1. Explain different features of Java.

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some exellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

The most important features of the Java language are as follow:

Compiled and Interpreted:-

Usually a computer language is either compiled or interpreted Java can be considered both a compiled and an interpreted language because its source code is first compiled into a binary, byte-code. Bytecode are not machine instructions and therefore, Java interpreter generates machine code that can be directly executed by the machine that is running the java program.

Platform-Independent and Portable:-

The most significant contribution of java over other languages is its portability. Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the size of the primitive data types are machine independent.

Object-Oriented

Java is a true object-oriented language. Almost everything in java is an object. All program code All program code and data reside within objects and classes. Java comes with an extensive set of classes, arranged in packages, that we can use in our programs by inheritance. The object model in java is simple and easy to extend.

Robust and Secure.

Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. It is designed as a garbage collected language relieving the programmers virtually all memory management problems.

Security becomes an important issue for a language that is used for programming on Internet. Threat of viruses and abuse of resources are everywhere. Java systems not only verify all memory access but program cannot gain access to memory location without proper authorization.

Distributed

Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java application can open and access remote objects on Internet as easily as they can do in a local system. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

Simple, Small and Familiar

Java is small and simple language. It is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language.

Familiarity is another striking feature of Java. To make the language look familiar to the existing programmers. It was modelled on C and C++ languages. Java uses many constructs of C and C++ and therefore, Java code "looks like a C++" code. In fact, Java is a simplified version of C++.

Multithreaded and Interactive:-

Multithreaded means handling multiple tasks simultaneously.. Java supports multithreaded programs. This means that we need not wait for the application to finish one task before beginning another. For example, we can listen to an audio clip while scrolling a page and at the same time download an applet from a distant computer. This features greatly improves the interactive performance of graphical applications.

High Performance:-

Java performance is impressive for an interpreted language, mainly due to use of intermediate bytecode. According to Sun, Java speed is comparable to the native C/C++. Java architecture is also designed to reduce overheads during runtime. Further, the incorporation of multireading enhances the overall execution speed of java programs.

Dynamic and Extensible:-

Java is a dynamic language. Java is capable of dynamically linking in new class libraries, methods, and objects. Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response.

Java programs support function written in other languages such as C and C++. These functions are known as native methods. This facility enables the programmers to use the efficient functions available in these languages. Native methods are linked dynamically at runtime.

2. What are different Data types in Java? Explain.

Data Types :-

Every variable in Java has a data type. Data types specify the size and type of values that can be stored. Java language is rich in its data types. The variety of data types available allow the programmer to select the type appropriate to the needs of the application.

Integer Types :-

Integer types can hold whole numbers such as 123, -96, and 5639. The size of the values that can be stored depends on the integer data type we choose. Java supports four types of integers, byte, short, int, and long. Java does not support the concept of unsigned types and therefore all Java values are signed meaning they can be positive or negative.

Floating Point Types :-

Integer types can hold only whole numbers and therefore we use another type known as floating point type to hold numbers containing fractional parts such as 27.59 and -1.375. There are two kinds of floating point storage in Java. The float type values are single-precision numbers while the double types represent double-precision numbers. Floating point numbers are treated as double-precision quantities. To force them to be in single-precision mode, we must append f or F to the numbers. For example; 1.23f, 7.56923e5F. Double-precision types are used when we need greater precision in storage of floating point numbers. All mathematical functions, such as sin, cos and sqrt return double type values. Floating point data types support a special value known as Not-a-Number (NaN). NaN is used to represent the result of operations such as dividing zero by zero, where an actual number is not produced. Most operations that have NaN as an operand will produce NaN as a result.

Character Type :-

In order to store character constants in memory, Java provides a character data type called char. The char type assumes a size of 2 bytes but, basically, It can hold only a single character.

Boolean Type :-

Boolean type is used when we want to test a particular condition during the execution of the program. There are only two values that a boolean type can take: true or false. Boolean type is denoted by the keyword boolean and uses only one bit of storage. All comparision operators return boolean values. Boolean values are often used in selection and iteration statements. The word true and false cannot be used as identifiers.

