CJ sem 4 external exam solution

1) Explain the history of JAVA

i) Java is a multi-platform, object-oriented, and network-centric language that can be used as a platform in itself. It is a fast, secure, reliable programming language.

ii) Java was developed by James Gosling, who is known as the father of Java. James Gosling and his team members started the project in 1991.

iii) Initially developed by James Gosling at Sun Microsystems In 1995, Java 1.0 was the first public execution. It pledged 'Write Once, Run Anywhere' on popular platforms with free runtimes.

iv) Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.

v) Firstly, it was called "Greentalk" by James Gosling, and the file extension was .gt. After that, it was called Oak and was developed as a part of the Green project.

vi)But they had to later rename it as "JAVA" as it was already a trademark by Oak Technologies**.**

vii) The Java language has experienced a few changes since JDK 1.0 just as various augmentations of classes and packages to the standard library.

viii) In Addition to the language changes, considerably more sensational changes have been made to the Java Class Library throughout the years.

ix) After the first release of Java, there have been many additional features added to the language like HotSpot JVM included Java Naming and Directory Interface.

x) Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

2) Explain the features of JAVA

Simple	Java is very easy to learn programming language, as its syntax is very simple and clear in understanding.
Object Oriented	Java is an object-oriented programming language, were everything is an object, which has some data and behaviour.
High Performance	Java is faster as its code is compiled into byte code which is optimized by the Java compiler for Java Virtual Machine (JVM) to execute its applications faster with the help of Just-In-Time (JIT) compiler.
Portable	Java is portable as it provides the Java byte code to execute on any platform without any implementation.
Platform Independent	Java code is compiled into byte code format, which does not bound it to any specific operating system.
Robust	With features like strong memory management, automatic garbage collection and mechanism like exception handling makes Java a robust language.
Secured	Java's secure features allow us to develop virus-free systems, as Java program runs in Java Runtime Environment (JRE) making it is more secure.
Multithreaded	Multithreading feature of Java makes it possible to write program that can perform many tasks simultaneously without occupying memory for each thread, instead it shares a common memory area, resulting in highly interactive and responsive applications.
Distributed	Java is a distributed language as it can create applications that can run over network, with the help of TCP/IP protocols used for communication.

3) Explain JVM and JRE in detail

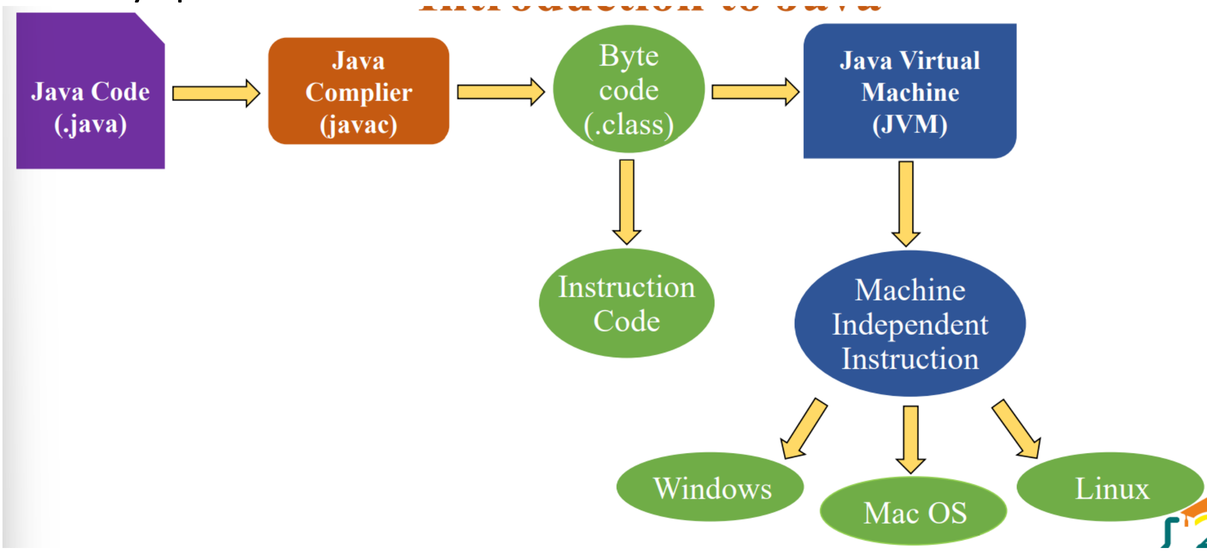
i) **JVM (JAVA virtual machine)**

a) JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

b) It is A specification where working of Java Virtual Machine is specified. Its implementation has been provided by Oracle and other companies.

c) In other programming languages, the compiler produces machine code for a particular system. However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.

d) First, Java code is compiled into bytecode. This bytecode gets interpreted on different machines Between host system and Java source, Bytecode is an intermediary language. JVM in Java is responsible for allocating memory space.

