1. Describe the general features of java language in detail. Briefly describe the history of java as programming language.

Ans: Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: J2EE for Enterprise Applications, J2ME for Mobile Applications.

The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere.**

Java is −

- **Object Oriented** − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** − Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

- **Multithreaded** − With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted** − Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** − With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** − Java is designed for the distributed environment of the internet.
- **Dynamic** − Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

History of Java

James Gosling initiated Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called 'Oak' after an oak tree that stood outside Gosling's office, also went by the name 'Green' and ended up later being renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.

On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May, 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

2. What do you mean by java applet? Describe applet life cycle.

Ans: An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following –

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet

Four methods in the Applet class gives you the framework on which you build any serious applet −

- **init** − This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start** − This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint** − Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

A "Hello, World" Applet

Following is a simple applet named HelloWorldApplet.java

```
import java.applet.*;

import java.awt.*;


public class HelloWorldApplet extends Applet {

  public void paint (Graphics g) {

    g.drawString ("Hello World", 25, 50);

  }

}
```

These import statements bring the classes into the scope of our applet class −

- java.applet.Applet
- java.awt.Graphics

3. **What do you mean by inheritance? What is the role of keword " extends" in inheritance. Explain with suitable example.**

Ans: Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

extends Keyword

extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax:

```
class Super {
  .....
  .....
}
class Sub extends Super {
  .....
  .....
}
```

Sample Code

Following is an example demonstrating Java inheritance. In this example, you can observe two classes namely Calculation and My_Calculation.

Using extends keyword, the My_Calculation inherits the methods addition() and Subtraction() of Calculation class.

Copy and paste the following program in a file with name My_Calculation.java

Example:

```
class Calculation {
  int z;

  public void addition(int x, int y) {
    z = x + y;
    System.out.println("The sum of the given numbers:"+z);
  }
  public void Subtraction(int x, int y) {
    z = x - y;
    System.out.println("The difference between the given numbers:"+z);
  }
}
public class My_Calculation extends Calculation {
  public void multiplication(int x, int y) {
    z = x * y;
    System.out.println("The product of the given numbers:"+z);
  }
  public static void main(String args[]) {
    int a = 20, b = 10;
```

```
    My_Calculation demo = new My_Calculation();

    demo.addition(a, b);

    demo.Subtraction(a, b);

    demo.multiplication(a, b);

  }

}
```

Compile and execute the above code as shown below.

```
javac My_Calculation.java
java My_Calculation
```

After executing the program, it will produce the following result −

Output

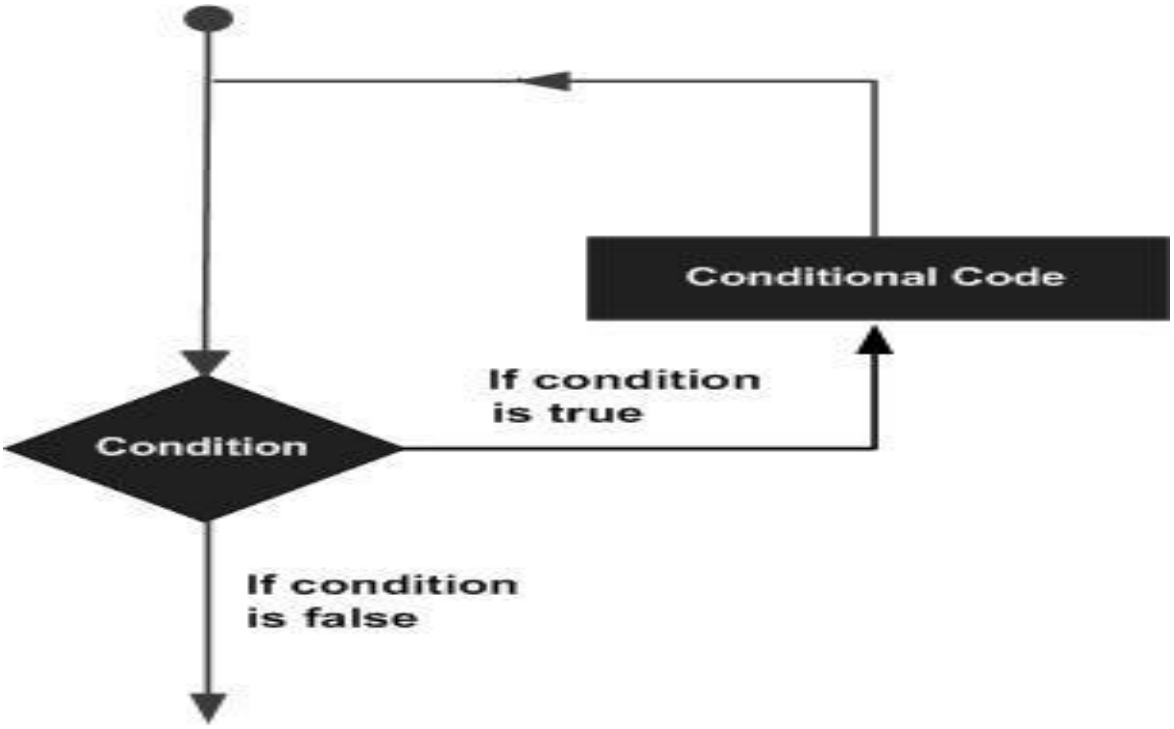
```
The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200
```

4. Describe the role of loops in java. How many loop statements are used in java? Describe with proper syntax. What is the role of loop control statements? How for loop is enhanced in java?

Ans: There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A **loop** statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −



Java programming language provides the following types of loop to handle looping requirements.

