RAMAIAH Institute of Technology

# Object Oriented Programming

## Lecture Notes – Module 1

**The Object Model Foundations of the Object Model:** Object-Oriented Programming, Object-Oriented Design, Object-Oriented Analysis.

**Elements of the Object Model:** Abstraction, Encapsulation, Modularity, Hierarchy, Typing, Concurrency, Persistence Applying the Object Model.

**Introduction to Java Programming:** Java Buzzwords, Overview of Java Datatypes, Variables, arrays, Control statements.

**Text Books:**


1. Object-Oriented Analysis And Design With applications, Grady Booch, Robert A Maksimchuk, Michael W Eagle, Bobbi J Young, 3rd Edition, 2013, Pearson education, ISBN :978-81-317-2287-93. 20
2. The Complete Reference - Java, Herbert Schildt 10th Edition, 2017, TMH Publications, ISBN: 9789387432291.


**Reference Book:**


1. Head First Java, Kathy Sierra and Bert Bates, 2nd Edition, 2014, Oreilly Publication , ISBN : 978817366602

## Chapter 1

**The Object Model Foundations of the Object Model:** Object-Oriented Programming, Object-Oriented Design, Object-Oriented Analysis.

## Object Oriented Programming

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:


- **Object**
- **Class**
- **Inheritance**
- **Polymorphism**
- **Abstraction**
- **Encapsulation**

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

- **Coupling**
- **Cohesion**
- **Association**
- **Aggregation**
- **Composition**

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

- Object

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects. Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

- Class

Collection of objects is called class. It is a logical entity. A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

- Inheritance


When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

- Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc. In Java, we use method overloading and method overriding to achieve polymorphism. Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

- 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.

- Encapsulation


Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines. A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

- Coupling

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

- Cohesion — Classes doing related work -High Cohesion, elseLow Cohesion — UTIL has more unwanted classes. IO has related to i/o classes

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

- Association — Object 1 association with one or more objs

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

- One to One
- One to Many
- Many to One, and
- Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many). Association can be unidirectional or bidirectional.

- Aggregation — Class A having Class B in its fns, where A and B are not related to each other

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a has-a relationship in Java. Like, inheritance represents the is-a relationship. It is another way to reuse objects.

