SYLLABUS

(Academic Session (2015-16)

JAVA PROGRAMMING [ETCS-307]

UNIT I

Overview and characteristics of Java, Java program Compilation and Execution Process Organization of the Java Virtual Machine, JVM as an interpreter and emulator, Instruction Set, class File Format, Verification, Class Area, Java Stack, Heap, Garbage Collection. Security Promises of the JVM, Security Architecture and Security Policy. Class loaders and security aspects, sandbox model.

[T1,R2] [No. of Hrs.: 11]

UNIT II

Java Fundamentals, Data Types & Literals Variables, Wrapper Classes, Arrays, Arithmetic Operators, Logical Operators, Control of Flow, Classes and Instances, Class Member Modifiers Anonymous Inner Class Interfaces and Abstract Classes, inheritance, throw and throws clauses, user defined Exceptions, The String Buffer Class, tokenizer, applets, Life cycle of applet and Security concerns.

[T1,T2][No. of Hrs.: 12]

UNIT III

Threads: Creating Threads, Thread Priority, Blocked States, Extending Thread Class, Runnable Interface, Starting Threads, Thread Synchronization, Synchronize Threads, Sync Code Block, Overriding Synced Methods, Thread Communication, wait, notify and notify all. AWT Components, Component Class, Container Class, Layout Manager Interface Default Layouts, Insets and Dimensions, Border Layout, Flow Layout, Grid Layout, Card Layout Grid Bag Layout AWT Events, Event Models, Listeners, Class Listener, Adapters, Action Event Methods Focus Event Key Event,Mouse Events, Window Event.

[T2][No. of Hrs.: 11]

UNIT IV

Input/Output Stream, Stream Filters, Buffered Streams, Data input and Output Stream, Print Stream Random Access File, JDBC (Database connectivity with MS-Access, Oracle, MS-SQL Server), Object serialization, Sockets, development of client Server applications, design of multithreaded server. Remote Method invocation, Java Native interfaces, Development of a JNI based application. Collection API Interfaces, Vector, stack, Hashtable classes, enumerations, set, List, Map, Iterators.

[T1][R1][No. of Hrs.: 10]

MODEL TEST PAPER I
FIRST TERM EXAMINATION
FIFTH SEMESTER (B.TECH)
JAVA PROGRAMMING [ETCS-307]

Time : 1 hrs. **M.M. : 30**

Note: *Attempt any three questions in total including Question No. 1 which is compulsory.*

Q.1.(a) Explain features of java programming language. **(5)**

Ans. Features of Java Programming Language

Simple :

- Java is Easy to write and more readable and eye catching.
- Java has a concise, cohesive set of features that makes it easy to learn and use.
- Most of the concepts are drew from C++ thus making Java learning simpler.

Secure :

- Java program cannot harm other system thus making it secure.
- Java provides a secure means of creating Internet applications.
- Java provides secure way to access web applications.

Portable :

- Java programs can execute in any environment for which there is a Java run-time system.(JVM)
- Java programs can be run on any platform (Linux,Window,Mac)
- Java programs can be transferred over world wide web (e.g applets)

Object-oriented :

- Java programming is object-oriented programming language.
- Like C++ java provides most of the object oriented features.
- Java is pure OOP. Language. (while C++ is semi object oriented)

Robust :

- Java encourages error-free programming by being strictly typed and performing run-time checks.

Multithreaded :

- Java provides integrated support for multithreaded programming.

Architecture-neutral :

- Java is not tied to a specific machine or operating system architecture.
- Machine Independent i.e Java is independent of hardware .

Interpreted :

- Java supports cross-platform code through the use of Java bytecode.
- Bytecode can be interpreted on any platform by JVM.

High performance :

- Bytecodes are highly optimized.
- JVM can executed them much faster .

Distributed :

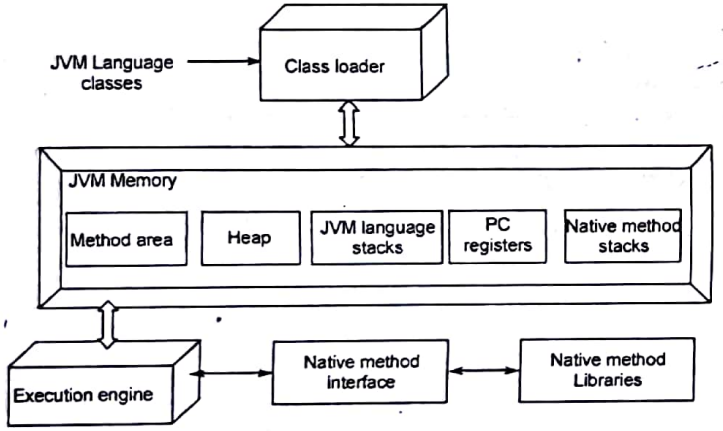
- Java was designed with the distributed environment.
- Java can be transmit run over internet.

Dynamic :

• Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time.

Q.1.(b) Explain JVM. **(5)**

Ans. A Java virtual machine (JVM) is an abstract computing machine. There are three notions of the JVM: specification, implementation, and instance. The specification is a book that formally describes what is required of a JVM implementation. Having a single specification ensures all implementations are interoperable. A JVM implementation is a computer program that meets the requirements of the JVM specification in a compliant and preferably performant manner. An instance of the JVM is a process that executes a computer program compiled into Java bytecode.



Q.2. (a) write short note on garbage collection in java. **(4)**

Ans. Java Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes

public static void gc()||

Q.2. (b) Explain class file format in java. **(6)**

Ans. Java class file format: Compiled Java is typically distributed in a very compact format called Class files. It is the assembler code for a virtual stack-oriented

Java-friendly CPU (Central Processing Unit) chip. Such chips do exist, but most of the time they are simulated with interpreters, JIT (Just In Time), Hotspots or AOT (Ahead Of Time) compilation on the target machines which have quite different architectures.

Class files are usually compressed and bundled into jar files.

Each class file is marked with a class file format version, which lets you know which JDK (Java Development Kit) it was compiled for. At offset 6 in the class file is a big-endian short that gives the major version number. You can display for the classes is in a jar with JarCheck.

A Java class file is a file (with the .class filename extension) containing Java bytecode that can be executed on the Java Virtual Machine (JVM). A Java class file is produced by a Java compiler from Java programming language source files (.java files) containing Java classes. If a source file has more than one class, each class is compiled into a separate class file.

JVMs are available for many platforms, and a class file compiled on one platform will execute on a JVM of another platform. This makes Java platform-independent.

Q.3. (a) Explain java stack. **(3)**

Ans. Stack is a subclass of Vector that implements a standard last-in, first-out stack.

Stack only defines the default constructor, which creates an empty stack. Stack includes all the methods defined by Vector, and adds several of its own.

Stack defines following methods:

Sl.No.	Methods with Description
1	**boolean empty()** Tests if this stack is empty. Returns true if the stack is empty, and returns false if the stack contains elements.
2	**Object peek()** Returns the element on the top of the stack, but does not remove it.
3	**Object pop()** Returns the element on the top of the stack, removing it in the process.
4	**Object push(Object element)** Pushes element onto the stack. element is also returned.
5	**int search(Object element)** Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, -1 is returned.

Q.3.(b) Explain classes and instances in java. **(7)**

Ans. Class and instance

```
import java.util.Date;
class DateApp {
 public static void main(String args[]) {
 Date today = new Date();
 System.out.println(today);
 }
}
```

The last line of the main() method uses the System class from the java.lang package to display the current date and time. First, let's break down the line of code that invokes the println() method, then look at the details of the argument passed to it.

Class Methods and Variables: Let's take a look at the first segment of the statement:

System.out.println(today);

The construct—System.out—is the full name for the out variable in the System class. Notice that the application never instantiated the System class and that out is referred to directly from the class name. This is because out is a *class variable*—a variable associated with the class rather than with an instance of the class. You can also associate methods with a class—*class methods.*

To refer to class variables and methods, you join the class's name and the name of the class method or class variable together with a period ('.').

Instance Methods and Variables: Methods and variables that are not class methods or class variables are known as *instance methods* and *instance variables*. To refer to instance methods and variables, you must reference the methods and variables from an instance of the class.

While System's out variable *is* a class variable, it *refers* to an instance of the PrintStream class (a member of the java.io package that implements the The Standard Output Stream

When the System class is loaded into the application, the class instantiates PrintStream and assigns the new PrintStream object to the out class variable. Now, you have an instance of a class so you can call one of its instance methods.

System.out.println()

As you see, you refer to an object's instance methods and variables similar to the way you refer class methods and variables. You join an object reference (System.out) and the name of the instance method or instance variable (println()) together with a period ('.').

The Java compiler allows you to cascade these references to class and instance methods and variables together and resulting in constructs like the one that appears in the sample program:

System.out.println()

Sum it Up: Class variables and class methods are associated with a class and occur once per class. Instance methods and variables occur once per instance of a class.

Q.4. (a) Show control of flow in java. **(5)**

Ans. Both C++ and Java support several different kinds of statements designed to alter or control the logical flow of the program, although in some cases, the behavior of those statements differs between Java and C++.

The ability to alter the logical flow of the program is often referred to as *Flow of Control.*

Statements that support flow of control: The following table lists the statements supported by Java (and by C++ with some exceptions) for controlling the logical flow of the program.

Statement	Type
if-else	selection
switch-case	selection
for	loop
while	looop
do-while	loop
try-catch-finally	exception handling
throw	exception handling
break	miscelanneous
continue	miscellaneous
label:	miscellaneous
return	misellaneous
goto	reserved by Java but not supported

Statement Type if-else selectionswitch-case selectionfor loopwhile loopdo-while looptry-catch-finally exception handlingthrow exception handlingbreak miscellaneouscontinue miscellaneouslabel: miscellaneousreturn miscellaneous*goto* reserved by Java but not supported

Q.4. (b) Explain security architecture in java. **(5)**

Ans. Java's security model is one of the language's key architectural features that makes it an appropriate technology for networked environments. Security is important because networks provide a potential avenue of attack to any computer hooked to them. This concern becomes especially strong in an environment in which software is downloaded across the network and executed locally, as is done with Java applets, for example. Because the class files for an applet are automatically downloaded when a user goes to the containing Web page in a browser, it is likely that a user will encounter applets from untrusted sources. Without any security, this would be a convenient way to spread viruses. Thus, Java's security mechanisms help make Java suitable for networks because they establish a needed trust in the safety of network-mobile code.