Sr.No.	Loop & Description
1	**while loop** Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	**for loop** Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

3	**do...while loop** Like a while statement, except that it tests the condition at the end of the loop body.

Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Java supports the following control statements.

Sr.No.	Control Statement & Description
1	**break statement** Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
2	**continue statement** Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Enhanced for loop in Java

As of Java 5, the enhanced for loop was introduced. This is mainly used to traverse collection of elements including arrays.

Following is the syntax of enhanced for loop −

```
for(declaration : expression) {
   // Statements
}
```

- **Declaration** − The newly declared block variable, is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.
- **Expression** − This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Example:

```
public class Test {
```

```
  public static void main(String args[]) {
    int [] numbers = {10, 20, 30, 40, 50};
    for(int x : numbers ) {
      System.out.print( x );
      System.out.print(",");
    } System.out.print("\n");
    String [] names = {"James", "Larry", "Tom", "Lacy"};
    for( String name : names ) {
      System.out.print( name );
     System.out.print(",");
    }
  }
}
```

Output: **10, 20, 30, 40, 50,**

```
James, Larry, Tom, Lacy,
```

5. Describe various data types in java. Also describe java literals in detail.

Ans: 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");

Java Literals:

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

Literals can be assigned to any primitive type variable. For example −

```
byte a = 68;
char a = 'A'
```

byte, int, long, and short can be expressed in decimal(base 10), hexadecimal(base 16) or octal(base 8) number systems as well.

Prefix 0 is used to indicate octal, and prefix 0x indicates hexadecimal when using these number systems for literals. For example −

```
int decimal = 100;
int octal = 0144;
int hexa =  0x64;
```

String literals in Java are specified like they are in most other languages by enclosing a sequence of characters between a pair of double quotes. Examples of string literals are −

Example

```
"Hello World"

"two\nlines"

"\"This is in quotes\""
```

String and char types of literals can contain any Unicode characters. For example −

```
char a = '\u0001';

String a = "\u0001";
```

Java language supports few special escape sequences for String and char literals as well. They are :

Notation	Character represented
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab

\"	Double quote
\'	Single quote
\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

6. What is the role of overriding in OOP? Explain its scope with an example.

Ans: If a class inherits a method from its superclass, then there is a chance to override the method provided that it is not marked final.

The benefit of overriding is: ability to define a behavior that's specific to the subclass type, which means a subclass can implement a parent class method based on its requirement.

In object-oriented terms, overriding means to override the functionality of an existing method.

Example:

```
class Animal {
  public void move() {
    System.out.println("Animals can move");
  }
}
class Dog extends Animal {
  public void move() {
    System.out.println("Dogs can walk and run");
  }
}
public class TestDog {
```

```
  public static void main(String args[]) {
    Animal a = new Animal();   // Animal reference and object
    Animal b = new Dog();   // Animal reference but Dog object
    a.move();   // runs the method in Animal class
   b.move();   // runs the method in Dog class
  }
}
```

Output of above program is:

```
Animals can move
Dogs can walk and run
```

7. Describe basic Rules for Method-Overriding. Write a program using the super Keyword in Method-Overriding.

Ans: Rules for Method Overriding:

- The argument list should be exactly the same as that of the overridden method.
- The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
- The access level cannot be more restrictive than the overridden method's access level. For example: If the superclass method is declared public then the overridding method in the sub class cannot be either private or protected.
- Instance methods can be overridden only if they are inherited by the subclass.
- A method declared final cannot be overridden.
- A method declared static cannot be overridden but can be re-declared.
- If a method cannot be inherited, then it cannot be overridden.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- A subclass in a different package can only override the non-final methods declared public or protected.

- An overriding method can throw any uncheck exceptions, regardless of whether the overridden method throws exceptions or not. However, the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
- Constructors cannot be overridden.

Using the super Keyword:

When invoking a superclass version of an overridden method the **super**keyword is used.

Example:

```
class Animal {
  public void move() {
    System.out.println("Animals can move");
  }
}
class Dog extends Animal {
  public void move() {
    super.move();   // invokes the super class method
    System.out.println("Dogs can walk and run");
  }
}
public class TestDog {
  public static void main(String args[]) {
    Animal b = new Dog();   // Animal reference but Dog object
    b.move();   // runs the method in Dog class
  }
}
```

This will produce the following result −

Output

```
Animals can move
Dogs can walk and run
```

8. Write the short notes on following: a) java interfaces b) java Exceptions c) Applets

Ans: **a) java interfaces**

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways −

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including −

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.

- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

b) java Exceptions :

An exception (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Based on these, we have three categories of Exceptions. You need to understand them to know how exception handling works in Java.

- **Checked exceptions** − A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.

For example, if you use **FileReader** class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a *FileNotFoundException* occurs, and the compiler prompts the programmer to handle the exception.

Example:

```
import java.io.File;
import java.io.FileReader;
```

```
public class FilenotFound_Demo {

   public static void main(String args[]) {

      File file = new File("E://file.txt");

      FileReader fr = new FileReader(file);

   }

}
```

If you try to compile the above program, you will get the following exceptions.

Output

```
C:\>javac FilenotFound_Demo.java
FilenotFound_Demo.java:8: error: unreported exception FileNotFoundException; must be caught or
declared to be thrown
      FileReader fr = new FileReader(file);
                      ^
1 error
```

Note − Since the methods **read()** and **close()** of FileReader class throws IOException, you can observe that the compiler notifies to handle IOException, along with FileNotFoundException.

- **Unchecked exceptions** − An unchecked exception is an exception that occurs at the time of execution. These are also called as **Runtime Exceptions**. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an *ArrayIndexOutOfBoundsExceptionexception* occurs.

