QUESTION BANK

UNIT-1

1. **Explain the history and versions of Java?**

Answer:

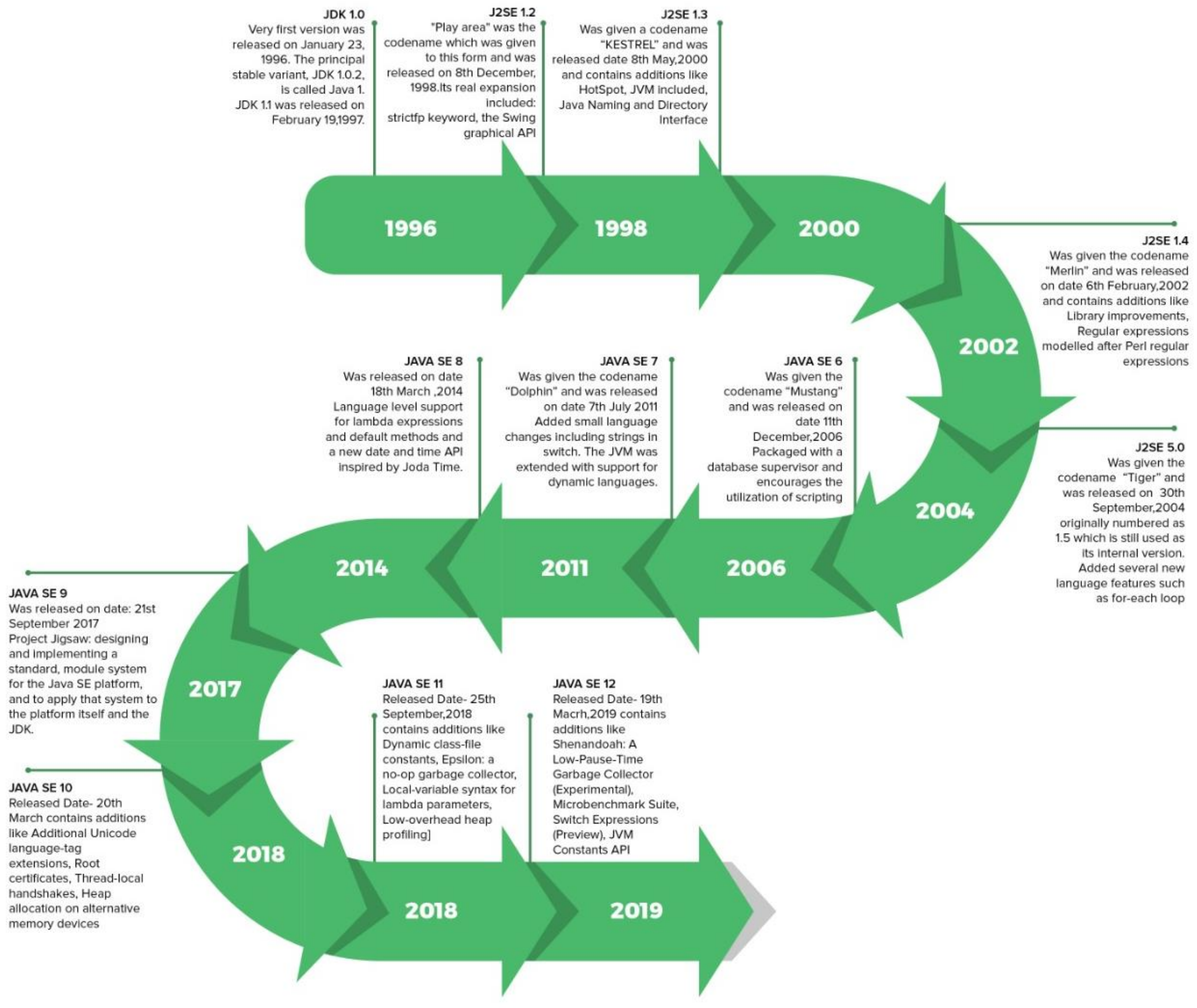
Java is an *Object-Oriented programming language* developed by **James Gosling** in the early *1990s*. The team initiated this project to develop a language for digital devices such as set-top boxes, television, etc. Originally C++ was considered to be used in the project but the idea was rejected for several reasons(For instance C++ required more memory). Gosling endeavoured to alter and expand C++ however before long surrendered that for making another stage called **Green**. James Gosling and his team called their project "**Greentalk**" and its file extension was **.gt** and later is known as "**OAK**".

