Module 1

The History and Evolution of Java: The Byte code, Features of Java An overview of Java: Object-Oriented Programming, Structure of a Java program, Data Types and Variables, Type conversion and casting, Arrays

Classes: Fundamentals, Declaring Objects, Assigning Object Reference Variables, Methods, Constructors, this Keyword, Garbage Collection, Stack application

Methods and Classes: Overloading Methods, Using Objects as Parameters, Argument Passing, Returning Objects, Access Control, static, final, Command-Line Arguments

The History and Evolution of Java

Java is related to C++, which is a direct descendant of C. Much of the character of Java is inherited from these two languages. From C, Java derives its syntax. Many of Java's object oriented features were influenced by C++.

C Language :

- Fundamentally changed the way programming was approached.
- Structured, efficient, high-level language that could replace assembly code.
- Invented and first implemented by Dennis Ritchie on UNIX operating system

C++

- The increasing complexity of programs has driven the need for better ways to manage that complexity.
- With structured programming methods, once a project reaches a certain size, its complexity exceeds.
- To solve this problem, a new way to program was invented, called *object-oriented programming (OOP)*.
- OOP is a programming methodology that helps organize complex programs through the use of inheritance, encapsulation, and polymorphism.
- C++ was invented by Bjarne Stroustrup in 1979.
- C++ extends C by adding object-oriented features.
- It includes all of C's features, attributes, and benefits.

Java

- Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems.
- The primary motivation was the need for a platform-independent (that is, architecture-neutral) language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls.
- The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target
- The problem is that compilers are expensive and time-consuming to create.
- While java was being developed the world wide web was emerging in parallel and web demanded portable programs.
- Java derives much of its character from C and C++.

How Java changed the internet

- Java had a profound effect on the Internet.
- Java simplified web programming and also made networked programs available through servlets and applets.

Applets

- An *applet* is a special kind of Java program that is designed to be transmitted over the Internet and automatically executed by a Java-compatible web browser.
- An applet is downloaded on demand, without further interaction with the user.
- Applets are intended to be small programs. They are typically used to display data provided by the server, handle user input, or provide simple functions, such as a loan calculator, that execute locally.
- Applet is a dynamic, self-executing program.
- Such a program that is downloaded automatically and run should be prevented from doing any harm and also be able to run in a variety of different environments and under different operating systems.

- Security is provided by confining an applet to the Java execution environment and not allowing it access to other parts of the computer.

The ByteCode

- The output of a Java compiler is not executable code, rather, it is bytecode.
- *Bytecode* is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the *Java Virtual Machine (JVM)*.
- Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments because only the JVM needs to be implemented for each platform.
- Bytecode has been highly optimized, the use of bytecode enables the JVM to execute programs much faster than one might expect.
- When a JIT compiler is part of the JVM, selected portions of bytecode are compiled into executable code in real time, on a piece-by-piece, demand basis.
- JIT compiler compiles code as it is needed, during execution.

Servlets

- A servlet is a small program that executes on the server.
- Just as applets dynamically extend the functionality of a web browser, servlets dynamically extend the functionality of a web server.
- Servlets are used to create dynamically generated content that is then served to the client.
- The servlet offers several advantages, including increased performance.
- Servlets (like all Java programs) are compiled into bytecode and executed by the JVM, they are highly portable. Thus, the same servlet can be used in a variety of different server environments.

The Java Buzzwords

1. Simple
    a. Java was designed to be easy for the professional programmer to learn and use effectively.
    b. With knowledge of C and Object oriented principles Java is a simple language to be learnt.
2. Secure
    a. Java confines the running of code to JVM only and does not let it to access the other parts of the computer.
3. Portable
    a. Java is known as a portable language because Java code can execute on all major platforms.
    b. Bytecode, A highly optimized set of instructions makes java a portable programming language.
    c. Java bytecode on any hardware that has a compliant JVM.
4. Object oriented
    a. The object oriented principles are Encapsulation, Abstraction, Inheritance and polymorphism.
    b. Encapsulation refers to binding together of code and data.
    c. Abstraction refers to hiding of complex features.
    d. Inheritance is a property through which one class can obtain the features of the other.
    e. Polymorphism means “One entity multiple forms”.
    f. The objected oriented model allows the user to incorporate all the features mentioned above into the code.
    g. Everything in Java is considered an object and hence programs are modular.