Example

```
public class Unchecked_Demo {

   public static void main(String args[]) {

      int num[] = {1, 2, 3, 4};

      System.out.println(num[5]);

   }

}
```

c) Applets: An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following −

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

9. What is difference between byte streams and character streams? Demonstrate the use of console class to get inputs and show results.

Answer: **Character Stream Vs Byte Stream in Java**

A **stream** is a way of sequentially accessing a file. In Streams you can process the data one at a time as bulk operations are unavailable with them. But, streams supports a huge range of source and destinations including disk file, arrays, other devices, other programs etc. In Java, a **byte** is not the same thing as a **char** . Therefore a byte stream is different from a character stream. So, Java defines two types of streams: **Byte Streams** and **Character Streams** .

Byte Streams

A **byte** stream access the file byte by byte. Java programs use byte streams to perform input and output of **8-bit** bytes. It is suitable for any kind of file, however not quite appropriate for text files. For example, if the file is using a **unicode encoding** and a character is represented with two bytes, the byte stream will treat these separately and you will need to do the conversion yourself. Byte oriented streams do not use any **encoding scheme** while Character oriented streams use character encoding scheme(UNICODE). All byte stream classes are descended from **InputStream** and **OutputStream** .

Character Streams

A **character stream** will read a file character by character. Character Stream is a higher level concept than **Byte Stream** . A Character Stream is, effectively, a Byte Stream that has been wrapped with logic that allows it to output characters from a specific **encoding** . That means, a character stream needs to be given the file's encoding in order to work properly. Character stream can support all types of character sets ASCII, Unicode, UTF-8, UTF-16 etc. All character stream classes are descended from **Reader** and **Writer** .

Using Console Class to get inputs and show results

It has been becoming a preferred way for reading user's input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).

Advantages:

- Reading password without echoing the entered characters.
- Reading methods are synchronized.
- Format string syntax can be used.

Drawback:

- Does not work in non-interactive environment (such as in an IDE).

```
// Java program to demonstrate working of System.console()
// Note that this program does not work on IDEs as
// System.console() may require console
public class Sample
{
   public static void main(String[] args)
   {
      // Using Console to input data from user
      String name = System.console().readLine();
    System.out.println(name);
   }
}
```

10. List the Programming paradigms. For any three state which programming languages are based on them and how?

Ans: **There are several kinds of major programming paradigms:**

1. Imperative 2. Logical 3. Object-Oriented 4. Functional

Imperative: The *imperative* programming paradigm assumes that the computer can maintain through environments of variables any changes in a computation process. Computations are performed through a guided sequence of steps, in which these variables are referred to or changed. The order of the steps is crucial, because a given step will have different consequences depending on the current values of variables when the step is executed**.**

***ImperativeLanguages*:**
Popular programming languages are imperative more often than they are any other paradigm studies in this course. There are two reasons for such popularity:

1. the imperative paradigm most closely resembles the actual machine itself, so the programmer is much closer to the machine;
2. because of such closeness, the imperative paradigm was the only one efficient enough for widespread use until recently

Logical: The *Logical Paradigm* takes a declarative approach to problem-solving. Various logical assertions about a situation are made, establishing all known facts. Then queries are made. The role of the computer becomes maintaining data and logical deduction.

- ***Logical Paradigm Programming:***
  A logical program is divided into three sections:
  1. a series of definitions/declarations that define the problem domain
  2. statements of relevant facts
  3. statement of goals in the form of a query

Object-Oriented: *OOP* is a paradigm in which real-world objects are each viewed as seperate entities having their own state which is modified only by built in procedures, called methods.

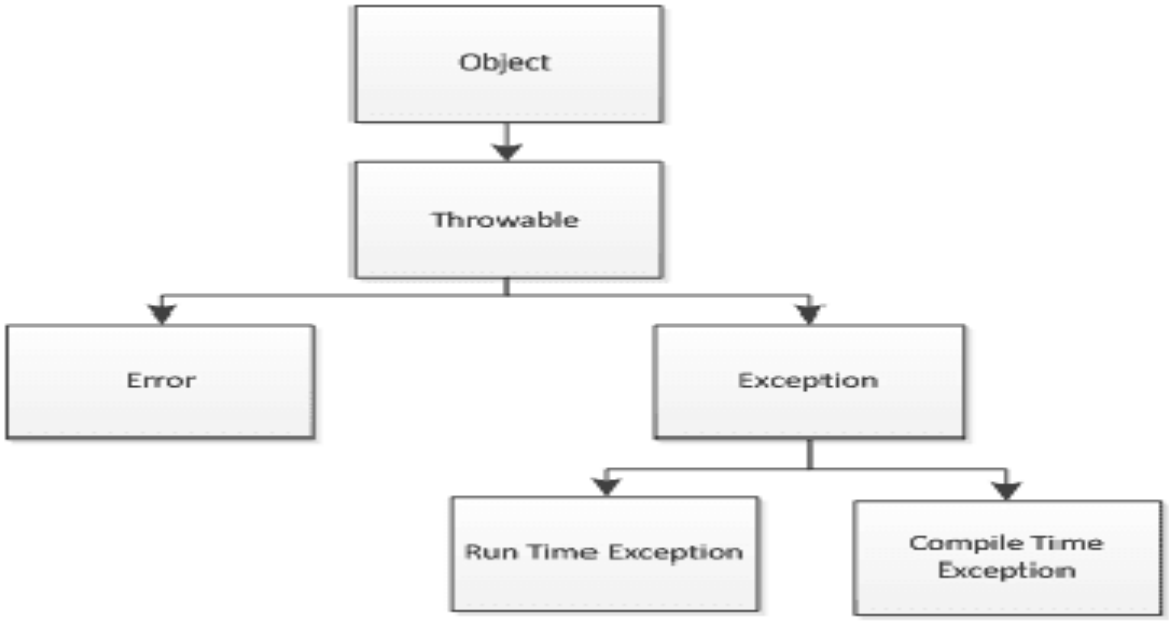
Because objects operate independently, they are encapsulated into modules which contain both local environments and methods. Communication with an object is done by message passing.