The name **Oak** was used by **Gosling** after an **oak tree** that remained outside his office. Also, Oak is an image of solidarity and picked as a national tree of numerous nations like the U.S.A., France, Germany, Romania, etc. But they had to later rename it as "**JAVA**" as it was already a trademark by **Oak Technologies**.

Gosling and his team did a brainstorm session and after the session, they came up with several names such as **JAVA, DNA, SILK, RUBY, etc.**

Java name was decided after much discussion since it was so unique. The name Java originates from a sort of **espresso bean**, Java. Gosling came up with this name while having a coffee near his office. Java was created on the principles like **Robust, Portable, Platform Independent, High Performance, Multithread, etc.** and was called one of the **Ten Best Products of 1995** by the **TIME MAGAZINE**.

Currently, Java is used in **internet programming, mobile devices, games, e-business solutions, etc.** The Java language has experienced a few changes since **JDK 1.0** just as various augmentations of classes and packages to the standard library. In Addition to the language changes, considerably more sensational changes have been made to the Java Class Library throughout the years, which has developed from a couple of hundred classes in JDK 1.0 to more than three thousand in J2SE 5.



JDK 1.0
Very first version was released on January 23, 1996. The principal stable variant, JDK 1.0.2, is called Java 1. JDK 1.1 was released on February 19,1997.
J2SE 1.2
"Play area" was the codename which was given to this form and was released on 8th December, 1998.Its real expansion included: strictfp keyword, the Swing graphical API
J2SE 1.3
Was given a codename "KESTREL" and was released date 8th May,2000 and contains additions like HotSpot, JVM included, Java Naming and Directory Interface
1996
1998
2000
J2SE 1.4
Was given the codename "Merlin" and was released on date 6th February,2002 and contains additions like Library improvements, Regular expressions modelled after Perl regular expressions
2002
J2SE 5.0
Was given the codename "Tiger" and was released on 30th September,2004 originally numbered as 1.5 which is still used as its internal version. Added several new language features such as for-each loop
2004
JAVA SE 6
Was given the codename "Mustang" and was released on date 11th December,2006 Packaged with a database supervisor and encourages the utilization of scripting
2006
JAVA SE 7
Was given the codename "Dolphin" and was released on date 7th July 2011 Added small language changes including strings in switch. The JVM was extended with support for dynamic languages.
2011
JAVA SE 8
Was released on date 18th March ,2014 Language level support for lambda expressions and default methods and a new date and time API inspired by Joda Time.
2014
JAVA SE 9
Was released on date: 21st September 2017 Project Jigsaw: designing and implementing a standard, module system for the Java SE platform, and to apply that system to the platform itself and the JDK.
2017
JAVA SE 10
Released Date- 20th March contains additions like Additional Unicode language-tag extensions, Root certificates, Thread-local handshakes, Heap allocation on alternative memory devices
2018
JAVA SE 11
Released Date- 25th September,2018 contains additions like Dynamic class-file constants, Epsilon: a no-op garbage collector, Local-variable syntax for lambda parameters, Low-overhead heap profiling]
2018
JAVA SE 12
Released Date- 19th Macrh,2019 contains additions like Shenandoah: A Low-Pause-Time Garbage Collector (Experimental), Microbenchmark Suite, Switch Expressions (Preview), JVM Constants API
2019

History of various Java versions:

VERSION	RELEASE DATE	MAJOR CHANGES
JDK Beta	1995	Initial Stage
JDK 1.0	January 1996	The Very first version was released on January 23, 1996. The principal stable variant, JDK 1.0.2, is called Java 1.
JDK 1.1	February 1997	Was released on February 19, 1997. There were many additions in JDK 1.1 as compared to version 1.0 such as • A broad retooling of the AWT occasion show • Inner classes added to the language • JavaBeans • JDBC • RMI
J2SE 1.2	December 1998	"Play area" was the codename which was given to this form and was released on 8th December 1998. Its real expansion included: strictfp keyword • the Swing graphical API was coordinated into the centre classes • Sun's JVM was outfitted with a JIT compiler out of the blue • Java module • Java IDL, an IDL usage for CORBA interoperability • Collections system
J2SE 1.3	May 2000	Codename-"KESTREL" Release Date- 8th May 2000 Additions: • HotSpot JVM included • Java Naming and Directory Interface • JPDA • JavaSound • Synthetic proxy classes
J2SE 1.4	February 2002	Codename-"Merlin" Release Date- 6th February 2002 Additions: Library improvements • Regular expressions modelled after Perl regular expressions • The image I/O API for reading and writing images in formats like JPEG and PNG • Integrated XML parser and XSLT processor (JAXP) (specified in JSR 5 and JSR 63) • Preferences API (java.util.prefs) Public Support and security updates for this version ended in October 2008.
J2SE 5.0	September 2004	Codename-"Tiger" Release Date- "30th September 2004" Originally numbered as 1.5 which is still used as its internal version. Added several new language features such as: • for-each loop • Generics • Autoboxing • Var-args
JAVA SE 6	December 2006	Codename-"Mustang" Released Date- 11th December 2006 Packaged with a database supervisor and encourages the utilization of scripting languages with the JVM. Replaced the name J2SE with ava SE and dropped the .0 from the version number. Additions:

		• Upgrade of JAXB to version 2.0: Including integration of a StAX parser. • Support for pluggable annotations (JSR 269). • JDBC 4.0 support (JSR 221)
JAVA SE 7	July 2011	Codename-"Dolphin" Release Date- 7th July 2011 Added small language changes including strings in the switch. The JVM was extended with support for dynamic languages. Additions: • Compressed 64-bit pointers. • Binary Integer Literals. • Upstream updates to XML and Unicode.
JAVA SE 8	March 2014	Released Date- 18th March 2014 Language level support for lambda expressions and default methods and a new date and time API inspired by Joda Time.
JAVA SE 9	Septemb er 2017	Release Date: 21st September 2017 Project Jigsaw: designing and implementing a standard, a module system for the Java SE platform, and to apply that system to the platform itself and the JDK.
JAVA SE 10	March 2018	Released Date- 20th March Addition: • Additional Unicode language-tag extensions • Root certificates • Thread-local handshakes • Heap allocation on alternative memory devices • Remove the native-header generation tool – javah. • Consolidate the JDK forest into a single repository.
JAVA SE 11	Septemb er 2018	Released Date- 25th September, 2018 Additions- • Dynamic class-file constants • Epsilon: a no-op garbage collector • The local-variable syntax for lambda parameters • Low-overhead heap profiling • HTTP client (standard) • Transport Layer Security (TLS) 1.3 • Flight recorder
JAVA SE 12	March 2019	Released Date- 19th March 2019 Additions- • Shenandoah: A Low-Pause-Time Garbage Collector (Experimental) • Microbenchmark Suite • Switch Expressions (Preview) • JVM Constants API • One AArch64 Port, Not Two • Default CDS Archives

2. Can main() method in Java can return any data?

Answer:

Java main method doesn't **return** anything, that's why it's **return type** is void . This has been done to keep things simple because once the **main method** is finished executing, **java** program terminates. So, there is no point in **returning** anything, there is nothing that can be done for the **returned** object by JVM.

