Q. 1 Explain the following terms

I) Classes
II) Objects
III) Abstraction
IV) Inheritance
V) Polymorphism

Object

- 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.
- An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

- ***Collection of objects* is called class. It is a logical entity.**
- A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space
- A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

  - **Fields**
  - **Methods**
  - **Constructors**
  - **Blocks**
  - **Nested class and interface**

Inheritance

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

Polymorphism

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

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

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.

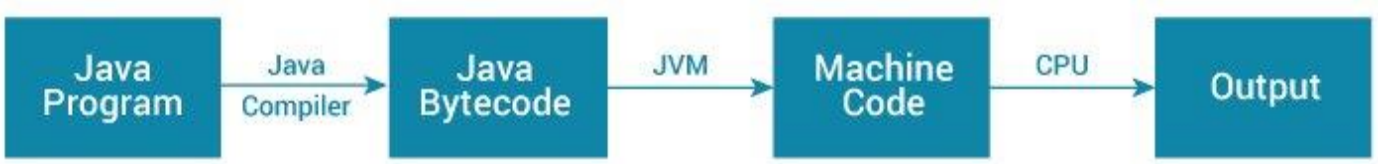
Q. 2 Explain JVM (java virtual machine) and importance of JVM in JRE.

- **What is JVM?**

JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.

When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

Java is a platform-independent language. It's because when you write Java code, it's ultimately written for JVM but not your physical machine (computer). Since JVM executes the Java bytecode which is platform-independent, Java is platform-independent.



- **What is JRE?**

JRE (Java Runtime Environment) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.

JRE is the superset of JVM.

If you need to run Java programs, but not develop them, JRE is what you need.
The components of JRE are as follows:

Deployment technologies, **User interface toolkits**, **Integration libraries**, **Other base libraries, Lang and util base libraries,Java Virtual Machine (JVM)**,

- **What is JDK?**

JDK (Java Development Kit) is a software development kit required to develop applications in Java. When you download JDK, JRE is also downloaded with it.

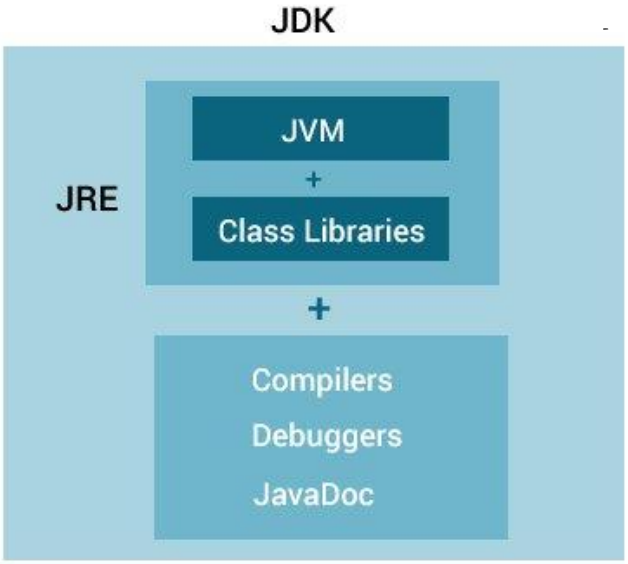
In addition to JRE, JDK also contains a number of development tools (compilers, JavaDoc, Java Debugger, etc).

- **Relationship between JVM, JRE, and JDK.**

Q. 3 Why JAVA is known as platform independent language? Explain.

What is a Platform, and What is a Platform-Independent Language?

The platform can be defined as a distinct combination of hardware, operating system, and software that provides an **environment to run programs**.