3. What is operator? Explain Arithmetic and Relational Operators in java.

Operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operator are used in programs to manipulate data and variables. They usually form a part of mathematical or logical expressions.

Arithmetic Operators:-

Arithmetic operators are used to construct mathematical expressions as in algebra. Java provides all the basic arithmetic operators. The operators +, -, *, and / all work the same way as they do in other languages. These can operate on any built-in numeric data type of Java. We cannot use these operators on boolean type.

Arithmetic operators are used as shown below:

a-b	a+b
a*b	a/b
a%b	-a*b

Here a and b may be variables or constants and are known as operands.

Operator	Meaning
+	Addition or unary plus.
-	Subtraction or unary minus.
*	Multiplication
/	Division
%	Modulo division (Remainder)

Relational Operators:-

We often compare two quantities, and depending on their relation, take certain decisions. For example, we may compare the age of two persons, or the price of two items, and so on. This comparisions can be done with the help of relational operators. We have already used the symbol '<' meaning 'less than'. An expression such as

$$a < b \quad \text{or} \quad x < 20$$

containing a relational operators is termed as a relational expression. The value of relational expression is either true or false. For example, if $x = 10$, then

$x < 20$ is true.

while

$20 < x$ is false

Java supports six relational operators in all.

A simple relational expression contains only on relational operator and is of the following form:

ae-1 relational operator ae-2

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

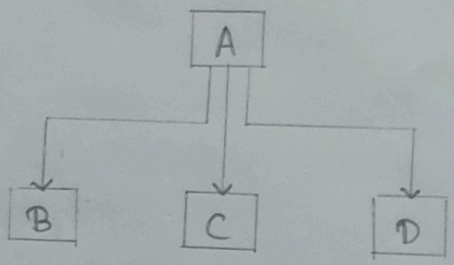
4. Explain the concept of Inheritance in java with suitable example.

The mechanism of deriving a new class from an old one is called inheritance. The old class is known as the base class or super class or parent class and the new one is called the subclass or derived class or child class.

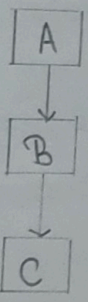
The inheritance allows subclasses to inherit all the variables and methods of their parent classes. Inheritance may take different forms:

- Single inheritance (only one super class)
- Multiple inheritance (several super classes)
- Hierarchical inheritance (one super class, many subclasses
- Multilevel inheritance (Derived from a derived class)

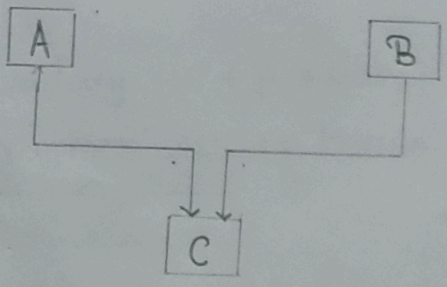
(a) Single inheritance



(b) Hierarchical inheritance



(c) Multilevel inheritance



(d) Multiple inheritance

Java does not directly implement multiple inheritance. However, this concept is implemented using a secondary inheritance path in the form of interfaces.

The syntax of Java Inheritance

```
class subclassname extends superclassname
{
    variables declaration;
    methods declaration;
}
```

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.
In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Java Inheritance Example:-

```
class Employee {
float salary = 40,000;
}
class Programmer extends Employee {
int bonus = 10000;
public static void main(String args[]){
Programmer p = new Programmer ();
System.out.println ("Programmer salary is:" + p.salary);
System.out.println("Bonus of Programmer is:" +p.bonus);
}
}
```

Output:

```
Programmer salary is: 40000.0
Bonus of programmer is: 10000
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

5. Explain the concept of Interface with syntax and example.

An interface is basically a kind of class. Like classes, interfaces contain methods and variables but with a major difference. The difference is that interfaces define only abstract methods and final fields. This means that interfaces do not specify any code to implement these methods and data fields contain only constants. Therefore, it is the responsibility of the class that implements an interface to define the code for implementation of these methods.

The syntax for defining an interface is very similar to that for defining a class. The general form of an interface definition is:

```
interface InterfaceName
{
    variables declaration;
    methods declaration;
}
```

Here, interface is the key word and InterfaceName is any valid Java variable. Variable are declared as follow:

```
static final type VariableName = Value;
```

Note that all variables are declared as constants. Methods declaration will contain only a list of methods without any body statements.

Ex.