Example:

```
public class Sample
{
   public static void main(String args[])
   {
      System.out.println("Contents of the main method");
      return 20;
   }
}
```

Output

```
Sample.java:4: error: incompatible types: unexpected return value
   return 20;
          ^
1 error
```

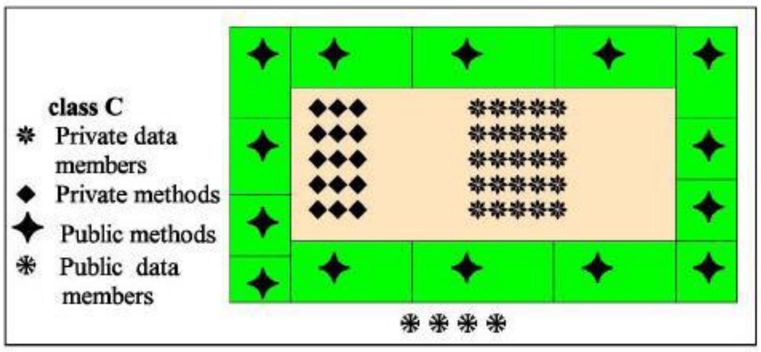
Therefore, you cannot return any value from main.

3. **Give an example of use of Pointers in Java class.**
   Answer:

4. **List and explain the applications of OOPs.**
   Answer:
   Main application areas of OOP are:
   - User interface design such as windows, menu etc.
   - Real Time Systems
   - Simulation and Modelling
   - Object oriented databases
   - AI and Expert System
   - Neural Networks and parallel programming
   - Decision support and office automation systems etc.

   Explanation:
   - It is easy to model a real system as real objects are represented by programming objects in OOP. The objects are processed by their member data and functions. It is easy to analyse the user requirements.
   - With the help of inheritance, we can reuse the existing class to derive a new class such that the redundant code is eliminated and the use of existing class is extended. This saves time and cost of program.
   - In OOP, data can be made private to a class such that only member functions of the class can access the data. This principle of data hiding helps the programmer to build a secure program that cannot be invaded by code in other part of the program.
   - With the help of polymorphism, the same function or same operator can be used for different purposes. This helps to manage software complexity easily.
   - Large problems can be reduced to smaller and more manageable problems. It is easy to partition the work in a project based on objects.
   - It is possible to have multiple instances of an object to co-exist without any interference i.e. each object has its own separate member data and function.

5. **What is meant by Encapsulation?**
   Answer:
   Encapsulation
   The packing of data and functions into a single component is known as encapsulation.
   C++ supports the features of encapsulation using classes. The data is not accessible by outside functions. Only those functions that are able to access the data are defined within the class. These functions prepare the interface between the object's data and the program.



6. **List out the uses and applications of java?**
   Answer:
   <u>Uses and Applications of Java</u>:
   Uses of java: According to Sun, 3 billion devices run java. There are many devices where java is currently used. Some of them are as follows:

i. Desktop Applications such as acrobat reader, media player, antivirus etc.
ii. Web Applications such as irctc.co.in, javatpoint.com etc.
iii. Enterprise Applications such as banking applications.
iv. Mobile
v. Embedded System
vi. Smart Card
vii. Robotics
viii. Games etc.

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

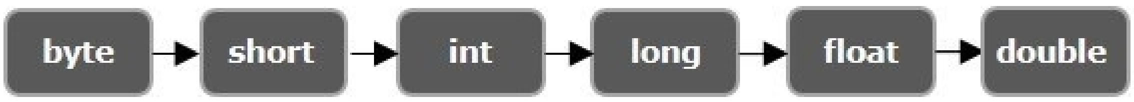
i) **Standalone Application**: It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.
ii) **Web Application**: An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.
iii) **Enterprise Application**: An application that is distributed in nature, such as banking applications etc. It has the advantage of high-level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.
iv) **Mobile Application**: An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

7. **With an example explain primitive type conversion and casting.**
Answer:
Java provides various datatypes to store various data values. It provides 7 primitive datatypes (stores single values) as listed below −

- boolean − Stores 1-bit value representing true or, false.
- byte − Stores twos compliment integer up to 8 bits.
- char − Stores a Unicode character value up to 16 bits.
- short − Stores an integer value upto 16 bits.
- int − Stores an integer value upto 32 bits.
- long − Stores an integer value upto 64 bits.
- float − Stores a floating-point value upto 32bits.
- double − Stores a floating-point value up to 64 bits.

