OOPJ Answer key:- Jan-Feb 2022

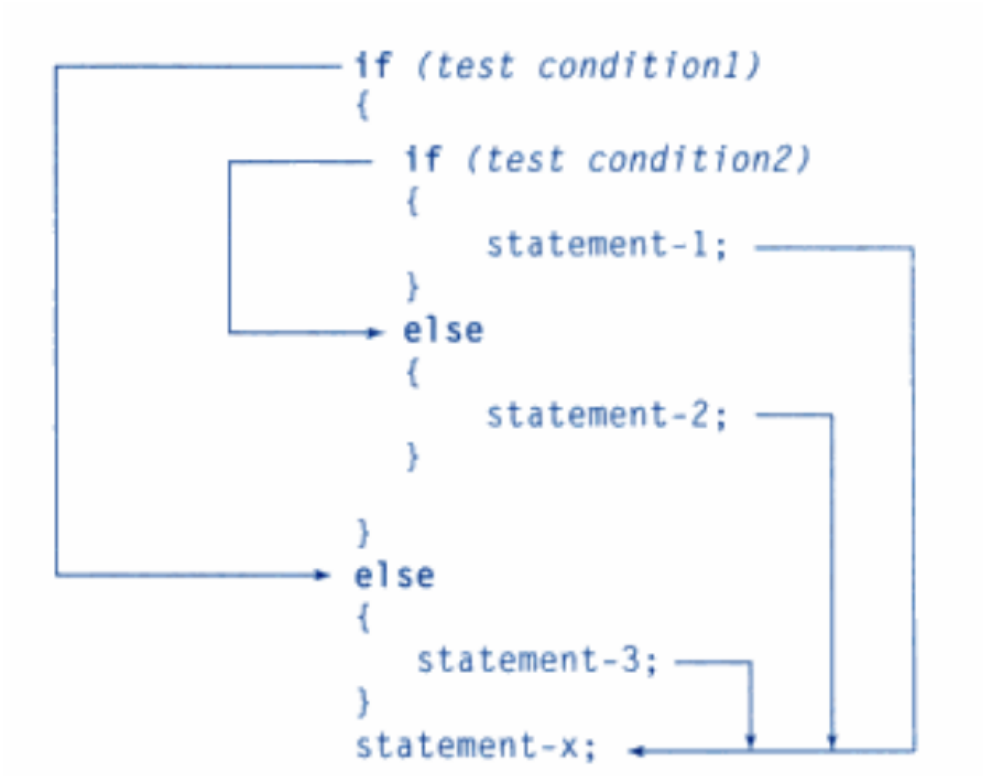
1) If else statement

If the test expression is true, then the true-block statements immediately following the if statement are executed otherwise the false-block statement are executed.

```
if(test expression)
{
   True-block statement(s)
}
else
{
   False-block statement(s)
}
statement-x
```

```
.....
.....
if(code == 1)
  boy = boy + 1;
else
  girl = girl + 1;
xxx;
.....
```

Nested if else statement



Example :

```
if(sex is female)
  if(balance > 5000)
    bonus = 0.05 * balance;
else
  bonus = 0.02 * balance;
  balance = balance + bonus;
```

b) Java Virtual Machine

- The Java compiler produces an intermediate code known as bytecode for a machine that does not exist. This machine is called the Java Virtual Machine and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer. **Bytecode** is also referred to as ***virtual machine code***.
-