5. Robust
    a. Java restricts the programmer in a few key areas thus helping them identify mistakes early in the program development.
    b. Compile time checks helps identifying programming errors.
    c. Run time errors can be checked through Exceptions handling mechanism.
    d. Memory management is a tedious task in C and C++. The programmer needs to manually allocate and free the dynamic memory.

e. Java's memory allocation is simple and deallocation of memory is automatic through garbage collection.
f. Exceptional conditions like file not found, array index out of bound, divide by zero can be easily handled through Exception handling mechanisms.

6. Multithreaded
    a. Java allows multithreading i.e. it allows the user to write programs that can do many things simultaneously.
    b. The Java run time provides solutions for multiprocess synchronizations that helps the user to construct interactive programs.
7. Architecture neutral
    a. " Write once run anywhere , anytime forever" was the goal with which Java was created and the goal is accomplished to a great extent.
    b. Operating system upgrades , processor upgrades and changes in the core system resources do not affect the bytecode runs.
8. Interpreted and high performance
    a. Java enables creation of cross platform programs by compiling into an intermediate representation called the bytecode.
    b. This code can be run on any machine with a JVM.
    c. Java bytecode was carefully designed so that it would be easy to translate directly into native machine code by using the Just in time compiler.
9. Distributed
    a. Java can handle the TCP/IP protocols.
    b. Java also supports Remote Method Invocation which enables a program to invoke methods across network.
10. Dynamic
    a. Java programs carry substantial amounts of run time information that is used to verify and resolve access to objects at run time.
    b. This helps in linking the code dynamically in a safe manner thus making the language robust.

An Overview of Java

Object-Oriented Programming

- Object-oriented programming (OOP) is at the core of Java.
- All computer programs consist of two elements: code and data.
- To manage increasing complexity *object-oriented programming* was introduced.
- Object-oriented programming organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data.
- An object-oriented program can be characterized as *data controlling access to code*.

OOP Principles

1. **Abstraction :**
    - An essential element of object-oriented programming is *abstraction*.
    - Data abstraction is the process of hiding certain details and showing only essential information to the user.
    - It helps to reduce programming complexity and effort.
    - For example, people do not think of a car as a set of tens of thousands of individual parts. They think of it as a well-defined object with its own unique behaviour.
    - They can ignore the details of how the engine, transmission, and braking systems work. Instead, they are free to utilize the object as a whole.
    - In Java abstraction is achieved through abstract classes and interface**.**

2. **Encapsulation**
    - Encapsulation is the mechanism that binds together code and the data it manipulates keeps data and code safe from outside interference and misuse.
    - Encapsulation can be thought of as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper.
    - It is also known as “Data hiding”.
    - In Java, the basis of encapsulation is the class. Each method or variable in a class may be marked private or public.

- The *public* interface of a class represents everything that external users of the class may know.
- The *private* methods and data can only be accessed by code that is a member of the class.

3. Inheritance

- Inheritance is the process by which one object acquires the properties of another object.
- For example, a Golden Retriever is part of the classification *dog*, which in turn is part of the *mammal* class, which is under the larger class *animal*.
- The idea behind inheritance in Java is that one can create new classes that are built upon existing classes.
- When the user inherits from an existing class, he can reuse methods and fields of the parent class. Also, one can add new methods and fields in the current class.
- Inheritance facilitates Reusability and hence is an important concept of OOPs.

4. Polymorphism

- The word polymorphism means having many forms.. Polymorphism allows us to perform a single action in different ways.
- For Example: Consider a stack (which is a last-in, first-out list). There might be a need for 3 types of stack, one for integer values one for double and one for string. The algorithm that implements each stack is the same, even though the data being stored differs. In a non–object-oriented language, one had to create three different sets of stack routines, with each set using different names. However, because of polymorphism, in Java one can specify a general set of stack routines that all share the same names.
- Polymorphism in java can be implemented by overloading and overriding.

