to encapsulation as using a "black box," or a device that you can use without regard to the internal mechanisms. A programmer can access and use the methods and data contained in the black box but cannot change them.

In other words: Wrapping up of data and functions into a single unit is called encapsulation.

Polymorphism

Java Polymorphism is one of easy concept to understand. Polymorphism definition is that Poly means many and morphos means forms.

Java provides two ways to implement polymorphism

1. Static Polymorphism (compile time polymorphism/ Method overloading):
The ability to execute different method implementations by altering the argument used with the method name is known as method overloading.

2. Dynamic Polymorphism (run time polymorphism/ Method Overriding):
When you create a subclass by extending an existing class, the new subclass contains data and methods that were defined in the original super class.

In other words, any child class object has all the attributes of its parent. Sometimes, however, the super class data fields and methods are not entirely appropriate for the subclass objects; in these cases, you want to override the parent class members.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

Inheritance

Inheritance is the ability to create classes that share the attributes and methods of existing classes, but with more specific features. Inheritance is mainly used for code reusability.

A First Simple Program:

```
/*
This is a simple Java program.
Call this file "Example.java".
*/
class Example {
// Your program begins with a call to main().
public static void main(String args[]) {
System.out.println("This is a simple Java program.");
                    }
}
```

Compiling the Program

- To compile the **Example** program, execute the compiler, **javac** specifying the name of the
- source file on the command line, as shown here:
- C:\>javac Example.java
- The **javac** compiler creates a file called **Example.class** that contains the byte code version of the program.
- The Java bytecode is the intermediate representation of program that contains instructions the Java Virtual Machine will execute.
- The byte code is not directly executed so need to run the byte code through
- Java interpreter called **java**.
- C:\>java Example
- When the program is run, the following output is displayed:
- This is a simple Java program.

A Closer Look at the First Sample Program

Lets have a look at simple program, line by line.

- The first two lines are multiline comments wgich are enclosed between /* and */

```
/*
This is a simple Java program.
Call this file "Example.java".
*/
```

- The next line is all about creating a class, here class is keyword and Example is identifier. The content of the class needs to put between opening curly brace ({) and the closing curly brace (}).

```
class Example {
        class definition comes here.
}
```

- The next line in the program is the *single-line comment, as* shown below:

```
// Your program begins with a call to main().
```

- All Java applications begin execution by calling main() method, lets have a look at different parts of main function

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

  - public : The **public** keyword is an *access specifier,* which allows the programmer to control the visibility of class members. When a class member is preceded by **public**, then that member may be accessed by code outside the class in which it is declared.

    The opposite of the public is private.

  - Static : The keyword **static** allows **main( )** to be called without having to instantiate a particular instance of the class. This is necessary since **main( )** is called by the Java Virtual Machine before any objects are made.

  - Void : The keyword **void** simply tells the compiler that **main( )** does not return a value.

  - **main( )** is the method called when a Java application begins. Keep in mind that Java is case-sensitive. Thus, **Main** is different from **main**.

  - Output statement is as follows

    ```
    System.out.println("This is a simple Java program.");
    ```

    - System is a predefined class that provides access to the System.

- out is the output stream that is connected to the console.
- Println is the output function which display string arguments in the concole.

A Second Short Program

```
/*
Here is another short example.
Call this file "Example2.java".
*/
class Example2 {
public static void main(String args[]) {
int num; // this declares a variable called num
num = 100; // this assigns num the value 100
System.out.println("This is num: " + num);
num = num * 2;
System.out.print("The value of num * 2 is ");
System.out.println(num);
}
}
```

When you run this program, you will see the following output:

```
This is num: 100
The value of num * 2 is 200
```

Line by line explanation of the program is as below.

- int num; : This declares a variable called num of type int.
- num = 100; This assigns num the value 100.
- System.out.println("This is num: " + num); : Displaying value of the variable num.
- num = num * 2; Here ***** operator to indicate multiplication. After this line executes, num will contain the value 200.

- System.out.print("The value of num * 2 is ") : Printing the line “The value of num * 2 is” on console, and line break is not happening here since we used print() instead of println()

- System.out.println(num): Printing the value of num with line break.

Sign Up

And

Download

Full

Notes

☺☺☺