**Core Java**

* **What is JAVA?**

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995. James Gosling is known as the father of Java.

Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

* **Example -**

**class** Simple{

**public** **static** **void** main(String args[]){

     System.out.println("Hello Java");

    }

}

* **Application**
* Desktop Applications such as acrobat reader, media player, antivirus, etc.
* Web Applications such as irctc.co.in, javatpoint.com, etc.
* Enterprise Applications such as banking applications.
* Mobile
* Embedded System
* Smart Card
* Robotics
* Games.
* **Types of Java Applications**

There are mainly 4 types of applications that can be created using Java programming:

**1) Standalone Application**

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

**2) Web Application**

An application that runs on the server side and creates a dynamic page is called a web application. Currently, [Servlet](https://www.javatpoint.com/servlet-tutorial), [JSP](https://www.javatpoint.com/jsp-tutorial), [Struts](https://www.javatpoint.com/struts-2-tutorial), [Spring](https://www.javatpoint.com/spring-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), etc. technologies are used for creating web applications in Java.

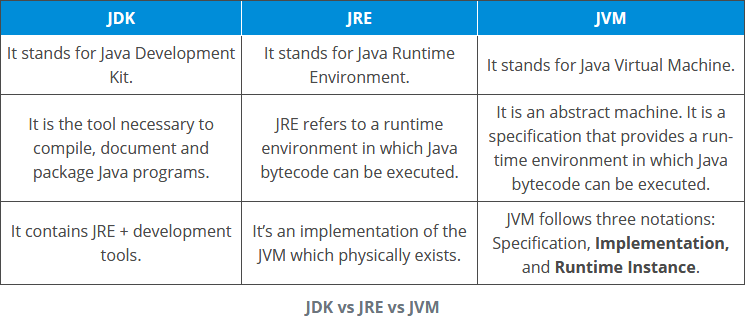
**3) Enterprise Application**

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, [EJB](https://www.javatpoint.com/ejb-tutorial) is used for creating enterprise applications.

**4) Mobile Application**

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

* **Features:-**
* [Simple](https://www.javatpoint.com/features-of-java#Simple)
* [Object-Oriented](https://www.javatpoint.com/features-of-java#Object-Oriented)
* [Portable](https://www.javatpoint.com/features-of-java#Portable)
* [Platform independent](https://www.javatpoint.com/features-of-java#Platform-independent)
* [Secured](https://www.javatpoint.com/features-of-java#Secured)
* [Robust](https://www.javatpoint.com/features-of-java#Robust)
* [Architecture neutral](https://www.javatpoint.com/features-of-java#Architecture-neutral)
* [Interpreted](https://www.javatpoint.com/features-of-java#Interpreted)
* [High Performance](https://www.javatpoint.com/features-of-java#High-Performance)
* [Multithreaded](https://www.javatpoint.com/features-of-java#Multithreaded)
* [Distributed](https://www.javatpoint.com/features-of-java#Distributed)
* [Dynamic](https://www.javatpoint.com/features-of-java#Dynamic)
* **JDK, JRE and JVM**



* **Variable**

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.

* **Types of Variables**

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial):

1. **local variable** - A variable declared inside the body of the method is called local variable.
2. **instance variable** - A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as [static](https://www.javatpoint.com/static-keyword-in-java).

It is called an instance variable because its value is instance-specific and is not shared among instances.

1. **static variable** - A variable that is declared as static is called a static variable. It cannot be local.

* **Data Types in Java-**

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

* **Java Primitive Data Types**
* There are 8 types of primitive data types:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

* **Operators in Java**

**Operator** in [Java](https://www.javatpoint.com/java-tutorial) is a symbol that is used to perform operations.

For example: +, -, \*, / etc.

There are many types of operators in Java which are given below:

|  |  |  |
| --- | --- | --- |
| Unary | postfix | *expr*++ *expr*-- |
| prefix | ++*expr* --*expr* +*expr* -*expr* ~ ! |
| Arithmetic | multiplicative | \* / % |
| additive | + - |
| Shift | shift | << >> >>> |
| Relational | comparison | < > <= >= instanceof |
| equality | == != |
| Bitwise | bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| Logical | logical AND | && |
| logical OR | || |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

* **Java Control Statements | Control Flow in Java**

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear.

