BEGINNING JAVA PROGRAMMING WITH HELLO WORLD EXAMPLE

- The process of Java programming can be simplified in three steps:
- Create the program by typing it into a text editor and saving it to a file – HelloWorld.java.
- Compile it by typing “javac HelloWorld.java” in the terminal window.
- Execute (or run) it by typing “java HelloWorld” in the terminal window.Below given program is the simplest program of Java printing “Hello World” to the screen. Let us try to understand every bit of code step by step.

```
/* This is a simple Java program.
Filename: "HelloWorld.java". */
class HelloWorld
{
// Your program begins with a call to main ().
// Prints "Hello, World" to the terminal window.
public static void main(String args[])
{
System.out.println("Hello, World");
}
}
```

Output:

Hello, World

The “Hello World!” program consists of three primary components: the HelloWorld class definition, the main method and source code comments. Following explanation will provide you with a basic understanding of the code:

Class definition: This line uses the keyword class to declare that a new class is being defined.

1. class HelloWorld
   HelloWorld is an identifier that is the name of the class. The entire class definition, including all of its members, will be between the opening curly brace {and the closing curly brace}.
2. main method: In Java programming language, every application must contain a main method whose signature is: public static void main(String[] args)
3. public: So that JVM can execute the method from anywhere.
4. static: Main method is to be called without object.

The modifiers public and static can be written in either order.

5. void: The main method doesn't return anything.
6. main(): Name configured in the JVM.
7. String[]: The main method accepts a single argument: an array of elements of type String.

Unit 3 [PPL]- Java Programs

Like in C/C++, main method is the entry point for your application and will subsequently invoke all the other methods required by your program.

The next line of code is shown here. Notice that it occurs inside main( ).

System.out.println("Hello, World");

This line outputs the string “Hello, World” followed by a new line on the screen. Output is actually accomplished by the built-in println( ) method. System is a predefined class that provides access to the system, and out is the variable of type output stream that is connected to the console.

8. Comments: They can either be multi-line or single line comments.

/* This is a simple Java program.

Call this file "HelloWorld.java". */

This is a multiline comment. This type of comment must begin with /* and end with */. For single line you may directly use // as in C/C++.

Important Points :

- The name of the class defined by the program is HelloWorld, which is same as name of file(HelloWorld.java).
- This is not a coincidence. In Java, all codes must reside inside a class and there is at most one public class which contain main() method.
- By convention, the name of the main class(class which contain main method) should match the name of the file that holds the program.

Compiling the program :

After successfully setting up the environment, we can open terminal in both Windows/Unix and can go to directory where the file – HelloWorld.java is present.

Now, to compile the HelloWorld program, execute the compiler – javac , specifying the name of the source file on the command line, as shown:

javac HelloWorld.java

The compiler creates a file called HelloWorld.class (in present working directory) that contains the bytecode version of the program. Now, to execute our program, JVM(Java Virtual Machine) needs to be called using java, specifying the name of the class file on the command line, as shown:

java HelloWorld

This will print “Hello World” to the terminal screen.

JAVA NAMING CONVENTIONS

- Below are some naming conventions of java programming language. They must be followed while developing software in java for good maintenance and readability of code. Java uses CamelCase as a practice for writing names of methods, variables, classes, packages and constants.
- Camel case in Java Programming: It consists of compound words or phrases such that each word or abbreviation begins with a capital letter or first word with a lowercase letter, rest all with capital.

1. Classes and Interfaces:

Class names should be nouns, in mixed case with the first letter of each internal word capitalised. Interfaces name should also be capitalized just like class names.

Use whole words and must avoid acronyms and abbreviations.

Examples:

a. interface Bicycle
b. class MountainBike implements Bicyle
c. interface Sport
d. class Football implements Sport

2. Methods:

Methods should be verbs, in mixed case with the first letter lowercase and with the first letter of each internal word capitalised.

Examples:

a. void changeGear(int newValue);
b. void speedUp(int increment);
c. void applyBrakes(int decrement);

3. Variables :

- Variable names should be short yet meaningful.
- Variables can also start with either underscore('_') or dollar sign '$' characters.
- Should be mnemonic i.e., designed to indicate to the casual observer the intent of its use.
- One-character variable names should be avoided except for temporary variables.
- Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.

Examples:

// variables for MountainBike class

a. int speed = 0;
b. int gear = 1;

4. Constant variables:

- Should be all uppercase with words separated by underscores ("_").

- There are various constants used in predefined classes like Float, Long, String etc.

Examples:

a. static final int MIN_WIDTH = 4;
   // Some Constant variables used in predefined Float class
b. public static final float POSITIVE_INFINITY = 1.0f / 0.0f;
c. public static final float NEGATIVE_INFINITY = -1.0f / 0.0f;
d. public static final float NaN = 0.0f / 0.0f;

5. Packages:

- The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, like com, edu, gov, mil, net, org.
- Subsequent components of the package name vary according to an organization's own internal naming conventions.