Java's security model is focused on protecting users from hostile programs downloaded from untrusted sources across a network. To accomplish this goal, Java provides a customizable "sandbox" in which Java programs run. A Java program must play only inside its sandbox. It can do anything within the boundaries of its sandbox, but it can't take any action outside those boundaries. The sandbox for untrusted Java applets, for example, prohibits many activities, including:

- Reading or writing to the local disk
- Making a network connection to any host, except the host from which the applet came
- Creating a new process
- Loading a new dynamic library and directly calling a native method

By making it impossible for downloaded code to perform certain actions, Java's security model protects the user from the threat of hostile code.

MODEL TEST PAPER I
SECOND TERM EXAMINATION
FIFTH SEMESTER (B.TECH)
JAVA PROGRAMMING [ETCS-307]

Time : 1 hrs. **M.M. : 30**

Note: *Attempt any three questions in total including Question No. 1 which is compulsory.*

Q.1. (a) Define abstract class. **(4)**

Ans. A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

abstract class A{}

Q.1.(b) show inheritance with example. **(6)**

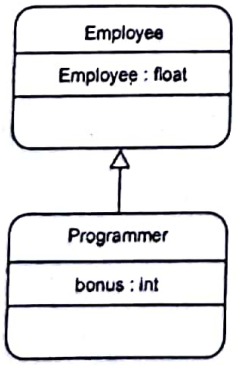
Ans. Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

class Subclass-name **extends** Superclass-name

```
{
//methods and fields
}
```



Q.2. (a) Define life cycle of applet. **(4)**

Ans. Following are the methods.

1. init() method
2. start() method
3. paint() method
4. stop() method
5. destroy() method

These methods are known as **life cycle methods**. These methods are defined in **java.applet.Applet** class except **paint()** method. The paint() method is defined in **java.awt.Component** class, an indirect super class of Applet.

The life cycle methods are called as **callback methods** as they are called implicitly by the browser for the smooth execution of the applet. Browser should provide an environment known as **container** for the execution of the applet. Following are the responsibilities of the browser.

1. It should call the callback methods at appropriate times for the smooth execution of the applet.

2. It is responsible to maintain the life cycle of the applet.

3. It should have the capability to communicate between applets, applet to JavaScript and HTML, applet to browser etc.

Q.2. (b) What is AWT? and name some packages under AWT. **(6)**

Ans. Abstract Window Toolkit (AWT) is a set of application program interfaces (API s) used by Java programmers to create graphical user interface (GUI) objects, such as buttons, scroll bars, and windows. AWT is part of the Java Foundation Classes (JFC) from Sun Microsystems, the company that originated Java. The JFC are a comprehensive set of GUI class libraries that make it easier to develop the user interface part of an application program.

A more recent set of GUI interfaces called Swing extends the AWT so that the programmer can create generalized GUI objects that are independent of a specific operating system's windowing system .

The AWT provides two levels of APIs:

• A general interface between Java and the native system, used for windowing, events, and layout managers. This API is at the core of Java GUI programming and is also used by Swing and Java 2D. It contains:

— The interface between the native windowing system and the Java application;

— The core of the GUI event subsystem;

— Several layout managers;

— The interface to input devices such as mouse and keyboard; and

— A java.awt.datatransfer package for use with the Clipboard and Drag and Drop.

• A basic set of GUI widgets such as buttons, text boxes, and menus. It also provides the AWT Native Interface, which enables rendering libraries compiled to native code to draw directly to an AWT Canvas object drawing surface.

AWT also makes some higher level functionality available to applications, such as:

• Access to the system tray on supporting systems; and

• The ability to launch some desktop applications such as web browsers and email clients from a Java application.

Neither AWT nor Swing are inherently thread safe. Therefore, code that updates the GUI or processes events should execute on the Event dispatching thread. Failure to do so may result in a deadlock or race condition. To address this problem, a utility class called SwingWorker allows applications to perform time-consuming tasks following user-interaction events in the event dispatching thread.

Q.3. (a) Explain Tokenizer. **(3)**

Ans. The **java.util.StringTokenizer** class allows you to break a string into tokens. It is simple way to break string.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class.

There are 3 constructors defined in the String Tokenizer class.

Constructor	Description
String Tokenizer(String str)	creates String Tokenizer with specified string.
String Tokenizer (String str, String delim)	creates String Tokenizer with specified string and delimeter.
StringTokenizer(String str, String delim, boolean returnValue)	creates StringTokenizer with specified string, delimeter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

Q.3. (b) Explain various mouse events in java. **(7)**

Ans. This event indicates a mouse action occurred in a component. This low-level event is generated by a component object for Mouse Events and Mouse motion events.

- a mouse button is pressed
- a mouse button is released
- a mouse button is clicked (pressed and released)
- a mouse cursor enters the unobscured part of component's geometry
- a mouse cursor exits the unobscured part of component's geometry
- a mouse is moved
- the mouse is dragged

Class declaration: Following is the declaration for **java.awt.event.MouseEvent** class:

public class MouseEvent
extends InputEvent

Field

Following are the fields for **java.awt.event.MouseEvent** class:

- **static int BUTTON1** —Indicates mouse button #1; used by getButton()
- **static int BUTTON2** —Indicates mouse button #2; used by getButton()
- **static int BUTTON3** —Indicates mouse button #3; used by getButton()
- **static int MOUSE_CLICKED** —The "mouse clicked" event
- **static int MOUSE_DRAGGED** —The "mouse dragged" event
- **static int MOUSE_ENTERED** —The "mouse entered" event
- **static int MOUSE_EXITED** —The "mouse exited" event

- **static int MOUSE_FIRST** —The first number in the range of ids used for mouse events
- **static int MOUSE_LAST** — The last number in the range of ids used for mouse events
- **static int MOUSE_MOVED** —The "mouse moved" event
- **static int MOUSE_PRESSED** — The "mouse pressed" event
- **static int MOUSE_RELEASED** —The "mouse released" event
- **static int MOUSE_WHEEL** —The "mouse wheel" event
- **static int NOBUTTON** —Indicates no mouse buttons; used by getButton()
- **static int VK_WINDOWS** —Constant for the Microsoft Windows "Windows" key.

Q.4. (a) Define Adapter classes in java. **(3)**

Ans. An adapter class provides the default implementation of all methods in an event listener interface. Adapter classes are very useful when you want to process only few of the events that are handled by a particular event listener interface.

```
package com.journaldev.design.adapter;
public class Volt {
   private int volts;
   public Volt(int v){
      this.volts=v;
   }
   public int getVolts() {
   return volts;
   }
   public void setVolts(int volts) {
      this.volts = volts;
   }
}
```

Q.4. (b) Define Border layout and grid layout in java. **(7)**

Ans.
```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Font;
import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;
public class CombinedLayoutManager extends JFrame {
public static void main(String[] args) {
   CombinedLayoutManager e = new CombinedLayoutManager();
   e.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    e.setSize(400, 300);
     e.setVisible(true);
     }
   public CombinedLayoutManager() {
     Container pane = getContentPane();
    pane.setLayout(new BorderLayout());
    pane.add(getHeader(), BorderLayout.NORTH);
    pane.add(getTextArea(), BorderLayout.CENTER);
    pane.add(getButtonPanel(), BorderLayout.SOUTH);
     }
  protected JComponent getHeader() {
    JLabel label = new JLabel("Embedded Layout Manager Test", JLabel.CENTER);
     label.setFont(new Font("Courier", Font.BOLD, 24));
    return label;
     }
  protected JComponent getTextArea() {
    return new JTextArea(10, 10);
}
  protected JComponent getButtonPanel() {
    JPanel inner = new JPanel();
    inner.setLayout(new GridLayout(1, 2, 10, 0));
    inner.add(new JButton("Ok"));
     inner.add(new JButton("Cancel"));
     return inner;}}
```

MODEL TEST PAPER I
END TERM EXAMINATION
FIFTH SEMESTER (B.TECH)
JAVA PROGRAMMING [ETCS-307]

Time : 3 hrs. **M.M. : 75**

Note: *Question 1 is compulsory and attempt any 5 questions from the rest.*

Q.1. (a) Define API interfaces with example program. **(10)**

There are a number of situations in software engineering when it is important for disparate groups of programmers to agree to a "contract" that spells out how their software interacts. Each group should be able to write their code without any knowledge of how the other group's code is written. Generally speaking, *interfaces* are such contracts.

For example, imagine a futuristic society where computer-controlled robotic cars transport passengers through city streets without a human operator. Automobile manufacturers write software (Java, of course) that operates the automobile—stop, start, accelerate, turn left, and so forth. Another industrial group, electronic guidance instrument manufacturers, make computer systems that receive GPS (Global Positioning System) position data and wireless transmission of traffic conditions and use that information to drive the car.

The auto manufacturers must publish an industry-standard interface that spells out in detail what methods can be invoked to make the car move (any car, from any manufacturer). The guidance manufacturers can then write software that invokes the methods described in the interface to command the car. Neither industrial group needs to know *how* the other group's software is implemented

In the Java programming language, an *interface* is a reference type, similar to a class, that can contain *only* constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be *implemented* by classes or *extended* by other interfaces. Extension is discussed later in this lesson.

Defining an interface is similar to creating a new class:

```
public interface OperateCar {
      // constant declarations, if any
      // method signatures

     // An enum with values RIGHT, LEFT
     int turn(Direction direction,
          double radius,
          double startSpeed,
          double endSpeed);
  int changeLanes(Direction direction,
          double startSpeed,
          double endSpeed);
  int signalTurn(Direction direction,
          boolean signalOn);
```

```
    int getRadarFront(double distanceToCar,
        double speedOfCar);
    int getRadarRear(double distanceToCar,
        double speedOfCar);
        .....
        // more method signatures
}
```

method signatures have no braces and are terminated with a semicolon.