Composition — A class with another obj as its member

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

Advantage of OOPs over Procedure-oriented programming language

1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

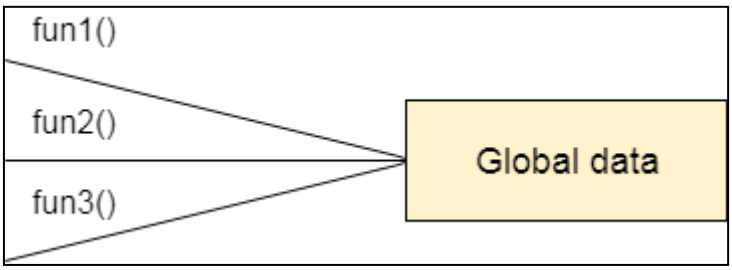



Figure: Data Representation in Procedure-Oriented Programming

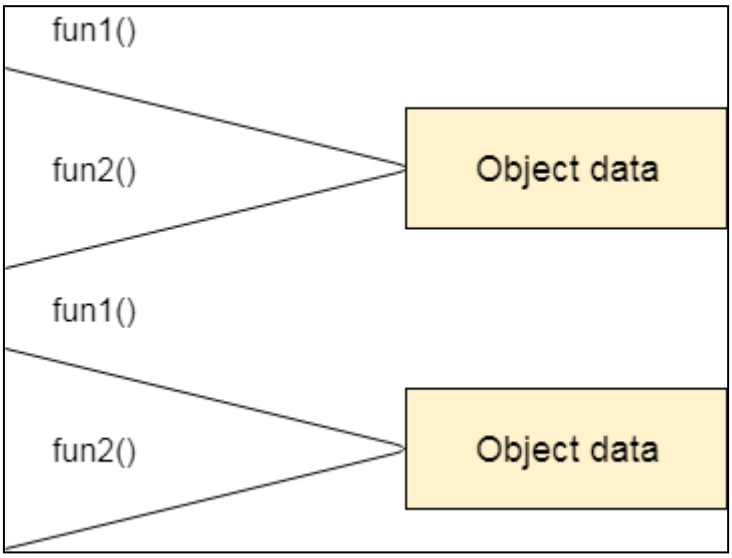



Figure: Data Representation in Object-Oriented Programming

3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

**Object-Oriented Design**

Object–Oriented Design (OOD) involves implementation of the conceptual model produced during object-oriented analysis. In OOD, concepts in the analysis model, which are technology–independent, are mapped onto implementing classes,

constraints are identified and interfaces are designed, resulting in a model for the solution domain, i.e., a detailed description of how the system is to be built on concrete technologies.

The implementation details generally include –

- Restructuring the class data (if necessary),
- Implementation of methods, i.e., internal data structures and algorithms,
- Implementation of control, and
- Implementation of associations.

Grady Booch has defined object-oriented design as *"a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design"*.

**Object-Oriented Analysis**

Object–Oriented Analysis (OOA) is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises of interacting objects.

The main difference between object-oriented analysis and other forms of analysis is that in object-oriented approach, requirements are organized around objects, which integrate both data and functions. They are modelled after real-world objects that the system interacts with. In traditional analysis methodologies, the two aspects - functions and data - are considered separately.

Grady Booch has defined OOA as, *"Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain"*.

The primary tasks in object-oriented analysis (OOA) are –

- Identifying objects
- Organizing the objects by creating object model diagram
- Defining the internals of the objects, or object attributes
- Defining the behavior of the objects, i.e., object actions Functions
- Describing how the objects interact

The common models used in OOA are use cases and object models.

## Chapter 2

**Elements of the Object Model:** Abstraction, Encapsulation, Modularity, Hierarchy, Typing, Concurrency, Persistence Applying the Object Model.

### Abstraction

Abstraction means to focus on the essential features of an element or object in OOP, ignoring its extraneous or accidental properties. The essential features are relative to the context in which the object is being used.

Grady Booch has defined abstraction as follows –

"An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer."

**Example** – When a class Student is designed, the attributes enrolment_number, name, course, and address are included while characteristics like pulse_rate and size_of_shoe are eliminated, since they are irrelevant in the perspective of the educational institution.

### Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. The class has methods that provide user interfaces by which the services provided by the class may be used.

### Modularity

Modularity is the process of decomposing a problem (program) into a set of modules so as to reduce the overall complexity of the problem. Booch has defined modularity as –

"Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules."

Modularity is intrinsically linked with encapsulation. Modularity can be visualized as a way of mapping encapsulated abstractions into real, physical modules having high cohesion within the modules and their inter–module interaction or coupling is low.

### Hierarchy

In Grady Booch's words, "Hierarchy is the ranking or ordering of abstraction". Through hierarchy, a system can be made up of interrelated subsystems, which can

have their own subsystems and so on until the smallest level components are reached. It uses the principle of "divide and conquer". Hierarchy allows code reusability.

The two types of hierarchies in OOA are –

- **"IS–A" hierarchy** – It defines the hierarchical relationship in inheritance, whereby from a super-class, a number of subclasses may be derived which may again have subclasses and so on. For example, if we derive a class Rose from a class Flower, we can say that a rose "is–a" flower.
- **"PART–OF" hierarchy** – It defines the hierarchical relationship in aggregation by which a class may be composed of other classes. For example, a flower is composed of sepals, petals, stamens, and carpel. It can be said that a petal is a "part–of" flower.

## Typing About data types

According to the theories of abstract data type, a type is a characterization of a set of elements. In OOP, a class is visualized as a type having properties distinct from any other types. Typing is the enforcement of the notion that an object is an instance of a single class or type. It also enforces that objects of different types may not be generally interchanged; and can be interchanged only in a very restricted manner if absolutely required to do so.