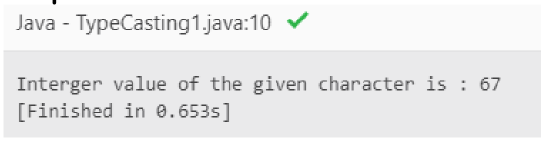
TYPE CASTING/TYPE CONVERSION

Converting one primitive datatype into another is known as type casting (type conversion) in Java. You can cast the primitive datatypes in two ways namely, Widening and, Narrowing.
Widening − Converting a lower datatype to a higher datatype is known as widening. In this case the casting/conversion is done automatically therefore, it is known as implicit type casting. In this case both datatypes should be compatible with each other.



Example:
```java
public class TypeCasting1
{
   public static void main(String args[])
   {
      char ch = 'C';
      int i = ch;
      System.out.println("Interger value of the given character is : "+i);
   }
}
```

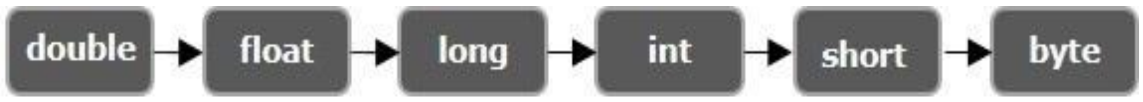
Output:



```
Java - TypeCasting1.java:10 ✓

Interger value of the given character is : 67
[Finished in 0.653s]
```

Narrowing – Converting a higher datatype to a lower datatype is known as narrowing. In this case the casting/conversion is not done automatically, you need to convert explicitly using the cast operator "( )" explicitly. Therefore, it is known as explicit type casting. In this case both datatypes need not be compatible with each other.



Example:

```
import java.util.Scanner;
public class NarrowingExample
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter an integer value: ");
        int i = sc.nextInt();
        char ch = (char) i;
        System.out.println("Character value of the given integer: "+ch);
    }
}
```

Output

```
Enter an integer value:
67
Character value of the given integer: C
```

8. With an example explain the different types of variables in java.

Answer:

A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used. Following is the basic form of a variable declaration:

data type variable [= value][, variable [= value] ...] ;

Here *data type* is one of Java's datatypes and *variable* is the name of the variable. To declare more than one variable of the specified type, you can use a comma-separated list.

Following are valid examples of variable declaration and initialization in Java –

Example

```
int a, b, c;         // Declares three integers, a, b, and c.
int a = 10, b = 10;  // Example of initialization
byte B = 22;         // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a';        // the char variable 'a' is initialized with value 'a'
```

There are three kinds of variables in Java –

- Local variables
- Instance variables
- Class/Static variables

Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.

- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Example

Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to only this method.

```
public class Test
{
    public void pupAge()
    {
            int age = 0;
            age = age + 7;
            System.out.println("Puppy age is : " + age);
    }
    public static void main(String args[])
    {
            Test test = new Test();
            test.pupAge();
    }
}
```

Output

```
Puppy age is: 7
```

Example

Following example uses *age* without initializing it, so it would give an error at the time of compilation.

```
public class Test
{
    public void pupAge()
    {
            int age;
            age = age + 7;
            System.out.println("Puppy age is : " + age);
    }
    public static void main(String args[])
    {
            Test test = new Test();
            test.pupAge();
    }
}
```

This will produce the following error while compiling it –

Output

```
Test.java:4:variable number might not have been initialized
age = age + 7;
       ^
1 error
```

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.

- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

Example

```
import java.io.*;
public class Employee
{
    public String name;      // this instance variable is visible for any child class.
    private double salary;     // salary  variable is visible in Employee class only.
    public Employee (String empName)      // The name variable is assigned in the constructor
    {
            name = empName;
    }
    public void setSalary(double empSal)   // The salary variable is assigned a value.
    {
            salary = empSal;
    }
    public void printEmp()                       // This method prints the employee details.
    {
            System.out.println("name  : " + name );
            System.out.println("salary :" + salary);
    }
    public static void main(String args[])
    {
            Employee empOne = new Employee("Ransika");
            empOne.setSalary(1000);
            empOne.printEmp();
    }
}
```

Output

```
name  : Ransika
salary :1000.0
```

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.

- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Example

```
import java.io.*;
public class Employee
{
    private static double salary;    // salary  variable is a private static variable
    public static final String DEPARTMENT = "Development ";   // DEPARTMENT is a constant
    public static void main(String args[])
    {
            salary = 1000;
            System.out.println(DEPARTMENT + "average salary:" + salary);
    }
}
```

Output

```
Development average salary:1000
```

9. Explain the structure of Java Program?

Answer:

A typical structure of a Java program contains the following elements:

- Package declaration
- Import statements
- Comments
- Class definition
- Class variables, Local variables
- Methods/Behaviors

Package declaration:

A class in Java can be placed in different **directories/packages** based on the module they are used. For all the classes that belong to a **single parent source directory**, a path from source directory is considered as **package declaration**.

Import statements:

There can be classes written in other **folders/packages** of our working java project and also there are many classes written by individuals, companies, etc which can be useful in our program. To use them in a class, we need to **import** the class that we intend to use. Many classes can be imported in a single program and hence multiple import statements can be written.

Comments:

The comments in Java can be used to **provide information about the variable, method, class or any other statement**. It can also be used to hide the program code for a specific time.

Class Definition:

A name should be given to a **class** in a java file. This name is used while creating an **object of a class**, in other classes/programs.

Variables:

The Variables are **storing the values of parameters** that are required during the execution of the program. Variables declared with modifiers have **different scopes,** which define the life of a variable.

Main Method:

Execution of a Java application starts from the main method. In other words, it's an **entry point for the class** or **program** that starts in **Java Run-time**.

Methods/Behaviors:

A set of instructions which form a **purposeful functionality** that can be required to run multiple times during the execution of a program. To not repeat the same set of instructions when the same functionality is required, the instructions are enclosed in a method. A method's behavior can be exploited by **passing variable values** to a method.

Example:

```
package abc; // A package declaration
import java.util.*; // declaration of an import statement
   // This is a sample program to understand basic structure of Java (Comment Section)
public class JavaProgramStructureTest // class name
{
    int repeat = 4; // global variable
    public static void main(String args[]) // main method
    {
        JavaProgramStructureTest test = new JavaProgramStructureTest();
        test.printMessage("Welcome to Tutorials Point");
    }
    public void printMessage(String msg) // method
    {
        Date date = new Date(); // variable local to method
        for(int index = 0; index < repeat; index++) // Here index - variable local to for loop
        {
            System.out.println(msg + "From" + date.toGMTString());
        }
    }
}
```

Output

```
Welcome to Tutorials Point from 2 Jul 2019 08:35:15 GMT
Welcome to Tutorials Point from 2 Jul 2019 08:35:15 GMT
Welcome to Tutorials Point from 2 Jul 2019 08:35:15 GMT
Welcome to Tutorials Point from 2 Jul 2019 08:35:15 GMT
```

10. Differentiate between JDK, JRE, and JVM

Answer:

Following are the important differences between JDK,JRE and JVM

Key	JDK	JRE	JVM
Definition	JDK (Java Development Kit) is a software development kit to develop applications in Java. In addition to JRE, JDK also contains number of development tools (compilers, JavaDoc, Java Debugger etc.).	JRE (Java Runtime Environment) is the implementation of JVM and is defined as a software package that provides Java class libraries, along with Java Virtual Machine (JVM), and other components to run applications written in Java programming.	JVM (Java Virtual Machine) is an abstract machine that is platform-dependent and has three notions as a specification, a document that describes requirement of JVM implementation, implementation, a computer program that meets JVM requirements, and instance, an implementation that executes Java byte code provides a runtime environment for executing Java byte code.
Prime functionality	JDK is primarily used for code execution and has prime functionality of development.	On other hand JRE is majorly responsible for creating environment for code execution.	JVM on other hand specifies all the implementations and responsible to provide these implementations to JRE.
Platform Independence	JDK is platform dependent i.e. for different platforms different JDK required.	Like of JDK JRE is also platform dependent.	JVM is platform independent.