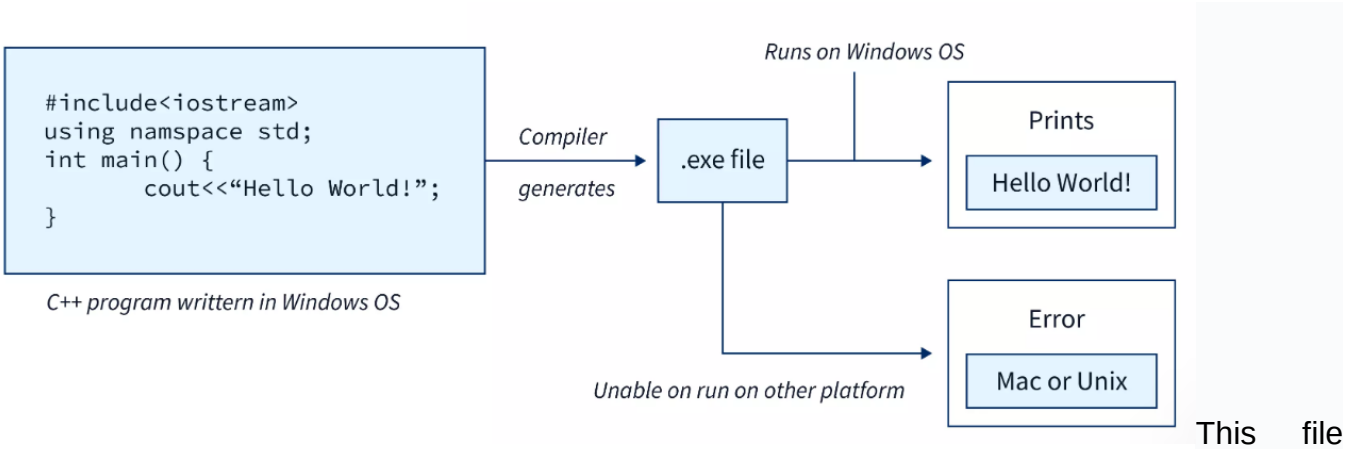
Java is called Platform Independent because programs written in Java can be run on multiple platforms without re-writing them individually for a particular platform, i.e., **Write Once Run Anywhere (WORA)**.

How Java Provides Platform Independence?

- To understand how Java facilitates platform independence, let us differentiate between the compilation process of other programming languages, mainly C/C++, and that of Java.
- The program written by the programmer, known as source code, is not understood by the computer. Source code is very similar to human language, consisting of words and phrases. It has to be converted into machine language code, which computers can easily understand and execute.
- Machine language code is a set of instructions to be executed by the computer. The compiler does this conversion. The process of **converting source code into machine code is called compilation**. Machine Language Code is unique for different platforms.

Ordinary Compilation Process

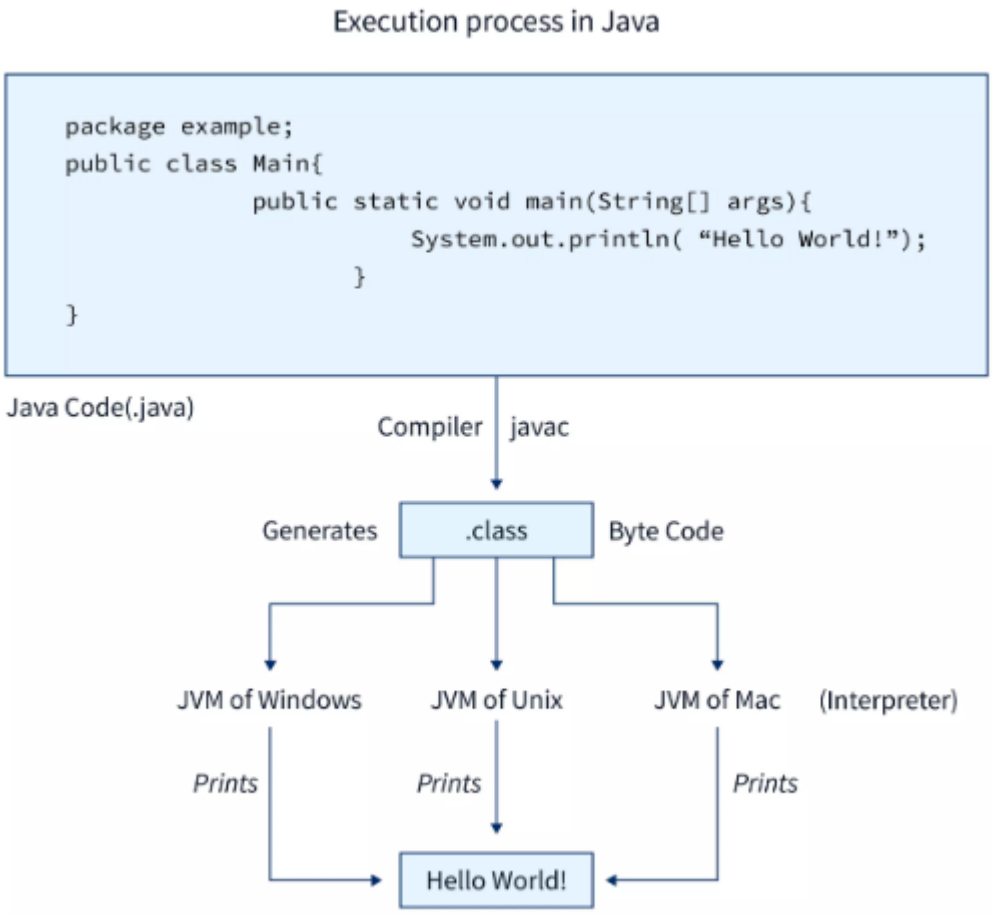
The code is first converted into machine-readable language when the C/C++ program is written and compiled in Windows OS. The compilation process will **generate natively executable code as a .exe file.** This natively generated code runs only on Windows Operating System.