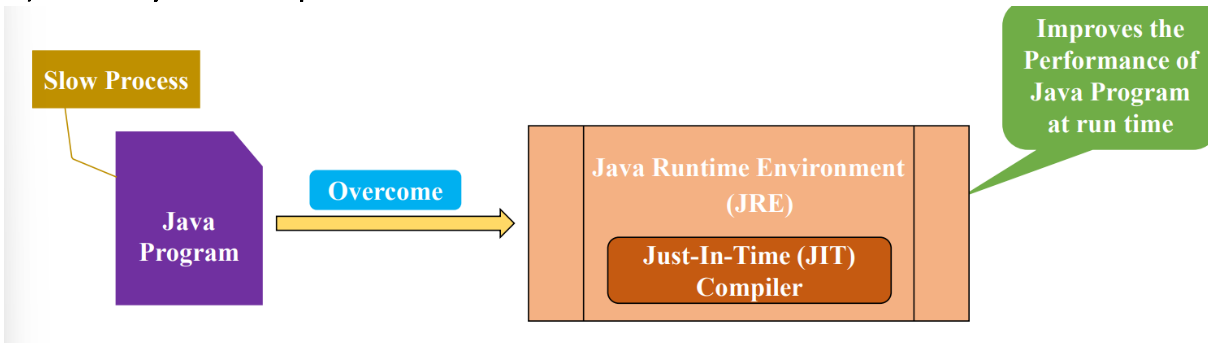


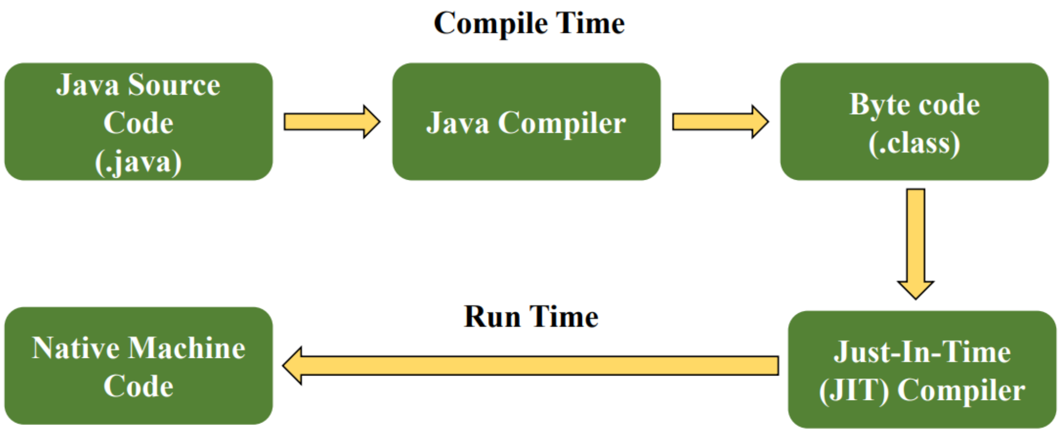
ii) **JRE (JAVA runtime environment)**

a) Java Run-time Environment (JRE) is the part of the Java Development Kit (JDK). It is a freely available software distribution which has Java Class Library, specific tools, and a stand-alone JVM.

b) It is the most common environment available on devices to run java programs. The source Java code gets compiled and converted to Java bytecode. If you wish to run this bytecode on any platform, you require JRE.

c) The JRE loads classes, verify access to memory, and retrieves the system resources. JRE acts as a layer on the top of the operating system.
d) after compiling, the compiler generates a .class file which has the bytecode. The bytecode is platform independent and runs on any device having the JRE. From here, the work of JRE begins.
e) The bytecode process in JRE is as follows:-


Slow Process
Java Program
Overcome
Java Runtime Environment (JRE)
Just-In-Time (JIT) Compiler
Improves the Performance of Java Program at run time



Compilation process of JRE

4) What is OOP Explain any one pillar of OOP

i) Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic.

ii) OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them.

iii) The fundamental idea behind object-oriented languages is to combine into a single unit both *data* and the *functions that operate on that data*. Such a unit is called an *object*.

iv) The first step in OOP is to collect all of the objects a programmer wants to manipulate and identify how they relate to each other.

v) Once an object is known, it is labelled with a class of objects that defines the kind of data it contains and any logic sequences that can manipulate it. In OOP classes are user-defined data types.

vi) One of the major pillar of OOP is:- inheritance

1. Inheritance is a mechanism in Java where a class can inherit properties and behaviors from another class.
2. The class that inherits properties and behaviors is called the subclass, while the class that is being inherited from is called the superclass.
3. Inheritance allows for code reusability, as the subclass can reuse the code from its superclass without having to rewrite it.
4. Inheritance creates an "is-a" relationship between classes, where the subclass is a type of the superclass.
5. Access modifiers such as public, private, and protected can be used to control access to superclass members by subclasses.
6. Constructors and initialization blocks are not inherited from the superclass, but can be called using the super keyword.
7. If a subclass overrides a method from its superclass, the method in the subclass will be called instead of the method in

the superclass when the method is called on an object of the subclass.

8. A class can only inherit from one superclass, but a superclass can have multiple subclasses.
9. If a superclass is modified, the changes will affect all its subclasses.
10. Inheritance can be used to implement polymorphism, where a subclass can be used in place of its superclass.

```
// Superclass
class Animal {
   public void makeSound() {
      System.out.println("The animal makes a sound");
   }
}

// Subclass
class Dog extends Animal {
   public void makeSound() {
      System.out.println("The dog barks");
   }
}

// Main class
class Main {
   public static void main(String[] args) {
      Animal animal = new Animal();
      animal.makeSound(); // Output: The animal makes a sound

      Dog dog = new Dog();
      dog.makeSound(); // Output: The dog barks

      // Polymorphism example
      Animal anotherDog = new Dog();
```

```
        anotherDog.makeSound(); // Output: The dog barks
    }
}
```

5) Explain looping in java with Syntax

i) while loop:

a) The Java while loop is used to iterate a part of the program repeatedly until the specified condition is true. As soon as the condition becomes false, the loop automatically stops.
b) The while loop is considered as a repeating if statement. If the number of iteration is not fixed, it is recommended to use the while loop.
c) Syntax:

```
while (condition)
{
    //code to be executed
}
```

ii) do-while loop:

a) The Java do-while loop is used to iterate a part of the program repeatedly, until the specified condition is true.
b) Java do-while loop is called an exit control loop. The Java do-while loop is executed at least once because condition is checked after loop body.
c) Syntax:

```
do{

//code to be executed / loop body
}while (condition);
```

iii) for loop:

a) The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

b) Syntax:

```
for(initialization; condition; increment/decrement)
{
        //statement or code to be executed
}
```

c) Example –

```
class Main {
   public static void main(String[] args) {
      for (int i = 1; i <= 10; i++) {
         System.out.println(i);
      }
   }
}
```

6) Explain control statements in java

if statement:

a) It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

b) Syntax:

```
if(condition)
{
  // Statements to execute if condition is true
}
```

c) Example of if else statement:

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = scanner.nextInt();

        if (num > 0) {
            System.out.println("The number is positive");
        } else if (num < 0) {
            System.out.println("The number is negative");
        } else {
            System.out.println("The number is zero");
        }
    }
}
```

switch statement:

a) The Java *switch statement* executes one statement from multiple conditions.
b) The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.
c) In other words, the switch statement tests the equality of a variable against multiple values.

Syntax:

```
switch(expression)
{
        case value1:
         //code to be executed;
        break;
```

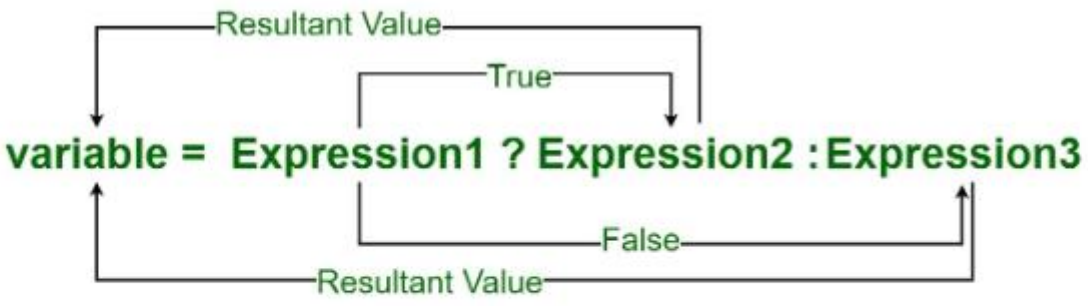
```
            case value2:
            //code to be executed;
            break;

            default:
            code to be executed if all cases are not matched;
}
```

c) **Ternary Operator**

a) Java ternary operator is the only conditional operator that takes three operands.

b) We can use the ternary operator in place of if-else conditions or even switch conditions using nested ternary operators.

c) Although it follows the same algorithm as of if-else statement, the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.



7)Inheritance in JAVA and its types

In Java, inheritance is a mechanism that allows a class to inherit properties and behavior from another class. This is achieved through the use of the **extends** keyword, where a subclass is created that inherits from a superclass. The subclass can then add its own properties and behavior, or override the superclass's behavior with its own implementation.

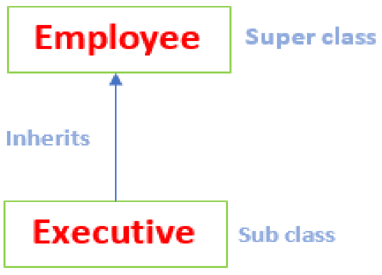
Here are some key points about inheritance in Java:

1. Inheritance allows code reuse and promotes code organization.

2. A subclass can inherit all non-private fields and methods of its superclass.
3. A subclass can override the methods of its superclass to provide its own implementation.
4. A subclass can add its own fields and methods in addition to those inherited from its superclass.
5. A subclass can access the public and protected members of its superclass.
6. A class can only inherit from one direct superclass, but can implement multiple interfaces.
7. The **super** keyword is used to call the superclass's constructor or methods from within the subclass.

Following are its types

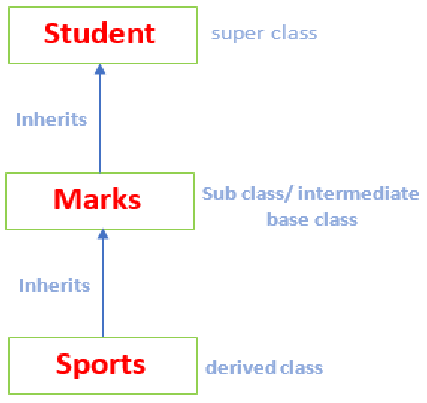
a) Single inheritance
    1. In single inheritance, a sub-class is derived from only one super class.
    2. it inherits the properties and behavior of a single-parent class. Sometimes it is also known as **simple inheritance**.
    3. In the above figu t class and Executive is a chi :lass inherits all the propertie:



Single Inheritance

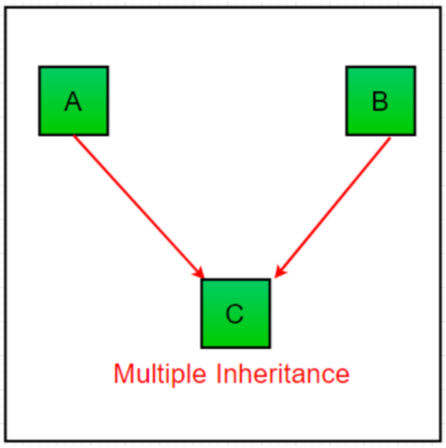
b) Multi-level inheritance
    1. In multi-level inh ved from a class which is als class is called multi-level inheritance.
    2. In simple words, we can say that a class that has more than one parent class is called multi-level inheritance.

3. Hence, there exists a single base class and single derived class but multiple intermediate base classes.



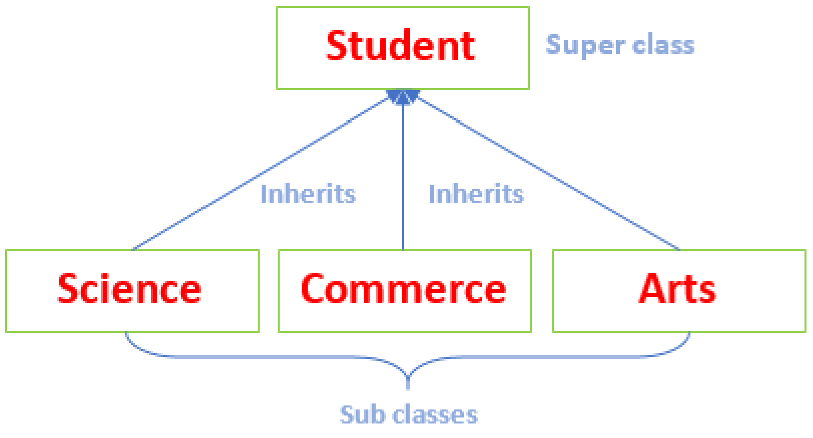
Multi-level Inheritance

c) Multiple inheritance
    1. Multiple inheritances is a type of inheritance where a subclass can inherit features from more than one parent class.
    2. Multiple inheritances should not be confused with multi-level inheritance, in multiple inheritances the newly derived class can have more than one superclass.
    3. And this newly derived class can inherit the features from these superclasses it has inherited from, so there are no restrictions.
    4. In java, multiple inheritances can be achieved through interfaces.



d) Heirarchial inheritance

1. If a number of classes are derived from a single base class, it is called hierarchical inheritance.

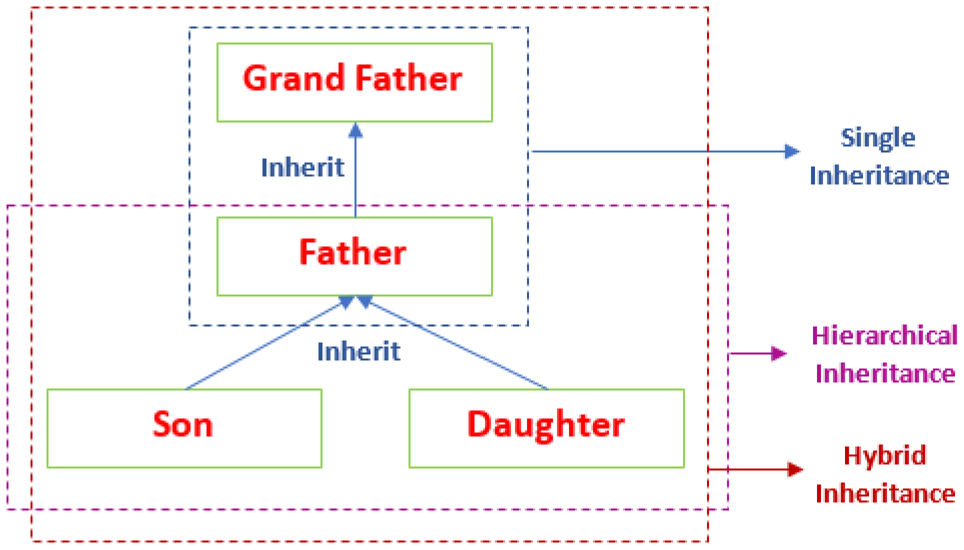


Hierarchical Inheritance

2.
3. In the above figure, the classes Science, Commerce, and Arts inherit a single parent class named Student.

e) Hybrid inheritance

1. Hybrid means consist of more than one. Hybrid inheritance is the combination of two or more types of inheritance.



2.

Hybrid Inheritance

3. In the above figure, GrandFather is a super class. The Father class inherits the properties of the GrandFather class. Since Father and GrandFather represents single inheritance. Further, the Father class is inherited by the Son and Daughter class. Thus, the Father becomes the parent class for Son and Daughter. These classes represent the hierarchical inheritance. Combinedly, it denotes the hybrid inheritance.

8) explain the concept of abstract class

i) Abstraction is a process of hiding the implementation details and showing only functionality to the user. It shows only essential things to the user and hides the internal details.

ii) This data abstraction can be implemented in JAVA using the abstract class. A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods.

iii) It needs to be extended and its method implemented. It cannot be instantiated. An abstract class must be declared with an abstract keyword.

iv) If a class contains at least one abstract method then compulsory should declare a class as abstract.

v) An abstract class can contain constructors in Java. And a constructor of an abstract class is called when an instance of an inherited class is created.

vi) In Java, we can have *an abstract class without any abstract method*. This allows us to *create classes that cannot be instantiated but can only be inherited*.

vii) For any abstract java class we are not allowed to create an object i.e., for abstract class instantiation is not possible.

viii) Implementation of abstract class through java program:-