```
return-type methodName1 (parameter_list);
```

Here is an example of an interface definition that contains two variables and one method:

```
Interface Item
{
    static final int code = 1001;
    static final String name = "Fan";
    void display();
}
```

Java Interface Example:

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){
System.out.println("Hello");}
public static void main(String args[]){
A6 obj = new A6();
obj.print();
}
}
```

Output:-

Hello

6. What is Package? Explain import statement.

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

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java packages provides access protection.
3) Java package removes naming collision.

Import Statement:-

Most of the Java programs starts with the statements having import keyword. It is similar to the preprocessor directives used in C or C++ programming. The import in Java is a keyword that allows the programmer to access packages available in Java. It is used to import a package, sub-package, a class, an interface or enum in the Java program.

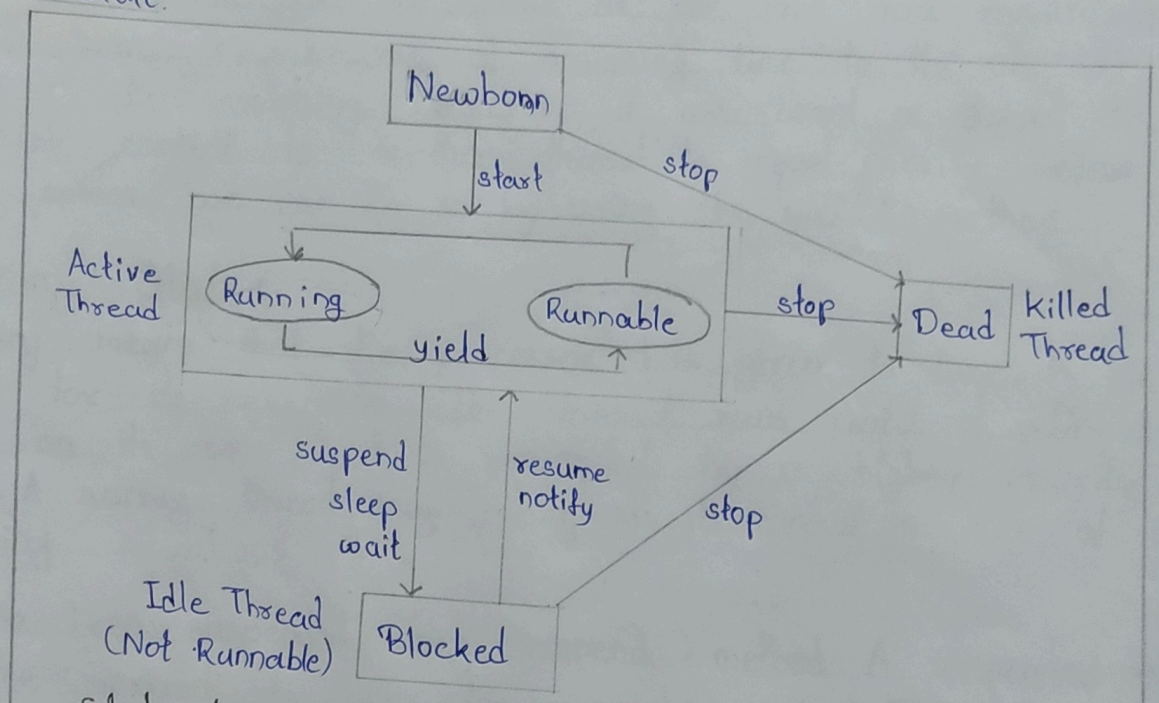
Syntax:-

```
import packageName.*;
```

7. Explain Life cycle of Thread.

During the life time of a thread, there are many states it can enter. They include:

1. Newborn state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state



State transition diagram of a thread

1. Newborn State:

When we create a thread object, the thread is born and is said to be in newborn state. The thread is not yet scheduled for running. At this state, we can do only one of the following things with it:

- Schedule it for running using start() method.
- Kill it using stop() method.

