

# **1. INTRODUCTION TO JAVA**

## **1.1 History of Java**

### **1.1.1. About Java**

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

### **1.1.2. Brief History**

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of java starts with Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. Currently, Java is used in internet, mobile devices, games, e-business solutions, etc.

There are given the significant points that describe the history of Java.

- James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991.
- Firstly, it was called "Greentalk" by James Gosling, and file extension was .gt.
- After that, it was called Oak and was developed as a part of the Green project.
- In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique.

### **1.1.3. Java Version History**

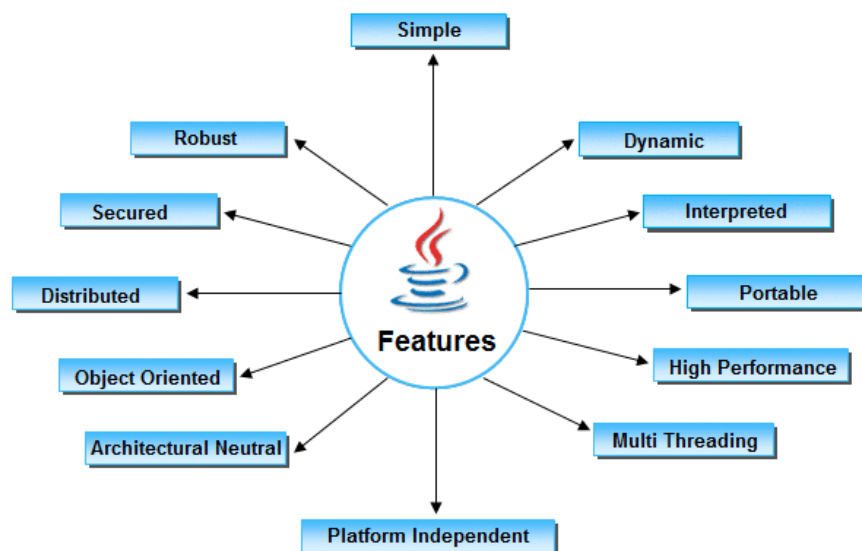
Many java versions have been released. The current release of Java is Java SE 10.

- a) JDK Alpha and Beta (1995)
- b) JDK 1.0 (23rd Jan 1996)

- c) JDK 1.1 (19th Feb 1997)
- d) J2SE 1.2 (8th Dec 1998)
- e) J2SE 1.3 (8th May 2000)
- f) J2SE 1.4 (6th Feb 2002)
- g) J2SE 5.0 (30th Sep 2004)
- h) Java SE 6 (11th Dec 2006)
- i) Java SE 7 (28th July 2011)
- j) Java SE 8 (18th March 2014)
- k) Java SE 9 (21st Sep 2017)
- l) Java SE 10 (20th March 2018)

## 1.2 Features of Java

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.



**Fig 1:** Features of Java

### 1.2.1. Simple

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

- a) Java syntax is based on C++ (so easier for programmers to learn it after C++).

- b) Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- c) There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

### **1.2.2. 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 incorporates both data and behaviour.

#### **a) Object :**

Any entity that has state and behaviour is known as an object. It can be physical or logical. An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. **Example:** A dog is an object because it has states like colour, name, breed, weight etc. as well as behaviours like wagging tail, barking, eating, etc.

#### **b) 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 in memory.

#### **c) Inheritance :**

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.

#### **d) Polymorphism :**

If one task is performed by 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 & method overriding to achieve polymorphism.

- If a class has multiple methods having same name but different in parameters, it is known **as** method overloading.
- If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

#### e) 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.

#### f) Encapsulation

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

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

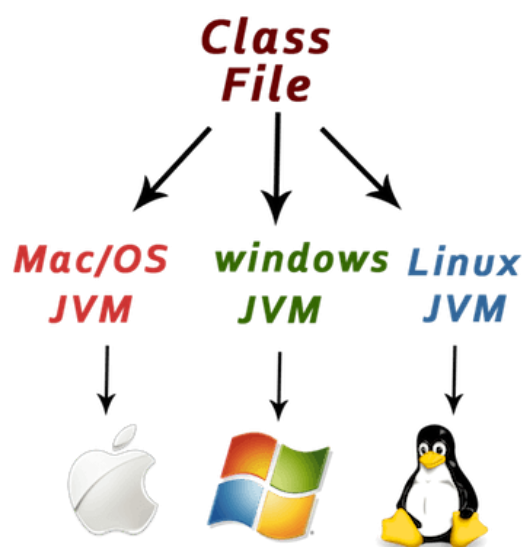


Fig 2: Platform Independent.

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

#### **1.2.4. Secured**

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

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

#### **1.2.5. Robust**

Robust simply means strong. Java is robust because:

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

#### **1.2.6. Portable**

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

#### **1.2.7. 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.

#### **1.2.8. High-Performance**

Java is faster than other traditional interpreted programming languages because Java byte code is "close" to native code. It is still a little bit slower than a compiled

language. Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

### 1.2.9. 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.

### 1.2.10. Dynamic

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++. Java supports dynamic compilation and automatic memory management (garbage collection).

## 1.3 C++ vs. Java

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

<b>Comparison Index</b>	<b>C++</b>	<b>Java</b>
Platform	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is used in window, web based, enterprise & mobile applications.
Go to	C++ supports.	Java doesn't support.
Multiple inheritance	C++ supports multiple - inheritance.	Java doesn't support. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You	Java supports pointer internally.

	can write pointer program in C++.	However, you can't write the pointer program in java. It means java has restricted pointer support.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses compiler and interpreter both. Java source code is converted into byte code at compilation time and interpreter executes this byte code at runtime and produces output. Java is interpreted that why it is platform independent.
Call by Value and reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference.
Structure-Union	C++ supports.	Java doesn't support.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party for thread.	Java has built-in thread support.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses a single inheritance tree always because all classes are child of Object class in java. The object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not interactive with h/w.

Table 1:C++ vs. Java

## 1.4. JDK, JRE, and JVM

### 1.4.1. JDK

The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development.

### 1.4.2. JRE

JRE stands for “Java Runtime Environment” and may also be written as “Java RTE.” The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, *and* supporting files.

### 1.4.3. JVM

It is a specification where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies. It is an implementation is a computer program that meets the requirements of the JVM specification. An Runtime Instance Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

To understand the difference between these three, let us consider the following figure.

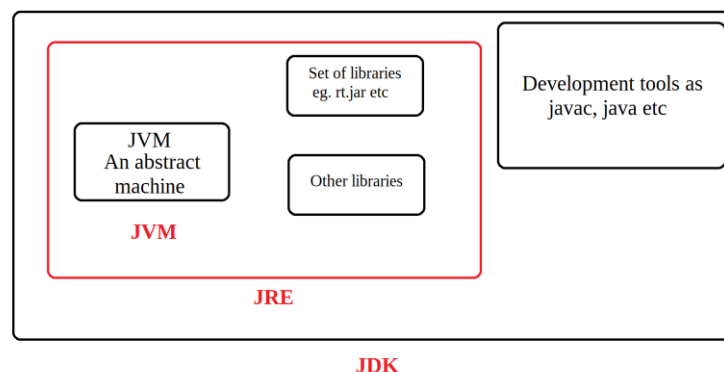


Fig 3: JDK vs. JVM vs. JRE.