The two types of typing are –

- **Strong Typing** – Here, the operation on an object is checked at the time of compilation, as in the programming language Eiffel.
- **Weak Typing** – Here, messages may be sent to any class. The operation is checked only at the time of execution, as in the programming language Smalltalk.

## Concurrency

Concurrency in operating systems allows performing multiple tasks or processes simultaneously. When a single process exists in a system, it is said that there is a single thread of control. However, most systems have multiple threads, some active, some waiting for CPU, some suspended, and some terminated. Systems with multiple CPUs inherently permit concurrent threads of control; but systems running on a single CPU use appropriate algorithms to give equitable CPU time to the threads

so as to enable concurrency.

In an object-oriented environment, there are active and inactive objects. The active objects have independent threads of control that can execute concurrently with threads of other objects. The active objects synchronize with one another as well as with purely sequential objects.

Persistence Existence of classes and objs even after program ends.

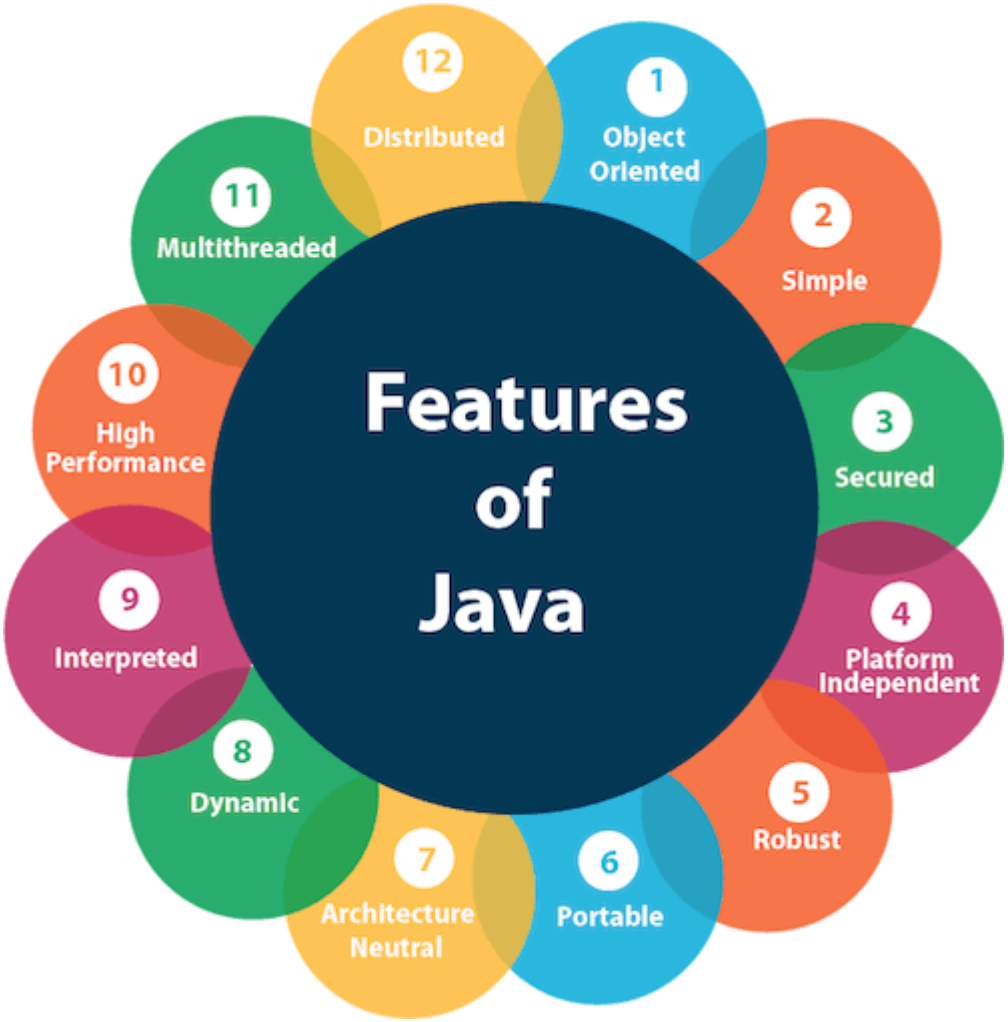
An object occupies a memory space and exists for a particular period of time. In traditional programming, the lifespan of an object was typically the lifespan of the execution of the program that created it. In files or databases, the object lifespan is longer than the duration of the process creating the object. This property by which an object continues to exist even after its creator ceases to exist is known as persistence.