This file cannot run on Mac, Unix, or other Operating Systems as the machine code is different for different Operating Systems. Thus, C/C++ is platform dependent programming language.

Step-by-Step Execution of Java Programs

- Contrary to other compilers, the Java compiler doesn't produce native executable files or code for a particular platform. The compilation process in Java instead **generates a special format called byte code**. This generated file is also referred to as a .class file.
- The Byte Code of Java is a set of machine instructions for a Java processor chip called Java Virtual Machine (JVM). Java Byte Code is very similar to machine language, but unlike machine language, Java byte code is absolutely the same on every platform. The byte code is not directly executable on any platform. Java code compiled into byte code still needs an interpreter to execute them on the platform.
- Java Virtual Machine or JVM is a special Java Interpreter. **Java Virtual Machine takes byte code as input. It interprets and executes the byte code and provides the output of the program**.
- **A compiler translates the entire program at once into machine-readable code. An Interpreter also converts the program to machine code, but it does so by translating it instruction-by-instruction or line-by-line.**
- The interpreter then converts the byte code into the native code. Native code is just like machine language code, which can be compiled to run with a particular processor and its set of instructions. Thus, native code can be executed by an individual operating system. Java Byte



Code is platform-independent but **natively executable code generated from byte code using an interpreter is highly platform-dependent** and cannot run on other platforms.

- For example, source code written in windows OS is compiled by the javac compiler of windows, and the .class file (byte code) is created. JVM of Unix, MAC, or any other platform can interpret this byte code.
- Hence, the JVM of that specific platform interprets the byte code and generates natively executable code for the platform. This native code is readily understood by the system and is executed, and it prints Hello World!
- Using byte code, a natively executable code can be generated on any platform using a JVM interpreter, irrespective of the platform on which the byte code was created. But the natively executable code of one platform cannot be used for other platforms, i.e., the native code of Unix cannot run on Windows or MAC.