## 1.5. Application of Java

The language has become the backbone of millions of applications across multiple platforms including Windows, Macintosh and UNIX-based desktops, Android-based mobiles, embedded systems and enterprise solutions.

### 1.5.1. Desktop GUI Applications:

Java provides GUI development through various means like Abstract Windowing Toolkit (AWT), Swing and Java FX. While AWT contains a number of pre-



constructed components such as menu, button, list, and numerous third-party components, Swing, a GUI widget toolkit

#### **1.5.2. Mobile Applications:**

Java Platform, Micro Edition (Java ME or J2ME) is a cross-platform framework to build applications that run across all Java supported devices, including feature phones and smart phones. Further, applications for Android, one of the most popular mobile operating systems, are usually scripted in Java using the Android Software Development Kit (SDK) or other environments.

#### **1.5.3. Embedded Systems:**

Embedded systems, ranging from tiny chips to specialized computers, are components of larger electromechanical systems performing dedicated tasks. Several devices, such as SIM cards, blue-ray disk players, utility meters and televisions, use embedded Java technologies. According to Oracle, 100% of Blu-ray Disc Players and 125 million TV devices employ Java. So, Java plays an important role in embedded systems, ranging from tiny chips to specialized computers.

#### **1.5.4. Web Applications:**

Java provides support for web applications through Servlets, Struts or JSPs. The easy programming and higher security offered by the programming language has allowed a large number of government applications for health, social security, education and insurance to be based on Java. Java also finds application in development of e-Commerce web applications using open-source e-Commerce platforms, such as Broadleaf.

#### **1.5.5. Scientific Applications:**

Java is the choice of many software developers for writing applications involving scientific calculations and mathematical operations. These programs are generally considered to be fast and secure, have a higher degree of portability and low maintenance. Applications like MATLAB use Java both for interacting user interface and as part of the core system.

## **2. Eclipse IDE – Tool for Java.**

### **2.1. Introduction to Eclipse IDE.**

#### **2.1.1. About Eclipse IDE**

In the context of computing, Eclipse is an integrated development environment (IDE) for developing applications using the Java programming language and other programming languages such as C/C++, Python, PERL, Ruby etc.

The Eclipse platform which provides the foundation for the Eclipse IDE is composed of plug-ins and is designed to be extensible using additional plug-ins. Developed using Java, the Eclipse platform can be used to develop rich client applications, integrated development environments and other tools. Eclipse can be used as an IDE for any programming language for which a plug-in is available.

The Java Development Tools (JDT) project provides a plug-in that allows Eclipse to be used as a Java IDE, PyDev is a plugin that allows Eclipse to be used as a Python IDE, C/C++ Development Tools (CDT) is a plug-in that allows Eclipse to be used for developing application using C/C++, the Eclipse Scala plug-in allows Eclipse to be used as an IDE to develop Scala applications and PHP eclipse is a plug-in to eclipse that provides complete development tool for PHP.

#### **2.1.2. Different Eclipse IDE distributions**

The different Eclipse IDE is following:

- a) The Eclipse IDE for Java development : Most people know Eclipse as an integrated development environment (IDE) for Java. In 2014 the Eclipse IDE is the leading development environment for Java with a market share of approximately 65%. The Eclipse IDE can be extended with additional software components. Eclipse calls these software components *plug-ins*. Plug-in can be grouped into *features*. Several projects and companies have extended the Eclipse IDE or created stand-alone applications (Eclipse Rich Client Platform) on top of the Eclipse framework.

- b) The Eclipse IDE distributions from the Eclipse Packaging Project (EPP) : The Eclipse IDE is also available as an IDE for other languages, ranging from C, C++ to Lua, Python, Perl and PHP. Several pre-packaged Eclipse distributions are available for download. These pre-packaged solutions are provided by an Eclipse project called the Eclipse Packaging Project (EPP).
- c) Developer and milestone downloads : The Eclipse project has a simultaneous release every year at the end of June. In June 2016 the Eclipse 4.6 (Neon) version was released. The top-level Eclipse project creates regular builds of the next releases including JDT, PDT and the Eclipse platform projects. This is called the Eclipse SDK.

### 2.1.3. Eclipse IDE releases

The Eclipse IDE open source project provides regular releases. It used to be one large release per year but in 2018 this was changed to one release every three months.

Release	Rename name	Release year
4.9	2018-09	2018
4.8	Photon	2018
4.7	Oxygen	2017
4.6	Neon	2016
4.5	Mars	2015
4.4	Luna	2014
4.3	Kepler	2013
4.2	Juno	2012

Table 2: Eclipse releases

## 2.2. Download and installation of the Eclipse IDE for Java Developers

The Eclipse IDE consists of several components. The Eclipse.org website provides pre-packaged Eclipse distributions to provide downloads for typical use cases. The Eclipse IDE for Java Developers distribution is specifically designed for standard

Java development. It contains typical required packages, like support for the Maven and Gradle build system and support for the Git version control system.

- I. Download the Eclipse IDE for Java Developers package from the following **URL:** <https://eclipse.org/downloads/eclipse-packages/>



Fig 4: Eclipse download website for Java Developers

Press on the link beside the package description, for example Window 64-Bit to start the download. The links which are displayed depend on your operating system.

- II. The download is a zip file, which is a compressed archive of multiple files.
- III. Installation procedure : After you downloaded the file with the Eclipse distribution, unpack it to a local directory.
- IV. Starting the Eclipse IDE: After you extracted the compressed file you can start Eclipse. No additional installation procedure is required, assuming that you Java installed on your machine.
- V. To start Eclipse, double-click the `eclipse.exe` (Microsoft Windows) in the directory where you unpacked Eclipse.
- VI. The Eclipse system prompts you for a *workspace*. The *workspace* is the location in your file system where Eclipse stores its preferences and other resources. For example, your projects can be stored in the workspace.

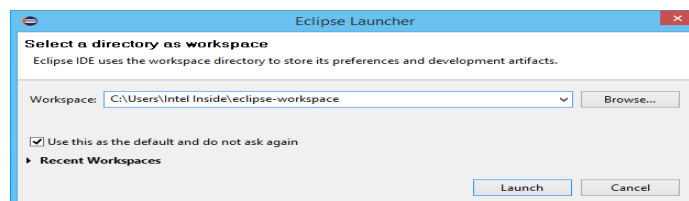


Fig 5: Eclipse Launcher

- VII. Select an empty directory and click the ok button.

- VIII. Eclipse starts and shows the Welcome page. Close this page by clicking the X beside Welcome.