## Chapter 3

**Introduction to Java Programming:** Java Buzzwords, Overview of Java Datatypes, Variables, arrays, Control statements.

### Java Buzzwords

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords. A list of the most important features of the Java language is given below.




- Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

---

- Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

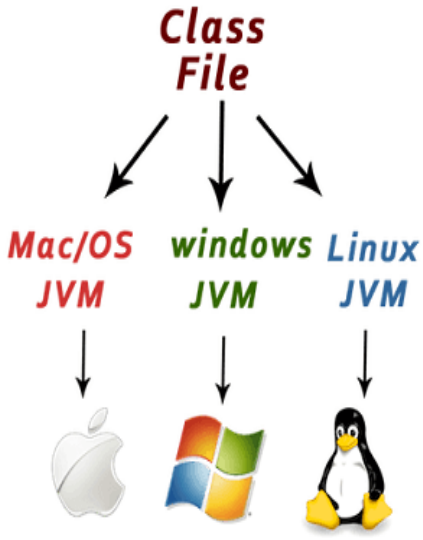
---

- Platform Independent

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two



components:

1. Runtime Environment
2. API(Application Programming Interface)

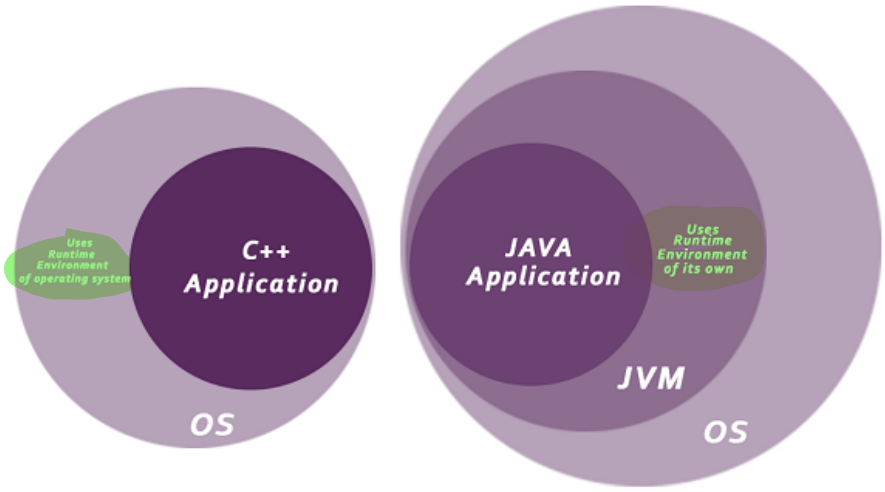
Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

---

- Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**




- **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be

provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

---

- Robust

The English mining of Robust is strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

---

- Architecture-neutral About bytes occupied by DATA TYPES

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

---

- Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

---

- High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

---

- Distributed

Java is distributed because it facilitates users to create distributed applications in

Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

---

- Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

---

- Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

**Overview of Java Datatypes**

First Java Program

Let us look at a simple code that will print the words *Hello World*.

Example

```
public class MyFirstJavaProgram
{
   /* This is my first java program.This will print 'Hello World' as the output*/
   public static void main(String []args) {
      System.out.println("Hello World"); // prints Hello World
   }
}
```

Let's look at how to save the file, compile, and run the program. Please follow the subsequent steps –

- Open notepad and add the code as above.
- Save the file as: MyFirstJavaProgram.java.
- Open a command prompt window and go to the directory where you saved the class. Assume it's C:\.