Key	JDK	JRE	JVM
Tools	As JDK is responsible for prime development so it contains tools for developing, debugging and monitoring java application.	On other hand JRE does not contain tools such as compiler or debugger etc. Rather it contains class libraries and other supporting files that JVM requires to run the program.	JVM does not include software development tools.
Implementation	JDK = Java Runtime Environment (JRE) + Development tools	JRE = Java Virtual Machine (JVM) + Libraries to run the application	JVM = Only Runtime environment for executing the Java byte code.

11. What kind of variables a class consists of? Explain briefly any two?

Answer:

There are three different types of variables a class can have in Java are **local variables, instance variables**, and **class/static** variables.

Local Variable:

A **local variable** in Java can be declared locally in **methods**, **code blocks**, and **constructors**. When the program control enters the **methods, code blocks**, and **constructors** then the local variables are **created** and when the program control leaves the methods, code blocks, and constructors then the local variables are **destroyed**. A local variable **must be initialized** with some value.

Example:

```
public class LocalVariableTest
{
    public void show()
        {
            int num = 100; // local variable
            System.out.println("The number is : " + num);
    }
    public static void main(String args[])
        {
            LocalVariableTest test = new LocalVariableTest();
            test.show();
    }
}
```

Output:

```
The number is : 100
```

Instance Variable:

An **instance variable** in Java can be declared **outside a block**, **method** or **constructor** but inside a class. These variables are **created** when the class **object is created** and **destroyed** when the class **object is destroyed**.

Example:

```
public class InstanceVariableTest
{
    int num; // instance variable
    InstanceVariableTest(int n)
        {
            num = n;
    }
    public void show()
        {
            System.out.println("The number is: " + num);
    }
    public static void main(String args[])
        {
            InstanceVariableTest test = new InstanceVariableTest(75);
            test.show();
    }
}
```

Output:

```
The number is : 75
```

Static/Class Variable:

A **static/class variable** can be defined using the **static** keyword. These variables are declared **inside a class** but **outside a method** and **code block**. A class/static variable can be **created** at the **start of the program** and **destroyed** at the **end of the program**.

Example:

```
public class StaticVaribleTest
{
    int num;
    static int count; // static variable
    StaticVaribleTest(int n)
        {
            num = n;
            count ++;
    }
    public void show()
        {
            System.out.println("The number is: " + num);
    }
    public static void main(String args[])
        {
            StaticVaribleTest test1 = new StaticVaribleTest(75);
            test1.show();
            StaticVaribleTest test2 = new StaticVaribleTest(90);
            test2.show();
            System.out.println("The total objects of a class created are: " + count);
    }
}
```

Output:

```
The number is: 75
The number is: 90
The total objects of a class created are: 2
```

12. **What is data encapsulation and what's its significance?**

Answer:

Encapsulation:

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java:

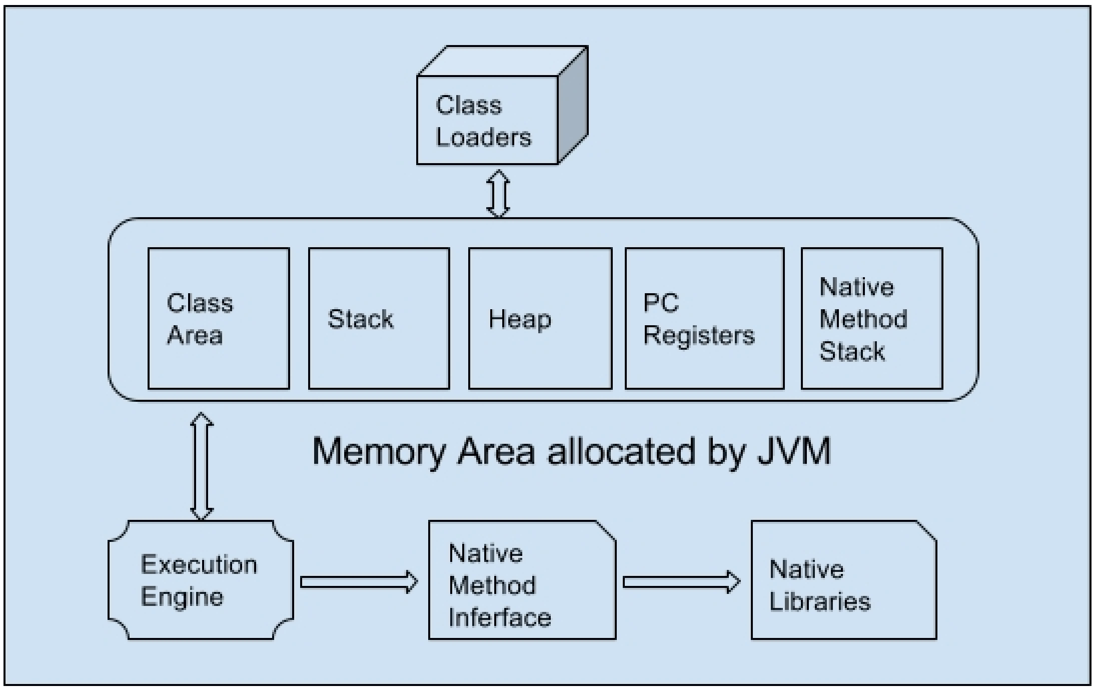
- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

Benefits of Encapsulation:

- The fields of a class can be made read-only or write-only.
- A class can have total control over what is stored in its fields.

13. **Explain the architecture of JVM?**

Answer:



- **Classloader** – Loads the class file into the JVM.
- **Class Area** – Storage areas for a class elements structure like fields, method data, code of method etc.
- **Heap** – Runtime storage allocation for objects.
- **Stack** – Storage for local variables and partial results. A stack contains frames and allocates one for each thread. Once a thread gets completed, this frame also gets destroyed. It also plays roles in method invocation and returns.
- **PC Registers** – Program Counter Registers contains the address of an instruction that JVM is currently executing.
- **Execution Engine** – It has a virtual processor, interpreter to interpret bytecode instructions one by one and a JIT, just in time compiler.
- **Native method stack** – It contains all the native methods used by the application.

14. What are the Data Types in Java?

Answer:

Data Types:

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in the memory.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java –

- Primitive Data Types
- Reference/Object Data Types

Primitive Data Types

There are eight primitive datatypes supported by Java. Primitive datatypes are predefined by the language and named by a keyword. Let us now look into the eight primitive data types in detail.

byte

- Byte data type is an 8-bit signed two's complement integer
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50

short

- Short data type is a 16-bit signed two's complement integer
- Minimum value is -32,768 (-2^15)

- Maximum value is 32,767 (inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer
- Default value is 0.
- Example: short s = 10000, short r = -20000

int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648 (-2^31)
- Maximum value is 2,147,483,647(inclusive) (2^31 -1)
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0
- Example: int a = 100000, int b = -200000

long

- Long data type is a 64-bit signed two's complement integer
- Minimum value is -9,223,372,036,854,775,808(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^63 -1)
- This type is used when a wider range than int is needed
- Default value is 0L
- Example: long a = 100000L, long b = -200000L

float

- Float data type is a single-precision 32-bit IEEE 754 floating point
- Float is mainly used to save memory in large arrays of floating-point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency
- Example: float f1 = 234.5f

double

- double data type is a double-precision 64-bit IEEE 754 floating point
- This data type is generally used as the default data type for decimal values, generally the default choice
- Double data type should never be used for precise values such as currency
- Default value is 0.0d
- Example: double d1 = 123.4

boolean

- boolean data type represents one bit of information
- There are only two possible values: true and false
- This data type is used for simple flags that track true/false conditions
- Default value is false
- Example: boolean one = true

char

- char data type is a single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Char data type is used to store any character
- Example: char letterA = 'A'

Reference Datatypes

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy, etc.
- Class objects and various type of array variables come under reference datatype.
- Default value of any reference variable is null.
- A reference variable can be used to refer any object of the declared type or any compatible type.
- Example: Animal animal = new Animal("giraffe");