To use an interface, you write a class that *implements* the interface. When a instantiable class implements an interface, it provides a method body for each of th methods declared in the interface. For example,

```
public class OperateBMW760i implements OperateCar {
    // the OperateCar method signatures, with implementation —
    // for example:
    int signalTurn(Direction direction, boolean signalOn) {
        // code to turn BMW's LEFT turn indicator lights on
        // code to turn BMW's LEFT turn indicator lights off
        // code to turn BMW's RIGHT turn indicator lights on
        // code to turn BMW's RIGHT turn indicator lights off
    }
    // other members, as needed — for example, helper classes not
    // visible to clients of the interface
}
```

Q.1. (b) Explain Remote method invocation in java. **(15)**

Ans. The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

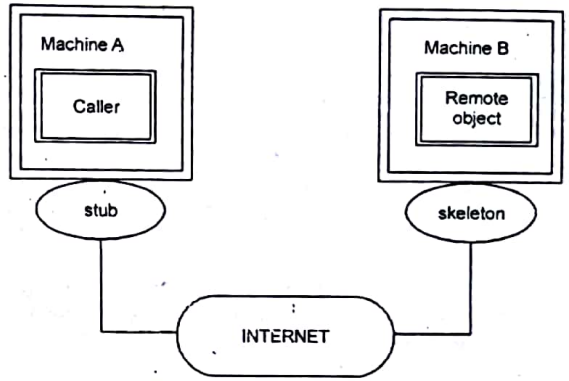
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming

request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

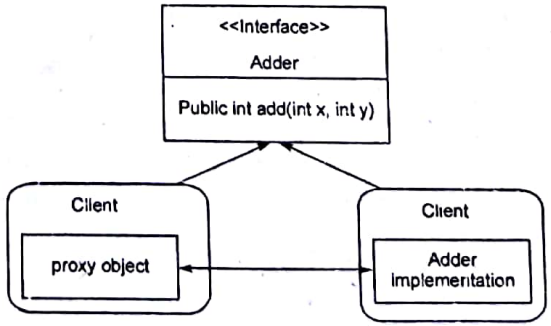


Steps to write the RMI program

These are 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

RMI Example: In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



Q.2. What is Grid bag layout in java. **(10)**

Ans. public class **GridBagLayout**

extends Object

implements LayoutManager2, Serializable

The GridBagLayout class is a flexible layout manager that aligns components vertically, horizontally or along their baseline without requiring that the components be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its *display area*.

Each component managed by a GridBagLayout is associated with an instance of GridBagConstraints. The constraints object specifies where a component's display area should be located on the grid and how the component should be positioned within its display area. In addition to its constraints object, the GridBagLayout also considers each component's minimum and preferred sizes in order to determine a component's size.

The overall orientation of the grid depends on the container's ComponentOrientation property. For horizontal left-to-right orientations, grid coordinate (0,0) is in the upper left corner of the container with x increasing to the right and y increasing downward. For horizontal right-to-left orientations, grid coordinate (0,0) is in the upper right corner of the container with x increasing to the left and y increasing downward.

To use a grid bag layout effectively, you must customize one or more of the GridBagConstraints objects that are associated with its components. You customize a GridBagConstraints object by setting one or more of its instance variables:

GridBagConstraints.gridx, GridBagConstraints.gridy

Specifies the cell containing the leading corner of the component's display area, where the cell at the origin of the grid has address gridx = 0, gridy = 0. For horizontal left-to-right layout, a component's leading corner is its upper left. For horizontal right-to-left layout, a component's leading corner is its upper right. Use GridBagConstraints. RELATIVE (the default value) to specify that the component be placed immediately following (along the x axis for gridx or the y axis for gridy) the component that was added to the container just before this component was added.

GridBagConstraints.gridwidth, GridBagConstraints.gridheight

Specifies the number of cells in a row (for gridwidth) or column (for gridheight) in the component's display area. The default value is 1. Use GridBagConstraints. REMAINDER to specify that the component's display area will be from gridx to the last cell in the row (for gridwidth) or from gridy to the last cell in the column (for gridheight). Use GridBagConstraints.RELATIVE to specify that the component's display area will be from gridx to the next to the last cell in its row (for gridwidth or from gridy to the next to the last cell in its column (for gridheight).

GridBagConstraints.fill

Used when the component's display area is larger than the component's requested size to determine whether (and how) to resize the component. Possible values are GridBagConstraints.NONE (the default), GridBagConstraints.HORIZONTAL (make the component wide enough to fill its display area horizontally, but don't change its height), GridBagConstraints.VERTICAL (make the component tall enough to fill its display area vertically, but don't change its width), and GridBagConstraints.BOTH (make the component fill its display area entirely).

GridBagConstraints.ipadx, GridBagConstraints.ipady

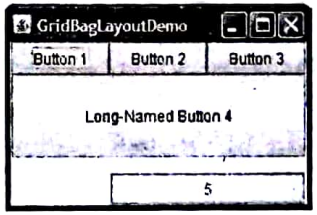
Specifies the component's internal padding within the layout, how much to add to the minimum size of the component. The width of the component will be at least its minimum width plus ipadx pixels. Similarly, the height of the component will be at least the minimum height plus ipady pixels.

GridBagConstraints.insets

Specifies the component's external padding, the minimum amount of space between the component and the edges of its display area.

GridBagConstraints.anchor

Specifies where the component should be positioned in its display area. There are three kinds of possible values: absolute, orientation-relative, and baseline-relative Orientation relative values are interpreted relative to the container's ComponentOrientation property while absolute values are not.



Q.3. Explain various Buffered Streams in java. **(10)**

In case of 'byte' and 'character' streams every byte or piece of data that's being read or write requires an direct support from underlying OS because of not having an intermediate buffer included. This makes a extensive use of memory and resources. On the other hand Buffered streams works as a wrapper to hold unbuffered streams and make it possible to store bunch of data or bytes in buffers (memory). The underlying OS resource are needed only when the buffer is full or empty and not on every instance of read or write.

The native input API is called only when the buffer is empty. Similarly, buffered output streams write data to a buffer, and the native output API is called only when the buffer is full. Unbuffered streams can be converted to buffered streams by wrapping them in a constructor of Buffered Stream class.

Example program converting 'byte Stream' to 'Buffered byte Stream'

```
package com.beingjavaguys.core;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
/**
 * @author Nagesh Chauhan
 *
 */
public class BufferedIOStream {
```

```
public static void main(String args[]) throws IOException {
    BufferedInputStream bis = null;
    BufferedOutputStream bos = null;
        try {
    bis = new BufferedInputStream(new FileInputStream(
        "files/source.txt"));
    bos = new BufferedOutputStream(new FileOutputStream(
        "files/destination.txt"));
    int data;
    while ((data = bis.read()) != -1) {
    System.out.println(data);
    bos.write(data);
    }
    } catch (FileNotFoundException e) {
    e.printStackTrace();
    } finally {
    if (bis != null)
     bis.close();
if (bos != null)
    bos.close();
    }
    }
    }
```

Q.4. Define Exception handling in java with example program. **(10)**

Ans. Exceptions can be handled by using 'try-catch' block. Try block contains the code which is under observation for exceptions. The catch block contains the remedy for the exception. If any exception occurs in the try block then the control jumps to catch block.

```
package com.myjava.exceptions;
public class MyExceptionHandle {
    public static void main(String a[]){
        try{
            for(int i=5;i>=0;i—){
                System.out.println(10/i);
            }
        } catch(Exception ex){
            System.out.println("Exception Message: "+ex.getMessage());
            ex.printStackTrace();
        }
        System.out.println("After for loop...");
        }
    }
```



```
2
3
5
10
Exception Message: / by zero
 java.lang.ArithmeticException:
/ by zero at com.myjava.exceptions.MyExceptionHandle.
main(MyExceptionHandle.java:12)
 After for loop...
```

Q.5. Define the following: **(10)**

Q.5. (*a*) Heap

Ans. Refer to Q.3(*a*) of First Term Examination of Model Paper-II.

Q.5. (*b*) List

Ans. A List is an ordered Collection (sometimes called a *sequence*). Lists may contain duplicate elements. In addition to the operations inherited from Collection, the List interface includes operations for the following:

Positional access — manipulates elements based on their numerical position in the list. This includes methods such as get, set, add, addAll, and remove.

Search — searches for a specified object in the list and returns its numerical position. Search methods include indexOf and lastIndexOf.

Iteration — extends Iterator semantics to take advantage of the list's sequential nature. The listIterator methods provide this behavior.

Range-view — The sublist method performs arbitrary *range operations* on the list.

The Java platform contains two general-purpose List implementations. ArrayList, which is usually the better-performing implementation, and LinkedList which offers better performance under certain circumstances.

```
List<Type> list3 = new ArrayList<Type>(list1);
list3.addAll(list2);
List<String> list = people.stream()
.map(Person::getName)
.collect(Collectors.toList());
```

Q.5. (*c*) Iterators

Ans. When the for-each loop is not available, and an explicit Iterator is needed, then iteration over a collection may be done with a while loop or a for loop.

The two styles have different advantages:

- the while loop is considerably more legible
- the for loop minimizes the scope of the Iterator to the loop itself

```
import java.util.*;
public final class LoopStyles {
    public static void main(String... aArguments) {
        List<String> flavours = new ArrayList<>();
        flavours.add("chocolate");
        flavours.add("strawberry");
        flavours.add("vanilla");
        useWhileLoop(flavours);
        useForLoop(flavours);
    }
```

```
    private static void useWhileLoop(Collection<String> aFlavours) {
        Iterator<String> flavoursIter = aFlavours.iterator();
        while (flavoursIter.hasNext()){
            System.out.println(flavoursIter.next());
        }
    }
    /**
    * Note that this for-loop does not use an integer index.
    */
    private static void useForLoop(Collection<String> aFlavours) {
    for (Iterator<String> flavoursIter = aFlavours.iterator(); flavoursIter.has
Next();){System.out.println(flavoursIter.next());
    }
    }
}
```

Q.6. Explain java native interfaces with example. **(10)**

Ans.

```
public class HelloJNI {
   static {
   System.loadLibrary("hello"); // Load native library at runtime
      // hello.dll (Windows) or libhello.so (Unixes)
   }
   // Declare a native method sayHello() that receives nothing and returns void
   private native void sayHello();
   // Test Driver
   public static void main(String[] args) {
      new HelloJNI().sayHello(); // invoke the native method
      }
}
```

The static initializer invokes System.loadLibrary() to load the native library "Hello" (which contains the native method sayHello()) during the class loading. It will be mapped to "hello.dll" in Windows; or "libhello.so" in Unixes. This library shall be included in Java's library path (kept in Java system variable java.library.path); otherwise, the program will throw a UnsatisfiedLinkError. You could include the library into Java Library's path via VM argument -Djava.library.path=*path_to_lib*.