- Type 'javac MyFirstJavaProgram.java' and press enter to compile your code. If there are no errors in your code, the command prompt will take you to the next line (Assumption : The path variable is set).
- Now, type ' java MyFirstJavaProgram ' to run your program.
- You will be able to see ' Hello World ' printed on the window.

Output

C:\> javac MyFirstJavaProgram.java

C:\> java MyFirstJavaProgram

Hello World

## Basic Syntax

About Java programs, it is very important to keep in mind the following points.

- Case Sensitivity – Java is case sensitive, which means identifier Hello and hello would have different meaning in Java.
- Class Names – For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

Example: *class MyFirstJavaClass*

- Method Names – All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

Example: *public void myMethodName()*

- Program File Name – Name of the program file should exactly match the class name.

When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).

But please make a note that in case you do not have a public class present in the file then file name can be different than class name. It is also not mandatory to have a public class in the file.

Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*

- public static void main(String args[]) – Java program processing starts from

the main() method which is a mandatory part of every Java program.

## Java Identifiers Variables

All Java components require names. Names used for classes, variables, and methods are called identifiers.

In Java, there are several points to remember about identifiers. They are as follows:

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- A key word cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of legal identifiers: age, $salary, _value, __1_value.
- Examples of illegal identifiers: 123abc, -salary.

## Java Modifiers Access specifiers

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers –

- Access Modifiers – default, public , protected, private
- Non-access Modifiers – final, abstract, strictfp

We will be looking into more details about modifiers in the next section.

## Java Variables

Following are the types of variables in Java –

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non-static Variables)

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java –

- Primitive Data Types
- Reference/Object Data Types

## Primitive Data Types

There are eight primitive datatypes supported by Java. Primitive datatypes are

predefined by the language and named by a keyword. Let us now look into the eight primitive data types in detail.

byte

- Byte data type is an 8-bit signed two's complement integer
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50

short

- Short data type is a 16-bit signed two's complement integer
- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767 (inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer
- Default value is 0.
- Example: short s = 10000, short r = -20000

int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648 (-2^31)
- Maximum value is 2,147,483,647(inclusive) (2^31 -1)
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0
- Example: int a = 100000, int b = -200000

long

- Long data type is a 64-bit signed two's complement integer
- Minimum value is -9,223,372,036,854,775,808(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^63 -1)
- This type is used when a wider range than int is needed
- Default value is 0L

- Example: long a = 100000L, long b = -200000L

float

- Float data type is a single-precision 32-bit IEEE 754 floating point
- Float is mainly used to save memory in large arrays of floating point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency
- Example: float f1 = 234.5f

double

- double data type is a double-precision 64-bit IEEE 754 floating point
- This data type is generally used as the default data type for decimal values, generally the default choice
- Double data type should never be used for precise values such as currency
- Default value is 0.0d
- Example: double d1 = 123.4

boolean

- boolean data type represents one bit of information
- There are only two possible values: true and false
- This data type is used for simple flags that track true/false conditions
- Default value is false
- Example: boolean one = true

char

- char data type is a single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Char data type is used to store any character
- Example: char letterA = 'A'

Reference Datatypes

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy, etc.
- Class objects and various type of array variables come under reference datatype.

- Default value of any reference variable is null.
- A reference variable can be used to refer any object of the declared type or any compatible type.
- Example: Animal animal = new Animal("giraffe");

Java Literals Values

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

Literals can be assigned to any primitive type variable. For example –

```
byte a = 68;
char a = 'A';
```

byte, int, long, and short can be expressed in decimal(base 10), hexadecimal(base 16) or octal(base 8) number systems as well.