```
abstract class Bike
```

```
{
      abstract void run();
}
class Honda4 extends Bike
{
     void run()
     {
        System.out.println("running safely");
     }
     public static void main(String args[])
     {
        Bike obj = new Honda4();
        obj.run();
     }
}
Output : running safely
```

ix) Explanation:

- An abstract class named Bike is defined, which has an abstract method named run().
- A concrete class named Honda4 is defined, which extends the Bike class and provides an implementation for the run() method.
- In the main() method, an object of the Honda4 class is created and assigned to a reference variable of the Bike type.
- The run() method of the Honda4 class is called using the reference variable obj.
- Since the run() method of the Honda4 class overrides the run() method of the Bike class, the implementation of the Honda4 class is executed, which prints the message "running safely" to the console.

9) Constructors and its types

i) A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created

ii) In Java, a constructor is a block of codes similar to the method. Every time an object is created using the new() keyword, at least one constructor is called.

iii) At the time of calling the constructor, memory for the object is allocated in the memory.

iv) In Java, constructors can be divided into 3 types:

a) Java No-Arg Constructors

1) If a constructor does not accept any parameters, it is known as a no-argument constructor.
2) If we don’t define a constructor in a class, then the compiler creates a constructor(with no arguments) for the class.
3)

```
class Geek
{
   int num;
   String name;
   Geek()
     {
          System.out.println("Constructor called");
     }
}

class GFG {
   public static void main(String[] args)
   {
     Geek geek1 = new Geek();
     System.out.println(geek1.name);
     System.out.println(geek1.num);
```

```
    }
  }
```

b)Java Parameterized Constructor

1) A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors
2) If we want to initialize fields of the class with our own values, then use a parameterized constructor.
3) class Main

```
{
   String languages;
   Main(String lang)
       {
               languages = lang;
               System.out.println(languages +
       "Programming Language");
       }

       public static void main(String[] args)
       {
       Main obj1 = new Main("Java");
        Main obj2 = new Main("Python");
       Main obj3 = new Main("C");
       }
 }
```

c)Default Constructor

1) If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program.
2) A constructor that has no parameters is known as default the constructor. A default constructor is invisible.
3)

```
class GFG
{
   GFG()
{
System.out.println("Default constructor");
}
   public static void main(String[] args)
   {