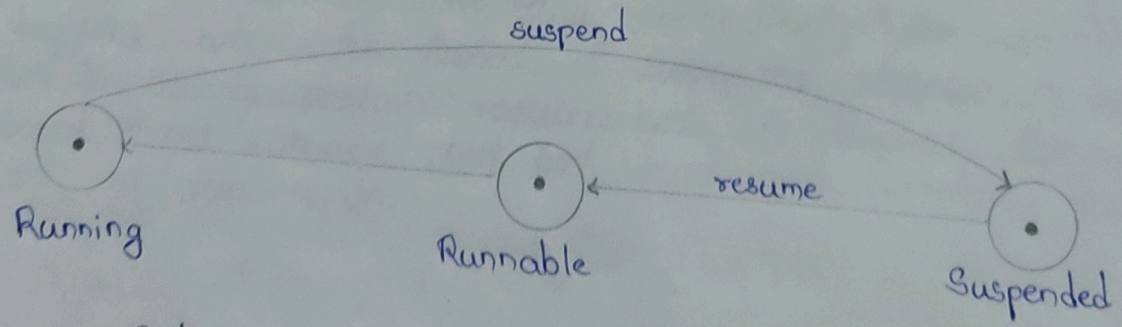
2. Runnable State:

The runnable state means that the thread is ready for execution and is waiting for the availability of the processor. That is, the thread has joined the queue of threads that are waiting for execution. If all threads have equal priority, then they are given time slots for execution in round robin fashion, i.e. first come, first-serve manner. The thread that relinquishes control joins the queue at the end and again waits for its turn. This process of assigning time to threads is known as time-slicing. However, if we want a thread to relinquish control to another thread to equal priority before its turn comes, we can do so by using the yield() method.

3. Running State:

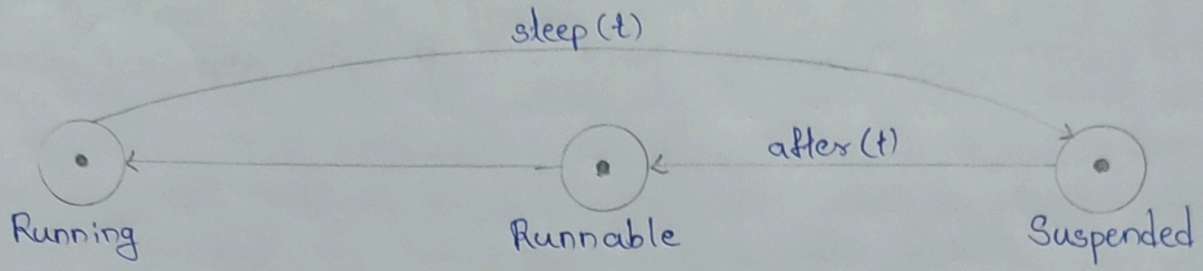
Running means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread. A running thread may relinquish its control in one of the following situations.

1. It has been suspended using suspend() method. A suspended thread can be revived by using the resume() method. This approach is useful when we want to suspend a thread for some time due to certain reason, but do not want to kill it.



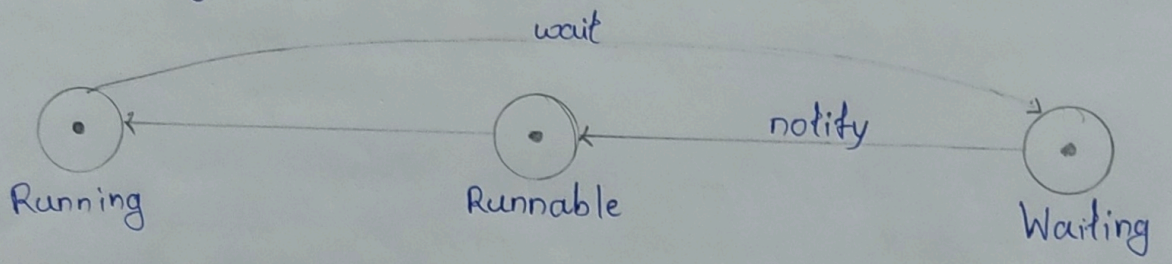
Relinquishing control using suspend() method