Prefix 0 is used to indicate octal, and prefix 0x indicates hexadecimal when using these number systems for literals. For example –

```
int decimal = 100;
int octal = 0144;
int hexa =  0x64;
```

String literals in Java are specified like they are in most other languages by enclosing a sequence of characters between a pair of double quotes. Examples of string literals are –

Example

```
"Hello World"
"two\nlines"
"\"This is in quotes\""
```

String and char types of literals can contain any Unicode characters. For example –

```
char a = '\u0001';
String a = "\u0001";
```

Java language supports few special escape sequences for String and char literals as well. They are –

| Notation | Character represented |
|---|---|
| **\n** | Newline (0x0a) |
| **\r** | Carriage return (0x0d) |
| **\f** | Formfeed (0x0c) |
| **\b** | Backspace (0x08) |
| **\s** | Space (0x20) |
| **\t** | tab |
| **\"** | Double quote |
| **\'** | Single quote |
| **** | backslash |
| **\ddd** | Octal character (ddd) |
| **\uxxxx** | Hexadecimal UNICODE character (xxxx) |

## Java Variables

A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used. Following is the basic form of a variable declaration –

data type variable [ = value][, variable [ = value] ...] ;

Here *data type* is one of Java's datatypes and *variable* is the name of the variable.

To declare more than one variable of the specified type, you can use a comma-separated list.

Following are valid examples of variable declaration and initialization in Java –

Example

```
int a, b, c;         // Declares three ints, a, b, and c.
int a = 10, b = 10;  // Example of initialization
byte B = 22;         // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a';        // the char variable a iis initialized with value 'a'
```

This chapter will explain various variable types available in Java Language. There are three kinds of variables in Java –

- Local variables
- Instance variables
- Class/Static variables

## Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Example

Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to only this method.

```
public class Test {
   public void pupAge() {
      int age = 0;
      age = age + 7;
```

```
      System.out.println("Puppy age is : " + age);
   }

   public static void main(String args[]) {
      Test test = new Test();
      test.pupAge();
   }
}
```

This will produce the following result –

Output

```
Puppy age is: 7
```

Example

Following example uses *age* without initializing it, so it would give an error at the time of compilation.

```
public class Test {
   public void pupAge() {
      int age;
      age = age + 7;
      System.out.println("Puppy age is : " + age);
   }

   public static void main(String args[]) {
      Test test = new Test();
      test.pupAge();
   }
}
```

This will produce the following error while compiling it –

Output

```
Test.java:4:variable number might not have been initialized
age = age + 7;
        ^
1 error
```

## Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

## Example

```
import java.io.*;
public class Employee {

   // this instance variable is visible for any child class.
   public String name;

   // salary  variable is visible in Employee class only.
   private double salary;
```

```
   // The name variable is assigned in the constructor.
   public Employee (String empName) {
      name = empName;
   }

   // The salary variable is assigned a value.
   public void setSalary(double empSal) {
      salary = empSal;
   }

   // This method prints the employee details.
   public void printEmp() {
      System.out.println("name  : " + name );
      System.out.println("salary :" + salary);
   }

   public static void main(String args[]) {
      Employee empOne = new Employee("Ransika");
      empOne.setSalary(1000);
      empOne.printEmp();
   }
}
```

This will produce the following result –

Output

```
name  : Ransika
salary :1000.0
```

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.

- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Example

```
import java.io.*;
public class Employee {

   // salary  variable is a private static variable
   private static double salary;

   // DEPARTMENT is a constant
   public static final String DEPARTMENT = "Development ";

   public static void main(String args[]) {
      salary = 1000;
      System.out.println(DEPARTMENT + "average salary:" + salary);
```

```
   }
}
```

This will produce the following result –

Output

```
Development average salary:1000
```

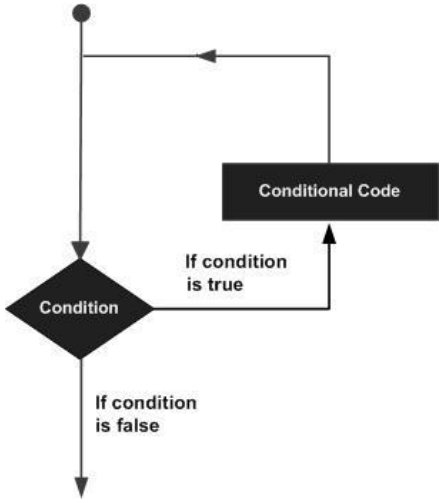
Note:– If the variables are accessed from an outside class, the constant should be accessed as Employee.DEPARTMENT