However, [Java](https://www.javatpoint.com/java-tutorial) provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

* **Java provides three types of control flow statements.**

1. Decision Making statements

* if statements
* switch statement

1. Loop statements

* do while loop
* while loop
* for loop
* for-each loop

1. Jump statements

* break statement
* continue statement
* **Object-oriented programming (OOPs)** is a methodology that simplifies software development and maintenance by providing some rules.
* Basic concepts of OOPs are:
* [Object](https://www.javatpoint.com/object-and-class-in-java) - 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.
* [Class](https://www.javatpoint.com/object-and-class-in-java#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](https://www.javatpoint.com/inheritance-in-java) - When one object acquires all the properties and behaviours of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java) - 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.
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java) - Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.
* [Encapsulation](https://www.javatpoint.com/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.

* **What is a method in Java?**

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code.

We write a method once and use it many times. We do not require to write code again and again.

It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

The most important method in Java is the **main()** method.

* **Access Specifier:** Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:
* **Public:** The method is accessible by all classes when we use public specifier in our application.
* **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
* **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
* **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.
* **Constructors in Java**

A constructor is a block of codes similar to the method. It is called when an instance of the [class](https://www.javatpoint.com/object-and-class-in-java)

is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

* **Rules for creating Java constructor-**

There are two rules defined for the constructor.

Constructor name must be the same as its class name

A Constructor must have no explicit return type

A Java constructor cannot be abstract, static, final, and synchronized

* **Types of Java constructors**

There are two types of constructors in Java:

1. **Default constructor (no-arg constructor)** - A constructor is called "Default Constructor" when it doesn't have any parameter.
2. **Parameterized** constructor - A constructor which has a specific number of parameters is called a parameterized constructor.

* **Java static keyword**

The **static keyword** in [Java](https://www.javatpoint.com/java-tutorial) is used for memory management mainly. We can apply static keyword with [variables](https://www.javatpoint.com/java-variables), methods, blocks and [nested classes](https://www.javatpoint.com/java-inner-class). The static keyword belongs to the class than an instance of the class.

The static can be:

* Variable (also known as a class variable)
* Method (also known as a class method)
* Block
* Nested class
* **this keyword in Java**

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.

* **Usage of Java this keyword**
* this can be used to refer current class instance variable.
* this can be used to invoke current class method (implicitly)
* this() can be used to invoke current class constructor.
* this can be passed as an argument in the method call.
* this can be passed as argument in the constructor call.
* this can be used to return the current class instance from the method.
* **Abstract and Interface**

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

* **Java Package**

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

* **Advantage of Java Package**

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

* **Java String**

String is basically an object that represents sequence of char values. An [array](https://www.javatpoint.com/array-in-java)

of characters works same as Java string. For example:

* **Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.
* **What is Exception Handling?**

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

* **Advantage of Exception Handling**

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

* **Types of Java Exceptions**

**1) Checked Exception**

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

**2) Unchecked Exception**

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

**3) Error**

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

* **Java Exception Keywords**

Java provides five keywords that are used to handle the exception. The following table describes each.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| try | The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature. |

* **Collections in Java**
* The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.
* Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
* Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes ([ArrayList](https://www.javatpoint.com/java-arraylist), Vector, [LinkedList](https://www.javatpoint.com/java-linkedlist), [PriorityQueue](https://www.javatpoint.com/java-priorityqueue), HashSet, LinkedHashSet, TreeSet).
* **Java JDBC Tutorial**

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

* JDBC-ODBC Bridge Driver
* Native Driver
* Network Protocol Driver
* Thin Driver

The **java.sql** package contains classes and interfaces for JDBC API.

* **Java Database Connectivity with 5 Steps**

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

1. Register the Driver class
2. Create connection
3. Create statement
4. Execute queries
5. Close connection

|  |
| --- |
| **Example –**   * Class.forName("oracle.jdbc.driver.OracleDriver); * Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",   "username","password");   * Statement stmt=con.createStatement(); * ResultSet rs=stmt.executeQuery("select \* from emp");   **while**(rs.next())  {  System.out.println(rs.getInt(1)+" "+rs.getString(2));  }   * con.close();   THANK YOU! |