      GFG hello = new GFG();
   }
}
```

10) What are arrays ? explain one loop hole in array

- An array is defined as a sequence of objects of the same data type.
- All the elements of an array are either of type int (whole numbers), or all of them are of type char, or all of them are of floating (decimal point) type, etc.
- An array cannot have a mixture of different data types as its elements.
- Also, array elements cannot be functions; however, they may be pointers to functions.
- In computer memory, array elements are stored in a sequence of adjacent memory blocks.
- Since all the elements of an array are of same data type, the memory blocks allocated to elements of an array are also of same size.
- Each element of an array occupies one block of memory.
- The size of memory blocks allocated depends on the data type and it is same as for different data types.

Loopholes in array

- Out-of-bounds exceptions: An array has a fixed size, and it is possible to try to access an element outside of the bounds of the array.
- If you try to access an element at an index that is less than 0 or greater than or equal to the length of the array, you will get an

ArrayIndexOutOfBoundsException

11)Explain thread life cycle with a diagram

i) A Thread is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.

ii) A thread in Java at any point of time exists in any one of the following states.

a) **New Thread:**

1) When a new thread is created, it is in the new state.
2) When a thread lies in the new state, its code is yet to be run and hasn't started to execute.

b) **Runnable State:**

1) A thread that is ready to run is moved to a runnable state.
2) In this state, a thread might actually be running or it might be ready to run at any instant of time.
3) Each and every thread runs for a short while and then pauses and relinquishes the CPU to another thread so that other threads can get a chance to run.

c) **Blocked/Waiting state:**

1) When a thread is temporarily inactive, then it's in one of the following states:

   Blocked

   Waiting

d) **Timed Waiting:**

1) A thread lies in a timed waiting state when it calls a method with a time-out parameter.
2) A thread lies in this state until the timeout is completed or until a notification is received.

e) **Terminated State**

A thread terminates because of either of the following reasons:

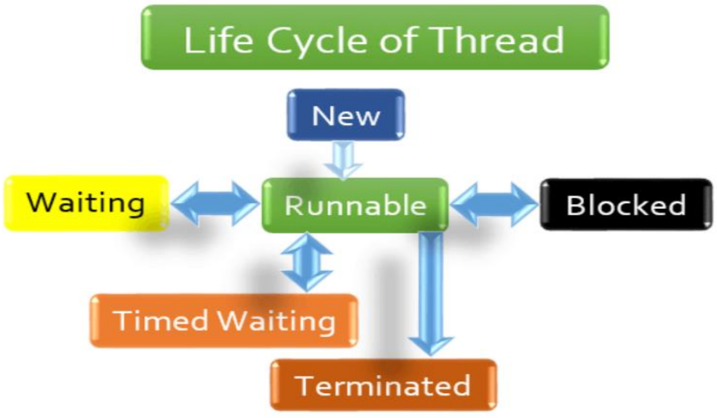
1) Because it exits normally. This happens when the code of the thread has been entirely executed by the program.

2) Because there occurred some unusual erroneous event, like segmentation fault or an unhandled exception.

Example :

```
class MyThread extends Thread
{
    public void run()
    {
            //Executing the job of MyThread class
            for(int i=0; i<10;i++)
            {
                    System.out.println("Child Thread -MyThread");
            }//i
    }//run()
}//class MyThread
public class T1
{
    public static void main(String [] args)
    {
            /*At this point in the program, number of thread =1
ie main() thread */

            MyThread t=new MyThread();

            //main() thread starts the child thread - MyThread
            t.start();

            /*At this point in the program, number of thread =2
ie 1) main() thread  2) MyThread*/
```

```
                //Executing the job of main()
                for(int i=0; i<10;i++)
                {
                        System.out.println("main() Thread - main()");
                }//i
        }//main()
}//class T1
```



Output:

```
Child Thread - MyThread
main() Thread - main()
Child Thread - MyThread
main() Thread - main()
Child Thread - MyThread
main() Thread - main()
...
```

This pattern will continue until both threads complete executing their respective loops. Since both threads are running concurrently, their output messages will be interleaved.

11)Any 5 types of combination of try catch and finally

TYPE - 1

```
try
{
}
catch(X e)
{
}
```

Compiles FINE

--
Type - 6
try
{
}
Compile Time Error : try without catch or finally

Type - 7
catch(X e)
{
}
Compile Time Error : catch without try

Type - 8
finally
{
}
Compile Time Error : finally without try

Type - 9
try
{
}
finally
{
}
Compile FINE

13) delegation event model with diagram.

i) The GUI programming in JAVA is event-driven; whenever a user initiates an activity such as a mouse activity, scrolling, etc., each is known as an event . The Delegation Event model is defined to handle events in GUI programming languages.

ii) It defines a standard and compatible mechanism to generate and process events. In this model, a source generates an event and forwards it to one or more listeners.

iii) The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it.

iv) The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.

v) In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them.

vi) The methods that receive and process events are defined in a set of interfaces found in java.awt.event.

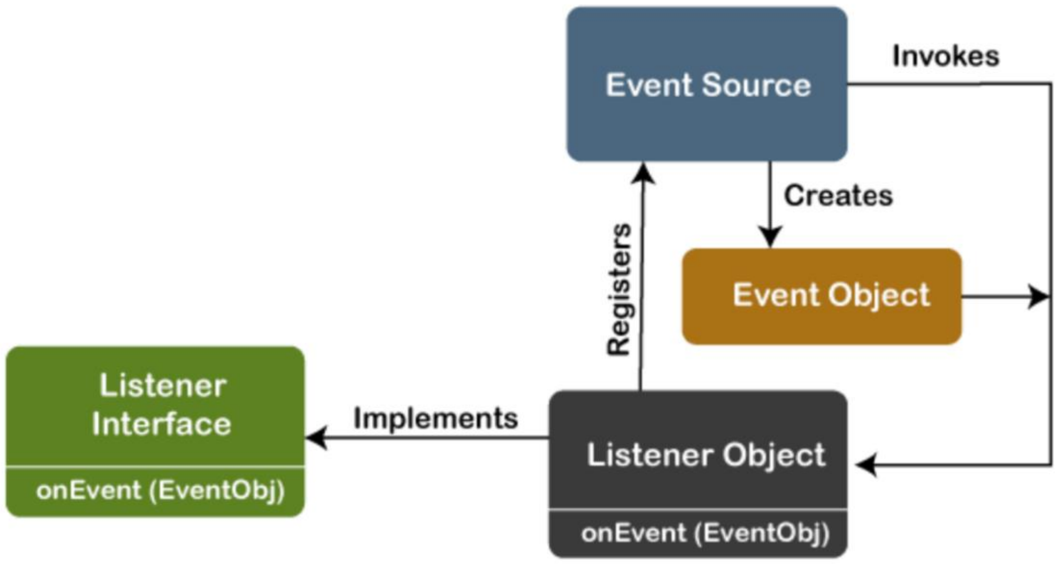
vii) A source is an object that causes and generates an event. There are various sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components, windows, etc., to generate events. Syntax is as follows:-

public void addTypeListener (TypeListener e1)

viii) An event listener is an object that is invoked when an event triggers. The listeners require two things; first, it must be registered with a source. Second, it must implement the methods to receive and process the received notifications.

ix) event listener are of 2 types which are:-

For KeyEvent we use *addKeyListener()* to register.
For ActionEvent we use *addActionListener()* to register.



14) Java AWT frames

i) Java AWT (Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.

ii) Java swing components are lightweight, platform-independent, provide powerful components like tables, scroll panels, buttons, lists, color chooser, etc.

iii) Frames are implemented using the JFrame class in the javax.swing package, which provides methods for creating and manipulating frames.

iv) This is the top-level window, with border and a title bar. JFrame class has various methods which can be used to customize it.

v) Frames can be customized by setting various properties such as title, size, location, background color, and border style.

vi) The java.awt.Frame component is a Windows graphics system which has a title bar and borders, behaving like a normal GUI window.

vii) They can also contain other components such as panels, which can be used to organize the layout of the frame.

viii) In JAVA frames can be created in 2 ways which are:-

a) By creating the object of Frame class (association)