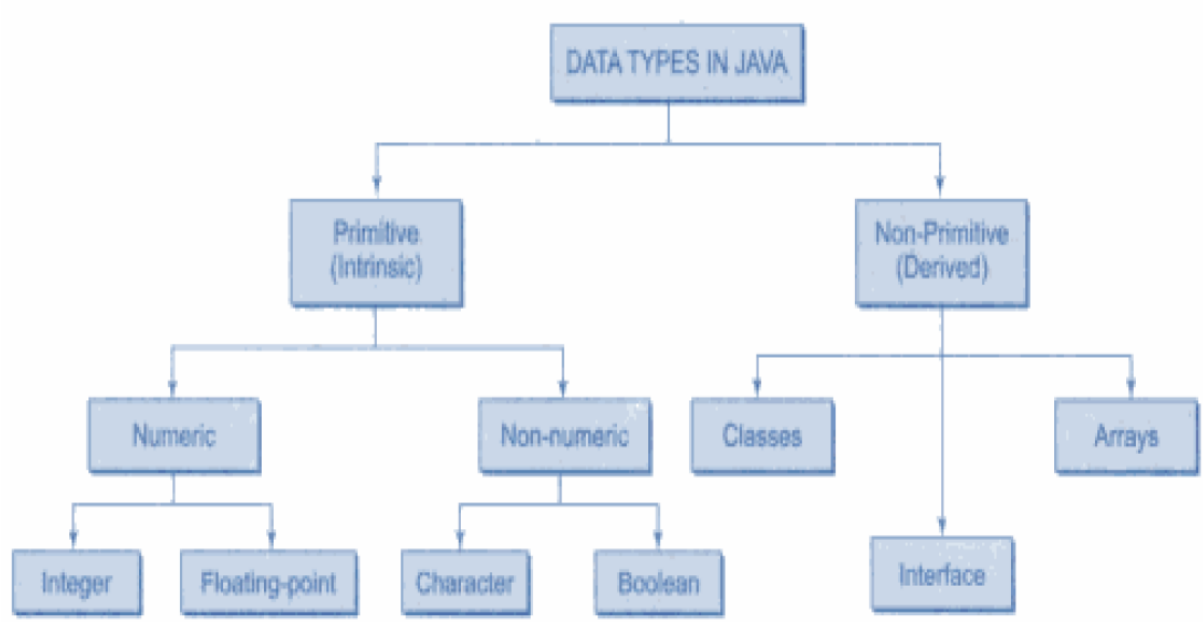
The ***virtual machine code*** is not machine specific. The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine as shown in the figure below.

- The Java object framework (Java API) acts as the intermediary between the user programs and the virtual machine which in turn acts as the intermediary between the operating system and the Java object framework.

c) Data types in Java

- Every variable in Java has a data type. Data types specify the size and type of values that can be stored. Data types in Java are divided into two main categories:

➢ Primitive types (intrinsic or built-In types)
➢ Derived types (reference types)



- Integer Data type: Integer type can bold whole numbers such as 123,-96, and 5639.The size of the values that can be stored depends on the integer data type chosen. Java supports four types of integer: ***byte, short, int, and long.***
- Java does not support unsigned types hence a java variable can a positive or negative value.
-

Type	*Size*	*Minimum value*	*Maximum value*
byte	One byte	–128	127
short	Two bytes	–32, 768	32, 767
int	Four bytes	–2, 147, 483, 648	2, 147, 483, 647
long	Eight bytes	–9, 223, 372, 036, 854, 775, 808	9, 223, 372, 036, 854, 775, 807

Floating point type

- Floating point type hold numbers containing fractional parts such as 27.59 and -1.375 (known as floating point constants).There are two kinds of floating point storage in Java:
- The ***float type*** values are single-precision numbers and the ***double type*** 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 to the numbers. Floating point data types support a special value known as ***Not-a-Number (NaN).***

- Most operations that have NaN as an operand will produce NaN as a result.

Character type

- Java provides a character data type called char. The char type assumes a size of 2 bytes and 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 comparison operator return Boolean type values.

2) For loop and while loop in java

The **for** loop is another *entry-controlled* loop that provides a more concise loop control structure. The general form of the **for** loop is

```
for (initialization ; test condition ; increment)
{
    Body of the loop
}
```

The execution of the **for** statement is as follows:

1. *Initialization* of the *control variables* is done first, using assignment statements such as i = 1 and count = 0. The variables **i** and **count** are known as loop-control variables.
2. The value of the control variable is tested using the *test condition*. The test condition is a relational expression, such as i < 10 that determines when the loop will exit. If the condition is *true*, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.
3. When the body of the loop is executed, the control is transferred back to the **for** statement after evaluating the last statement in the loop. Now, the control variable is *incremented* using an assignment stement such as i = i + 1 and the new value of the control variable is again tested to see whether it satisfies the loop condition. If the condition is satisfied, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test condition.

Consider the following segment of a program

```
for (x = 0 ; x <= 9; x = x+1)
{
    System.out.println(x);
}
```

The simplest of all the looping structures in Java is the **while** statement. The basic format of the **while** statement is

```
Initialization:
While (test condition)
{
      Body of the loop
}
```

The **while** is an *entry-controlled* loop statement. The *test condition* is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop. On exit, the program continues with the statement immediately after the body of the loop.

The body of the loop may have one or more statements. The braces are needed only if the body contains two or more statements. However, it is a good practice to use braces even if the body has only one statement.