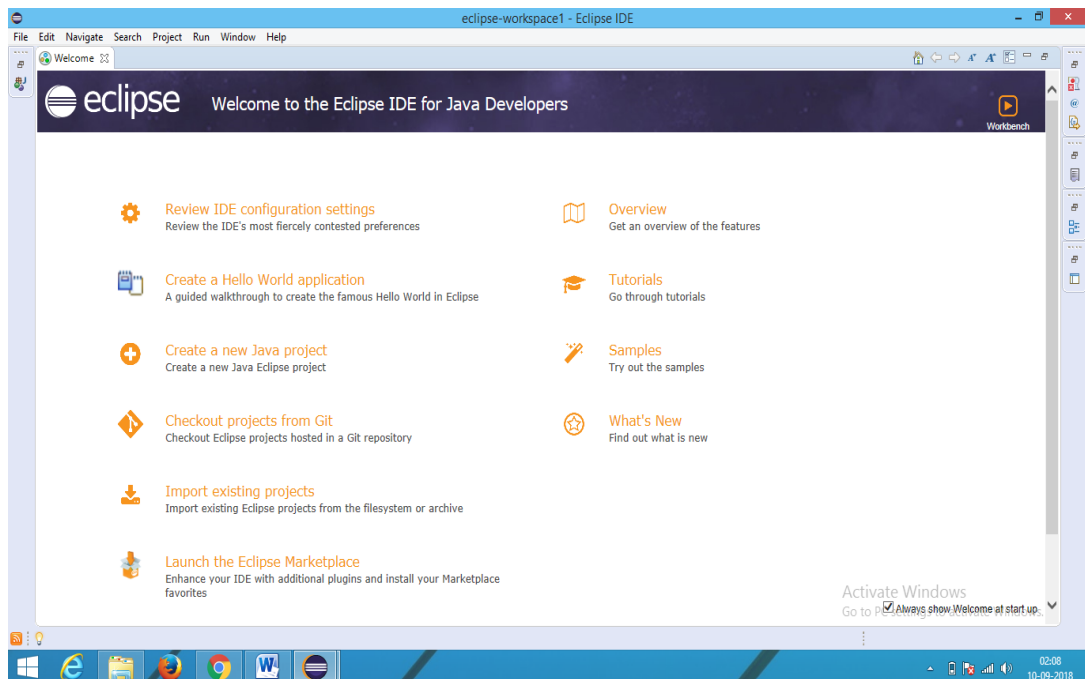


Fig 6: Eclipse Welcome Page

- IX. After closing the welcome screen, the application should look as following screenshot.

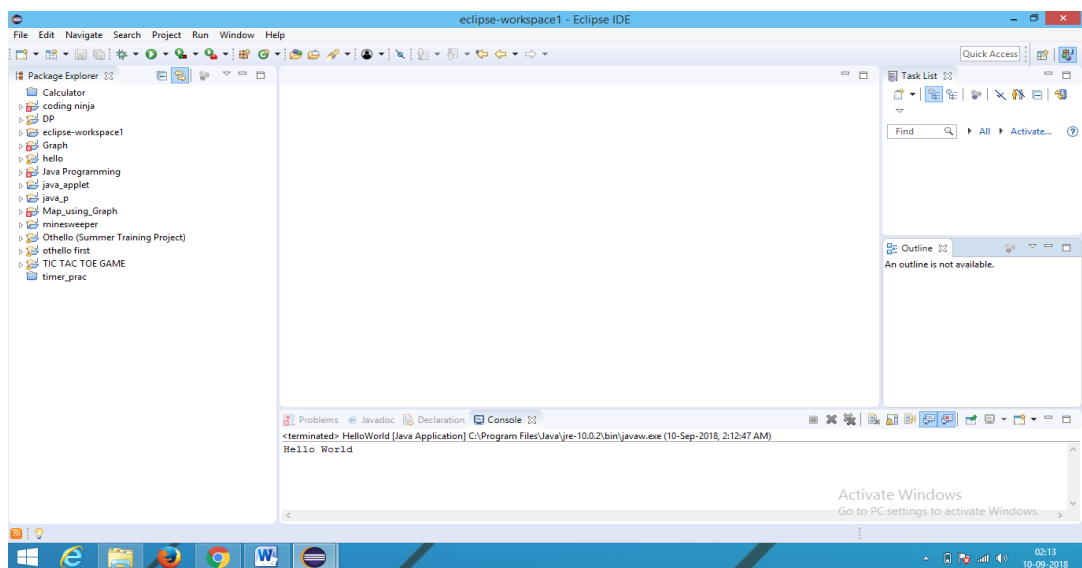


Fig 7: Eclipse Editor

## 3. Programming Concepts

### 3.1. Structure of Java Program

#### 3.1.1. Basic Structure Overview

Section	Description
Documentation Section	You can write a comment in this section. Comments are beneficial for the programmer because they help them understand the code..
Package statement	You can create a package with any name. A package is a group of classes that are defined by a name.
Import statements	This line indicates that if you want to use a class of another package, then you can do this by importing it directly into your program. <u>Example:</u> import calc.add;
Interface statement	Interfaces are just like classes that include a group of method declarations. It's an optional section and can be used when we want to implement multiple inheritances within a program.
Class Definition	A Java program may contain several class definitions. Classes are the main and essential elements of any Java program.

Table 3: Java program structure

#### 3.1.2. Main Method Class

Every Java stand-alone program requires the main method as the starting point of the program. This is an essential part of a Java program. There may be many classes in a Java program, and only one class defines the main method. Methods contain data type declaration and executable statements.

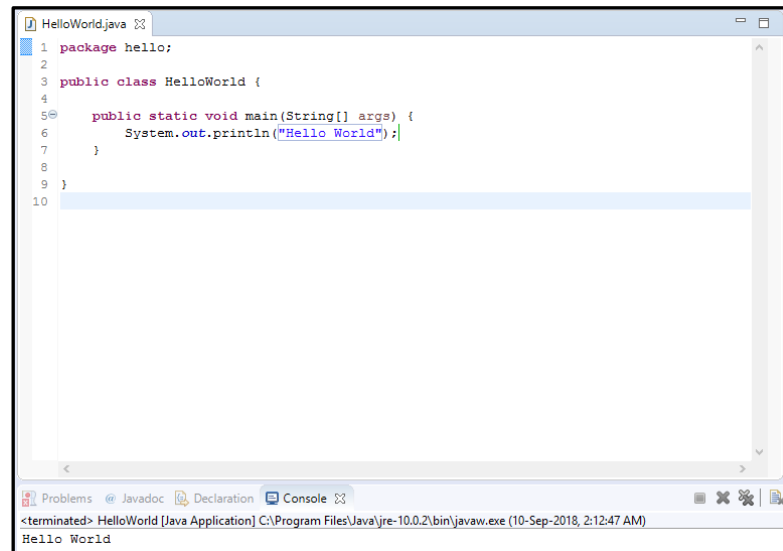
##### a) public static void main:

When the main method is declared public, it means that it can also be used by code outside of its class, due to which the main method is declared public. The word static used when we want to access a method without creating its object, as we call the main method, before creating any class objects. The word void indicates that a method does not return a value

### b) String[] args :

It is an array where each element of it is a string, which has been named as "args". If your Java program is run through the console, you can pass input parameter, and main() method takes it as input.