Objects are organized into classes, from which they inherit methods and equivalent variables. The object-oriented paradigm provides key benefits of reusable code and code extensibility.

11. Define the term exception. State the advantage of exception handling. What are types of exceptions ?

Ans: **Definition**: An **exception** is an event that occurs during the execution of a program that disrupts the normal flow of instructions. ... An**exception** handler is considered appropriate if the **type** of the **exception** thrown is the same as the **type** of **exception** handled by the handler.

Java is an object oriented programming language. The exception is object created at the time of exceptional/error condition which will be thrown from the program and halt normal execution of the program. Java exceptions object hierarchy is as below:



Advantage 1: Separating Error Handling Code from "Regular" Code

Advantage 2: Propagating Errors Up the Call Stack

Advantage 3: Grouping Error Types and Error Differentiation

Types of exceptions: Java's exceptions can be categorized into two types:

- Checked exceptions
- Unchecked exceptions

Generally, checked exceptions are subject to the catch or specify a requirement, which means they require catching or declaration. This requirement is optional for unchecked exceptions. Code that uses a checked exception will not compile if the catch or specify rule is not followed.

Unchecked exceptions come in two types:

- Errors
- Runtime exceptions

12. State the use of the following methods for programming applet. Give example of using each of these, init( ), start( ), paint( ), stop( ), destroy( ), update( ).

Ans: init() method in java applet : When an applet is loaded by the **Java** plugin of a browser or by an applet viewer, it will first call the Applet.**init method**. Any initializations that are required to use the applet should be executed here. The **init method** is the first **method** called in an Applet or JApplet.
Init method is a predefine **method** to **initialize** an object after its creation. **Init Method** is a life cycle **method** for Servlets for java. It is invoked only once. It is used for **initialization** of servlets.

start() in java: The **java**.lang.Thread.**start()** method causes this thread to begin execution, the**Java** Virtual Machine calls the run method of this thread.The result is that two threads are running concurrently: the current thread (which returns from the call to the **start** method) and the other thread (which executes its run method).

paint() in java: If you have done anything to change the look of the component, but not its size (like changing color, animating, etc.) then call this method. The **paint()** method supports**painting** via a Graphics object. The repaint**()** method is used to cause **paint()** to be invoked by the AWT **painting** thread.

paint() in java: In today's **Java** version, You can **stop** a thread by using a boolean volatile variable. If you remember, threads in **Java** start execution from run**()** method and**stop**, when it comes out of run**()** method, either normally or due to any exception. You can leverage this property to **stop** the thread.

destroy() method: It is called by the browser just before an applet is terminated. Your applet will override this method if it needs to perform any cleanup prior to its destruction. The destroy( ) method is called by the appletviewer or the web browser just before the execution of an applet is terminated. The **destroy()** method is called exactly once in an **applet's** life, just before the browser unloads the **applet**.

update( ) method: To eliminate flashing, you must override the**update() method**. The reason lies in the way the AWT requests that a Component (such as an Applet, Canvas, or Frame) repaint itself. The AWT requests a repaint by calling the Component's **update() method**. The **update() method** is defined by the AWT and is called when your applet has requested that a portion of its window be redrawn.

13. State the use of the following constructs in Java with example :
(1) final method declaration in super class while inheritance
(2) abstract class declaration
(3) method overriding

Final method declaration in super class while inheritance: **Final** is a keyword in java used for restricting some functionality. We can declare variables, methods and classes with final keyword super and final keywords are two popular and useful keywords in Java. They also play a significant role in dealing with Java programs and their classes. In this chapter, you will learn about how to use super and final within a Java program.

I have a big confusion in the usage of "final" keyword between classes and methods i.e.why final methods only support inheritance but not final classes

```
final class A{
   void print(){System.out.println("hello world");}
}

class Final extends A{
   public static void main(String[] args){
      System.out.println("hello world");
   }
}
```

ERROR:

cannot inherit from Final A class Final extennds A{ FINAL METHOD IS.

Abstract class declaration in java: A **class** which contains the **abstract** keyword in its **declaration** is known as **abstract class**. But, if a **class** has at least one **abstract** method, then the **class** must be declared **abstract**. If a **class** is declared **abstract**, it cannot be instantiated. Following is an example of the abstract method declaration:

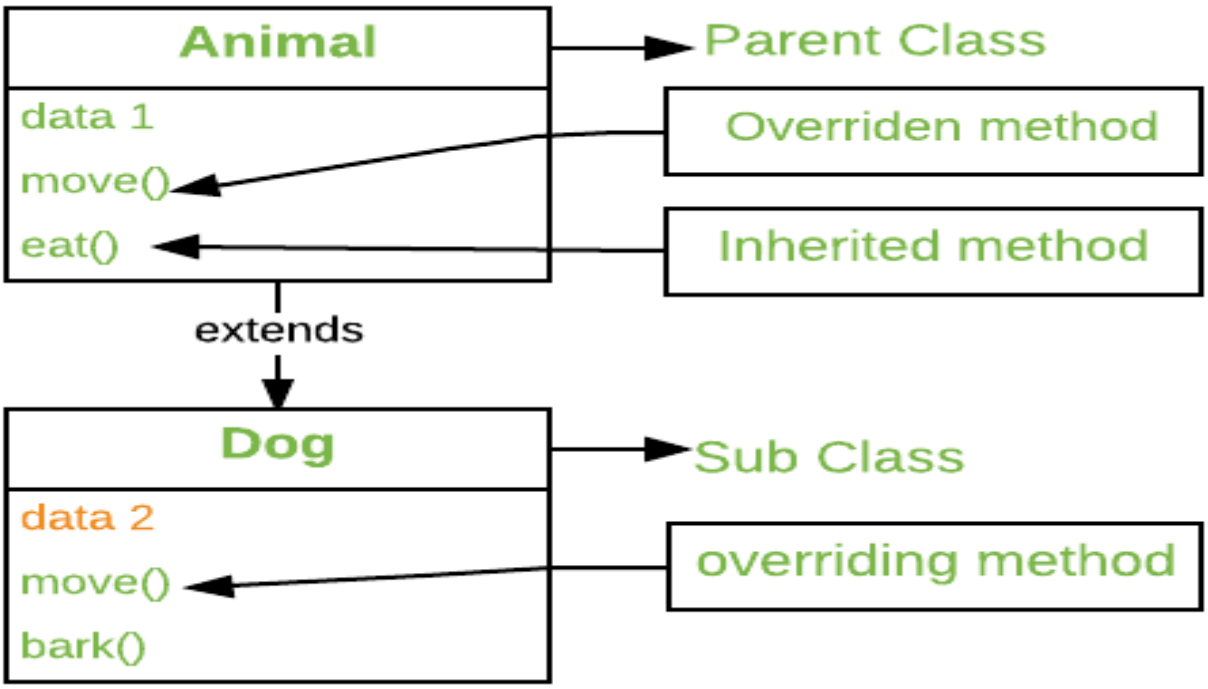
```
public abstract class Employee {

   private String name;

   private String address;

   private int number;

    public abstract double computePay();

   // Remainder of class definition

}
```

Method Overriding in Java: In any object-oriented **programming** language, **Overriding** is a feature that allows a subclass or child class to provide a specific implementation of a **method** that is already provided by one of its super-classes or parent classes.

Final methods can not be overridden : If we don't want a method to be overridden, we declare it as final.

```
class Parent
{
   // Can't be overridden
   final void show() {  }
}
```

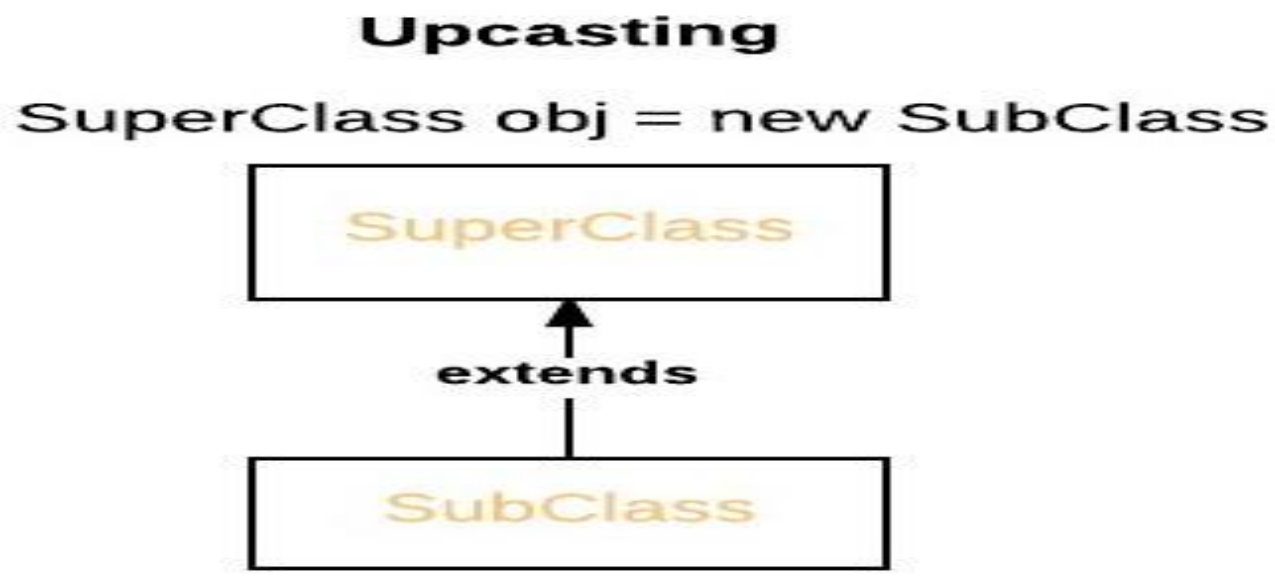
```
class Child extends Parent
{
   // This would produce error
   void show() {  }
}
```



14. Explain the concept of dynamic dispatch while overriding method in inheritance. Give example and advantages of doing so.

Ans: Dynamic Method Dispatch or Runtime Polymorphism in Java: Method overriding is one of the ways in which Java supports Runtime Polymorphism. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.
- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
- A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.



If a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

Advantages of Dynamic Method Dispatch:

1. Dynamic method dispatch allow Java to support overriding of methods which is central for run-time polymorphism.
2. It allows a class to specify methods that will be common to all of its derivatives, while allowing subclasses to define the specific implementation of some or all of those methods.
3. It also allow subclasses to add its specific methods subclasses to define the specific implementation of some.

15. What are challenges for Programming in Large ? How these are addressed by programming languages ?

Ans: **1. Larger projects need more control**

For the most part, bigger projects are more complex, uncertain and risky, either because the technical domain is more obscure, the impact of the project is greater, the scale of the team is larger or because there are more users and stakeholders.