2. It has been made to sleep. We can put a thread to sleep for a specified time period using the method sleep(time) where time is in milliseconds. This means that the thread is out of the queue during this time period. The thread re-enters the runnable state as soon as this time period is elapsed.



Relinquishing control using sleep() method

3. It has been told to wait until some event occurs. This is done using the wait() method. The thread can be scheduled to run again using the notify() method.



4. Blocked State:

A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements. A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

5. Dead State:

Every thread has a life cycle. A running thread ends its life when it has completed executing its run() method. It is a natural death. However, we can kill it by sending the stop message to it at any state thus causing a premature death to it. A thread can be killed as soon it is born, or while it is running, or even when it is in "not runnable" (blocked) condition.

8. Explain different errors which may occur while handling exception in Java.

An error may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash. It is therefore important to detect and manage properly all the possible error conditions in the program so that the program will not terminate or crash during execution.

Types of Errors

Errors may broadly be classified into two categories:

- Compile-time errors
- Run-time errors

Compile-Time Errors

All syntax errors will be detected and displayed by the Java compiler and therefore these errors are known as compile-time errors. Whenever the compiler displays an error, it will not create the .class file. It is therefore necessary that we fix all the errors before we can successfully compile and run the program.

Most of the compile-time errors are due to typing mistakes. Typographical errors are hard to find. We may have to check the code word by word or even character by character. The most common problems are:

- Missing semicolons.
- Missing (or mismatch of) brackets in classes and methods.
- Misspelling of identifiers and keywords
- Missing double quotes in strings.
- Use of undeclared variables
- Incompatible types in assignments/initialization
- Bad references to objects
- Use of = in place of == operator
- And so on

Run-Time Errors:

Sometimes, a program may compile successfully creating the .class file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow. Most common run-time errors are:

- Dividing an integer by zero
- Accessing an element that is out of the bounds of an array.
- Trying to store a value into an array of an incompatible class or type.
- Trying to cast an instance of a class to one of its subclasses.
- Passing a parameter that is not in a valid range or value for a method.
- Trying to illegally change the state of a thread
- Attempting to use a negative size for an array.
- Using a null object reference as a legitimate object reference to access a method or a variable.
- Converting invalid string to a number.
- Accessing a character that is out of bounds of a string
- And may more.

Exception Handling:

An exception is a condition that is caused by a run-time error in the program. When the Java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it (i.e. informs us that an error has occurred)
If the exception object is not caught and handled properly, the interpreter will display an error message and will terminate the program.
If we want the program to continue with the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as exception handling.
The purpose of exception handling mechanism is to provide a means to detect and report an "exceptional circumstance" so that appropriate action can be taken. The mechanism suggests incorporation of a

separate error handling code that performs the following tasks:

1. Find the problem (Hit the exception).
2. Inform that an error has occurred (Throw the exception)
3. Receive the error information (Catch the exception)
4. Take corrective actions (Handle the exception).

The error handling code basically consists of two segments, one to detect errors and to throw exception and the other to catch exceptions and to take appropriate actions.

