My Knowledge on Java

Basic Java Notes

Aong Cho Marma

2023

My Knowledge On Java

Aong Cho Marma

13 July, 2023

Contents

Chapter 1

INTRODUCTION

Java is a General Purpose Programming Language (GPL) and also powerful programming language. Java developed at **Sun Microsystems** which was purchased by **Oracle** in 2010. Java is GPL because it is used to solve a wide variety of problems and build software.

1.1 JAVA LANGUAGE SPECIFICATION

1.1.1 THE SYNTAX & SEMANTICS

To write English we should follow some rules (Grammar). Also, to write java we should follow some rules that is called **syntax & semantics**
Example:

- ◇ He are playing ⇒ Syntax error.(Grammar)
- ◇ He is hello and bye ⇒ Semantic error.(Meaning)

1.1.2 API

Application Programming Interface (API) also known as **library**. It contains predefined Java code that we can use to develop Java programs. It makes faster and easier development process. Because we do not need to write everything from scratch.

1.1.3 EDITIONS OF JAVA

Java comes in three editions

- ◇ **Standard Edition(SE):** Develop applications that run on desktop.
- ◇ **Enterprise Edition(EE):** Develop server-side applications.
- ◇ **Micro Edition(ME):** Develop applications for mobile devices.

NOTE: Java SE is the foundation of all other editions.

1.1.4 JDK

Java Development Kit (JDK)

- ◇ Set of programs that enable us to develop our programs.
- ◇ Contains **JRE(Java Runtime Environment)** that is used to run out programs.
- ◇ **JDK & JRE** contain **JVM (Java Virtual Machine)**.
- ◇ **JVM** executes our java programs on different machines that makes Java independent.

1.1.5 IDE

Integrated Development Environment (IDE) is a program that allows us to-

- ◇ **Write**: Write source code
- ◇ **Compile**: Translate source code to machine code
- ◇ **Debug**: Tools to find errors
- ◇ **Build**: Files that can be executed by JVM
- ◇ **Run**: Execute program

IDE makes development faster and easier. NetBeans, Eclipse, IntelliJ IDE are the popular Java IDEs.

> **NOTE:** The Java source code first compiled into a binary byte code using Java compiler, then this byte code runs on the JVM, Which is a software based interpreter. So Java is considered as both interpreted and compiled.

1.2 ANATOMY OF JAVA PROGRAM

1.2.1 CLASS

A blueprint to create *OBJECTS*.

CLASS STRUCTURE

```
class class_name {
  code block
}
// "class" is keyword.
```

1.2.2 OBJECTS

An instance of a *CLASS*.

1.2.3 METHOD

Group of instruction to do a specific task.

METHOD STRUCTURE

Each method consists of 4 main parts.

- ◇ Return Type
- ◇ Method Name
- ◇ Parameter
- ◇ Code Block

```
return_type method_name(parameter) {
  code block
}
```

NOTE : Every method is written inside a *CLASS*.

CALLING A METHOD

It is basically using the method

```
method_name(parameter);
```

NOTE : The *main()* method is automatically called when we run the JAVA program.

- It is the first method that is called.
- It is the starting point of execution of a program.

1.2.4 ACCESS MODIFIERS

The access modifiers in JAVA specifies the accessibility of a field, method, constructor, or class. There are four types of JAVA access modifiers :

- ◇ **Private :** Access level only within the class, can not be accessed from outside the class.
- ◇ **Default :** Access level only within the package. If do not specify any access leve then it will be the default.
- ◇ **Protected :** Access level within the package and outside the package *through child class*. If do not make the child class, it can not be accessed from outside the package.
- ◇ **Public :** Access level everywhere, within or outside the class and package.

1.2.5 NAMING CONVENTIONS

How to write name in programming.

- ◇ **Pascal Case Convention :**
  - ThisIsAName
  - Naming *Class*

◇ **Camel Case Convention :**

  – thisIsAName
  – Naming *Methods & Variables*

◇ **Snake Case Convention :**

  – this_is_name

1.2.6 JAVA PROGRAM STRUCTURE

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

1.2.7 PACKAGE

A container for Classes.

NOTE :

- Package contains Classes
- Class contains Methods &
- Method contains code blocks

Chapter 2

File Handling In Java

2.1 Create, Delete and Get The Path Of The Dir.

```java
package fileHandling;

import java.io.File;

public class A_CreateDir {

  public static void main(String[] args) {
    // It will create the directory at the current project directory.
    File dir = new File("Test"); // we can use directory path also
    dir.mkdir();

    //---- Get The File Directory
    String dirPath = dir.getAbsolutePath();
    String name = dir.getName();
    System.out.println("Dir Name: "+name+"\nPath: "+dirPath);

    //---- Delete Directory If Exist
    if(dir.delete()) {
      System.out.println(name+" directory has been deleted.");
    }

  }
}
```

2.2 Create File and Delete File

```java
package fileHandling;

import java.io.File;
import java.util.*;

public class B_CreateFile {

  public static void main(String[] args) {
    //---- File Must Be Exist Other Wise Create Directory
    //---- Set The Directory path
    File dir = new File("TextFile");

    //------ Get The Directory Path
    String path = dir.getAbsolutePath();

    //---- Set File Name And Path
    File file1 = new File(path+"/file1.txt");
    File file2 = new File(path+"/file2.txt");

    try {

      // ---- Create New File
      file1.createNewFile();
      file2.createNewFile();

    } catch (Exception e) {
      System.out.println(e);
    }

    //--- Check If A File Exist
    if(file2.exists()) {
      //---- Delete File
      file2.delete();
    }
  }
}
```

2.3 Ways To Write Data Into a File in Java

2.3.1 FileWriter Class

- *FileWriter* is used for writing character data into a file.
- It‘s suitable for writing simple text-based data.
- You can write strings and characters directly to the file.