Examples:

a. com.sun.eng
b. com.apple.quicktime.v2
   // java.lang packet in JDK
c. java.lang

Stream Classes in Java | Byte Stream Classes

Streams in Java represent an ordered sequence of data. Java performs input and output operations in the terms of streams.
It uses the concept of streams to make I/O operations fast. For example, when we read a sequence of bytes from a binary file, actually, we're reading from an input stream.

Similarly, when we write a sequence of bytes to a binary file, we're writing to an output stream.

Byte Streams and Character Streams

Modern versions of Java platform define two types of I/O streams:

- Byte streams
- Character streams

Let's understand first the meaning of byte streams and character streams one by one.

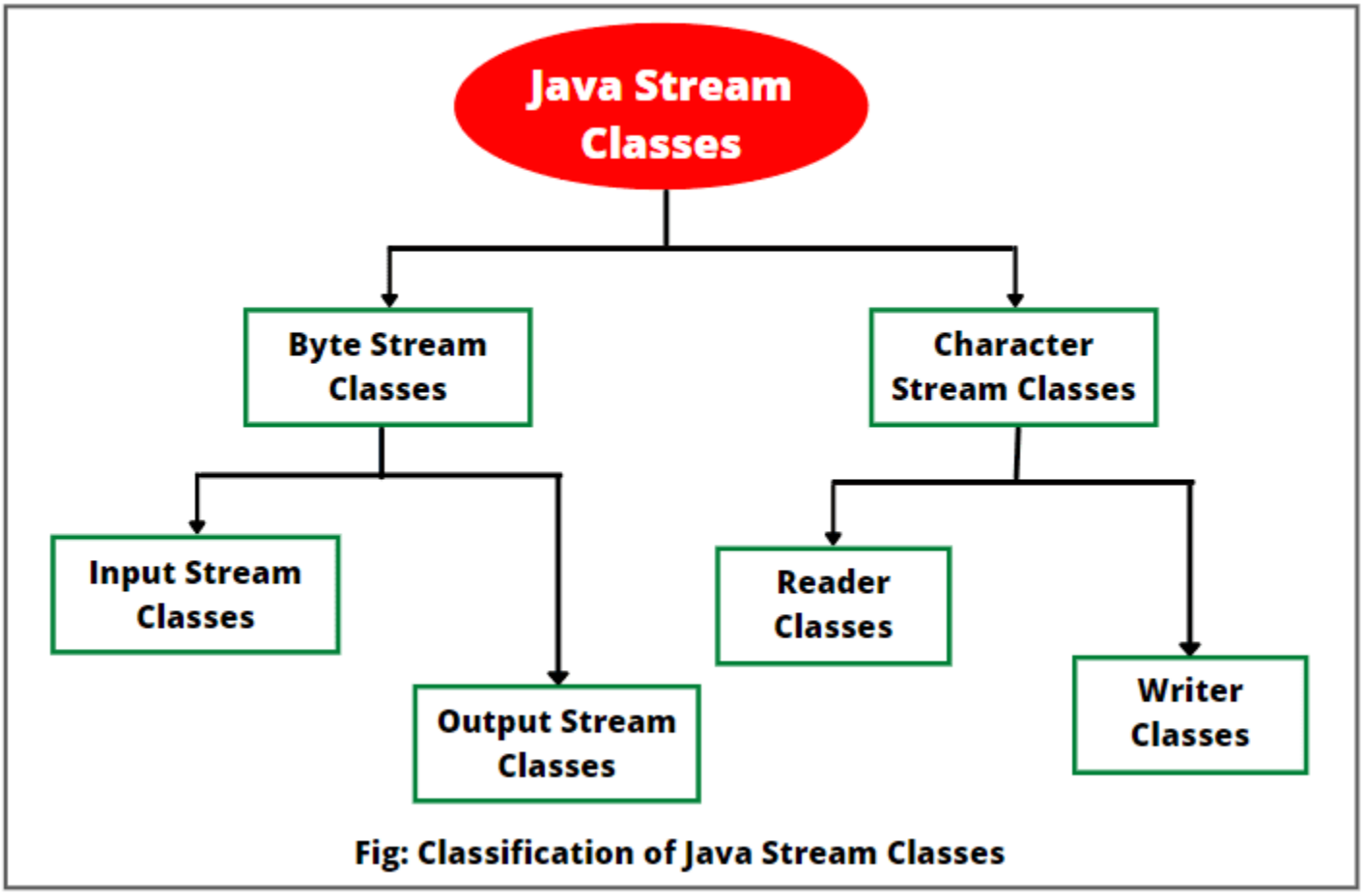

Fig: Classification of Java Stream Classes

JAVA USER INPUT

- In the Java program, there are 3 ways we can read input from the user in the command line environment to get user input,
  Java BufferedReader Class, Java Scanner Class, and Console class.
- We use the Scanner class to obtain user input. This program asks the user to enter an integer, a string, and float, and it will be printed on display.
- The scanner class in java.util is present so that we can add this package to our software.