In order to manage this added complexity there is a need for the project manager to utilize a mature project management method so that they can exercise a greater level of control. Let's face it. Project management is a set of methods and approaches that allows us to control a risky and uncertain initiative.

2. Find the right experts

Let's first look at the implications of running a project that is technically more challenging or that has a more complex domain.

This could be a project where you are using new technology, new materials, new processes, new designs or new techniques. This is clearly a step up in complexity from your smaller or medium-sized projects where the project manager sometimes double-up as subject matter expert.

3. More testing is required

A more technically complex domain also means that more effort is required to QA and test the product or service that you are going to provide.
Make sure everyone on the team understands how the product will be quality assured and what is required of them. You need to agree whose role it is to create the test plans and who will be carrying out the actual tests. Wherever possible, involve the end users in the testing in addition to the project team.
.

4. You have a team that needs your leadership

Another significant change when running a bigger project is that you now have a team.
This means however that there is a need for you to step up and be a real leader – not just a taskmaster. A team needs to be lead by an inspiring PM who makes them feel that they belong in the team and that their contributions matter.
Set time aside for one-2-one conversations, team building activities and make sure that you truly engage people in the project. If you have never worked in a bigger team before or been responsible for guiding others, seek advice from a senior manager within your organization or find a mentor who you can meet with on a regular basis. There are also lots of podcasts and books on the topic of building great teams.

5. Analyze and manage the stakeholders

On bigger projects you are also likely to have more stakeholders to interface with – that's people who have an interest in the project or who are affected by it.
To get your head around all the stakeholders you need a formal approach for understanding who they are, what their needs are and how to best involve them and communicate with them. Some people would be happy to receive a weekly status report, others would prefer a regular meet-up and others again demand that you call them whenever something significant happens.
Make an effort to tailor your communication to each of the key stakeholders.

6. Put in place formal governance

On a bigger project you also need a formal governance structure and escalation path, as it may no longer be appropriate to just walk into your boss's office to get guidance. A governance structure is all about agreeing who should attend the weekly working group meeting and who sits on the steering committee. Whereas the working group will meet weekly to work through the detail of the project, the steering committee is a monthly forum for the project's senior decision-makers.

7. Control scope

As there are more moving parts on a larger project – more dependencies, more stakeholders and more uncertainty – it also means that scope is more likely to change as you progress through the project.
Changes to scope aren't necessarily a problem as long as you are on top of them and ensure that the changes make sense and don't erode the business case.

8. Stay on top of the risks

We have already mentioned that the level of risk is likely to be much higher on a larger project, and that for this reason you need a more formal and structured project management approach. One of the elements that form part of such a structured approach is a formal risk management process that will help you stay on top of it all and do something about the many things that could go wrong.

16. Write a program in Java to perform the addition of two matrices (multidimensional arrays).

```
public class AddMatrices {
  public static void main(String[] args) {
      int rows = 2, columns = 3;
      int[][] firstMatrix = { {2, 3, 4}, {5, 2, 3} };
      int[][] secondMatrix = { {-4, 5, 3}, {5, 6, 3} };
      // Adding Two matrices
      int[][] sum = new int[rows][columns];
      for(int i = 0; i < rows; i++) {
         for (int j = 0; j < columns; j++) {
            sum[i][j] = firstMatrix[i][j] + secondMatrix[i][j];
         }
      }
      // Displaying the result
      System.out.println("Sum of two matrices is: ");
      for(int[] row : sum) {
         for (int column : row) {

System.out.print(column + "    ");
         }
         System.out.println();
      }
   }
}
```

When you run the program, the output will be:

```
Sum of two matrices is:

-2   8   7
```

```
10        8   6
```

17. What are generic data structures and generic algorithms ? How C++ implements this generic programming constructs ? Give example of each.

Ans: **Generic data structures and algorithms:**

The C preprocessor is a very powerful tool. One handy way to use it can be generating generic data structures and algorithms. Here you can find some conventions that are used in all such generic structures in libUCW, and also hints for use of these structures.

1) **General Idea:** The idea is simple. If you have some code, you can customize it a little by preprocessor macros. You can change constants, data types it operates on, whole expressions, or you can compile parts of the code conditionally. You can generate new function names using macros.

2) **How to use them:** The use is best explained with an example, so we will suppose there is a header file array.h, which contains a generic array data type and an indexing function, which returns a pointer to n'th element. To get an array of integers, we need to provide macro for used data type and macro that will provide prefixes for identifier names. Then we include the file. Then we could get another array with unsigned integers, so we will do the same:

```
#define ARRAY_TYPE int
#define ARRAY_PREFIX(name) intarray_##name
#include <array.h>
#define ARRAY_TYPE uint
#define ARRAY_PREFIX(name) uintarray_##name
#include <array.h>
```

3) **How it is implemented:**

For those who want to write their own or are just interested, how it works, here is the array.h header and some description to it.

```
#define ARRAY_A_TYPE ARRAY_PREFIX(t)
typedef ARRAY_TYPE *ARRAY_A_TYPE

static ARRAY_TYPE *ARRAY_PREFIX(index)(ARRAY_A_TYPE array, uint index)
{
  return array + index;
}

#undef ARRAY_A_TYPE
#undef ARRAY_TYPE
#undef ARRAY_PREFIX
```

Generic programming in C++:

A basic example of generic programming are templates of containers: In a statically typed language like C++ you would have to declare separate containers that hold integers, floats, and other types or deal with pointers to **void** and therefore losing all type information. Templates which are the C++ way of generic programming leverage this constraint by letting you define classes where one or more parameters are unspecified at the time you define the class. When you instance the template later you tell the compiler which type it should use to create the class out of the template. Example:

```
1.  template<typename T>
2.  class MyContainer
3.  {
4.     // Container that deals with an arbitrary type T
5.  };
6.
7.  void main()
8.  {
9.     // Make MyContainer take just ints.
10.    MyContainer<int> intContainer;
11. }
```