Code:

```java
package fileHandling;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Formatter;

public class WriteInFile {
  public static void main(String[] args) {
    FileWriter writer = null;

    try {
      writer = new FileWriter("Output.txt");

      char c = 'A';
      writer.write(c); //----- write single character
      writer.write('\n');

      char[] charArry = {'X', 'Y', 'Z'};
      writer.write(charArry); //--- Write array of characters
      writer.write('\n');

      String str = "This is a string";
      writer.write(str);  //--- write string
      writer.write('\n');

      writer.flush(); //- Ensure the data is immediately written to the file
      writer.close();
    } catch (IOException e) {
      System.out.println(e);
    }
  }
}
```

2.3.2 BufferedWriter Class

- This approach combines a **BufferedWriter** with a **FileWriter** to improve efficiency when writing large amounts of text data.
- It reduces the number of disk writes and is useful for optimizing performance

Code:

```java
package fileHandling;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class WriteInFile {

  public static void main(String[] args) {

    FileWriter fWriter = null;
    BufferedWriter bWriter = null;

    try {
      fWriter = new FileWriter("Output.txt");
      bWriter = new BufferedWriter(fWriter);

      bWriter.write('A');
      bWriter.newLine();
      bWriter.write("This is a String");
      bWriter.flush();

      fWriter.close();
      bWriter.close();
    } catch (IOException e) {
      System.out.println(e);
    }
  }

}
```

2.3.3 PrintWriter Class

- **PrintWriter** is useful for writing formatted text data into a file.
- It provides methods like **`printf`** and **`println`** for formatting output.

Code:

```java
package fileHandling;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class WriteInFile {

  public static void main(String[] args) {

    FileWriter fWriter = null;
    PrintWriter pWriter = null;

    try {
      fWriter = new FileWriter("Output.txt");
      pWriter = new PrintWriter(fWriter);


      pWriter.println("Hello World");
      pWriter.println("This is PrintWriter");
      pWriter.printf("Formatted Output: %s : %d + %d = %d %n", "Sum",5,7,12);
      pWriter.write("New Line");
      pWriter.flush();

      fWriter.close();
      pWriter.close();
    } catch (IOException e) {
      System.out.println(e);
    }
  }

}
```

2.4 Java Formatter Class

The Java *Formatter* class is defined in the *java.util* package and is declared final. It, therefore, cannot be extended or sub-classed.
With the help of this class, we can send formatted outputs to other outputs streams or devices, such as a GUI component or to a file apart from standard output.
Formatter Construction

- **Formatter() :**
- **Formatter(Appendable a) :**
- **Formatter(Appendable a, Locale loc) :**
- **Formatter(File file) :** The file parameter of this constructor designates a reference to a open file where the output will be streamed.

Using Formatter

- **%S or %s :** Specifies String
- **%X or %x :** Specifies hexadecimal integer
- **%o :** Specifies Octal integer
- **%d :** Specifies Decimal integer
- **%c :** Specifies character
- **%T or %t :** Specifies Time and Date
- **%n :** Insets newline character
- **%B or %b :** Specifies Boolean
- **%A or %a :** Specifies floating point hexadecimal
- **%f :** Specifies Decimal floating point

2.4.1 A Few Quick Examples

Using argument_index

```java
Formatter f2 = new Formatter();
f2.format("%2$s %1$s %3$s", "fear", "weakness","strengthen");
System.out.println(f2);
f2.close();

// Output: weakness fear strengthen
```

Regionalize Date

```java
Formatter f3=new Formatter();
f3.format(Locale.FRENCH,"%1$te %1$tB, %1$tY", Calendar.getInstance());
System.out.println(f3);
f3.close();

Formatter f4=new Formatter();
f4.format(Locale.ENGLISH,"%1$te %1$tB, %1$tY",Calendar.getInstance());
System.out.println(f4);
f4.close();

// Output: 7 octobre, 2023
//         7 October, 2023
```

Using %n and %% Specifiers

```java
Formatter f = new Formatter();
f.format("Format%n %.2f%% complete", 46.6);
System.out.println(f);
f.close();

// Output: Format
//      46.60% complete
```

Write in a file

```java
public class WriteInFile {
  public static void main(String[] args) {
    //--- "TestFile" directory must be exist in the project directory
    File file = new File("TextFile");
    String path = file.getAbsolutePath();
    System.out.println(path);

    try {
      Formatter formatter = new Formatter(path+"/file1.txt");
      formatter.format("%s %s %s\n", "21701002","Aong Cho","CSE");
      formatter.format("%s %s %s\n", "21701001","Taqi Ismile","CSE");
      formatter.close();
    } catch (Exception e) {
      System.out.println(e);
    }
  }
}
```