A Simple Java program

```
/* This is a simple Java program. */
```

```
class Example {

public static void main(String args[]) {

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

To compile

C:\>javac Example.java

The **javac** compiler creates a file called **Example.class** that contains the bytecode version of the program

To run the code call the java interpreter: C:\>java Example

Output : This is a simple Java program.

/* This is a simple Java program*/ : This is a *comment*. Like most other programming languages, Java lets one to enter a remark into a program's source file. The contents of a comment are ignored by the compiler.

class Example { : This line uses the keyword **class** to declare that a new class is being defined. **Example** is an *identifier* that is the name of the class. The entire class definition, including all of its members, will be between the curly braces { }.

public static void main(String args[ ]) : This line begins the **main( )** method. The **public** keyword is an *access modifier*, 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 keyword **static** allows **main( )** to be called without having to instantiate a particular instance of the class. The keyword **void** simply tells the compiler that **main( )** does not return a value. **String args[ ]** declares a parameter named **args**, which is an array of instances of the class **String**.

Data types and variables.

Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean.

These can be put in four groups:

• **Integers**: This group includes byte, short, int, and long, which are for whole-valued signed numbers.

• **Floating-point numbers**: This group includes float and double, which represent numbers with fractional precision.

• **Characters**: This group includes char, which represents symbols in a character set, like letters and numbers.

• **Boolean:** This group includes boolean, which is a special type for representing true/false values.

Type Conversion and Casting

It is possible to assign a value of one type to a variable of another type. If the two types are compatible, then Java will perform the conversion automatically. For example, it is always possible to assign an **int** value to a **long** variable. However, not all types are compatible, and thus, not all type conversions are implicitly allowed. For instance, there is no automatic conversion defined from **double** to **byte**. But it is still possible to obtain a conversion between incompatible types. To do so, a cast must be used, which performs an explicit conversion between incompatible types.

Java's Automatic Conversions

When one type of data is assigned to another type of variable, an *automatic type conversion* will take place if the following two conditions are met:

- The two types are compatible.
- The destination type is larger than the source type.

When these two conditions are met, a *widening conversion* takes place. For example, the **int** type is always large enough to hold all valid **byte** values, so no explicit cast statement is required.

For widening conversions, the numeric types, including integer and floating-point types, are compatible with each other. However, there are no automatic conversions from the numeric types to **char** or **boolean**. Also, **char** and **boolean** are not compatible with each other.

Casting Incompatible Types

Although the automatic type conversions are helpful, they will not fulfill all needs. For example, if there is a need to assign an **int** value to a **byte** variable, this conversion will not be performed automatically, because a **byte** is smaller than an **int**. This kind of conversion is called a *narrowing conversion*, since the value is explicitly made narrower, so that it will fit into the target type. To create a conversion between two incompatible types, a cast must be used. A *cast* is simply an explicit type conversion. It has this general form:

(*target-type*) *value*

Here, *target-type* specifies the desired type to convert the specified value to. For example, the following fragment casts an **int** to a **byte**. If the integer's value is larger than the range of a **byte**, it will be reduced modulo (the remainder of an integer division by the) **byte**'s range.

```
int a;
byte b;
// …
b = (byte) a;
```

A different type of conversion will occur when a floating-point value is assigned to an integer type: *truncation*. Integers do not have fractional components. Thus, when a floating-point value is assigned to an integer type, the fractional component is lost.

```
class Conversion {
public static void main (String args[]) {
        byte b;
        int i = 257;
        double d = 323.142;
        System.out.println("\nConversion of int to byte.");
        b = (byte) i;

        System.out.println("i and b " + i + " " + b);
        System.out.println("\nConversion of double to int.");
        i = (int) d;

        System.out.println("d and i " + d + " " + i);
        System.out.println("\nConversion of double to byte.");
        b = (byte) d;
        System.out.println("d and b " + d + " " + b);
        }
}
```

This program generates the following output:

```
Conversion of int to byte.
i and b 257 1
Conversion of double to int.
d and i 323.142 323
Conversion of double to byte.
d and b 323.142 67
```

When the value 257 is cast into a byte variable, the result is the remainder of the division of 257 by 256 (the range of a byte), which is 1 in this case. When the d is converted to an int, its fractional component is lost. When d is converted to a byte, its fractional component is lost, and the value is reduced modulo 256, which in this case is 67.

Arrays

An *array* is a group of like-typed variables that are referred to by a common name. A specific element in an array is accessed by its index. The general form of a one-dimensional array declaration is *type var-name[ ];*

```
int month_days[];
```

Although this declaration establishes the fact that **month_days** is an array variable, no array actually exists. In fact, the value of **month_days** is set to **null**, which represents an array with no value. To link **month_days** with an actual, physical array of integers, user must allocate one using **new** and assign it to **month_days**. **new** is a special operator that allocates memory.

```
month_days = new int[12];
```

A simple program on array

```
class Array {
public static void main(String args[]) {
int a[] = new int[5];
for(int i=0; i<4; i++){
a[i] = i;
System.out.println(a[i]);
        }
}
```

Declaring a 2D array:

```
int twoD[][] = new int[4][5];
```

This allocates a 4 by 5 array and assigns it to twoD.