Q. 4. What is an operator? Explain the different types of operators.

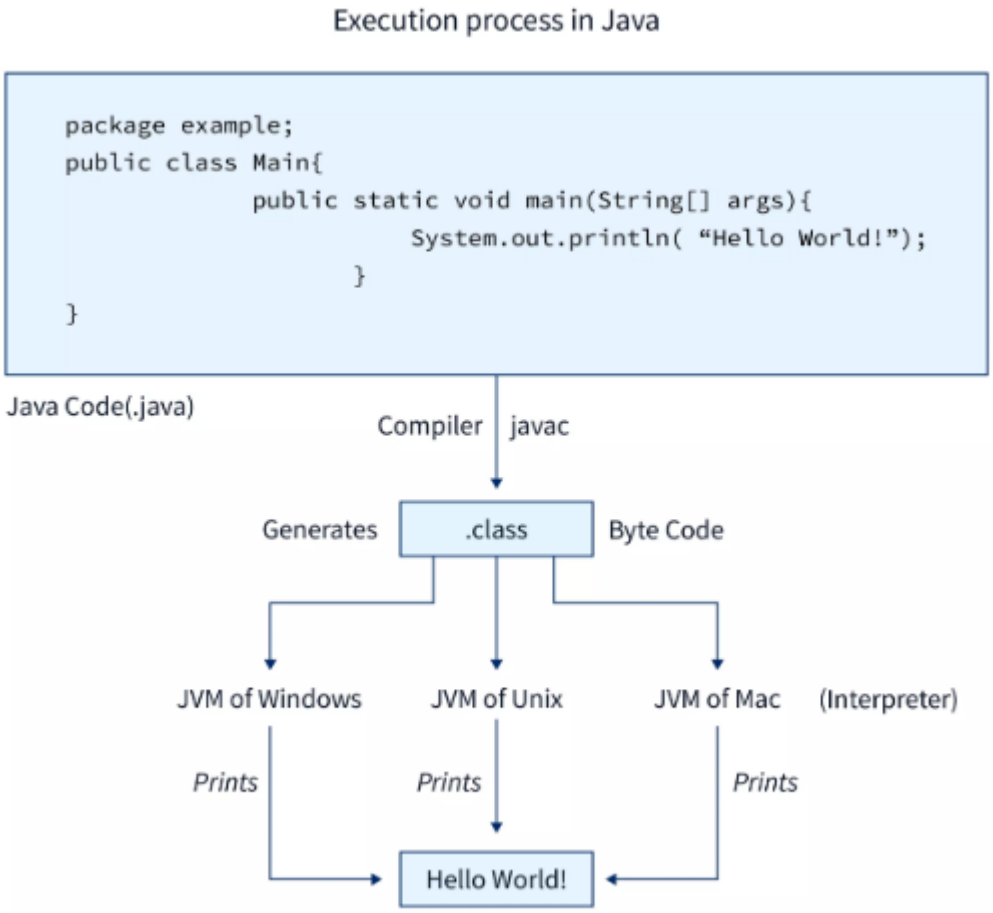
Refer https://www.javatpoint.com/

Q. 5/6. Explain steps for executing a JAVA program. What are source file and byte code file?

Step-by-Step Execution of Java Programs

- Contrary to other compilers, the Java compiler doesn't produce native executable files or code for a particular platform. The compilation process in Java instead **generates a special format called byte code**. This generated file is also referred to as a .class file.
- The Byte Code of Java is a set of machine instructions for a Java processor chip called Java Virtual Machine (JVM). Java Byte Code is very similar to machine language, but unlike machine language, Java byte code is absolutely the same on every platform. The byte code is not directly executable on any platform. Java code compiled into byte code still needs an interpreter to execute them on the platform.
- Java Virtual Machine or JVM is a special Java Interpreter. **Java Virtual Machine takes byte code as input. It interprets and executes the byte code and provides the output of the program**.

- **A compiler translates the entire program at once into machine-readable code. An Interpreter also converts the program to machine code, but it does so by translating it instruction-by-instruction or line-by-line.**
- The interpreter then converts the byte code into the native code. Native code is just like machine language code, which can be compiled to run with a particular processor and its set of instructions. Thus, native code can be executed by an individual operating system. Java Byte Code is platform-independent but **natively executable code generated from byte code using**



an interpreter is highly platform-dependent and cannot run on other platforms.

- For example, source code written in windows OS is compiled by the javac compiler of windows, and the .class file (byte code) is created. JVM of Unix, MAC, or any other platform can interpret this byte code.
- Hence, the JVM of that specific platform interprets the byte code and generates natively executable code for the platform. This native code is readily understood by the system and is executed, and it prints Hello World!
- Using byte code, a natively executable code can be generated on any platform using a JVM interpreter, irrespective of the platform on which the byte code was created. But the natively executable code of one platform cannot be used for other platforms, i.e., the native code of Unix cannot run on Windows or MAC.