Next, we declare the method sayHello() as a native instance method, via keyword native, which denotes that this method is implemented in another language. A native method does not contain a body. The sayHello() is contained in the native library loaded.

The main() method allocate an instance of HelloJNI and invoke the native method sayHello().

Compile the "HelloJNI.java" into "HelloJNI.class".

native functions are implemented in separate .c or .cpp files. (C++ provides a slightly simpler interface with JNI.) When the JVM invokes the function, it passes a JNIEnv pointer, a jobject pointer, and any Java arguments declared by the Java method. A JNI function may look like this:

```
JNIEXPORT void JNICALL Java_ClassName_MethodName
(JNIEnv *env, jobject obj)
{
/*Implement Native Method Here*/
}
```


The *env* pointer is a structure that contains the interface to the JVM. It includes all of the functions necessary to interact with the JVM and to work with Java objects. Example JNI functions are converting native arrays to/from Java arrays, converting native strings to/from Java strings, instantiating objects, throwing exceptions, etc. Basically, anything that Java code can do can be done using JNIEnv, albeit with considerably less ease.

Q.7. How multithreaded server is designed in java. **(10)**

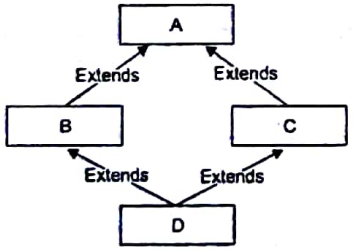
```
package servers;
import java.net.ServerSocket;
import java.net.Socket;
import java.io.IOException;
public class MultiThreadedServer implements Runnable{
    protected int serverPort = 8080;
    protected ServerSocket serverSocket = null;
    protected boolean isStopped = false;
    protected Thread runningThread= null;
public MultiThreadedServer(int port){
    this.serverPort = port;
}
public void run(){
    synchronized(this){
        this.runningThread = Thread.currentThread();
}
openServerSocket();
while(! isStopped()){
    Socket clientSocket = null;
    try {
        clientSocket = this.serverSocket.accept();
    } catch (IOException e) {
        if(isStopped()) {
            System.out.println("Server Stopped.") ;
            return;
        }
        throw new RuntimeException(
            "Error accepting client connection", e);
    }
    new Thread(
    new WorkerRunnable(
        clientSocket, "Multithreaded Server")
    ).start();
    }
    System.out.println("Server Stopped.") ;
    }
private synchronized boolean isStopped() {
    return this.isStopped;
}
public synchronized void stop(){
    this.isStopped = true;
    try {
        this.serverSocket.close();
    } catch (IOException e) {
```

```
            throw new RuntimeException("Error closing server", e);
        }
    }
    private void openServerSocket() {
        try {
            this.serverSocket = new ServerSocket(this.serverPort);
        } catch (IOException e) {
            throw new RuntimeException("Cannot open port 8080", e);
        }
    }
}
```

Q.8. Show multiple inheritance in java. **(10)**

Ans. Multiple inheritance is the ability of a single class to inherit from multiple classes. Java does ***not*** have this capability.

The designers of Java considered multiple inheritance to be too complex, and not in line with the goal of keeping Java simple. One specific problem that Java avoids by not having multiple inheritance is called the diamond problem.



What a Java class does have is the ability to implement multiple interfaces – which is considered a reasonable substitute for multiple inheritance, but without the added complexity.

```
interface X
{
    public void myMethod();
}
interface Y
{
    public void myMethod();
}
class Demo implements X, Y
{
    public void myMethod()
    {
        System.out.println(" Multiple inheritance example using interfaces");
    }
}
```

A class can implement any number of interfaces. In this case there is no ambiguity even though both the interfaces are having same method because methods in an interface are always abstract by default, which doesn't let them to give their implementation (or method definition) in interface itself.

MODEL TEST PAPER II
FIRST TERM EXAMINATION
FIFTH SEMESTER (B.TECH)
JAVA PROGRAMMING [ETCS-307]

Time : 1 hrs. **M.M. : 30**

Note: *Attempt any three questions in total including Question No. 1 which is compulsory.*

Q.1. (a) Compilation and execution process in java. **(5)**

Ans. A Java application program is made of one or more classes and zero or more interfaces. One of the class must have the main() method as the execution starting point of the program. There are 3 steps involved in creating the classes and interfaces of a Java program, and running the program:

- **Creating Classes and Interfaces:** Creating a text file for each class or interface that contains the definition of the class or interface written in Java language statements. Such a file is called source code file, which must have a file name identical to the name of the class or interface defined in the file, and an extension of ".java".
- **Compiling Classes and Interfaces:** Converting a source code file into a bytecode file that contains the same definition of the class or interface written Java Virtual Machine (JVM) instructions. A bytecode file must have a file name identical to the name of the class or interface defined in the file, and an extension of ".class".
- **Executing a Java Program:** Loading bytecode files into JVM, and executing its JVM instructions starting from the entry point of the specified class.

Q.1. (b) JVM acts as emulator, interpreter or both. **(5)**

Ans. Java typically interpret instead of compile? The main advantage of compilation is that you end up with raw machine language code that can be efficiently executed on your machine. However, it can only be executed on one type of machine architecture (Intel Pentium, PowerPC). A primary advantage of a compiling to an intermediate language like Java bytecode and then interpreting is that you can achieve *platform independence*: you can interpret the same .class file on differently types of machine architectures. However, interpreting the bytecode is typically slower than executing pre-compiled machine language code. A second advantage of using the Java bytecode is that it acts as a buffer between your computer and the program. This enables you to download an untrusted program from the Internet and execute it on your machine with some assurances. Since you are running the Java interpreter (and not raw machine language code), you are protected by a layer of security which guards against malicious programs. It is the combination of Java and the Java bytecode that yield a platform-independent and secure environment, while still embracing a full set of modern programming abstractions.

Q.2. (a) Explain java instruction sets.

Ans. Java Virtual Machine Instruction Set

Table of Contents

A Java Virtual Machine instruction consists of an opcode specifying the operation to be performed, followed by zero or more operands embodying values to be operated upon. This chapter gives details about the format of each Java Virtual Machine instruction and the operation it performs.

Java bytecode is the instruction set of the Java virtual machine. Each bytecode i composed by one, or in some cases two, bytes that represent the instruction (opcode along with zero or more bytes for passing parameters.

Of the 256 possible byte-long opcodes, 198 are currently in use (~77%), 51 ar reserved for future use, and 3 instructions (~1%) are set aside as permanentl unimplemented

There are also a few instructions for a number of more specialized tasks such a exception throwing, synchronization, etc.

Many instructions have prefixes and/or suffixes referring to the types of operand they operate on. These are as follows:

Prefix/Suffix	Operand Type
i	integer
l	long
s	short
b	byte
c	character
f	float
d	double
z	boolean
a	reference

Q.2. (*b*) Explain security policy in java. (5)

Ans. Refer to Q.4(b) of First Term Examination of Model Paper-I.

Q.3. (*a*) Explain java heap. (3

Ans. Java objects reside in an area called *the heap*. The heap is created when th JVM starts up and may increase or decrease in size while the application runs. Whe the heap becomes full, *garbage* is *collected*. During the garbage collection objects tha are no longer used are cleared, thus making space for new objects.

Note that the JVM uses more memory than just the heap. For example Java method thread stacks and native handles are allocated in memory separate from the heap, a well as JVM internal data structures.

The heap is sometimes divided into two areas (or *generations*) called the *nursery* (o *young space*) and the *old space*. The nursery is a part of the heap reserved for allocatio of new objects. When the nursery becomes full, garbage is collected by running a specia *young collection*, where all objects that have lived long enough in the nursery are *promote* (moved) to the old space, thus freeing up the nursery for more object allocation. Whe the old space becomes full garbage is collected there, a process called an *old collection*

The reasoning behind a nursery is that most objects are temporary and short lived A young collection is designed to be swift at finding newly allocated objects that are sti alive and moving them away from the nursery. Typically, a young collection frees a give amount of memory much faster than an old collection or a garbage collection of a single generational heap (a heap without a nursery).

Q.3. (*b*) Explain sandbox model. (7

Ans. A security measure in the Java development environment. The sandbox is set of rules that are used when creating an applet that prevents certain functions whe the applet is sent as part of a Web page. When a browser requests a Web page wit

applets, the applets are sent automatically and can be executed as soon as the page arrives in the browser. If the applet is allowed unlimited access to memory and operating system resources, it can do harm in the hands of someone with malicious intent. The sandbox creates an environment in which there are strict limitations on what system resources the applet can request or access. Sandboxes are used when executable code comes from unknown or untrusted sources and allow the user to run untrusted code safely.

The Java sandbox relies on a three-tiered defense. If any one of these three elements fails, the security model is completely compromised and vulnerable to attack:

- **byte code verifier** — This is one way that Java automatically checks untrusted outside code before it is allowed to run. When a Java source program is compiled, it compiles down to platform-independent Java byte code, which is verified before it can run. This helps to establish a base set of security guarantees.
- **applet class loader** — All Java objects belong to classes, and the applet class loader determines when and how an applet can add classes to a running Java environment. The applet class loader ensures that important elements of the Java run-time environment are not replaced by code that an applet tries to install.
- **security manager** — The security manager is consulted by code in the Java library whenever a dangerous operation is about to be carried out. The security manager has the option to veto the operation by generating a security exception.