Here is an example of the Hello Java program



```
1 package hello;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         System.out.println("Hello World");
7     }
8
9 }
10
```

The screenshot shows an IDE window titled 'HelloWorld.java'. The code is as follows:

```
1 package hello;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         System.out.println("Hello World");
7     }
8
9 }
10
```

At the bottom, there is a console window showing the output: 'Hello World'.

Figure 8: Basic Hello World Program

### 3.1.2. Naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers.

Name	Convention
class name	Start with uppercase letter and be a noun. e.g. String, Color, Button, System, Thread etc.
interface name	Start with uppercase letter and be an adjective. e.g. Runnable, Remote, ActionListener etc.
method name	Start with lowercase letter and be a verb. e.g. actionPerformed(), main(), print(), println() etc.
variable name	Start with lowercase letter. e.g. firstName, order.
package name	Should be in lowercase letter e.g. java, lang.
constants name	Should be in uppercase letter. e.g. RED, YELLOW.

Table 4: Naming conventions in java

## 3.2. Decision making

**a) If statements:** If statements in Java is used to control the program flow based on some condition, it's used to execute some statement code block if the expression evaluated to true; otherwise, it will get skipped.

**Syntax :**

```
if(test_expression)
{
    statement 1;
    statement 2;
}
```

**b) If else statements:** If else statements is also used to control the program flow based on some condition, only the difference is: it's used to execute some statement code block if the expression is evaluated to true, otherwise executes else statement.

**Syntax :**

```
if(test_expression)
{
    //execute your code
}else{
    //execute your code
}
```

**c) else if statements :** else if statements in Java is like another if condition, it's used in the program when if statement having multiple decisions.

**Syntax :**

```
if(test_expression)
{
    //execute your code
}else if(test_expression n){
    //execute your code
}else{
    //execute your code
}
```

**d) Java switch statement:** Java switch statement is used when you have multiple possibilities for the if statement.



**Syntax :**

```
switch(variable)
{
    case 1:
        //execute your code
        break;
    case n:
        //execute your code
        break;
    default:
        //execute your code
}
```

### 3.3. Looping Statement

**a) while loop :** while loop is the most basic loop in Java. It has one control condition and executes as long the condition is true. The condition of the loop is tested before the body of the loop is executed; hence it is called an entry-controlled loop.

**Syntax :**

```
While (condition)
{
    statement(s);
    Incrementation;
}
```

**b) do-while:** do-while loops are very similar to the while loops, but it always executes the code block at least once and furthermore as long as the condition remains true.

**Syntax :**

```
do
{
    statement(s);
}
while( condition );
```

**c) for:** for loops is very similar to Java while loops in that it continues to process a block of code until statement becomes false, and everything is defined in a single line.

**Syntax :**

```
for ( init; condition; increment )  
{  
    statement(s);  
}
```

**3.4. Access Modifiers**

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor, variable, and method or data member. There are four types of access modifiers available in java:

**3.4.1. Default:**

When no access modifier is specified for a class , method or data member – It is said to be having the default access modifier by default. The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible only within the same package.

**3.4.2. Private:**

The private access modifier is specified using the keyword private. The methods or data members declared as private are accessible only within the class in which they are declared. Any other class of same package will not be able to access these members. Classes or interface can not be declared as private.

**3.4.3. Protected:**

The protected access modifier is specified using the keyword protected. The methods or data members declared as protected are accessible within same package or sub classes in different package.

**3.4.4. Public:**

The public access modifier is specified using the keyword public. The public access modifier has the widest scope among all other access modifiers. Classes, methods or data members which are declared as public are accessible from everywhere in the program. There is no restriction on the scope of a public data members.

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Fig 9: Access Modifiers

### 3.5. Wrapper Classes

A Wrapper class is a class whose object wraps or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types. In other words, we can wrap a primitive value into a wrapper class object. Since J2SE 5.0, auto-boxing and unboxing feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known as auto-boxing and vice-versa unboxing.

The eight classes of java.lang package are known as wrapper classes in java. The lists of eight wrapper classes are given below:

Primitive Type	Wrapper class
Boolean	Boolean
Char	Character
Byte	Byte
Short	Short
Int	Integer
Long	Long
Float	Float
Double	Double

Table 5: List of Wrapper classes.

### **Need of Wrapper Classes?**

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method.
2. The class in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

## **3.6. Java useful keywords.**

### **3.6.1. Final**

The final keyword in java is used to restrict the user. The java final keyword can be used in many contexts. Final can be:

1. Variable: If you make any variable as final, you cannot change the value of final variable (It will be constant).
2. Method: If you make any method as final, you cannot override it.
3. Class: If you make any class as final, you cannot extend it.

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

### **3.6.2. Static**

The static keyword in Java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than an instance of the class. The static can be:

1. Variable (also known as a class variable): If you declare any variable as static, it is known as a static variable. The static variable can be used to refer to the common property of all objects. The static variable gets memory only once in the class area at the time of class loading.

2. Method (also known as a class method); If you apply static keyword with any method, it is known as static method. Static method belongs to the class rather than the object of a class. Static method can be invoked without the need for creating an instance of a class. Static method can access static data member and can change the value of it.
3. Block: Is used to initialize the static data member. It is executed before the main method at the time of class-loading. So, we can write statements which we want to execute before main.

### **3.6.3. This**

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object. The 6 usage of java this keywords are

1. this can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.
2. this can passed as argument in method call. It is mainly used in the event handling.
3. this can be passed as argument in the constructor call.
4. this can be used to return the current class instance from the method. In such case, return type of the method must be the class type and the current class instance is returned .

### **3.6.4. Super**

The super keyword in java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable. Usage of java super Keyword are:

1. super can be used to refer immediate parent class instance variable. We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

### 3.7. Java Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type text & send message.

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

#### 3.7.1. Abstract Classes

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to remember about abstract classes:

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Example of abstract class is given below:

```
abstract class Bank{
    abstract int getRateOfInterest();
}
class SBI extends Bank{
    int getRateOfInterest(){return 7;}
}
class ICICI extends Bank{
    int getRateOfInterest(){return 7;}
}
```

```

class TestBank{
    public static void main(String args[]){
        Bank b;
        b=new SBI();
        System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
    }
}

```

### 3.7.2. Interface

An interface in java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritances in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class. In Java 8, we can have default and static methods in an interface.

Why use Java interface? There are mainly three reasons to use interface.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

An interface is declared by using the interface keyword. It provides total abstraction means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

#### Syntax:

```

interface <interface_name>{
    // declare constant fields
    // declare methods that abstract
    // by default.
}

```

## 4. Collections in Java

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects. All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion, etc. can be achieved by Java Collections. Java Collection means a single unit of objects. Java Collection framework provides many interfaces and classes.