8. Differentiate between Procedural Language and Object Oriented Programming

S.no.	On the basis of	Procedural Programming	Object-oriented programming
1.	**Definition**	It is a programming language that is derived from structure programming and based upon the concept of calling procedures. It follows a step-by-step approach in order to break down a task into a set of variables and routines via a sequence of instructions.	Object-oriented programming is a computer programming design philosophy or methodology that organizes/ models software design around data or objects rather than functions and logic.
2.	**Security**	It is less secure than OOPs.	Data hiding is possible in object-oriented programming due to abstraction. So, it is more secure than procedural programming.
3.	**Approach**	It follows a top-down approach.	It follows a bottom-up approach.
4.	**Data movement**	In procedural programming, data moves freely within the system from one function to another.	In OOP, objects can move and communicate with each other via member functions.
5.	**Orientation**	It is structure/procedure-oriented.	It is object-oriented.
6.	**Access modifiers**	There are no access modifiers in procedural programming.	The access modifiers in OOP are named as private, public, and protected.
7.	**Inheritance**	Procedural programming does not have the concept of inheritance.	There is a feature of inheritance in object-oriented programming.
8.	**Code reusability**	There is no code reusability present in procedural programming.	It offers code reusability by using the feature of inheritance.
9.	**Overloading**	Overloading is not possible in procedural programming.	In OOP, there is a concept of function overloading and operator overloading.
10.	**Importance**	It gives importance to functions over data.	It gives importance to data over functions.

11.	**Virtual class**	In procedural programming, there are no virtual classes.	In OOP, there is an appearance of virtual classes in inheritance.
12.	**Complex problems**	It is not appropriate for complex problems.	It is appropriate for complex problems.
13.	**Data hiding**	There is not any proper way for data hiding.	There is a possibility of data hiding.
14.	**Program division**	In Procedural programming, a program is divided into small programs that are referred to as functions.	In OOP, a program is divided into small parts that are referred to as objects.
15.	**Examples**	Examples of Procedural programming include C, Fortran, Pascal, and VB.	The examples of object-oriented programming are -.NET, C#, Python, Java, VB.NET, and C++.

9. What is Primitive data type conversion and how does it differs from casting? Explain with an Example?

Type Casting:

In typing casting, a data type is converted into another data type by the programmer using the casting operator during the program design. In typing casting, the destination data type may be smaller than the source data type when converting the data type to another data type, that's why it is also called narrowing conversion.

Syntax/Declaration:-

destination_datatype = (target_datatype)variable;

(): is a casting operator.
target_datatype: is a data type in which we want to convert the source data type.

Type Casting example –

```
float x;

byte y;

...

...

y=(byte)x;  //Line 5
```

In Line 5: you can see that, we are converting **float(source) data type** into **byte(target) data type**.

2. Type conversion: In type conversion, a data type is automatically converted into another data type by a compiler at the compiler time. In type conversion, the destination data type cannot be smaller than the source data type, that's why it is also called widening conversion. One more important thing is that it can only be applied to compatible data types.

Type Conversion example –

```
int x=30;
float y;
y=x;  // y==30.000000.
```

S.N.	Type Casting	Type Conversion
1	Type casting is a mechanism in which one data type is converted to another data type using a casting () operator by a programmer.	Type conversion allows a compiler to convert one data type to another data type at the compile time of a program or code.
2	It can be used both compatible data type and incompatible data type.	Type conversion is only used with compatible data types, and hence it does not require any casting operator.
3	It requires a programmer to manually casting one data into another type.	It does not require any programmer intervention to convert one data type to another because the compiler automatically compiles it at the run time of a program.
4	It is used while designing a program by the programmer.	It is used or take place at the compile time of a program.
5	When casting one data type to another, the destination data type must be smaller than the source data.	When converting one data type to another, the destination type should be greater than the source data type.
6	It is also known as narrowing conversion because one larger data type converts to a smaller data type.	It is also known as widening conversion because one smaller data type converts to a larger data type.
7	It is more reliable and efficient.	It is less efficient and less reliable.