Consider the following code segment:

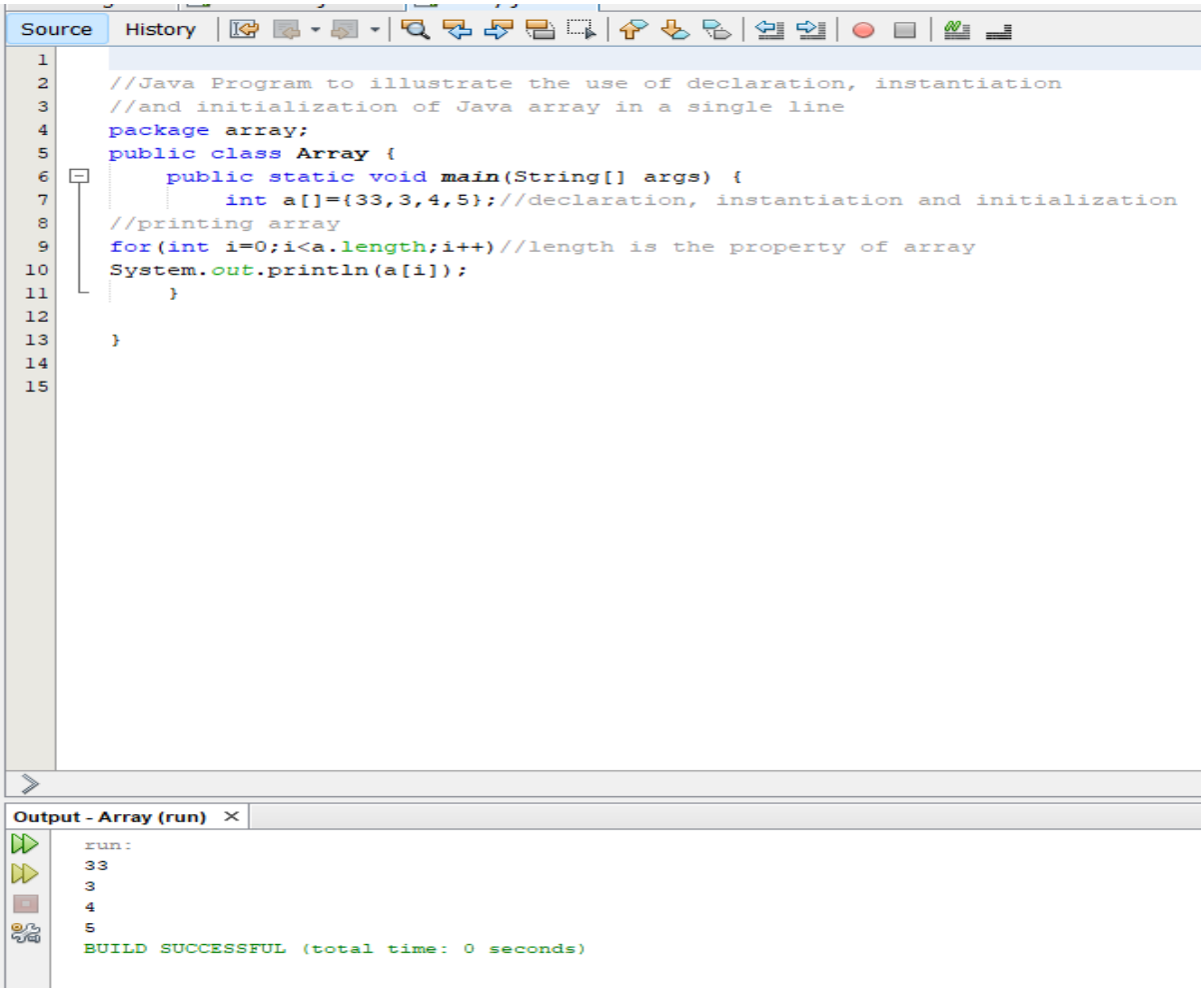
```
.....
.....
sum = 0;
n = 1;
while(n <= 10)
{
      sum = sum + n * n;
      n = n+1;
}
System.out.println("Sum = "+ sum);
.....
.....
```

The body of the loop is executed 10 times for n = 1, 2, ..., 10 each time adding the square of the value of n, which is incremented inside the loop. The test condition may also be written as n < 11; the result would be the same. Program 7.1 illustrates the use of the **while** for reading a string of characters from the keyboard. The loop terminates when c = '\ n', the newline character.

What is an array? Explain the steps involved in declaring and displaying elements in an array.

- **Java array** is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.
- It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.
- In Java, array is an object of a dynamically generated class. Java array inherits the Object class and implements the Serializable interfaces. We can store primitive values or objects in an array in Java.