Q.4. (*a*) Explain wrapper classes in java. (4)

Ans. A **wrapper class** is defined as a class in which a **primitive value** is wrapped up. These **primitive wrapper classes** are used to represent primitive data type values as objects. The Java platform provides wrapper classes for each of the primitive data types. For example, **Integer wrapper class** holds primitive 'int' data type value. Similarly, **Float wrapper class** contain 'float' primitive values, **Character wrapper class** holds a 'char' type value, and **Boolean wrapper class** represents 'boolean' value.

```
int i = 26; // Primitive data type 'int'
// Integer Wrapper class instantiation
Integer i_Obj = new Integer(i);
// Unwrapping primitive data 'int' from wrapper object
int i2 = i_Obj.intValue();
```

Q.4. (*b*) Explain use of arithmetic and logical operators in java with the help of example program. (6)

Ans.

```
class ArithmeticDemo {
 public static void main (String[] args) {
    int result = 1 + 2;
    // result is now 3
     System.out.println("1 + 2 = " + result);
    int original_result = result;
    result = result - 1;
    // result is now 2
    System.out.println(original_result + " - 1 = " + result);
    original_result = result;
    result = result * 2;
     // result is now 4
```

```
        System.out.println(original_result + " * 2 = " + result);
        original_result = result;
        result = result / 2;
        // result is now 2
        System.out.println(original_result + " / 2 = " + result);
        original_result = result;
        result = result + 8;
        // result is now 10
        System.out.println(original_result + " + 8 = " + result);
        original_result = result;
        result = result % 7;
        // result is now 3
        System.out.println(original_result + " % 7 = " + result);
    }
}
```

This program prints the following:

```
1 + 2 = 3
3 - 1 = 2
2 * 2 = 4
4 / 2 = 2
2 + 8 = 10
10 % 7 = 3
int a =23; // 23 is represented in binary as 10111
int b = ~a; // this will reverts the bits 01000 which is 8 in decimal
boolean x = true;
boolean y = !x; // This will assign false value to y as x is true
```

MODEL TEST PAPER II
SECOND TERM EXAMINATION
FIFTH SEMESTER (B.TECH)
JAVA PROGRAMMING [ETCS-307]

Time : 1 hrs. **M.M. : 30**

Note: *Attempt any three questions in total including Question No. 1 which is compulsory.*

Q.1. (a) Difference between throw and throws in exception handling. **(5)**

Ans. 1. **Throws clause** is used to declare an exception and **throw** keyword is used to throw an exception explicitly.

2. If we see syntax wise than **throw** is followed by an instance variable and **throws** is followed by exception class names.

3. The keyword **throw** is used inside method body to invoke an exception and **throws clause** is used in method declaration

Throw

```
....
static{
try {
throw new Exception("Something went wrong!!");
} catch (Exception exp) {
System.out.println("Error: "+exp.getMessage());
}
}
....
```

Throws

```
public void sample() throws ArithmeticException{
 //Statements
.....
 //if (Condition : There is an error)
ArithmeticException exp = new ArithmeticException();
 throw exp;
...
}
```

Q.1. (b) Explain user defined exceptions. **(5)**

Ans. User defined exceptions in java are also known as **Custom exceptions.** Most of the times when we are developing an application in java, we often feel a need to create and throw our own exceptions. These exceptions are known as **User defined or Custom exceptions.**

```
class MyException extends Exception{
    String str1;
    MyException(String str2) {
     str1=str2;
```

```
    }
    public String toString(){
    return ("Output String = "+str1);
    }
}
class CustomException{
    public static void main(String args[]){
        try{
            throw new MyException("Custom");
            // I'm throwing user defined custom exception above
        }
        catch(MyException exp){
            System.out.println("Hi this is my catch block");
            System.out.println(exp);
        }
    }
}
```

Output:

Hi this is my catch block

Output String = Custom

Q.2. (a) Show the use of String Buffer class. ((

Ans. A string buffer is like a String, but can be modified. At any point in time contains some particular sequence of characters, but the length and content of t sequence can be changed through certain method calls.

String buffers are safe for use by multiple threads. The methods are synchroniz where necessary so that all the operations on any particular instance behave as if th occur in some serial order that is consistent with the order of the method calls made each of the individual threads involved.

The principal operations on a StringBuffer are the append and insert method which are overloaded so as to accept data of any type. Each effectively converts a giv datum to a string and then appends or inserts the characters of that string to the stri buffer. The append method always adds these characters at the end of the buffer; t insert method adds the characters at a specified point.

For example, if z refers to a string buffer object whose current contents are "star then the method call z.append("le") would cause the string buffer to contain "startl whereas z.insert(4, "le") would alter the string buffer to contain "starlet".

In general, if sb refers to an instance of a StringBuffer, then sb.append(x) has t same effect as sb.insert(sb.length(), x).

Whenever an operation occurs involving a source sequence (such as appending inserting from a source sequence) this class synchronizes only on the string buff performing the operation, not on the source.

public final class **StringBuffer**
extends Object
implements Serializable, CharSequence

Q.2. (b) Difference between final and finally in java. **(4)**

Ans. Final:- It is used in the following cases:

- If the final keyword is attached to a variable then the variable becomes constant i.e. its value cannot be changed in the program.
- If a method is marked as final then the method cannot be overridden by any other method.
- If a class is marked as final then this class cannot be inherited by any other class.
- If a parameter is marked with final it becomes a read only parameter.

Finally: If an exception is thrown in try block then the control directly passes to the catch block without executing the lines of code written in the remainder section of the try block. In case of an exception we may need to clean up some objects that we created. If we do the clean-up in try block, they may not be executed in case of an exception. Thus finally block is used which contains the code for clean-up and is always executed after the try ...catch block.

Q.3. (a) Explain thread priority by example program. **(5)**

Ans. class thrun implements Runnable

```
{
    Thread t;
    boolean runn = true;
    thrun(String st,Int p)
    {
t = new Thread(this,st);
t.setPriority(p);
t.start();
    }
     publicvoid run()
     {
System.out.println("Thread name : " + t.getName());
System.out.println("Thread Priority : " + t.getPriority());
    }
}
class priority
{
    publicstaticvoid main(String args[])
    {
Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
thrun t1 = new thrun("Thread1",Thread.NORM_PRIORITY + 2);
thrun t2 = new thrun("Thread2",Thread.NORM_PRIORITY - 2);
System.out.println("Main Thread :" + Thread.currentThread());
System.out.println("Main Thread Priority : " +
Thread.currentThread().getPriority());
}
```

```
/*
   Output
Thread name : Thread1
Main Thread : Thread[main,10,main]
Thread name : Thread2
Thread Priority : 7
Main Thread Priority : 10
Thread Priority : 3
*/
```

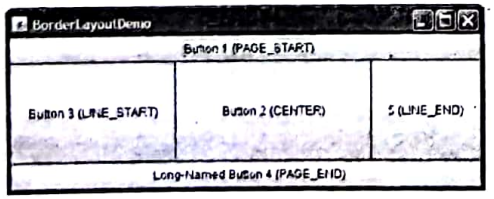
Q.3. (b) Explain layout manager. **(5)**

Ans. Several AWT and Swing classes provide layout managers for general use:

BorderLayout
BoxLayout
CardLayout
FlowLayout
GridBagLayout
GridLayout
GroupLayout
SpringLayout

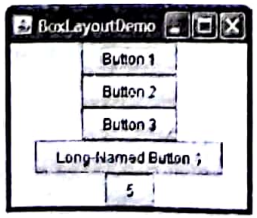
Examples

Border Layout



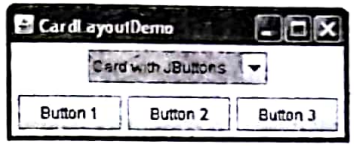
Every content pane is initialized to use a BorderLayout. (As Using Top-Level Containers explains, the content pane is the main container in all frames, applets, and dialogs.) A BorderLayout places components in up to five areas: top, bottom, left, right, and center. All extra space is placed in the center area. Tool bars that are created using JToolBar must be created within a BorderLayout container, if you want to be able to drag and drop the bars away from their starting positions. For further details, see How to Use BorderLayout.

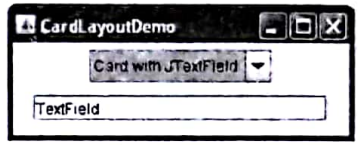
BoxLayout



The BoxLayout class puts components in a single row or column. It respects the components' requested maximum sizes and also lets you align components. For further details, see How to Use BoxLayout.

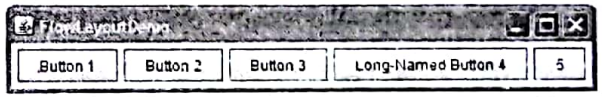
CardLayout





The CardLayout class lets you implement an area that contains different components at different times. A CardLayout is often controlled by a combo box, with the state of the combo box determining which panel (group of components) the CardLayout displays. An alternative to using CardLayout is using a tabbed pane, which provides similar functionality but with a pre-defined GUI. For further details, see How to Use CardLayout.

FlowLayout



FlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row, starting a new row if its container is not sufficiently wide. Both panels in CardLayoutDemo, shown previously, use FlowLayout. For further details, see How to Use FlowLayout.

Q.4. Explain: **(3)**

Q.4. (a) Wait()

Ans. wait() tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls **notify()**.

Q.4. (b) Notify()

Ans. notify() wakes up the first thread that called **wait()** on the same object.

Q.4. (c) Notify all()

Ans. notifyAll() wakes up all the threads that called **wait()** on the same object. The highest priority thread will run first.

These methods are declared within Object, as shown here:

final void wait() throws InterruptedException
final void notify()
final void notifyAll()

Q.4. (b) Show the use of Runnable interface. **(7)**

Ans. Creating thread by implementing Runnable interface

1. Let your class implement "Runnable" interface.

2. Now override the "public void run()" method and write your logic there (This is the method which will be executed when this thread is started).