3 Ways of Java User Input

There are three ways to read the User Input:

1. Java BufferedReader Class
2. Java Scanner Class
3. Using console Class

These three class are mentioned below; let us discuss them in detail:

1. Java BufferedReader Class

- It extends reader class.
- BufferedReader reads input from the character-input stream and buffers characters so as to provide an efficient reading of all the inputs.
- The default size is large for buffering. When the user makes any request to read, the corresponding request goes to the reader, and it makes a read request of the character or byte streams; thus, BufferedReader class is wrapped around another **input streams such as FileReader** or **InputStreamReaders.**

For example:

```
BufferedReader reader = new BufferedReader(new
FileReader("foo.in"));
```

BufferedReader can read data line by line using the method readLine() method.

BufferedReader can make the performance of code faster.

Constructors

BufferedReader has two constructors as follows:

Unit 3 [PPL]- Java Programs

1. BufferedReader(Reader reader): Used to create a buffered input character stream that uses the default size of an input buffer.

2. BufferedReader(Reader reader, input size): Used to create a buffered input character stream that uses the size provided for an input buffer.

Functions

- **int read:** It is used for reading a single character.
- **int read(char[] cbuffer, int offset, int length):** It is used for reading characters in the specified part of an array.
- **String readLine ():** Used to reading input line by line.
- **boolean ready():** Used to test whether the input buffer is ready to read.
- **long skip:** Used for skipping the characters.
- **void close():** It closes the input stream buffer and system resources associated with the stream.

When the user enters the character from the keyboard, it gets read by the device buffer and then from System.in it passes on to the buffered reader or input stream reader and gets stored in the input buffer.

Code:

```
import java.util.*;
import java.lang.*;
import java.io.*;
```

Unit 3 [PPL]- Java Programs

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
/*package whatever //do not write package name here */
class BufferedReaderDemo
{
public static void main (String[] args) throws
NumberFormatException, IOException {
System.out.println("Enter your number");
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
int t = Integer.parseInt(br.readLine());
System.out.println("Number you entered is: " + t);
System.out.println("Enter your string: ");
String s  = br.readLine();
System.out.println("String you entered is: " + s);
}
}
```

Output:

```
Enter your number:
7
Number you entered is: 7

Enter your string:
Java
String you entered is: Java
```

Program with reading from InputStreamReader and BufferedReader:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class BufferedReaderDemo {
public static void main(String args[]) throws IOException{
InputStreamReader reader = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(reader);
System.out.println("What is your name?");
String name=br.readLine();
System.out.println("Welcome "+name);
}
}
```

Output:

```
What is your name?
Alfiya
Welcome Alfiya
```

2. Java Scanner Class

- java.util. scanner class is one of the classes used to read user input from the keyboard. It is available at the util package.
- Scanner classes break the user input using a delimiter that is mostly whitespaces by default. The scanner has many methods to read console input of many primitive types such as double, int, float, long, Boolean, short, byte, etc.

- It is the simplest way to get input in java. Scanner class **implements Iterator** and Closeable interfaces. The scanner provides nextInt() and many primitive type methods to read inputs of primitive types.
- The next() method is used for string inputs.

Constructors

- **Scanner(File source):** It constructs a scanner to read from a specified file.
- **Scanner(File source, String charsetName):** It constructs a scanner to read from a specified file.
- **Scanner(InputStream source), Scanner(InputStream source, String charsetName):** It constructs a scanner to read from a specified input stream.
- **Scanner(0Readable source):** It constructs a scanner to read from a specified readable source.
- **Scanner(String source):** It constructs a scanner to read from a specified string source.
- **Scanner(ReadableByteChannel source):** It constructs a scanner to read from a specified channel source.
- **Scanner(ReadableByteChannel source, String charsetName):** It constructs a scanner to read from a specified channel source.

Functions

Below are mentioned the method to scan the primitive types from console input through Scanner class.

- nextInt(),
- nextFloat(),
- nectDouble(),
- nextLong(),
- nextShort(),
- nextBoolean(),

- nextDouble(),
- nextByte(),

Program to read from Scanner Class:

```
//Using scanner class.
import java.util.Scanner;
/*package whatever //do not write package name here */
class ScannerDemo {
public static void main (String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter your number");
int t = sc.nextInt();
System.out.println("Number you entered is: " + t);
System.out.println("Enter your string: ");
String s  = sc.next();
System.out.println("String you entered is: " + s);
}
}
```

Output:

```
Enter your number:
7
Number you entered is: 7
Enter your string:
Computer
String you entered is: Computer
```

Unit 3 [PPL]- Java Programs

3. Using console Class

Using the console class to read the input from the command-line interface. It does not work on IDE.

Code:

```
public class Main
{
public static void main(String[] args)
{
// Using Console to input data from user
System.out.println("Enter your data:");
String name = System.console().readLine();
System.out.println("You entered: "+name);
}
}
```

Output:

```
Enter your data:
Computer
You entered: Computer
```