- Like C/C++, we can also create single dimensional or multidimensional arrays in Java



What are constructors? Explain their use with an example

- Because the requirement for initialization is so common, Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor.
- It has the same name as the class in which it resides and is syntactically similar to a method. Once defined, the constructor is automatically called when the object is created, before the new operator completes.
- Constructors look a little strange because they have no return type, not even void. This is because the implicit return type of a class' constructor is the class type itself.

class Box { double width; double height; double depth;

Box() { System.out.println("Constructing Box"); width = 10; height = 10;depth = 10;}

double volume() {return width * height * depth;}

```
}
class BoxDemo6
{    public static void main(String args[])
{ Box mybox1 = new Box(); Box mybox2 = new Box();
    double vol;
    vol = mybox1.volume();
    System.out.println("Volume is " + vol);
        vol = mybox2.volume();
        System.out.println("Volume is " + vol);
}
}
```

3) How is exception handling done in java?

Exception Handling is a mechanism to handle runtime errors.

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

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)

Syntax of exception handling code

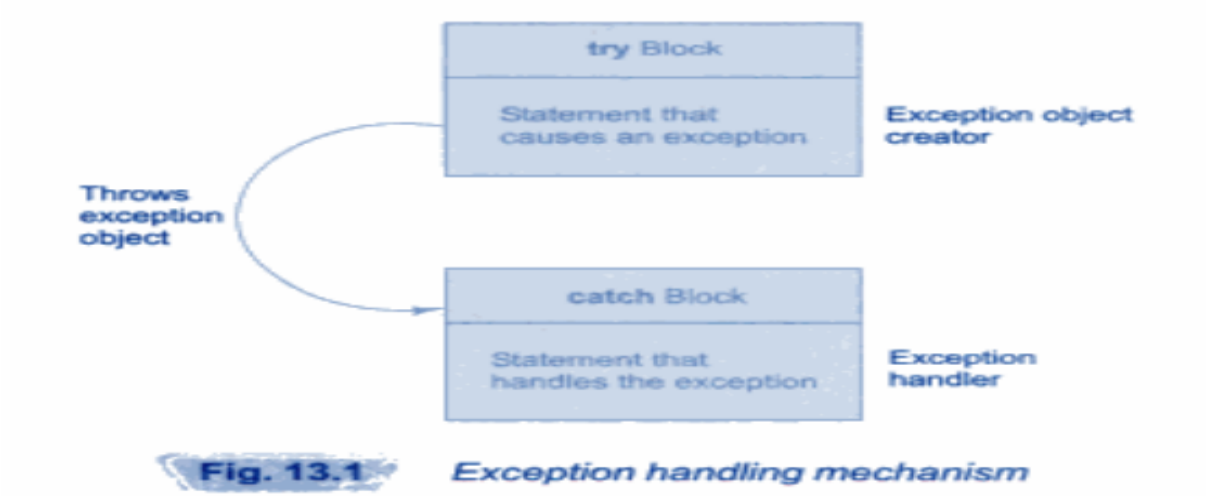


Fig. 13.1 *Exception handling mechanism*

Java uses a keyword **try** to preface a block of code that is likely to cause an error condition and "throw" an exception. A catch block defined by the keyword **catch** "catches" the exception "thrown" by thee try block and handles it appropriately. The catch block is added immediately after the try block.

What is a java package? Explain the types of java packages.

- 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.

Table 11.1 Java System Packages and Their Classes

Package name	*Contents*
java.lang	Language support classes. These are classes that Java compiler itself uses and therefore they are automatically imported. They include classes for primitive types, strings, math functions, threads and exceptions.
java.util	Language utility classes such as vectors, hash tables, random numbers, date, etc.
java.io	Input/output support classes. They provide facilities for the input and output of data.
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.applet	Classes for creating and implementing applets.

Explain how a thread can be created by extending the thread.

Extending a thread class

- Extending thread gives access to all the thread methods directly.

1) Declares the class as extending the Thread class.
2) Implement the run() method that is responsible for executing the sequence of code that the thread will execute.
3) Create a thread object and call the start() method to initiate the thread execution.