That's it, now you can start this thread as given below

1. Create an object of the above class
2. Allocate a thread object for our thread

3. Call the method "start" on the allocated thread object.

```
public class FirstThread implements Runnable
{
//This method will be executed when this thread is executed
public void run()
{
//Looping from 1 to 10 to display numbers from 1 to 10
for (int i=1; i<=10; i++)
{
//Displaying the numbers from this thread
System.out.println("Messag from First Thread : " +i);
/*taking a delay of one second before displaying next number
*
* "Thread.sleep(1000);" - when this statement is executed,
* this thread will sleep for 1000 milliseconds (1 second)
* before executing the next statement.
*
* Since we are making this thread to sleep for one second,
* we need to handle "InterruptedException". Our thread
* may throw this exception if it is interrupted while it
* is sleeping.
*
*/
try
{
Thread.sleep (1000);
}
catch (InterruptedException interruptedException)
{
  /*Interrupted exception will be thrown when a sleeping or waiting
   *thread is interrupted.
   */
  System.out.println("First Thread is interrupted when it is sleeping"
+interruptedException);
    }
  }
 }
}
```

MODEL TEST PAPER II
END TERM EXAMINATION
FIFTH SEMESTER (B.TECH)
JAVA PROGRAMMING [ETCS-307]

Time : 3 hrs. **M.M. : 75**

Note: *Question 1 is compulsory and attempt any 6 questions from the rest.*

Q.1. (a) Explain component class in java. **(10)**

Ans. The Component class is found under java.awt package. The container class is the subclass of Component class. All non-menu-related elements that comprise a graphical user interface are derived from the abstract class Component. The Component class defines a number of methods for handling events, changing window bounds, controlling fonts and colors, and drawing components and their content.

A container is a component that can accommodate other components and also other containers. Containers provide the support for building complex hierarchical graphical user interface. Container provides the overloaded method add () to include components in the container

Three steps are necessary to create and place a GUI component:

1. Declare the component with an *identifier* (*name*);

2. Construct the component by invoking an appropriate constructor via the new operator;

3. Identify the container (such as Frame or Panel) designed to hold this component. The container can then add this component onto itself via *aContainer*.add(*aComponent*) method. Every container has a add(Component) method. Take note that it is the container that actively and explicitly adds a component onto itself, instead of the other way.

```
Label lblInput; // Declare an Label instance called lblInput
lblInput = new Label("Enter ID"); // Construct by invoking a constructor via the new operator
add(lblInput); // this.add(lblInput) - "this" is typically a subclass of Frame
lblInput.setText("Enter password"); // Modify the Label's text string
lblInput.getText(); // Retrieve the Label's text string
```

Q.1.(b) Explain Wrapper classes. **(5)**

Ans. Refer Model Paper-II Q.4(a) First Term Examination.

Q.2. Define Sandbox model. **(10)**

Ans. Refer Model Paper-II Q.3(b) First Term Examination.

- A sandbox in which the program has access to the CPU, the screen, keyboard, and mouse, and to its own memory. This is the minimal sandbox — it contains just enough resources for a program to run.

- A sandbox in which the program has access to the CPU and its own memory as well as access to the web server from whicn it was loaded. This is often thought of as the default state for the sandbox.

- A sandbox in which the program has access to the CPU, its memory, its web server, and to a set of program-specific resources (local files, local machines, etc.). A

word-processing program, for example, might have access to the docs directory on the local filesystem, but not to any other files.

• An open sandbox, in which the program has access to whatever resources the host machine normally has access to.

The sandbox, then, is not a one-size-fits-all model. Expanding the boundaries of the sandbox is always based on the notion of trust: when my one-year-old niece comes to visit, there's very little in the sandbox for her to play with, but when my six-year-old godchild comes to visit, I trust that I might give her more things to play with. In the hands of some visitors, a toy with small removable parts would be dangerous, but when I trust the recipient, it's perfectly reasonable to include that item in the sandbox. And so it is with Java programs: in some cases, I might trust them to access my filesystem; in other cases, I might trust them to access only part of my filesystem; and in still other cases, I might not trust them to access my filesystem at all.

Applets, Applications, and Programs: In early versions of Java, only applets were run within a sandbox. In the Java 2 platform, all programs have the potential to run in a sandbox. Applets that run through the Java Plug-in or the appletviewer will always run in a sandbox, and applications that are run via the command line (or by clicking an icon on the desktop) may optionally be set up to use a sandbox. Applications also have the option of programmatically installing new versions of the sandbox.

Hence, in the Java 2 platform there is little distinction between the security level of an applet and an application. There are programmatic differences, of course, but both are subject to the same security model, and the security model for both is administered and programmed in the same way. There is one significant difference, however: applets always run with Java's security model (even if that model has been administered such that the applet is allowed to do anything it wants to), and an application will only run under the security model if it is told to do so. This is typically done by the end user by specifying a command-line parameter; it may be done by the program developer who specifies that parameter in a script that starts the application, and it may be done by the developer who inserts code into his program.

Q.3. Define life cycle of Applet with example program. **(10)**

Ans. Refer ans 2 a second term examination dummy 1.

```
import java.applet.Applet;
import java.awt.Graphics;
/*
*
* Applet can either run by browser or appletviewer application.
* Define <applet> tag within comments as given below to speed up
* the testing.
*/
/*
<applet code="AppletLifeCycleExample" width=100 height=100>
</applet>
*/
public class AppletLifeCycleExample extends Applet{
    /*
    * init method is called first.
```



```
    * It is used to initialize variables and called only once.
*/
public void init() {
    super.init();
    }
    /*
    * start method is the second method to be called. start method is
    * called every time the applet has been stopped.
    */
    public void start() {
    super.start();
    }
    /*
    * stop method is called when the the user navigates away from
    * html page containing the applet.
    */
    public void stop() {
    super.stop();
    }
    /* paint method is called every time applet has to redraw its
    * output.
    */
    public void paint(Graphics g) {
    super.paint(g);
    }
    /*
    * destroy method is called when browser completely removes
    * the applet from memeory. It should free any resources initialized
    * during the init method.
    */
    public void destroy() {
        super.destroy();
    }
}
```

Q.4. Explain various Action Events in java. ? **(10)**

Ans. Components such as the Button and JButton fire off ActionEvents to indicate some kind of component-defined action. For example, the Button fires off an ActionEvent whenever the user presses it. The entire point of an event is to inform a listener that something has happened to a component in the GUI. An event includes all of the information that a listener needs to figure out what happened and to whom it happened (the what and who of the event). An event must give enough information to fully describe itself. That way, a listener can figure out what exactly happened and respond in a meaningful way.

The ActionEvent includes methods for learning the action's command string, modifiers, and identification string. The getActionCommand() method returns the command string that indicates the event's intended action, such as print or copy (the what). The getSource() method returns the object that generates the event (the who).

In order to receive an ActionEvent, a listener must implement the ActionListener interface and register itself with the component. Furthermore, a component must keep track of its listeners in order to notify them of an event.

By using the ActionEvent example as a model, we can easily see the pieces necessary for a component to generate an event and a listener to listen for an event. At a high level, there are three pieces:

1. The component
2. The event class
3. The listener interface

Example:

```
EventListenerList xxxListeners = new EventListnerList();
public void addXXXListener(XXXListener listener)
{
 xxxListeners.add(XXXListener.class, listener);
}
public void removeXXXListener(XXXListener listener)
{
 xxxListeners.remove(XXXListener.class, listener);
}
protected void fireXXX(XXXEvent xxxEvent)
{
    Object[] listeners = xxxListeners.getListenerList();
    // loop through each listener and pass on the event if needed
    Int numListeners = listeners.length;
    for (int i = 0; i<numListeners; i+=2)
    {
        if (listeners[i]==XXXListener.class)
        {
            // pass the event to the listeners event dispatch method
            ((XXXListener)listeners[i+1]).dispatchXXX(xxxEvent);
        }
    }
}
```

The example defines a generic recipe that all components can follow. However, in order for the example to work, you must define an XXXEvent and an XXXListener interface.

The event class: The event holds all of the information necessary for a listener to figure out what happened. The information included is really event specific. Just think about the event carefully and design the event class to hold whatever information is necessary to fully describe the event to a listener. Events normally extend the java.awt.AWTEvent event class.

The listener interface: An event listener interface defines the methods used by a component to dispatch events. Each event type will have at least one corresponding dispatch method in a listener interface.

```
public interface XXXListener
    extends EventListener
{
    // event dispatch methods
    somethingHappened(XXXEvent e);
    somethingElseHappened(XXXEvent e);
    ...
}
```

Q.5. Explain the use of adapter class in java with the help of example program. **(10)**

Ans. An adapter class is most easily defined as an inner class, placed inside the class which requires the adapter.

```
public class FixedStack {
    Object array[];
    int top = 0;
 FixedStack(int fixedSizeLimit) {
     array = new Object[fixedSizeLimit];
 }
 public void push(Object item) {
     array[top++] = item;
 }
 public boolean isEmpty() {
     return top == 0;
 }
 // other stack methods go here...
 /** This adapter class is defined as part of its target class,
 * It is placed alongside the variables it needs to access.
 */
 class Enumerator implements java.util.Enumeration {
     int count = top;
     public boolean hasMoreElements() {
         return count > 0;
     }
     public Object nextElement() {
         if (count == 0)
             throw new NoSuchElementException("FixedStack");
         return array[—count];
     }
 }
```

```
public java.util.Enumeration elements() {
    return new Enumerator();
    }
}
```

The interface java.util.Enumeration is used to communicate a series of values to client. Since FixedStack does not (and should not!) directly implement the Enumeratio interface, a separate adapter class is required to present the series of elements, in th form of an Enumeration. Of course, the adapter class will need some sort of access to th stack's array of elements. If the programmer puts the definition of the adapter clas inside of FixedStack, the adapter's code can directly refer to the stack object's instanc variables.

In Java, a class's non-static members are able to refer to each other, and they al take their meaning relative to the current instance this. Thus, the instance variabl array of FixedStack is available to the instance method push and to the entire body o the inner class FixedStack.Enumerator. Just as instance method bodies "know" thei current instance this, the code within any inner class like Enumerator "knows" it *enclosing instance*, the instance of the enclosing class from which variables like arra are fetched.

Q.6. Write short note on Object serialization. **(10)**

Ans. To *serialize* an object means to convert its state to a byte stream so that th byte stream can be reverted back into a copy of the object. A Java object is *serializabl* if its class or any of its superclasses implements either the java.io.Serializable interfac or its subinterface, java.io.Externalizable. *Deserialization* is the process of convertin the serialized form of an object back into a copy of the object.