```
import java.awt.*;
class P2
{
	P2()
	{
		//Frame class
		Frame f=new Frame();
		Button b=new Button("Click Me");
		b.setBounds(30,50,80,30);
		f.add(b);
		f.setSize(300,300);
		f.setLayout(null);
		f.setVisible(true);
	}
	public static void main(String args[])
	{
		P2 p=new P2();
	}
}
```

b) By extending Frame class (inheritance)

```
import java.awt.*;
```

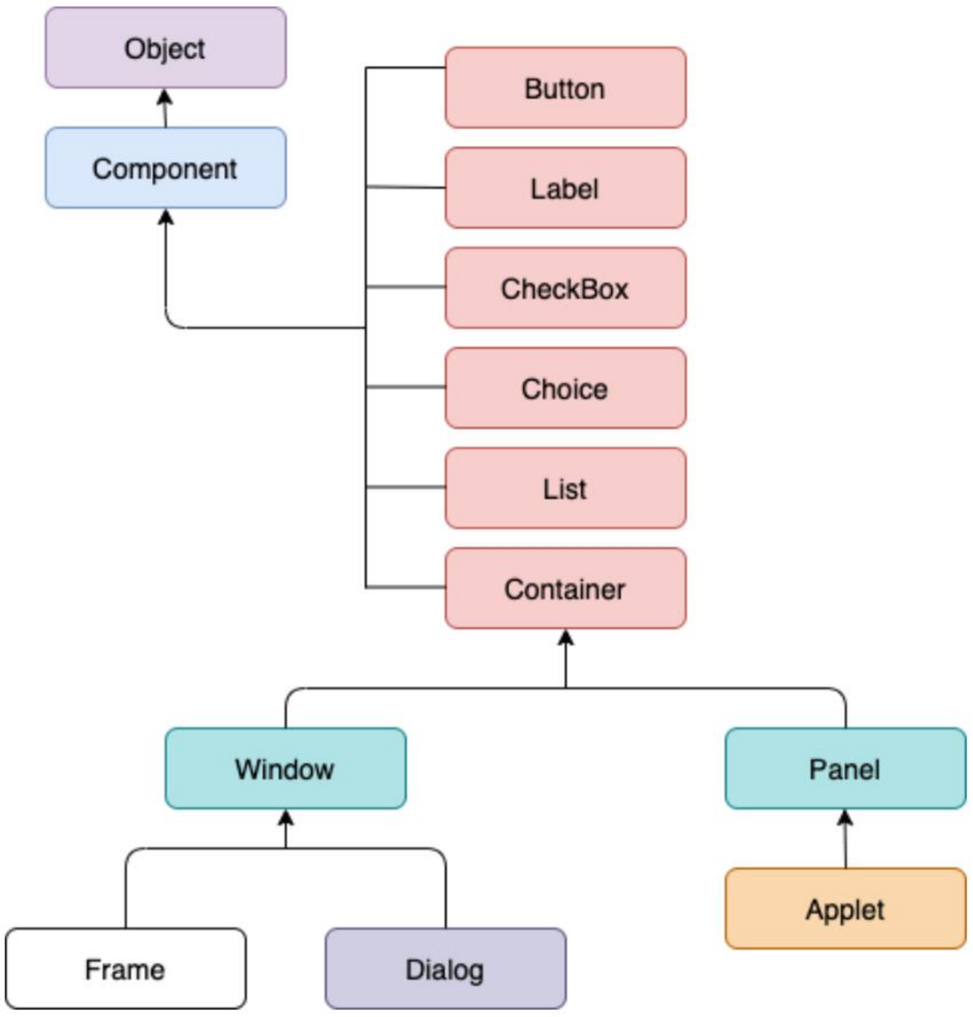
```
class P1 extends Frame
{
	P1()
	{
		Button b=new Button("Sumeet");

		b.setBounds(30,100,80,30);

		add(b);

		setSize(300,300);
		setLayout(null);
		setVisible(true);
	}
	public static void main(String args[])
	{
		P1 p=new P1();
	}
}
```

15) Java AWT hierarchy with a suitable block diagram

AWT stands for Abstract window toolkit is an Application programming interface (API) for creating Graphical User Interface (GUI) in Java. It allows Java programmers to develop window-based applications.

AWT provides various components like button, label, checkbox, etc. used as objects inside a Java Program. AWT components use the resources of the operating system, i.e., they are platform-dependent, which means, component's view can be changed according to the view of the operating system. The classes for AWT are provided by the Java.awt package for various AWT components.

The following image represents the hierarchy for Java AWT.



Component Class
The component class stands at the top of the AWT hierarchy, is an abstract class that contains all the properties of the component visible on the screen. The Component object contains information about the currently selected foreground and background color. It also has information about the currently selected text color.

Container
The container is a component that contains other components like button, textfield, label, etc. However, it is a subclass of the Component class.

Panel
The panel can be defined as a container that can be used to hold other components. However, it doesn't contain the title bar, menu bar, or border.

Window
A window can be defined as a container that doesn't contain any border or menu bar. It creates a top-level view. However, we must have a frame, dialog, or another window for creating a window.

Frame
The frame is a subclass of Window. It can be defined as a container with components like button, textfield, label, etc. In other words, AWT applications are mostly created using frame container.

16)