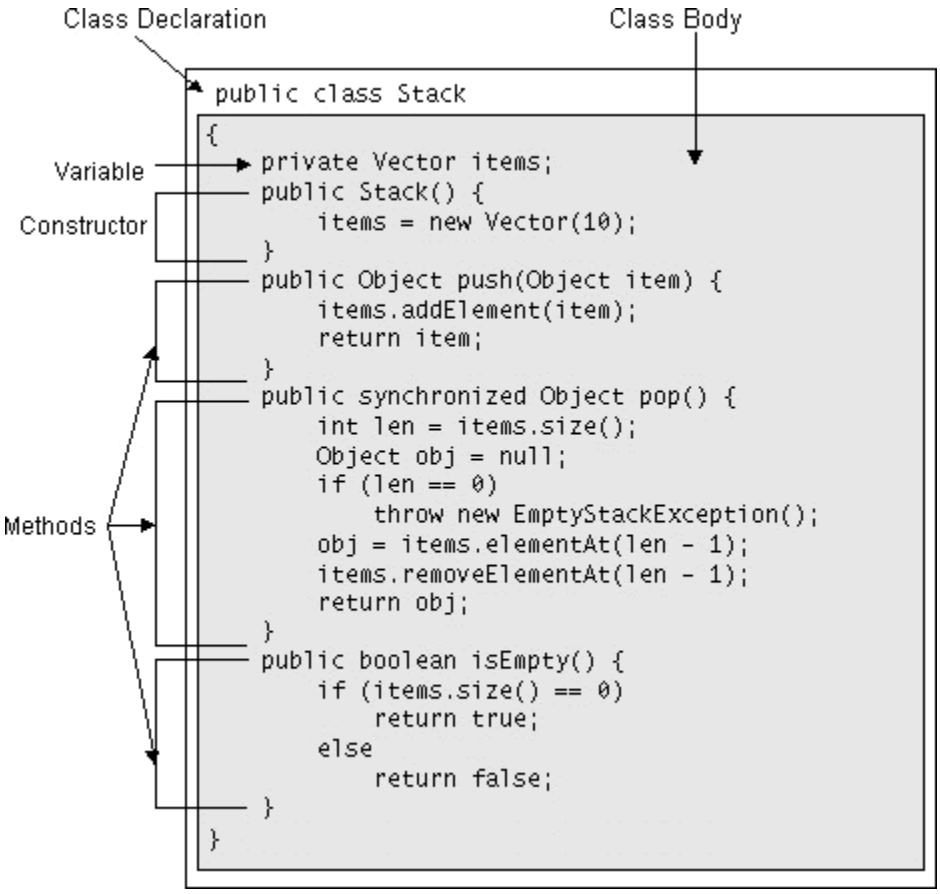
Templates are **generic** because the compiler translates the template into actual code. Note that in the case you don't instantiate your template no code will be generated for it *at all*. On the other hand, if you declare a MyContainer<int>, a MyContainer<float>, and a MyContainer<String> the compiler will create **three** versions of your code each of them having a different type. There will be some optimizations involved but basically your template code will be instantiated to three new types.

Lets see a function to add two int (Easiest Example):

```
1.  int add(int a,int b){
2.          return a+b;
3.  }
```

18. State *two* major differences in class and an interface. "Interface gives multiple inheritance facility just as in C++" justify.

Ans: Classes in Java are almost the same as C++ classes, in that they define an abstract datatype, with its particular fields and methods. Each object is an instance of a class, and follows the class prototype that defines the variables and methods common to all objects of a certain kind. Each instance of a class must be instantiated, after it is declared, unlike in C++. This is usually done with the keyword **"new"**.



An interface, or protocol as it is sometimes called, is a device that is used to allow unrelated objects to interact with one another, by implementing an agreed upon system of behavior. When a class implements an interface, the class agrees to implement all of the methods defined in the interface. Interfaces are useful since they capture similiarity between unrelated objects without forcing a class relationship. Furthermore, interfaces may consist of either abstract methods or entire abstract classes. One class uses an interface by using the **"implements"** keyword, and example might look : class Window implements ActionListener {}

The most difficult to avoid complication that arises when using multiple inheritance is that sometimes the programmers interested in using this technique to extend the existing code are forced to learn some of the implementation's details. The second trivial problem that might appear when using this technique is the creation of ***ambiguities***:

class A { virtual void f(); };
class B { virtual void f(); };

class C : public A ,public B { void f(); };

19. What is interpretation and translation process ? With neat diagram state the purpose of each activity in language processing with interpretation and translation.

Ans: **Translators:**

Translators generally receive their material in file format (e.g. a Word document), type in the translated text and deliver a file back. Translators use specific glossaries and other relevant reference materials in their work. They can also use a translation tool, which may be a suitable way of ensuring that the right terminology is used or that repeated phrases are translated consistently throughout the text.