For example, the java.awt.Button class implements the Serializable interface, s you can serialize a java.awt.Button object and store that serialized state in a file Later, you can read back the serialized state and deserialize into a java.awt.Buttor object.

The Java platform specifies a default way by which serializable objects ar serialized. A (Java) class can override this default serialization and define its own wa of serializing objects of that class. The Object Serialization Specification describe object serialization in detail.

When an object is serialized, information that identifies its class is recorded in th serialized stream. However, the class's definition ("class file") itself is not recorded. I is the responsibility of the system that is deserializing the object to determine how t locate and load the necessary class files

Binding a Serializable Object: You can store a serializable object in the director if the underlying service provider supports that action, as does Oracle's LDAP servic provider.

The following example invokes Context.bind to bind an AWT button to the nam "cn=Button". To associate attributes with the new binding, you use DirContext.bind. T overwrite an existing binding, use Context.rebind and DirContext.rebind.

```
// Create the object to be bound
Button b = new Button("Push me");
// Perform the bind
ctx.bind("cn=Button", b);
```

You can then read the object back using Context.lookup, as follows.


```
// Check that it is bound
Button b2 = (Button)ctx.lookup("cn=Button");
System.out.println(b2);
```

Running this example produces the following output.

```
# java SerObj
java.awt.Button[button0,0,0,0x0,invalid,label=Push me]
```

Q.7. Explain hash table classes in java. **(10)**

Ans. A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key. It implements the Map interface and extends Dictionary class.

This class implements a hashtable, which maps keys to values. Any non-null object can be used as a key or as a value.

To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

An instance of Hashtable has two parameters that affect its efficiency: its *capacity* and its *load factor*. The load factor should be between 0.0 and 1.0. When the number of entries in the hashtable exceeds the product of the load factor and the current capacity, the capacity is increased by calling the rehash method. Larger load factors use memory more efficiently, at the expense of larger expected time per lookup.

If many entries are to be made into a Hashtable, creating it with a sufficiently large capacity may allow the entries to be inserted more efficiently than letting it perform automatic rehashing as needed to grow the table.

- It contains only unique elements.
- It may have not have any null key or value.
- It is synchronized.

```
import java.util.*;
class TestCollection16{
public static void main(String args[]){
Hashtable<Integer,String> hm=new Hashtable<Integer,String>();
hm.put(100,"Amit");
hm.put(102,"Ravi");
hm.put(101,"Vijay");
hm.put(103,"Rahul");
for(Map.Entry m:hm.entrySet()){
System.out.println(m.getKey()+" "+m.getValue());
}
}
}
```

Output:

```
103 Rahul
102 Ravi
101 Vijay
100 Amit
```

Q.8. Explain JDBC connectivity with MS-SQL server in java. (10

Ans. To connect MS SQL Server using windows authentication, the first step is t setup ODBC. You can go to Control panel -> Administrative tools -> ODBC. Add a ne DSN to connect MS SQL Server using windows authentication account following wizar setup.

The second step is the similar with using SQL Server authentication. The onl change is that the connection string is: jdbc:odbc:dsn-name. There is no need to u username/password any longer, because it is already connected to the server!

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class Database {
    public Connection conn = null;
    private String dbName = null;

    public Database(){
    }

    public Database(String dbName, String dbURL){
        this.dbName = dbName;
        try {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            this.conn = DriverManager.getConnection(dbURL);//here put the ne
simple url.
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public ResultSet runSql(String sql) throws SQLException {
        Statement sta = conn.createStatement();
        return sta.executeQuery(sql);
    }
}
```

FIRST TERM EXAMINATION [SEPT. 2015]
FIFTH SEMESTER [B.TECH]
JAVA PROGRAMMING [ETCS–307]

Time. 1.5 Hours **M.M. : 30**

Note: Q No.1 is compulsory. Attempt any two from the remaining three quations.

Q.1. (a) What is method overriding in java? (2)

Ans. Method Overriding in Java: If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java.**

In other words, if subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Usage of java method overriding:

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism.

Example:

One of the simplest example – Here Boy class extends Human class. Both the classes have a common method void eat(). Boy class is giving its own implementation to the eat() method or in other words it is overriding the method eat().

```
class Human{
  public void eat()
  {
    System.out.println("Human is eating");
  }
}
class Boy extends Human{
  public void eat(){
    System.out.println("Boy is eating");
  }
  public static void main( String args[]) {
    Boy obj = new Boy();
    obj.eat();
  }
}
```

Output:

Boy is eating

Q.1. (b) Difference between abstract class and interface? (2)

Ans.

Abstract Classes	Interfaces
1. Abstract class can extend only one class or one abstract class at a time	Interface can extend any number of interfaces at a time
2. Abstract class can extend from a class or from an abstract class	Interface can extend only from an interface
3. Abstract class can have both abstract and concrete methods	Interface can have only abstract methods

4. A class can extend only one abstract class.	A class can implement any number of interfaces
5. In abstract class keyword 'abstract' is mandatory to declare a method as an abstract	In an interface keyword 'abstract' is optional to declare a method as an abstract
6. Abstract class can have protected. public and public abstract methods	Interface can have only public abstract methods *i.e* by default

Q 1. (c) What is java sandbox model? **(2)**

Ans. A security measure in the Java development environment. The sanbox is a set of rules that are when creating an applet that prevents certain functions when the applet is sent as part of a Web page. When a browser requests a Web page with applets, the applets are sent automatically and can be executed as soon as the page arrives in the browser. if the applet is allowed unlimited access to memory and operating system resources, it can do harm in the hands of someone with malicious intent. The sandbox creates an environment in which there are strict limitations on what system resources the applet can request or access. Sandboxes are used when executable code comes from unknown or untrusted sources and allow the user to run untrusted code safely.

The Java sandbox relies on a three-tiered defense. If any one of these three elements fails, the secuirty model is completely compromised and vulnerable to attack.

- **Byte code verifier**
- **applet class loader**
- **security manager**

Q. 1. (d) Can we initialize final static variable inside the constructor? Explain. **(2)**

Ans. No, we can not initialize final static variable inside constructor.

Static final variables cannot be assigned value in constructor; they must be assigned a value with their declaration. Non-static final variables can be assigned a value either in constructor or with the declaration.

```
public class A
{
    private static final int x;

    public A()
    {
        x = 5;
    }
}
```

Final means the variable can only be assigned once (in the constructor).

Static means it's a class instance.

A constructor will be called each time an instance of the class is created. Thus, the above code means that the value of x will be re-initialized each time an instance is created. But because the variable is declared final (and static), you can only do this

```
class A {
    private static final int x;
    static {
        x = 5;
    }
}
```

But, if you remove static, you are allowed to do this:


```
class A {
    private final int x;
    public A() {
        x = 5;
    }
}
```

Q.1. (e) What is the purpose of using Wrapper classes?

Ans. A wrapper class is defined as a class in which a primitive value is wrapped up. These primitive wrapper classes are used to represent primitive data type values as objects. The Java platform provides wrapper classes for each of the primitive data types. For example, Integer wrapper class holds primitive 'int' data type value. Similarly, Float wrapper class contain 'float' primitive values, Character wrapper class holds a 'char' type value, and Boolean wrapper class represents 'boolean' value.

```
int i = 26; // Primitive data type 'int'
// Integer Wrapper class instantiation
Integer i_Obj= new Integer(i);
// Unwrapping primitive data 'int' from wrapper object
int i2 = i_Obj.intValue();
```

Q.2. (a) Explain Various uses of final keyword with example. **(2)**

Ans. Final keyword in Java has three different uses: create constants, prevent inheritance and prevent methods from being overridden. Following is a list of uses of final keyword.

1. Using final to define constants: If you want to make a local variable, class variable (static field), or instance variable (non-static field) constant, declare it final. A final variable may only be assigned to once and its value will not change and can help avoid programming errors.

Once a final variable has been assigned, it always contains the same value.

```
1. class Bike9{
2. final int speedlimit=90;//final variable
3. void run(){
4. speedlimit=400;
5. }
6. public static void main(String args[]){
7. Bike9 obj=new Bike9();
8. obj.run();
9. }
10. }//end of class
```

Output:Compile Time Error

2. Using final to prevent inheritance: If you find a class's definition is complete and you don't want it to be sub-classed, declare it final. A final class cannot be inherited, therefore, it will be a compile-time error if the name of a final class appears in the extends clause of another class declaration; this implies that a final class cannot have any subclasses.

It is a compile-time error if a class is declared both final and abstract, because the implementation of such a class could never be completed.

```
final class XYZ{
}
class ABC extends XYZ{
  void demo(){
```

```
        System.out.println("My Method");
    }
    public static void main(String args[]){
        ABC obj= new ABC();
        obj.demo();
    }
}
```

Output:

The type ABC cannot subclass the final class XYZ

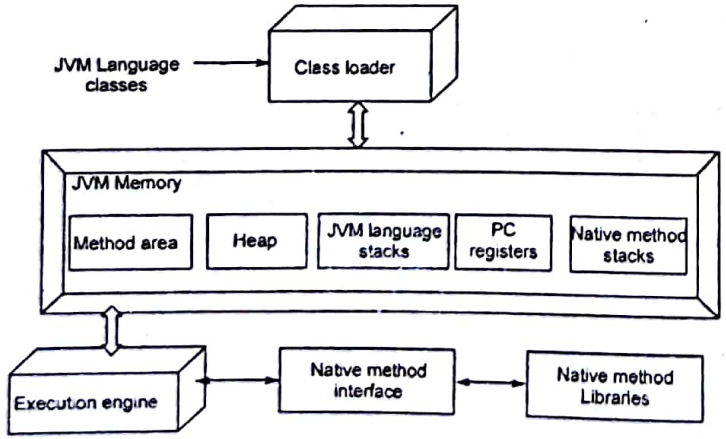
3. Using final to prevent overriding: When a class is extended by other classes, its methods can be overridden for reuse. There may be circumstances when you want to prevent a particular method from being overridden, in that case, declare that method final. Methods declared as final cannot be overridden.