## Java Control statements

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages –




Java programming language provides the following types of loop to handle looping requirements. Click the following links to check their detail.

| Sl.No. | Loop & Description |
|---|---|
| 1 | while loop<br>Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body. |

| | |
|---|---|
| 2 | for loop<br>Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| 3 | do...while loop<br>Like a while statement, except that it tests the condition at the end of the loop body. |

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Java supports the following control statements. Click the following links to check their detail.

| Sl.No. | Control Statement & Description |
|---|---|
| 1 | break statement<br>Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch. |
| 2 | continue statement<br>Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |

## Enhanced for loop in Java

As of Java 5, the enhanced for loop was introduced. This is mainly used to traverse collection of elements including arrays.

Syntax

Following is the syntax of enhanced for loop –

```
for(declaration : expression) {
   // Statements
}
```

- Declaration – The newly declared block variable, is of a type compatible with

the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.

- Expression – This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Example

```
public class Test {
   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         System.out.print( x );
         System.out.print(",");
      }
      System.out.print("\n");
      String [] names = {"James", "Larry", "Tom", "Lacy"};
      for( String name : names ) {
         System.out.print( name );
         System.out.print(",");
      }
   }
}
```

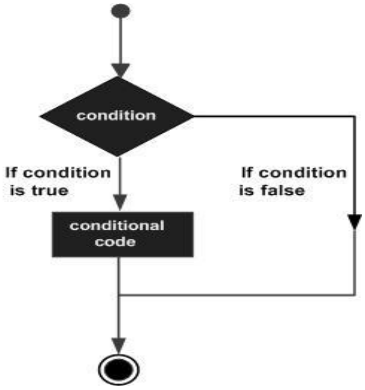
This will produce the following result –

Output

```
10, 20, 30, 40, 50,
James, Larry, Tom, Lacy,
```

Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages –




Java programming language provides following types of decision making statements. Click the following links to check their detail.

| Sl.No. | Statement & Description |
|---|---|
| 1 | if statement<br>An if statement consists of a boolean expression followed by one or more statements. |
| 2 | if...else statement<br>An if statement can be followed by an optional else statement, which executes when the boolean expression is false. |
| 3 | nested if statement<br>You can use one if or else if statement inside another if or else if statement(s). |
| 4 | switch statement<br>A switch statement allows a variable to be tested for equality against a list of values. |

The ? : Operator

We have covered conditional operator ? : in the previous chapter which can be used to replace if...else statements. It has the following general form –

Exp1 ? Exp2 : Exp3;

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

To determine the value of the whole expression, initially exp1 is evaluated.

- If the value of exp1 is true, then the value of Exp2 will be the value of the whole expression.
- If the value of exp1 is false, then Exp3 is evaluated and its value becomes the value of the entire expression.

## Java Arrays

Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

This tutorial introduces how to declare array variables, create arrays, and process arrays using indexed variables.

Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable –

Syntax

```
dataType[] arrayRefVar;   // preferred way.
or
dataType arrayRefVar[];  // works but not preferred way.
```

Note – The style dataType[] arrayRefVar is preferred. The style dataType arrayRefVar[] comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers.