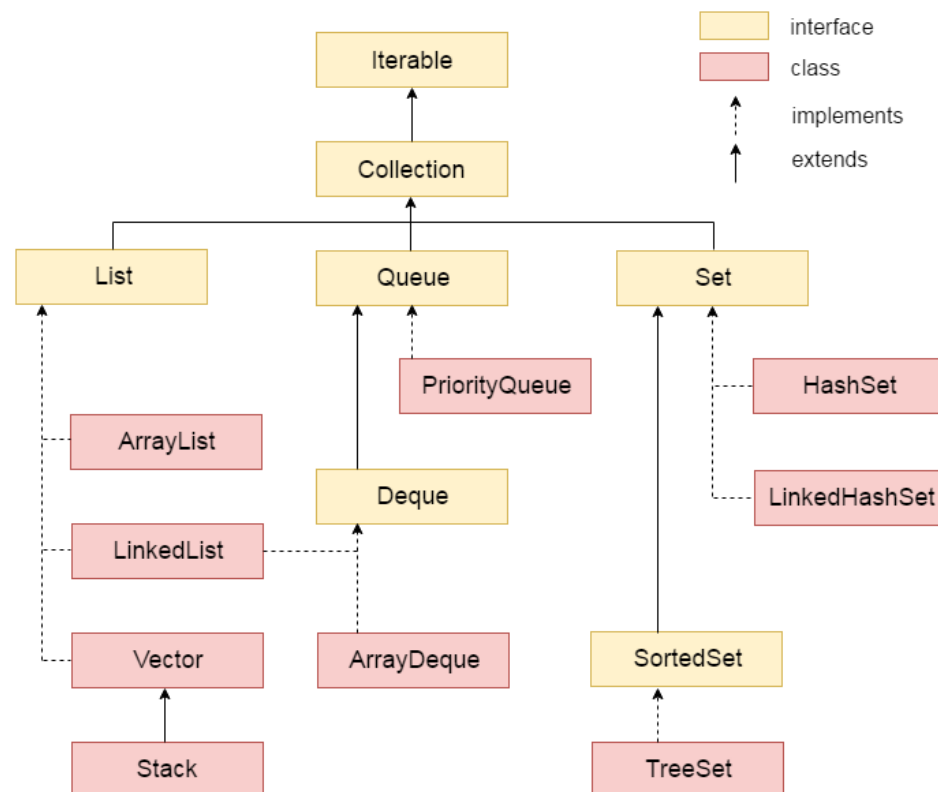


Fig 10: Collections in Java

### 4.1. ArrayList

ArrayList is a part of collection framework and is present in java.util package. It provides us dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.

- ArrayList inherits Abstract List class and implements List interface.
- Java ArrayList allows us to randomly access the list.
- ArrayList cannot be used for primitive types, like int, char, etc. We need a wrapper class for such cases.



#### 4.1.1. Constructors in Java ArrayList:

- a) ArrayList(): This constructor is used to build an empty array list
- b) ArrayList(Collection c): This constructor is used to build an array list initialized with the elements from collection c
- c) ArrayList(int capacity): This constructor is used to build an array list with initial capacity being specified

#### 4.1.2. Java program to demonstrate working of ArrayList in Java

```
import java.io.*;
import java.util.*;
class arrayli
{
    public static void main(String[] args)
    {
        //declaring ArrayList with initial size 5
        ArrayList<Integer> arrli = new ArrayList<Integer>(5);

        // Appending the new element at the end of the list
        for (int i=1; i<=n; i++)
            arrli.add(i);

        // Printing elements
        System.out.println(arrli);

        // Remove element at index 3
        arrli.remove(3);

        // Displaying ArrayList after deletion
        System.out.println(arrli);

        // Printing elements one by one
        for (int i=0; i<arrli.size(); i++)
            System.out.print(arrli.get(i)+" ");
    }
}
```

#### **4.1.3. Methods in Java ArrayList:**

- a) void clear(): This method is used to remove all the elements from any list.
- b) void add(int index, Object element): This method is used to insert a specific element at a specific position index in a list.
- c) Object[] toArray(): This method is used to return an array containing all of the elements in the list in correct order.
- d) boolean add(Object o): This method is used to append a specified element to the end of a list.

#### **4.2. LinkedList**

Linked List are linear data structures where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node. Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays. It also has few disadvantages like the nodes cannot be accessed directly instead we need to start from the head and follow through the link to reach to a node we wish to access. To store the elements in a linked list we use a doubly linked list which provides a linear data structure and also used to inherit an abstract class and implement list and deque interfaces.

##### **4.2.1. Constructors for Java LinkedList:**

- a) LinkedList(): Used to create an empty linked list.
- b) LinkedList(Collection C): Used to create a ordered list which contains all the elements of a specified collection, as returned by the collection's iterator.

##### **4.2.2. Java code for Linked List implementation**

```
import java.util.*;

public class Test
{
    public static void main(String args[])
    {
        // Creating object of class linked list
        LinkedList<String> object = new LinkedList<String>();
```

```

// Adding elements to the linked list
object.add("A");
object.addLast("C");
object.addFirst("D");
object.add(2, "E");
System.out.println("Linked list : " + object);

// Removing elements from the linked list
object.remove("A");
object.remove("B");
object.removeFirst();
object.removeLast();
System.out.println("Linked list after deletion: " + object);

// Finding elements in the linked list
if(object.contains("E"))
    System.out.println("List contains the element 'E' ");

// Number of elements in the linked list
System.out.println("Size of linked list = " + object.size());
}
}

```

#### **4.2.3. Methods for Java LinkedList:**

1. `int size()`: It returns the number of elements in this list.
2. `void clear()`: It clears all the elements in this list.
3. `boolean contains(Object element)`: It returns true if element is present in list.
4. `boolean add(Object element)`: It appends the element to the end of the list.
5. `void add(int index, Object element)`: It inserts the element at the position 'index' in the list.
6. `Object remove()`: It is used to remove and return the element from head of list.

### 4.3. HashMap

HashMap is a part of Java's collection since Java 1.2. It provides the basic implementation of Map interface of Java. It stores the data in (Key, Value) pairs. To access a value one must know its key. HashMap is known as HashMap because it uses a technique called Hashing. Hashing is a technique of converting a large String to small String that represents same String. A shorter value helps in indexing and faster searches. HashSet also uses HashMap internally. It internally uses a link list to store key-value pairs already explained in HashSet in detail and further articles.

#### 4.3.1. Few important features of HashMap are:

- a) HashMap is a part of java.util package.
- b) HashMap extends an abstract class AbstractMap which also provides an incomplete implementation of Map interface.
- c) HashMap doesn't allow duplicate keys but allows duplicate values. That means A single key can't contain more than 1 value but more than 1 key can contain a single value.