```
//-------P1.java------------------------------------------------------------------------
--------------------------
//By extending Frame class (inheritance)
import java.awt.*;
class P1 extends Frame
{
	P1()
	{
		//Button class
		Button b=new Button("Sumeet");

		//Setting button position
/*setBounds(int x, int y, int width, int height) - Specifies the size of the frame and the location
 of the upper left corner x axis to the right and y axis to the bottom*/
```

```
		b.setBounds(30,100,80,30);

		//Adding button into frame
		add(b);

		//Frame size - 300 width and 300 height
		setSize(300,300);

		//No layout manager
		setLayout(null);

		//Now frame will be visible, by default not visible
		setVisible(true);
	}
	public static void main(String args[])
	{
		P1 p=new P1();
	}
}

//-------P2.java--------------------------------------------------------------------
-------------------------

//By creating the object of Frame class (association)
import java.awt.*;
class P2
{
	P2()
	{
		//Frame class
		Frame f=new Frame();
		Button b=new Button("Click Me");
		b.setBounds(30,50,80,30);
		f.add(b);
		f.setSize(300,300);
		f.setLayout(null);
		f.setVisible(true);
	}
	public static void main(String args[])
	{
		P2 p=new P2();
	}
}
```

-------P3.java--

```
//Labels and Buttons
import java.awt.*;
class P3
{
	public static void main(String args[])
	{
		Frame f= new Frame("Labels and Buttons");

		//Labels
		Label l1,l2;
		l1=new Label("Label 1");
		l1.setBounds(50,100, 100,30);

		l2=new Label("Label 2");
		l2.setBounds(50,150, 100,30);

		f.add(l1);
		f.add(l2);

		//Buttons
		Button b1, b2;
		b1=new Button("Label 1");
		b1.setBounds(160,100, 100,30);

		b2=new Button("Label 2");
		b2.setBounds(160,150, 100,30);

		f.add(b1);
		f.add(b2);

		f.setSize(400,400);
		f.setLayout(null);
		f.setVisible(true);
	}
}
```

-------P4.java--

```
//TextField
import java.awt.*;
class P4
{
	public static void main(String args[])
	{
		Frame f= new Frame("TextField");

		//Labels
		Label l1,l2;
		l1=new Label("First Name");
		l1.setBounds(50,100, 100,30);

		l2=new Label("Last Name");
		l2.setBounds(50,150, 100,30);

		f.add(l1);
```

```
            f.add(l2);

            //TextField
            TextField t1,t2;
            t1=new TextField();  //Textbox
            t1.setBounds(160,100, 100,30);

            t2=new TextField();
            t2.setBounds(160,150, 100,30);

            f.add(t1);
            f.add(t2);

            //Buttons
            Button b1, b2;
            b1=new Button("Save");
            b1.setBounds(290,100, 100,30);

            b2=new Button("Save");
            b2.setBounds(290,150, 100,30);

            f.add(b1);
            f.add(b2);

            f.setSize(600,400);
            f.setLayout(null);
            f.setVisible(true);
        }
}
```

-------P5.java---

```
//TextArea
import java.awt.*;
class P5
{
	public static void main(String args[])
	{
		Frame f= new Frame();
		TextArea t=new TextArea();
		t.setBounds(10,30, 300,200);
		f.add(t);
		f.setSize(400,400);
		f.setLayout(null);
		f.setVisible(true);
	}
}
```

-------P6.java---

```
//Checkbox and Radio button (CheckboxGroup)
import java.awt.*;
class P6
{
	public static void main(String args[])
	{
	Frame f= new Frame("Checkbox and Radio button 
(CheckboxGroup)");

	Checkbox chk1, chk2, chk3;
		chk1 = new Checkbox("Core Java");
	chk1.setBounds(100,100, 100,50);
}


        chk2 = new Checkbox("DBMS");
        chk2.setBounds(100,140, 50,50);

                chk3 = new Checkbox("TOC");
        chk3.setBounds(100,180, 50,50);

        f.add(chk1);
        f.add(chk2);
                f.add(chk3);

                //CheckboxGroup class converts checkbox into radio
button
                CheckboxGroup cbg = new CheckboxGroup();
        Checkbox rbtn1, rbtn2, rbtn3;

                rbtn1 = new Checkbox("Core Java", cbg, true);
        rbtn1.setBounds(200,100, 100,50);

        rbtn2 = new Checkbox("DBMS", cbg, false);
        rbtn2.setBounds(200,140, 50,50);

                rbtn3 = new Checkbox("TOC", cbg, false);
        rbtn3.setBounds(200,180, 50,50);

        f.add(rbtn1);
        f.add(rbtn2);
                f.add(rbtn3);

        f.setSize(600,400);
        f.setLayout(null);
        f.setVisible(true);
        }
}
```

-------P7.java---

```
//Checkbox and Radio button (CheckboxGroup)
import java.awt.*;
class P6
{
        public static void main(String args[])
        {
        Frame f= new Frame("Checkbox and Radio button
(CheckboxGroup)");

        Checkbox chk1, chk2, chk3;
                chk1 = new Checkbox("Core Java");
        chk1.setBounds(100,100, 100,50);

        chk2 = new Checkbox("DBMS");
        chk2.setBounds(100,140, 50,50);

                chk3 = new Checkbox("TOC");
        chk3.setBounds(100,180, 50,50);

        f.add(chk1);
        f.add(chk2);
                f.add(chk3);

                //CheckboxGroup class converts checkbox into radio
button
                CheckboxGroup cbg = new CheckboxGroup();
        Checkbox rbtn1, rbtn2, rbtn3;
```