Example

The following code snippets are examples of this syntax –

```
double[] myList;   // preferred way.
```

or

```
double myList[];   // works but not preferred way.
```

## Creating Arrays

You can create an array by using the new operator with the following syntax –

Syntax

```
arrayRefVar = new dataType[arraySize];
```

The above statement does two things –

- It creates an array using new dataType[arraySize].
- It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below –

```
dataType[] arrayRefVar = new dataType[arraySize];
```

Alternatively you can create arrays as follows –

```
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

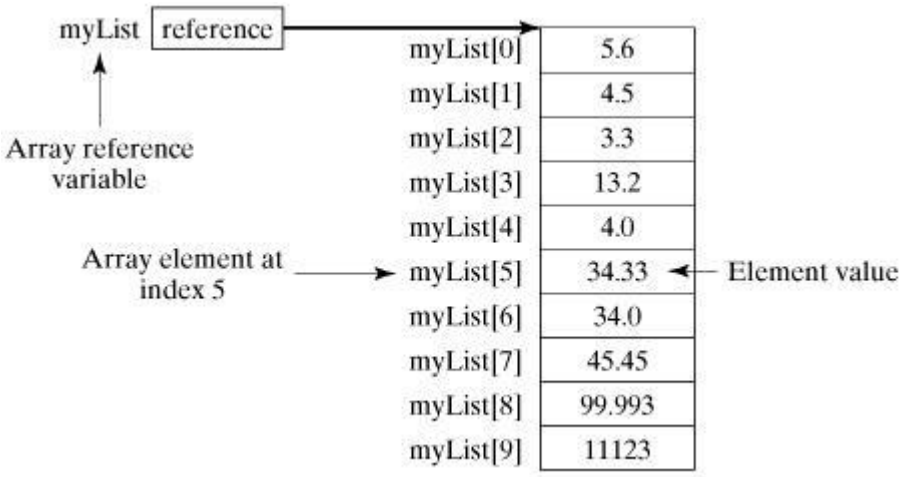
The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayRefVar.length-1.

Example

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList –

```
double[] myList = new double[10];
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.




## Processing Arrays

When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

## Example

Here is a complete example showing how to create, initialize, and process arrays –

```
public class TestArray {

  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};

    // Print all the array elements
    for (int i = 0; i < myList.length; i++) {
      System.out.println(myList[i] + " ");
    }

    // Summing all elements
    double total = 0;
    for (int i = 0; i < myList.length; i++) {
      total += myList[i];
    }
    System.out.println("Total is " + total);
```

```
      // Finding the largest element
      double max = myList[0];
      for (int i = 1; i < myList.length; i++) {
        if (myList[i] > max) max = myList[i];
      }
      System.out.println("Max is " + max);
   }
}
```

This will produce the following result –

Output

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

The foreach Loops

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

Example

The following code displays all the elements in the array myList –

```
public class TestArray {
   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (double element: myList) {
        System.out.println(element);
      }
   }
}
```

This will produce the following result –

Output

```
1.9
2.9
3.4
3.5
```

Passing Arrays to Methods

Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an int array –

Example

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

You can invoke it by passing an array. For example, the following statement invokes the printArray method to display 3, 1, 2, 6, 4, and 2 –

Example

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Returning an Array from a Method

A method may also return an array. For example, the following method returns an array that is the reversal of another array –

Example

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
    result[j] = list[i];
  }
  return result;
}
```

## The Arrays Class

The java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements. These methods are overloaded for all primitive types.

| Sl.No. | Method & Description |
|---|---|
| 1 | public static int binarySearch(Object[] a, Object key)<br>Searches the specified array of Object ( Byte, Int , double, etc.) for the specified value using the binary search algorithm. The array must be sorted prior to making this call. This returns index of the search key, if it is contained in the list; otherwise, it returns ( – (insertion point + 1)). |
| 2 | public static boolean equals(long[] a, long[] a2)<br>Returns true if the two specified arrays of longs are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. This returns true if the two arrays are equal. Same method could be used by all other primitive data types (Byte, short, Int, etc.) |
| 3 | public static void fill(int[] a, int val)<br>Assigns the specified int value to each element of the specified array of ints. The same method could be used by all other primitive data types (Byte, short, Int, etc.) |
| 4 | public static void sort(Object[] a)<br>Sorts the specified array of objects into an ascending order, according to the natural ordering of its elements. The same method could be used by all other primitive data types ( Byte, short, Int, etc.) |