#### 4.3.2. Constructors in HashMap

- a) HashMap() : It is the default constructor which creates an instance of HashMap with initial capacity 16 and load factor 0.75.
- b) HashMap(int initial capacity) : It creates a HashMap instance with specified initial capacity and load factor 0.75.
- c) HashMap(int initial capacity, float loadFactor) : It creates a HashMap instance with specified initial capacity and specified load factor.
- d) HashMap(Map map) : It creates instance of HashMap with same mappings as specified map.

#### 4.3.3. Java program to illustrate

```
import java.util.HashMap;  
import java.util.Map;
```

```
public class GFG  
{
```

```

public static void main(String[] args)
{
    HashMap<String, Integer> map = new HashMap<>();
    map.put("vishal", 10);
    map.put("sachin", 30);
    System.out.println("Size of map is:- " + map.size());
    print(map);
    if (map.containsKey("vishal"))
        System.out.println("value for key \"vishal\" is:- " + map.get("vishal"));
    map.clear();
    print(map);
}

public static void print(Map<String, Integer> map)
{
    if (map.isEmpty())
        System.out.println("map is empty");
    else
        System.out.println(map);
}
}

```

#### 4.3.3. Methods in HashMap

- a) void clear(): Used to remove all mappings from a map.
- b) boolean containsKey(Object key): Used to return True if for a specified key, mapping is present in the map.
- c) boolean containsValue(Object value): Used to return true if one or more key is mapped to a specified value.
- d) boolean isEmpty(): Used to check whether the map is empty or not. Returns true if the map is empty.
- e) Set entrySet(): It is used to return a set view of the hash map.
- f) Object get(Object key): It is used to retrieve or fetch the value mapped by a particular key.
- g) Set keySet(): It is used to return a set view of the keys.
- h) int size(): It is used to return the size of a map.

#### **4.3.4. Time complexity of HashMap**

HashMap provides constant time complexity for basic operations, get and put, if hash function is properly written and it disperses the elements properly among the buckets. Iteration over HashMap depends on the capacity of HashMap and number of key-value pairs. Basically it is directly proportional to the capacity + size. Capacity is the number of buckets in HashMap. So it is not a good idea to keep high number of buckets in HashMap initially.

#### **4.4. Pair Class in Java**

In C++, we have `std::pair` in the utility library which is of immense use if we want to keep a pair of values together. We were looking for an equivalent class for pair in Java but Pair class did not come into existence till Java 7.

JavaFX 2.2 has the `javafx.util.Pair` class which can be used to store a pair. We need to store the values into Pair using the parameterized constructor provided by the `javafx.util.Pair` class.

Note that the `<Key, Value>` pair used in HashMap/TreeMap. Here, `<Key, Value>` simply refers to a pair of values that are stored together.

#### **Methods provided by the `javafx.util.Pair` class**

- a) `Pair (K key, V value)` : Creates a new pair
- b) `boolean equals()` : It is used to compare two pair objects. It does a deep comparison, i.e., it compares on the basis of the values (`<Key, Value>`) which are stored in the pair objects.
- c) `String toString()` : This method will return the String representation of the Pair.
- d) `K getKey()` : It returns key for the pair.
- e) `V getValue()` : It returns value for the pair.
- f) `int hashCode()` : Generate a hash code for the Pair.

## 5. Othello Game in Java

### 5.1. Introduction to Othello Game.

#### 5.1.1. History

Othello is the trading name of a much older board game called Reversi. In the 19th century, John W. Mollett or Lewis Waterman invented the game under the name Reversi. It was then published in 1898 by the Ravensburger Company as one of its first titles. In 1970, Goro Hasegawa renamed the game as Othello and the game got marketed by the Japanese company Tsukuda. This new version modified some of the play of the reversi game and its popularity started to spread rapidly to Europe and North America. The first world championship was already organized in 1977.

#### 5.1.2. Motto of Game

The motto of Othello is "a minute to learn, a lifetime to master".

#### 5.1.3. About Othello

Othello is a classic board game and can only be played by 2 players just like chess, checkers and go. The game board is made up of 8 rows and 8 columns. The board must be populated with 64 discs. They are colored black on one side and white on the other. This way, they can be flipped to the other color easily. One player plays discs black side up, the other white side up.

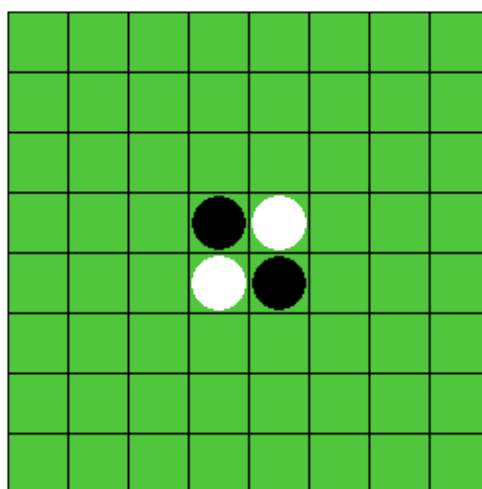


Fig 11: Othello Game

After the first 4 initial discs are placed, black open the game. You can play a disc when you flank one or more opponent's discs between your new disc and any other of your own discs, in the same horizontal, vertical or diagonal line. The opponent's discs that are flanked will be turned upside-down and change colour. When there is no possible legal move, the turn is given back to the opponent. When both players need to pass because there is no legal move left, the game has ended. The discs are counted. Whoever has the most discs wins the board game.

The irony of Othello is that even though the entire board and all the pieces are in full visible, the best move is often far from obvious. Deciding upon the right move can be analogous to hunting for buried treasures: you need to know what it is you are looking for and how to go about finding it. The best way to learn Othello is to read the rules of the game and when you have assimilated the rules then read some articles about strategy and tactics and of course practice as much as possible.

#### **5.1.4. Rules of Othello**

- Black always moves first.
- A move is made by placing a disc of the player's color on the board in a position that "out-flanks" one or more of the opponent's discs.
- A disc or row of discs is outflanked when it is surrounded at the ends by discs of the opposite color.
- A disc may outflank any number of discs in one or more rows in any direction (horizontal, vertical, diagonal).
- All the discs which are outflanked will be flipped, even if it is to the player's advantage not to flip them.
- Discs may only be outflanked as a direct result of a move and must fall in the direct line of the disc being played.
- If you can't outflank and flip at least one opposing disc, you must pass your turn. However, if a move is available to you, you can't forfeit your turn.
- Once a disc has been placed on a square, it can never be moved to another square later in the game.
- When a player runs out of discs, but still has opportunities to outflank an opposing disc, then the opponent must give the player a disc to use.



### **5.1.5. End of the Game**

- When it is no longer possible for either player to move, the game is over.
- The discs are now counted and the player with the majority of his or her color discs on the board is the winner.
- A tie is possible.

## **5.2. Purpose**

For a computerised player, games of skill are challenges of: analysis of the game; generation of the possible responses; decision on which is the best course of action to take. Furthermore, look and feel of the game playing.