```
		rbtn1 = new Checkbox("Core Java", cbg, true);
	rbtn1.setBounds(200,100, 100,50);

	rbtn2 = new Checkbox("DBMS", cbg, false);
	rbtn2.setBounds(200,140, 50,50);

		rbtn3 = new Checkbox("TOC", cbg, false);
	rbtn3.setBounds(200,180, 50,50);

	f.add(rbtn1);
	f.add(rbtn2);
		f.add(rbtn3);

	f.setSize(600,400);
	f.setLayout(null);
	f.setVisible(true);
	}
}
```

-------P8.java--

```
//Checkbox and Radio button (CheckboxGroup)
import java.awt.*;
class P6
{
	public static void main(String args[])
	{
        Frame f= new Frame("Checkbox and Radio button
(CheckboxGroup)");

        Checkbox chk1, chk2, chk3;
                chk1 = new Checkbox("Core Java");
        chk1.setBounds(100,100, 100,50);

        chk2 = new Checkbox("DBMS");
        chk2.setBounds(100,140, 50,50);

                chk3 = new Checkbox("TOC");
        chk3.setBounds(100,180, 50,50);

        f.add(chk1);
        f.add(chk2);
                f.add(chk3);

                //CheckboxGroup class converts checkbox into radio
button
                CheckboxGroup cbg = new CheckboxGroup();
        Checkbox rbtn1, rbtn2, rbtn3;

                rbtn1 = new Checkbox("Core Java", cbg, true);
        rbtn1.setBounds(200,100, 100,50);

        rbtn2 = new Checkbox("DBMS", cbg, false);
        rbtn2.setBounds(200,140, 50,50);

                rbtn3 = new Checkbox("TOC", cbg, false);
        rbtn3.setBounds(200,180, 50,50);

        f.add(rbtn1);
        f.add(rbtn2);
                f.add(rbtn3);
        f.setSize(600,400);
        f.setLayout(null);
        f.setVisible(true);
        }
}
```

-------P9.java--

```
//Java BorderLayout
/*The BorderLayout is used to arrange the components in five
regions: north, south, east, west and center. Each region (area) may
contain one component only. It is the default layout of
frame or window. The BorderLayout provides five constants for each
region:

        public static final int NORTH
        public static final int SOUTH
        public static final int EAST
        public static final int WEST
        public static final int CENTER
*/

import java.awt.*;

class P9
{
        public static void main(String[] args)
        {
                Frame f=new Frame();
```

```
		Button b1=new Button("NORTH");
		Button b2=new Button("SOUTH");
		Button b3=new Button("EAST");
		Button b4=new Button("WEST");
		Button b5=new Button("CENTER");

		f.add(b1,BorderLayout.NORTH);
		f.add(b2,BorderLayout.SOUTH);
		f.add(b3,BorderLayout.EAST);
		f.add(b4,BorderLayout.WEST);
		f.add(b5,BorderLayout.CENTER);

		f.setSize(500,500);
		f.setVisible(true);
	}
}
```

-------P10.java---

```
//Java GridLayout
/*The GridLayout is used to arrange the components in rectangular
grid. One component is displayed in each rectangle.*/

import java.awt.*;

class P10
{
    public static void main(String[] args)
    {
            Frame f=new Frame();

            Button b1=new Button("1");
            Button b2=new Button("2");
            Button b3=new Button("3");
            Button b4=new Button("4");
            Button b5=new Button("5");
   Button b6=new Button("6");
    Button b7=new Button("7");
            Button b8=new Button("8");
   Button b9=new Button("9");

            f.add(b1);
            f.add(b2);
            f.add(b3);
            f.add(b4);
            f.add(b5);
            f.add(b6);
            f.add(b7);
            f.add(b8);
            f.add(b9);

            //Setting grid layout of 3 rows and 3 columns
            f.setLayout(new GridLayout(3,3));

            f.setSize(300,300);
            f.setVisible(true);
    }
}
```

-------P11.java--

```
//Java GridLayout
/*The GridLayout is used to arrange the components in rectangular
grid. One component is displayed in each rectangle.*/

import java.awt.*;

class P10
{
	public static void main(String[] args)
	{
		Frame f=new Frame();

		Button b1=new Button("1");
		Button b2=new Button("2");
		Button b3=new Button("3");
		Button b4=new Button("4");
		Button b5=new Button("5");
        Button b6=new Button("6");
          Button b7=new Button("7");
		Button b8=new Button("8");
        Button b9=new Button("9");

		f.add(b1);
		f.add(b2);
		f.add(b3);
		f.add(b4);
		f.add(b5);
		f.add(b6);
		f.add(b7);
		f.add(b8);
		f.add(b9);

		//Setting grid layout of 3 rows and 3 columns
		f.setLayout(new GridLayout(3,3));

		f.setSize(300,300);
		f.setVisible(true);
	}
}
```

-------P12.java--

```
//Java CardLayout
/*The CardLayout class manages the components in such a manner that only one component
is visible at a time. It treats each component as a card that is why it is known as CardLayout.*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

//class extends JFrame and implements actionlistener
public class P12 extends JFrame implements ActionListener
{
	//Declaration of objects of CardLayout class.
	CardLayout card;
```

```
        //Declaration of objects of JButton class.
        JButton b1,b2,b3;

        // Declaration of objects of Container class to get the content
        Container c;
        P12()
        {
                //Gets the content
                c=getContentPane();

                //CardLayout class object with 40 horizontal space and 30
vertical space .
                card=new CardLayout(40,30);
                c.setLayout(card);

                b1=new JButton("Core Java");
                b2=new JButton("is very");
                b3=new JButton("Easy ....");

                b1.addActionListener(this);
                b2.addActionListener(this);
                b3.addActionListener(this);

                c.add(b1);
                c.add(b2);
                c.add(b3);
        }
        public void actionPerformed(ActionEvent e)
                {
                //Call the next card
                card.next(c);
        }

        public static void main(String[] args)
```

```
	{
		P12 cl=new P12();
		cl.setSize(400,400);
		cl.setVisible(true);

		 //Function to set default operation of JFrame.
		cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
	}
}
```