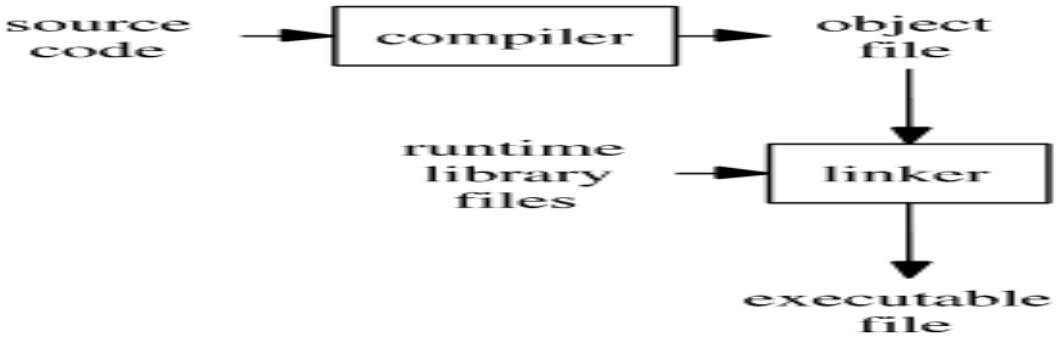
It is important that translators are good writers and are able to express themselves well when writing in the target language. Although many translators work with more than one language combination, they therefore only work in one language direction. In other words, they translate from other languages *into* their own native language.

The Translation Process: A computer system can only understand machine code. A program written for example in a high level Language such as Java cannot be run directly. To execute a computer program written in any programming Language, it must first be translated. The source code which is written by the programmer needs to be translated. When translated, the source Code becomes object code which is understandable by the computer system. This process can be seen in the following diagram;

Interpreters:

Interpreters work directly with their customers. An interpreter can attend meetings or conferences in person, or can participate by telephone or video. Examples of common interpreting situations include conferences, where a team of interpreters works from an interpreter booth to provide interpretation direct to the participants' wireless headsets, during doctor's visits where the doctor and the patient do not speak the same language, or during study visits where an interpreter uses whispered interpreting to convey what is being said to one or more people.

The following illustrates the programming process for a compiled programming language.



The same goal:

Neither interpreting nor translation involves simply replacing a word in one language with the corresponding word in another language. Obviously, a good translation or interpretation always reflects intentions, word choices, style and nuances. Both interpreters and translators must therefore be well versed in all the variations of the source language in order to reproduce the content in the best way. This is one thing they have in common. However, the medium is different. Since the two roles require different skills, most professional interpreters and translators choose to deal exclusively with one or the other, although there are some exceptions.

20. What are abstract data types ? How C++ implements abstract data types ? Give example.

Ans: Abstract Data Types:

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation independent view. The process of providing only the essentials and hiding the details is known as abstraction.

- An ADT is a collection of data and associated operations for manipulating that data
- ADTs support abstraction, encapsulation, and information hiding
- They provide equal attention to data and operations
- Common examples of ADTs: – Built-in types: boolean, integer, real, array – User-defined types: stack, queue, tree, list.

Example:

```
class AbstractClass {
public:
```

```
  virtual void AbstractMemberFunction() = 0; // Pure virtual function makes

// this class Abstract class.
  virtual void NonAbstractMemberFunction1(); // Virtual function.

  void NonAbstractMemberFunction2();
};
```

In general an abstract class is used to define an implementation and is intended to be inherited from by concrete classes.

21. **Write the short notes on: 1)** Runtime Polymorphism in Java

2) A **java package** 3) **Java Access Specifiers**

Ans: Runtime Polymorphism in Java

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Let's first understand the upcasting before Runtime Polymorphism.

Upcasting

When reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:


```
1.  class A{}
2.  class B extends A{}
```

```
1.  A a=new B();//upcasting
```

Example of Java Runtime Polymorphism

In this example, we are creating two classes Bike and Splendar. Splendar class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, subclass method is invoked at runtime.

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

```
1.  class Bike{
2.    void run(){System.out.println("running");}
3.  }
4.  class Splender extends Bike{
5.    void run(){System.out.println("running safely with 60km");}
6.
7.    public static void main(String args[]){
8.      Bike b = new Splender();//upcasting
9.      b.run();
10.   }
11. }
```

A **java package** :

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

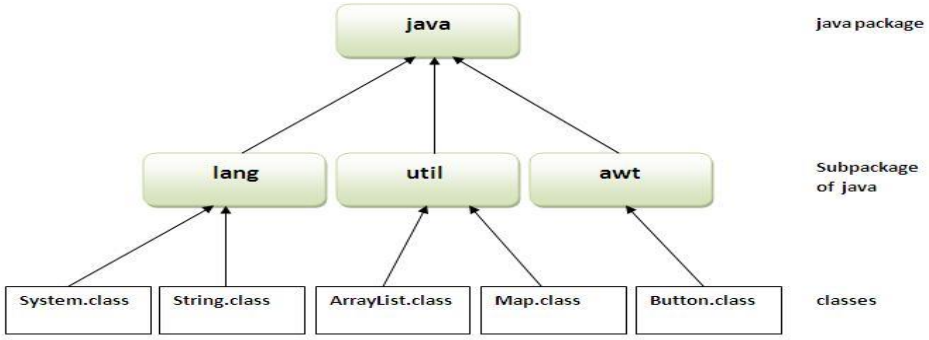
Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



Java Access Specifiers:
Java Access Specifiers (also known as Visibility Specifiers) regulate access to classes, fields and methods in Java.These Specifiers determine whether a field or method in a class, can be used or invoked by another method in another class or sub-class. Access Specifiers can be used to restrict access. Access Specifiers are an integral part of object-oriented programming.

Types Of Access Specifiers :

In java we have four Access Specifiers and they are listed below.

1. public
2. private
3. protected
4. default(no specifier)

We look at these Access Specifiers in given table:

Access Modifiers	Default	private	protected	public
Accessible inside the class	yes	yes	yes	yes
Accessible within the subclass inside the same package	yes	no	yes	yes
Accessible outside the package	no	no	no	yes
Accessible within the subclass outside the package	no	no	yes	yes