Since we know, Board games have distinct boundaries. Living in a complex society, children need clear limits to feel safe. Board games can help your child weave her wild and erratic side into a more organized, mature, and socially acceptable personality. After all, staying within the boundaries (not intruding on others' space, for example) is crucial to leading a successful social and academic life. Therefore I decided to make a computer game like Othello so that the children can play with in boundaries and this game will increase their thinking ability, their co-ordination, and feel of the game play.

The following project applies these general computer game play principles to the game of Othello. It also tries to function as a portable substitute for a real game board. As computing power and programming research advances more games of skill become playable. Study of simpler games can however reveal principles that can guide the development of programs to play more complicated games.

The main aim of this project is to develop an interesting computer game so that the players have a great experience and to initiate a step towards the world of computers. It is a simple java application that is based on the principles of the game of Othello. This game will increase their thinking ability, their co-ordination, feel of the game play, maturity, and make the players more organized and socially acceptable personality

### 5.3. Classes and Description

Let us decide the key entities about which our Othello game revolve around. The key entities must be:

1. Player Class: Two players who play the game
2. Board Class: On which the game is conducting
3. Othello Class: Manager class who conduct the game

#### 5.3.1. Player Class:

There are two player objects that would play the game. The name and symbol of both the player object must be different and cannot be null respectively.

##### Data members

- Name of players: name (String)
- Symbol used by the players: symbol (char)
- Count of win games. winGame (int)

##### Member Functions:

- Default Constructor: Player()
- Parameterised Constructor: public Player(String name, char symbol)
- Function to set name: public void setName(String name)
- Function to set symbol: public void setSymbol(char symbol)
- Function to get name: public String getName()
- Function to get symbol: public char getSymbol()
- Function to set count of win games: setWinGames()
- Function to get count of win games: getWinGames()

#### 5.3.2. Board Class:

On which the game is conducting. It must be an 8X8 board. For each new game a board object must be setup.

##### Import packages:

- java.util.ArrayList
- javafx.util.Pair

**Data members:**

- Board 2D array to hold the game: board[][] (char)
- Size of board: BOARD\_SIZE = 8 (final int)
- Symbol of both the players: p1Symbol, p2Symbol (char)
- Count for Total Moves: private totalMoves (int)

**Member Functions:**

- public int noOfValidMoves(char symbol): Calculating number of Valid Moves. It will count the total possible moves that a player can move on the board with the help of list of valid moves.
- public Board(char p1Symbol, char p2Symbol): Constructor for setting up the Othello Board. It will set up a board and make 4 prior moves.
- public void printBoard(): Displaying Current Board. It is used to print the current status of the board.
- public boolean checkMove (char symbol, int x, int y): helper function to generate list of Valid Moves. It helps to generate the the list of valid moves by checking each location for validity.
- public Array List<Pair<Integer, Integer>> validMoves(char symbol). This function will return a ArrayList of pair of points giving a location on board.
- public boolean move(char symbol,int x, int y) .it will check if the given location can hold a valid move. If yes,it will perform the move and increase the totalMoves count.
- public boolean completeGame(). It will check that the board is full or not.
- public int countSymbol(char symbol). This functions will return the total count of the symbols which is needed to decide which player wins the game.

**5.3.3. Othello Class:**

It is the manager class who will conduct the game. It must have two objects of Player class who play the game and an object of Board class upon which the game is conducting.

**Import packages:**

- java.util.ArrayList

- java.util.Scanner
- javafx.util.Pair

### Data members:

- Object of Board class for conducting Othello: Board board
- Objects of Player class to play: player1, player2
- Check for new game: boolean anotherGame = true
- Count for Total Number of Games: No\_of\_Games (static int)
- Count for Number of Draw Games: Draw

### Member Functions:

- public static void main(String[] args)  
It comprises of 3 steps:
  - a) Taking Player Information
  - b) Creating Board with the Given Symbols & Conduct Game.
  - c) Display Score Board.
- public void startGame(): It will first make 2 new objects of Player class and then take their information and call create\_Board() for conducting the game.
- private Player take\_player\_info(int n): Taking the private information for the Players such as name symbol etc.
- public void create\_Board() : Create a new board object to play a new game.
- Private void printScoreBoard(): Used for displaying the score of both the players after the game is end.  
private void printHint(ArrayList<Pair<Integer, Integer>> validMoves): Used to printing the calculated valid moves for the required Player

## 5.4. Outputs.

```

eclipse-workspace1 - Othello (Summer Training Project)/src/othello/Othello.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Othello [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (11-Sep-2018, 2:33:55 PM)
Console
OTHELLO GAME :
Enter Player 1's name :
Shivansh
Enter Player 1's Symbol :
B
Enter Player 2's name :
Shivansh
Enter Player 2's Symbol :
B
Name already taken !! Choose another name for Player 2 !!
Enter Player 2's name :
Nikhil
Symbol already taken !! Choose another Symbol for Player 2 !!
Enter Player 2's Symbol :
W

OTHELLO Board : 8 X 8 Board
X|Y 0 1 2 3 4 5 6 7
+-----+
0 | | | | | | | |
+-----+
1 | | | | | | | |
+-----+
2 | | | | | | | |
+-----+
3 | | | B | W | | |
+-----+
4 | | | W | B | | |
+-----+
5 | | | | | | | |
+-----+
6 | | | | | | | |
+-----+