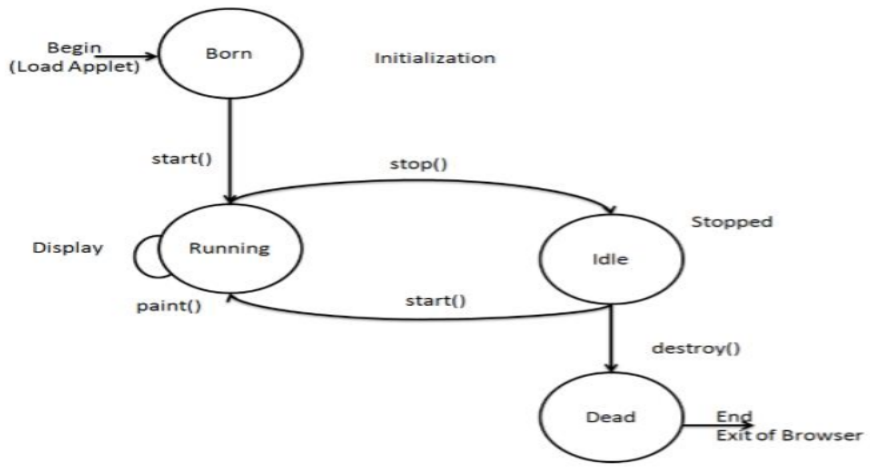
To declare the class

```
Class Mythread extends thread
        {
                ....
        }
```

4) Describe the life cycle of an applet with a diagram

- Every java applets 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.
- The applet states include:

1) Born or initialization state
2) Running state
3) Idle state
4) Dead or destroyed state



1) 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:

   a. Create objects needed by the Applet
   b. Set up initial values

c. Load images or fonts
d. Set up colors

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

```
public void init()
   {
   ……//action
   }
```

2) Running state

Apple 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. Unlike init(), start() method can be called more than once. We may override the start() method to create a thread to control the applet.

```
public void start()
{
……//action
}
```

3) Idle or Stopped state

An applet becomes idle when it is stopped from running.

Stopping occurs automatically when we leave the page containing the current 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
}
```

4) Dead 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.

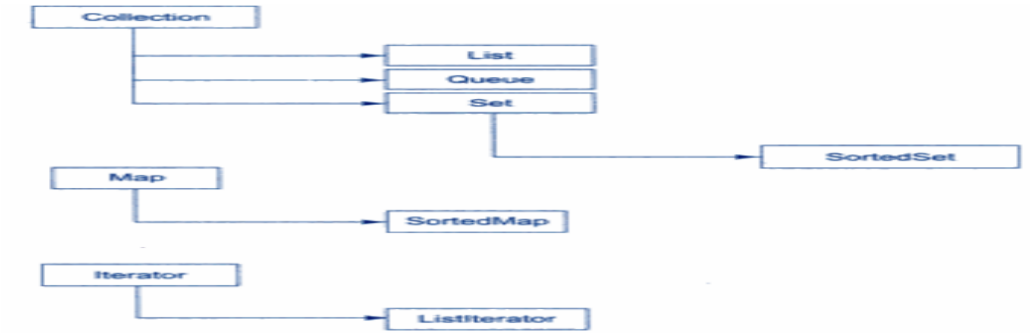
```
Public void destroy()
{
…….//action
}
```

5) Display State

Applets moves to the display state whenever it has to perform some output operations on the screen. This happens immediately after the applet enter into running state.

Write a note on java collections

- The collection framework which is contained in the java.util package is one of java's most powerful sub-systems.
- The collection framework defines a set of interfaces and their implementations to manipulate collections, which serve as a container for a group of objects such as a set of words in a dictionary or a collection of mails.
- The collections framework also allows us to store, retrieve , and update a set of objects.
- It provides an API to work with the data structures, such as lists, trees, maps and sets.
- In this will shall discuss the interfaces, classes and algorithms available in the collections framework.



- The collections framework contains many interfaces such as collection, map, and iterator. other interfaces of the framework extend these interfaces.
- The interface available in the collections framework can be structured as
- The interfaces list and set are the subinterface of the collection interface.
- The SortedMap interface is the subinterface of the map interface.
- The LIstIterator interface is the subinterface of the iterator interface.

Describe the 3 methods in the AWT graphics class with examples

5) Explain the stage, node and scene graph with examples.

The Stage and Scene Classes

- All JavaFX applications have at least one stage and one scene. These elements are encapsulated in the JavaFX API by the Stage and Scene classes. To create a JavaFX application, you will, at minimum, add at least one Scene object to a Stage.
- **Stage** is a top-level container. All JavaFX applications automatically have access to one Stage, called the primary stage.
- The primary stage is supplied by the run-time system when a JavaFX application is started. Although you can create other stages, for many applications, the primary stage will be the only one required.

Scene is a container for the items that comprise the scene. These can consist of controls, such as push buttons and check boxes, text, and graphics. To create a scene, you will add those elements to an instance of **Scene**

Nodes and Scene Graphs

- The individual elements of a scene are called nodes. For example, a push button control is a node. However, nodes can also consist of groups of nodes.

- A node can have a child node. In this case, a node with a child is called a parent node or branch node.
- Nodes without children are terminal nodes and are called leaves. The collection of all nodes in a scene creates what is referred to as a scene graph, which comprises a tree.
- The root node is a special type of node and the top-level node in the scene graph and is the only node in the scene graph that does not have a parent. Thus, with the exception of the root node, all other nodes have parents, and all nodes either directly or indirectly descend from the root node.
- The base class for all nodes is Node. There are several other classes that are, either directly or indirectly, subclasses of Node. These include Parent, Group, Region, and Control, to name a few.

Layouts

- JavaFX provides several layout panes that manage the process of placing elements in a scene. For example, the FlowPane class provides a flow layout and the GridPane class supports a row/column grid-based layout. Several other layouts, such as BorderPane (which is similar to the AWT's BorderLayout), are available. The layout panes are packaged in javafx.scene.layout.

Write note on InetAddress class, RowSet interface

- The InetAddress class is used to encapsulate both the numerical IP address and the domain name for that address. You interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address.
- The InetAddress class hides the number inside. InetAddress can handle both IPv4 and IPv6 addresses.

- Factory Methods

The InetAddress class has no visible constructors. To create an InetAddress object, you have to use one of the available factory methods. Factory methods are merely a convention whereby static methods in a class return an instance of that class.

Three commonly used InetAddress factory methods are shown here

`static InetAddress getLocalHost( )`
`  throws UnknownHostException`

→ The **getLocalHost( )** method simply returns the InetAddress object that represents the local host.

`static InetAddress getByName(String hostName)`
`  throws UnknownHostException`

→ The **getByName( )** method returns an InetAddress for a host name passed to it. If these methods are unable to resolve the host name, they throw an UnknownHostException.

`static InetAddress[ ] getAllByName(String hostName)`
`  throws UnknownHostException`

→ The **getAllByName( )** factory method returns an array of InetAddresses that represent all of the addresses that a particular name resolves to. It will also throw an UnknownHostException if it can't resolve the name to at least one address

- InetAddress also includes the factory method getByAddress( ), which takes an IP address and returns an InetAddress object.
- The following example prints the addresses and names of the local machine and two well-known Internet web sites:

```
// Demonstrate InetAddress.
package networking;
import java.net.*;
public class InetAddressTest {
public static void main(String args[]) throws UnknownHostException {
InetAddress Address = InetAddress.getLocalHost();
System.out.println(Address);
Address = InetAddress.getByName("javatpoint.com");
System.out.println(Address);
InetAddress SW[] = InetAddress.getAllByName("www.gmail.com");
    for (InetAddress SW1 : SW) {
        System.out.println(SW1);
    }
}

}
```

```
Output - Networking (run)
run:
LAPTOP-G03MESPL/192.168.1.105
javatpoint.com/104.21.79.8
www.gmail.com/142.250.193.69
BUILD SUCCESSFUL (total time: 1 second)
```

Explain how to pass parameters to applets with an example.

We can supply user-defined parameters to an applet using <PARAM...> tags. Each <PARAM...> tag has a ***name*** attribute such as ***color***, and a ***value*** attribute such as ***red***. Inside the applet code, the applet can refer to that parameter by name to find its value. For example, we can change the colour of the text displayed to red by an applet by using a <PARAM...> tag as follows:

```
<APPLET .....>
<PARAM = color VALUE = "red">
</APPLET>
```

Similarly, we can change the text to be displayed by an applet by supplying new text to the applet through a <PARAM...> tag as shown below:

```
<PARAM NAME = text VALUE = "I love Java">
```

Passing parameters to an applet code using <PARAM> tag is something similar to passing parameters to the **main( )** method using command line arguments. To set up and handle parameters, we need to do two things:

1. Include appropriate <PARAM...> tags in the HTML document.
2. Provide Code in the applet to parse these parameters.

Parameters are passed on an applet when it is loaded. We can define the **init( )** method in the applet to get hold of the parameters defined in the <PARAM> tags. This is done using the **getParameter( )** method, which takes one string argument representing the **name** of the parameter and returns a string containing the value of that parameter.

6) Explain event handling in java.

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc.

The **java.awt.event** package provides many event classes and Listener interfaces for event handling.

Two event handling mechanism

- The 1.0 method of event handling is still supported, but it is not recommended for new programs.
- Also, Many of the methods that support the old 1.0 event model have been deprecated.
- The modern approach is the way that events should be handled by all new programs and thus is the method employed by programs

The Delegation Event Model

- The modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events.
- Its quite simple: **a source generates an event and sends it to one or more listeners.**
- **In this scheme, the listener simply waits until it receives an event. Once an event is received, the listener processes the event and then returns.**
- The advantage of this design is that **the application logic that processes events** is cleanly **separated from the user interface logic that generates those events**.

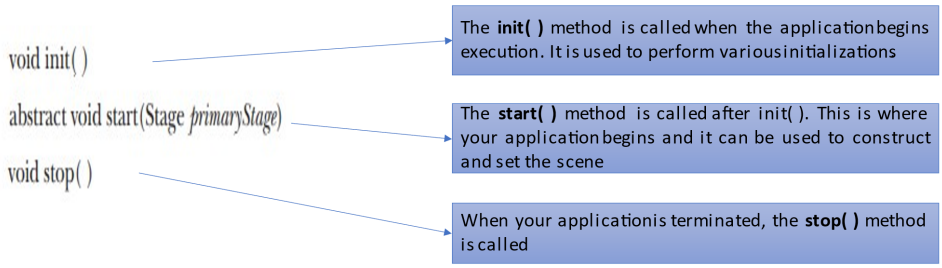
What is an applet? How do applets differ from applications? Write an applet program to display "hello".

- Applets are small java programs that are primarily used for internet computing.
- Can perform arithmetic operations, display graphics, play sounds, accept user input, create animation, and play interactive games.
- There are two types of applet
  Local applet
  Remote applet
- The main difference between Applet and Application is that **the applet is a small java program that can be executed by a Java-compatible web browser** while the application is a standalone program that can directly run on the machine.
- Applets cannot communicate with other servers on the network.
- Applets cannot read from or write to the files in the local computer.
- Applets cannot run any programs from the local computer.

Describe the life cycle of the javaFX Application class.

The Application Class and the Life-cycle Methods

- A JavaFX application must be a subclass of the Application class, which is packaged in javafx.application. Thus, your application class will extend Application.
- The Application class defines three life-cycle methods that your application can override. These are called **init( ), start( ), and stop( )**



7) What is method overloading? Explain with a program.

In Java, it is possible to create methods that have the same name, but different parameter lists and different definitions. This is called *method overloading*. Method overloading is used when objects are required to perform similar tasks but using different input parameters. When we call a method in an object, Java matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute. This process is known as *polymorphism*.

To create an overloaded method, all we have to do is to provide several different method definitions in the class, all with the same name, but with different parameter lists. The difference may either be in the number or type of arguments. That is, each parameter list should be unique. Note that the method's return type does not play any role in this. Here is an example of creating an overloaded method.

```
class Room
{
    float length;
    float breadth;
    Room(float x, float y)              // constructor1
    {
        length = x;
        breadth = y;
    }
    Room(float x)                       // constructor2
    {
        length = breadth = x;
    }
    int area( )
    {
        return (length * breadth);
    }
}
```

Here, we are overloading the constructor method **Room( )**. An object representing a rectangular room will be created as

```
Room room1 = new Room(25.0, 15.0);    // using constructor1
```

On the other hand, if the room is square, then we may create the corresponding object as

```
Room room2 = new Room (20.0);         // using constructor2
```

Note on any three:

Interfaces

- An interface is like a class but includes group of method declaration.
- The interface defines **contains only constants, abstract methods, and final field** and do not specify any code to implement the methods and data fields.
- Responsibility of class to define the code for implementation of these methods
- It is used to **achieve abstraction and multiple inheritance** in java.

interface *InterfaceName*

{

variables declaration;

methods declaration;

}

- Here **interface** is a keyword and *InterfaceName* is any valid java variable.

The break statement

The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement transferring the control to next statement .

The paint() method

The paint( ) method has one **parameter of type Graphics**. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required. where g is an object reference of class Graphic.

TCP/IP sockets

TCP/IP Client Sockets

- TCP/IP sockets are used to implement reliable, bidirectional, persistent, point-to-point, stream-based connections between hosts on the Internet. A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet.
- There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients. The ServerSocket class is designed to be a "listener," which waits for clients to connect before doing anything. Thus, ServerSocket is for servers. The Socket class is for clients. It is designed to connect to server sockets and initiate protocol exchanges.
- The creation of a Socket object implicitly establishes a connection between the client and server. There are no methods or constructors that explicitly expose the details of establishing that connection. Here are two constructors used to create client sockets:

Socket(String *hostName*, int *port*) throws UnknownHostException, IOException	Creates a socket connected to the named host and port.
Socket(InetAddress *ipAddress*, int *port*) throws IOException	Creates a socket using a preexisting **InetAddress** object and a port.

8) Explain switch statement with a java program

We have seen that when one of the many alternatives is to be selected, we can design a program using **if** statements to control the selection. However, the complexity of such a program increases dramatically when the number of alternatives increases. The program becomes difficult to read and follow. At times, it may confuse even the designer of the program. Fortunately, Java has a built-in multiway decision statement known as **switch**. The **switch** statement tests the value of a given variable (or expression) against a list of **case** values and when a match is found, a block of statements associated with that **case** is executed. The general form of the **switch** statement is as shown below:

```
switch (expression)
{
        case value-1:
                    block-1
                    break;
        case value-2:
                    block-2
                    break;
        ......
        ......
        default:
                    default-block
                    break;
}
statement-x;
```

The *expression* is an integer expression or characters. *value-1*, *value-2* .... are constants or constant expressions (evaluable to an integral constant) and are known as *case labels*. Each of these values should be unique within a **switch** statement. *block-1*, *block-2* .... are statement lists and may contain zero or more statements. There is no need to put braces around these blocks but it is important to note that case labels end with a colon (:).

Briefly explain java stream classes

- The java.io package contains a large number of stream classes that provide capabilities for processing all types of data.
- These classes may be recognized into two groups based on the data type on which they operate.

1) Byte stream classes that provide support for handling I/O operations on bytes.

2)Character stream classes that provide support for managing I/O operations on characters

Byte stream classes

- Byte stream classes have been designed to provide functional features for creating and manipulating streams and files for reading and writing bytes.
- Since the streams are unidirectional, they can transmit bytes in only one direction and therefore java provides two kinds of byte stream classes: **input stream classes and output stream classes.**

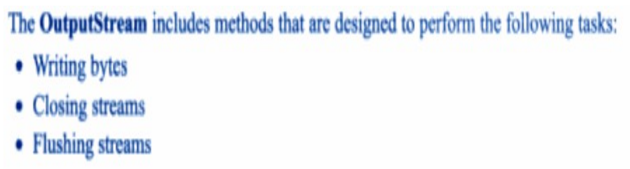
Input Stream Classes

- Input stream classes that are used to read 8 bit bytes include a superclass known as InputStream and a number of subclasses for supporting various input-related functions.

Output stream classes

Output stream classes

- Like InputStream, the OutputStream is an abstract class and therefore we cannot instantiate it.
- The several subclasses of the outputStream can be used for performing the output operations.



Write a note on any three:

Bytecode

- The Java compiler produces an intermediate code known as bytecode for a machine that does not exist. This machine is called the Java Virtual Machine and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer. **Bytecode** is also referred to as ***virtual machine code***.
- The ***virtual machine code*** is not machine specific. The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine as shown in the figure below.
- The Java object framework (Java API) acts as the intermediary between the user programs and the virtual machine which in turn acts as the intermediary between the operating system and the Java object framework.

Multithreading

- **Multithreading in java** is a process of executing multiple threads simultaneously.(multitasking)

- A thread is a lightweight sub-process, the smallest unit of processing
- A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.
- The are two distinct types of multitasking: process-based and thread-based

1) Process-based Multitasking (Multiprocessing)

- Process-based multitasking is the feature that allows your computer to run two or more programs concurrently.
- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists.

2) Thread-based Multitasking (Multithreading)

- In a thread-based multitasking means that a single program can perform two or more tasks simultaneously.
- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

Inner class

Java inner class or nested class is a class that is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.

Additionally, it can access all the members of the outer class, including private data members and methods.

PreparedStatement interface

A `PreparedStatement` enables you to create compiled SQL statements that execute more efficiently than `Statements`. `PreparedStatements` can also specify parameters, making them more flexible than `Statements`—you can execute the same query repeatedly with different parameter values. For example, in the `books` database, you might want to locate all book titles for an author with a specific last and first name, and you might want to execute that query for several authors. With a `PreparedStatement`, that query is defined as follows:

```
PreparedStatement authorBooks = connection.prepareStatement(
   "SELECT LastName, FirstName, Title " +
   "FROM Authors INNER JOIN AuthorISBN " +
      "ON Authors.AuthorID=AuthorISBN.AuthorID " +
   "INNER JOIN Titles " +
      "ON AuthorISBN.ISBN=Titles.ISBN " +
   "WHERE LastName = ? AND FirstName = ?");
```