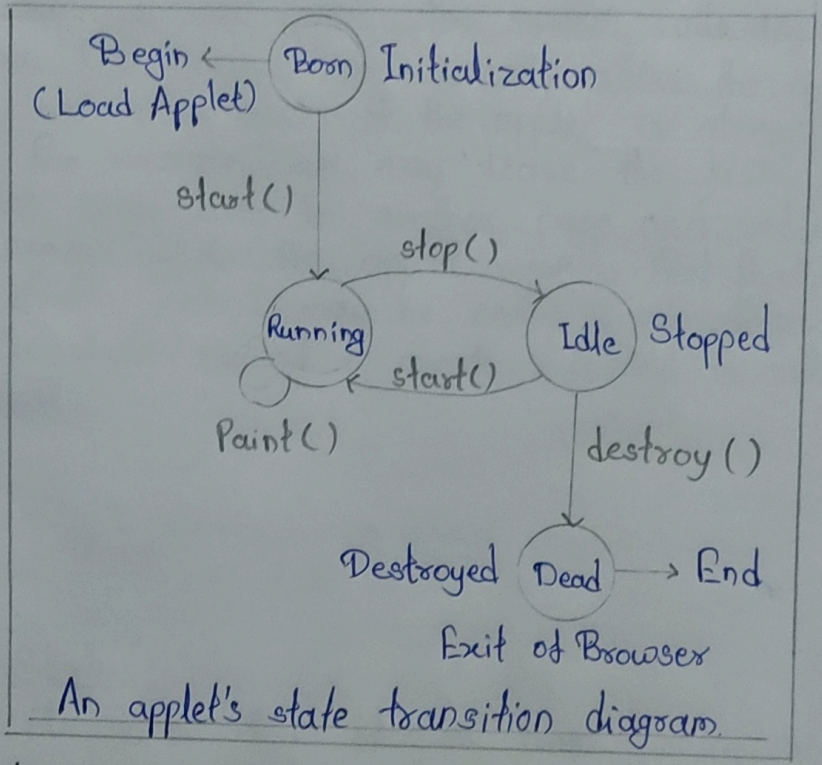
When writing programs, we must always be on the lookout for places in the program where an exception could be generated.

9. Explain Life cycle of an Applet.

Every java applet inherits a set of default behaviours from the Applet class. As a result, when an applet is loaded, it undergoes a series of changes in its state as shown in Figure below:

The applet state include:

- Born on initialization state
- Running state
- Idle state
- Dead or destroyed state



An applet's state transition diagram.

Initialization State:

Applet enters the initialization state when it is first loaded. This is achieved by calling the init() method of Applet Class. The applet is born. At this stage we may do the following, if required.

- Create objects needed by the applet.
- Set up initial values
- Load images or fonts
- Set up colors

The initialization occurs only once in the applet's life cycle. To provide any of the behaviours mentioned above, we must override the init() method:

```
public void init ()
{
    ----------
    ----------          (Action)
}
```

Running State:

Applet enters the running state when the system calls the start() method of Applet Class. This occurs automatically after the applet is initialized. Starting can also occur if the applet is already in 'stopped' (idle) state. For example, we may leave the Web page containing the applet temporarily to another page and return back to the page. This again starts the applet running. Not that, unlike init() method, the start() method may be called more than once. We may override the start() method to create a thread to control the applet.

```
public void start()
{
    ----------
    ----------
    ----------          (Action)
}
```

Idle or Stopped State:

An applet becomes idle when it is stopped from running. Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling the stop() method explicitly. If we use a thread to run the applet, then we must use stop() method to terminate the thread. We can achieve this by overriding the stop() method:

```
public void stop ( )
{
    ----------
    ----------          (Action)
    ----------
}
```

Dead or Destroyed State:

An applet is said to be dead when it is removed from memory. This occurs automatically by invoking the destroy() method when we quit the browser. Like initialization, destroying stage occurs only once in the applet's life cycle. If the applet has created any resources, like threads, we may override the destroy() method to clean up these resources.

```
public void destroy()
{
    ----------
    ---------- (Action)
    ----------
}
```

Display State:

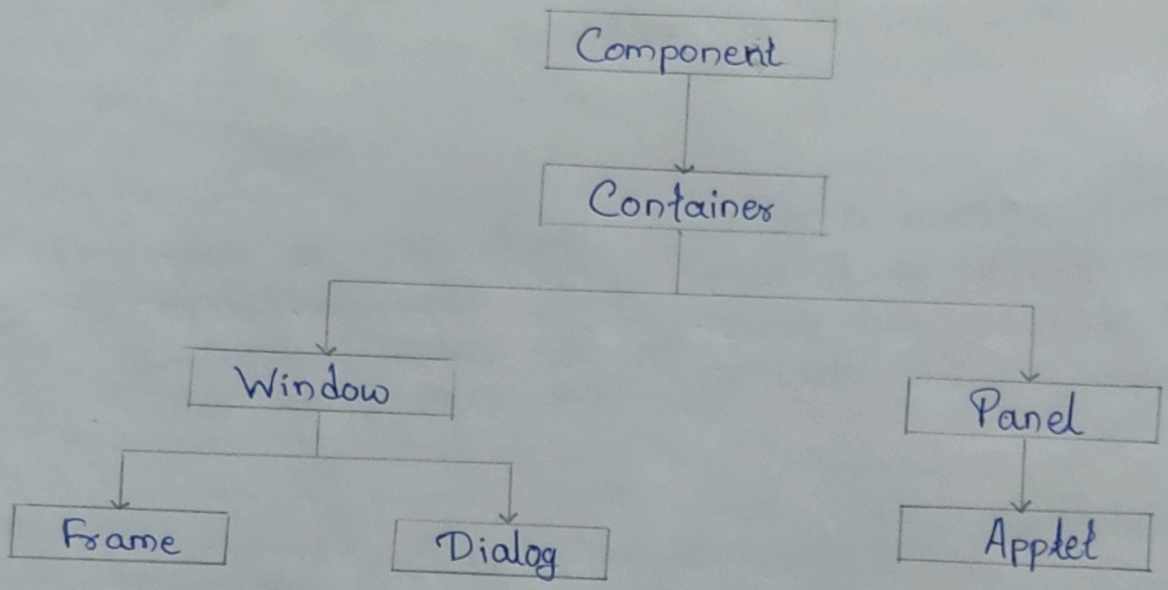
Applet moves to the display state whenever it has to perform some output operations on the screen. This happens immediately after the applet enters into the running state. The paint() method is called to accomplish this task. Almost every applet will have a applet paint() method. Like other methods in the life cycle, the default version of paint() method does absolutely nothing. We must therefore override this method if we want anything to be displayed on the screen.

```
public void paint (Graphics g)
{
    ----------
    ---------- (Display statements)
    ----------
}
```

It is to be noted that the display state is not considered as a part of the applet's life cycle. In fact, the paint() method is defined in the Applet class. It is inherited from the Component class, a super class of Applet.

10. Explain in detail AWT Package Hierarchy with diagram.

The Abstract Window Toolkit (AWT) package in Java enables the programmers to create GUI-based applications. It contains a number of classes that help to implement common Windows-based tasks, such as manipulating windows, adding scroll bars, buttons, list items, text boxes, etc. All the classes are contained in the java.awt package. These classes are hierarchically arranged inside the awt package in such a manner that each successive level in th hierarchy adds certain attributes to the GUI application.



AWT Hierarchy

AWT provides support for both standard and applet windows. Above figure shows how their corresponding classes are hierarchically arranged in the awt package.

Component:-

Component class is the super class to all the other classes from which various GUI elements are realized. It is primarily responsible for effecting the display of a graphic object on the screen. It also handles the various keyboard and mouse events of the GUI application.

Containers:-
As the name suggests, the Container object contains the other awt components. It manages the layout and placement of the various awt components within the container. A container object can contain other containers objects as well; thus allowing nesting of containers.

Window:-
The Window object realizes a top-level window but without any border or menu bar. It just specifies the layout of the window. A typical window that you would want to create in your application is not normally derived from the Window class but from its subclass, i.e, Frame.

Panel:-
The super class of applet, Panel represents a window space on which the application's output is displayed. It is just like a normal window having no border, title bar, menu bar, etc. A panel can contain within itself other panels as well.

Frame:-
The Frame object realizes a top-level window complete with border and menu bar. It supports common window-related events such as close, open, activate, deactivate, etc. Almost all the programs that we created while discussing applets and graphics programming used one or more classes of the awt package.