```

Fig 12: Taking Player info in Othello

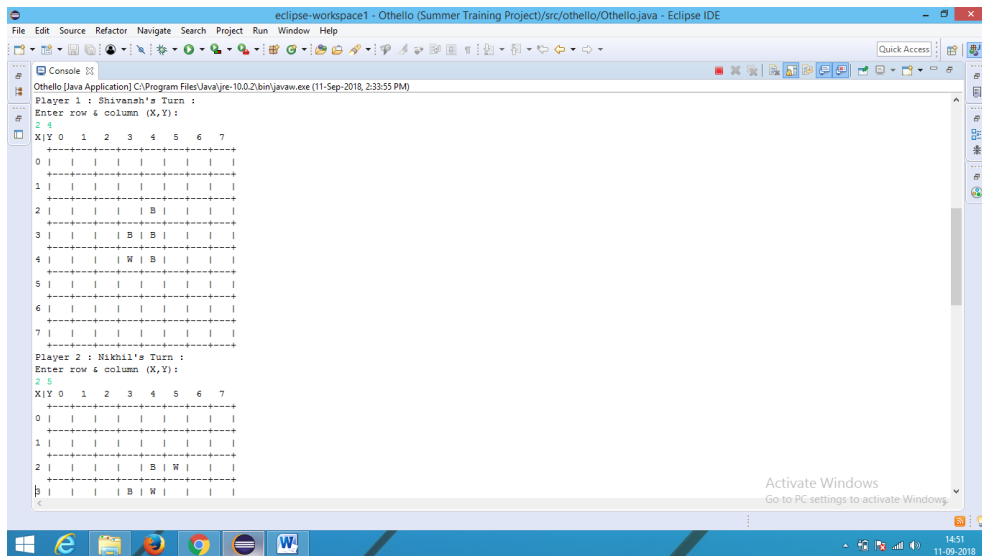


Fig 13: Taking Move one by one

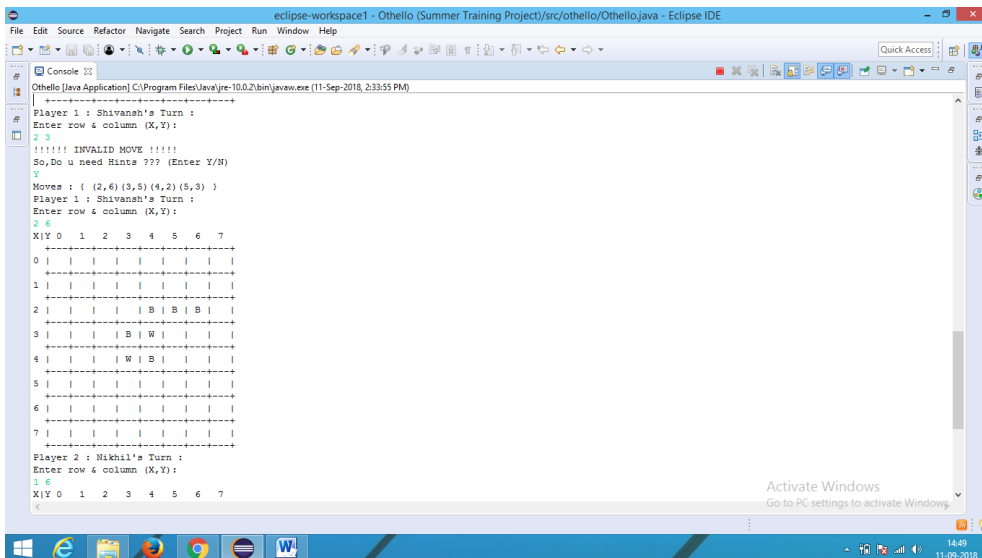


Fig 14: Taking help when you have invalid move

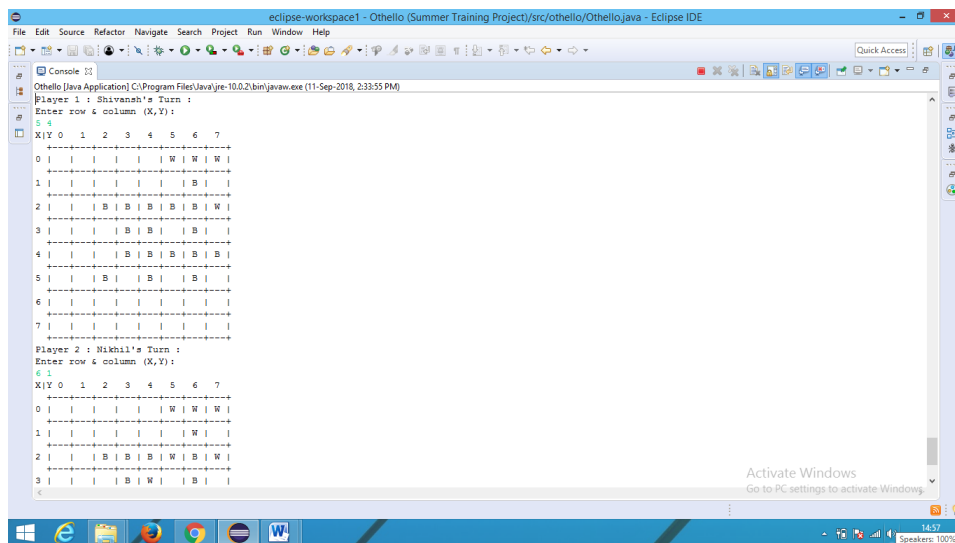


Fig 15: Continuing game

The screenshot shows the Eclipse IDE with the console output of the Othello game. The game has ended with Player 2 (Nikhil) winning. The console displays the final 8x8 board state, the score board, and a prompt to play a new game.

```

Othello [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (11-Sep-2018, 2:33:55 PM)
7 | B | B | W | W | W | W | W |
+-----+
Player 2 : Nikhil's Turn :
Enter row & column (X,Y):
0 1
X\Y 0 1 2 3 4 5 6 7
0 | B | W | W | W | W | W | W |
+-----+
1 | B | W | W | W | B | B | B | B |
+-----+
2 | B | W | B | B | B | B | W | W |
+-----+
3 | B | W | B | B | B | B | B | W |
+-----+
4 | B | B | W | W | W | W | W | W |
+-----+
5 | B | W | B | W | W | W | W | W |
+-----+
6 | B | B | W | W | W | W | W | W |
+-----+
7 | B | B | W | W | W | W | W | W |
+-----+
!!!! GAME OVER !!!!
PLAYER 2 Nikhil WINS !!!!!
SCORE BOARD
Total number of games = 1
Shivansh won 0 times
Nikhil won 1 times
Number of tied games = 0
Want to play a new Game ? (Enter Y/N)

```

Fig 16: Game over and display score board.

The screenshot shows the Eclipse IDE with the console output of the Othello game. The game has ended with Player 2 (Nikhil) winning. The console displays the final 8x8 board state, the score board, and a prompt to play a new game. The user has entered 'Y' to play a new game, and the console displays the new 8x8 board state.

```

Othello [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (11-Sep-2018, 2:33:55 PM)
!!!! GAME OVER !!!!
PLAYER 2 Nikhil WINS !!!!!
SCORE BOARD
Total number of games = 1
Shivansh won 0 times
Nikhil won 1 times
Number of tied games = 0
Want to play a new Game ? (Enter Y/N)
Y
OTHELLO Board : 8 X 8 Board
X\Y 0 1 2 3 4 5 6 7
0 | | | | | | | |
+-----+
1 | | | | | | | |
+-----+
2 | | | | | | | |
+-----+
3 | | | B | W | | | |
+-----+
4 | | | W | B | | | |
+-----+
5 | | | | | | | |
+-----+
6 | | | | | | | |
+-----+
7 | | | | | | | |
+-----+
Player 1 : Shivansh's Turn :
Enter row & column (X,Y):

```

Fig 17: Asking for new game

## **Conclusion**

This report is about the fundamentals of Core Java that is what basic things one should be familiar with in order to start java applications. I have used these fundamentals as well as advanced concepts in order to complete my summer training project i.e. “Othello game in Java” which will basically give all the details about this game and used concepts.

Also while training in Core java and Data structures at Coding Ninjas, I learned a lot of new skills technically as well as professionally and how it feels to work in such a environment.

## References

- <https://www.tutorialspoint.com>
- <https://www.javatpoint.com>
- <https://www.codingninjas.in>
- <https://www.stackoverflow.com>
- <https://www.geeksforgeeks.org>
- <https://github.com>
- <https://youtube.com>
- <https://google.com>