8	There is a possibility of data or information being lost in type casting.	In type conversion, data is unlikely to be lost when converting from a small to a large data type.

8	float b = 3.0;
	int a = (int) b
		int x = 5, y = 2, c;
		float q = 12.5, p;
		p = q/x;

10. Explain various data type supported in java with example

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

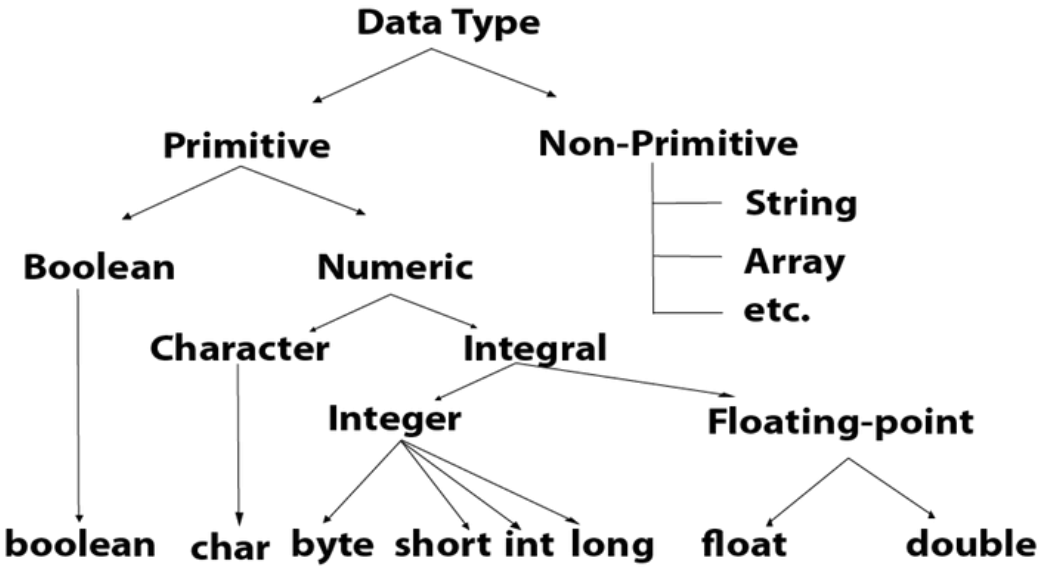
1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type



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

Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example:

Boolean one = **false**

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example:

byte a = 10, **byte** b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example:

short s = 10000, **short** r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is -2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example:

int a = 100000, **int** b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example:

long a = 100000L, **long** b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example:

float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example:

double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example:

char letterA = 'A'

11. Write a for, while, do while program to compute the following program

i) 1*2*3* ……* 20 **ii) 4 + 8 + 12 +….. + 80**

I)

Code –

```
public class Example
{
public static void main(String args[])
{
        int m=1;
```

```
        for(int i=1;i<=20;i++)
        {
        m = m*i;
        }
System.out.println(“Product = “ +m);
}
}
```

Output –

To compile - javac Example.java
To Run - java Example

ii)

Code –

```
public class Example
{
public static void main(String args[])
{
        int s=0;
        for(int i=4;i<=80;i=i+4)
        {
        s + s + I;
        }
System.out.println(“Sum = “ +s);
}
}
```

Output –

To compile - javac Example.java
To Run - java Example

12. Write a java program to print the sum of square of first 10 even numbers.

Code –

```
public class Example
{
public static void main(String args[])
{
        int s=0;
        for(int i=2;i<=20;i=i+2)
        {
        s = s+i*i;
        }
```

```
System.out.println(“Sum = “ +s);
}
}
```

Output –

To compile - javac Example.java
To Run - java Example

13. Write a JAVA program that displays area, circumference of circle whose diameter is 10 cm.

```
public class Example
{
public static void main(String args[])
{
        int d=10;
        float area = 3.14 * (d/2) * (d/2);
        float circumference = 2 * 3.14 * (d/2);

        System.out.println(“Area of Circle = “ +area);
        System.out.println(“Circumference of Circle = “ +circumference);
}
}
```

Output –

To compile - javac Example.java
To Run - java Example