```
class XYZ{
  final void demo(){
    System.out.println("XYZ Class Method");
  }
}

class ABC extends XYZ{
  void demo(){
    System.out.println("ABC Class Method");
  }
  public static void main(String args[]){
    ABC obj= new ABC();
    obj.demo();
  }
}
```

The above program would throw a compilation error.

Q.2. (b) What is JVM and class loader architecture? (3)

Ans. A Java virtual machine (JVM) is an abstract computing machine. There are three notions of the JVM: specification, implementation, and instance. The specification



is a book that formally describes what is required of a JVM implementation. Having a single specification ensure all implementations are interoperable. A JVM implementation is a computer program that meets the requirements of the JVM specification in a compliant and preferably performant manner. An instance of the JVM is a process that executes a computer program complied into Java bytecode.

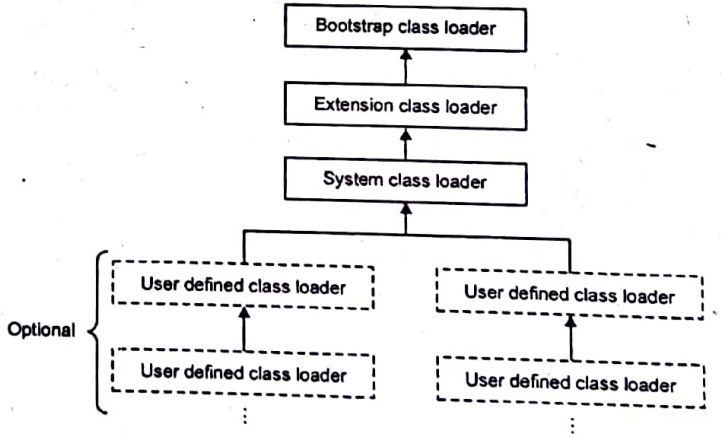
Class loader architecture: Java ClassLoader loads a java class file into java virtual machine. It is as simple as that. It is not a huge complicated concept to learn and every java developer must know about the java class loaders and how it works.

Like NullPointerException, one exception that is very popular is ClassNotFoundException. At least in your beginner stage you might have got umpteen number of ClassNotFoundException. Java class loader is the culprit that is causing this exception.

Types (Hierarchy) of Java Class Loaders.

Java class loaders can be broadly classified into below categories:

- **Bootstrap Class Loader:** Bootstrap class loader loads Java's core classes like java lang. java.util etc These are classes that are part of java runtime environment. Bootstrap class loader is native implementation and so they may differ across different JVMs.
- **Extensions Class Loader:** JAVA_HOME/jre/lib/ext contains jar packages that are extensions of standard core java classes. Extensions class loader loads classes from this ext folder. Using the system environment properly java ext.dirs you can add 'ext' folders and jar files to be loaded using extensions class loader.
- **System Class Loader:** Java classes that are available in the java classpath are loaded using System class loader.



Class loader architecture

Q.2. (c) How exception handing is done in java? Explain use of throw, throws, try, catch, finally. (5)

Ans. Refer Q.2. (a) of End Term Examination 2015

Q.3. (a) What is type coversion? Explain implicit and explicit conversion is example. (2)

Ans. Changing a value from one data type to another type is known as data type conversion. Data type conversions are either widening or narrowing, it depends on the data capacities of the data types involved. There are different ways of, implicitly or explicitly, changing an entity of one data type into another data type.

An important consideration with a type conversion is whether the result of the conversion is within the range of the destination data type.

Widening Conversion: If a value of narrower (lower size) data type converted to a value of a broader (higher size) data type without loss of information, is called Widening conversion. This is done implicitly by the JVM and also known as implicit casting. For example an integer data type is directly converted to a double.

```
int a = 100;
double b = a;
System.out.println(b);
```

byte → short → int → long → float → double

Widening

Narrowing Conversion

If a value of broader (higher size) data type converted to a value of a narrower (lower size) data type which can result in loss of information, is called Narrowing conversion. For example double data type explicitly converted into integer

```
double a = 100.7;
int b = (int) a;
System.out.println(b);
```

double → float → long → int → short → byte

Narrowing

Q. 3. (b) Explain ues of String, String Buffer and String Tokenizer with example (3)

Ans. USE of String:

Java String provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compareTo, intern, substring etc.

In java, string is basically an object that represents sequence of char values.

An array of characters works same as java string. For example:

1. **char**[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=**new** String(ch);

The java. lang. String class implements Serializable, Comparable and Char Sequence interfaces.

Java String class methods

Method	Description
Char char At (int index)	returns char valus for the particular index.
int length()	returns string length
static String format (String format, Object ... args)	returns formatted string
static String format(locale l, String format, Object...args)	returns formatted string with given locale
String substring (int beginIndex)	returns substring for given begin index
Srting substring(int beginIndex, int end Index)	returns substring for given begin index and end index

String Buffer class: StringBuffer class is used to create a **mutable** string object. A string that can be modified or changed is known as mutable string. StringBuffer and

StringBuilder classes are used for creating mutable string. It represents growable and writable character sequence. So if we do a lot of changes with **String** objects, we will end up with a lot of memory leak.

Important methods of StringBuffer class

1. Public synchronized StringBuffer append(String s): Is used to append the specified string with this string. The append () method is overloaded like append (char), append (boolean), append (int), append (float), append(double) etc.

2. Public synchronized StringBuffer insert(int offset, String s): Is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert (int, double) etc.

3. Public synchronized StringBuffer replace(int startIndex, int endIndex, String str): Is used to replace the string from specified startIndex and endIndex.

4. Public synchronized StringBuffer delete(int startIndex, int endIndex): Is used to delete the string from specified startIndex and endIndex.

```
class A{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

String Tokenizer in Java:

The java.util.StringTokenizer class allows you to break a string into tokens. It is simple way to break string.

Methods of StringTokenizer class

The 6 useful methods of StringTokenizer class are as follows:

Public method	Description
Boolean has More Tokens()	Checks if there is more tokens available.
String nextToken()	returns the next token from the string token Tokenizer object.
String next Token (string delim)	returns the next token based on delimeter
Example:	
Boolean hasMoreElements()	Same as hasMoreToken() method.

```
package com.myjava.stokenizerr;
import java.util.StringTokenizer;
public class MyStringTokens {
  public static void main(String a[]){
    String msg = "http://10.123.43.67:80/";
    StringTokenizer st = new StringTokenizer(msg,"://.");
    while(st.hasMoreTokens()){
        System.out.println(st.nextToken());
    }
  }
}
```

Q.3. (c) How applet is different from java application? Explain Applet life cycle with example. (3)

Ans.

Applet	Application
Small Program	Large Program
Used to run a program on client Browser	Can be executed on stand alone computer system
Applet is portable and can be executed by supported browser.	Need JDK, JRE, JVM installed on JAVA client machine
Applet applicationss are executed in a Restricted Environment	Application can access all the resources of the computer
Applets are created by extending the java. applet. Applet	Applications are created by writing public static void main(String[] S) method.
Applet application has 5 methods which will be automatically invoked on occurance of specific event	Application has a single start point which is main methods

Example:

```
import java.awt. *;
import java. applet. *;
Public class MyClass extends Applet
{
        public void init () { }
        public void start () { }
        public void step () { }
        public void destroy () { }
        public void paint (Graphics g) { }
}
```

```
Public class Myclass
{
public static void main (string args []) { }
}
```

Q. 4. (a) What is byte code and explain class file format? (2)

Ans. Java class file formate: Compiled Java is typically distributed in a very compact format called Class files. It is the assembler code for a virtual stack-oriented Java-friendly CPU (Cental Processing Unit) chip. Such chips do exist, but most of the time they are simulated with interpreters, JIT (Just In Time), Hotspots or AOT (Ahead Of Time) compilation on the target machines which have quite diffeent architectures.

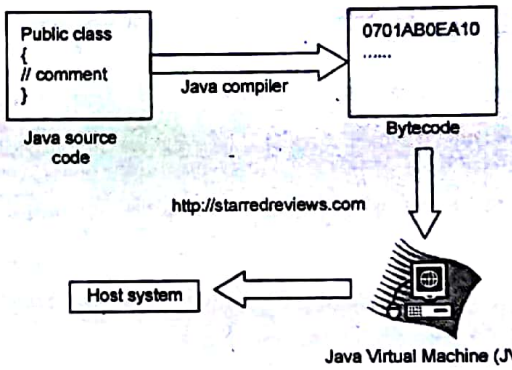
Class files are usually compressed and bundled into jar files.

Each class file is marked with a class file format version, which lets you know which JDK (Java Development Kit) it was compiled for. At offset 6 in the class file is a big-endian short that gives the major version number. You can display for the classes is in a jar with JarCheck.

A java class file is a file (with the class filename extension) containing Java bytecode that can be executed on the Virtual Machine (JVM). A Java clss file is produced by a Java compiler from Java programming language sources files (java files) containing Java classes. if a source file has more than one class, each class is compiled into a separate class file.

JVMs are available for many platforms, and a class file compiled on one platform will execute on a JVM of another platform. This makes JaVa platform-independent.

BYTE CODE: Bytecode is nothing but the intermediate representation of Java source code which is produced by the Java compiler by compiling that source code. This byte code is an machine independent code.It is not an completely a compiled code but it is an intermediate code somewhere in the middle which is later interpreted and executed by JVM.Bytecode is a machine code for JVM.But the machine code is platform specific whereas bytecode is platform independent that is the main difference between them.It is stored in class file which is created after compiling the source code.Most developers although have never seen byte code (nor have ever wanted to see it!) One way to view the byte code is to compile your class and then open the .class file in a hex editor and translate the bytecodes by referring to the virtual machine specification.



Q. 4. (b) What is inner class explain with example? (3)

Ans. Inner Class: Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.

Given below is the program to create an inner class and access it. In the given example, we make the inner class private and access the class through a method.

```
class Outer_Demo{
  int num;
  //inner class
  private class Inner_Demo{
    public void print(){
      System.out.println("This is an Inner class");
    }
  }
  //Accessing he Inner class from the method within
  void display_Inner(){
    Inner_Demo inner = new Inner_Demo();
    inner.print();
  }
}
public class My_class{
  public static void main(String args[]){
    //Instantiating the outer class
    Outer_Demo outer = new Outer_Demo();
    //Accessing the display_Inner() method.
    outer.display_Inner();
  }
```

}

If you compile and execute the above program, you will get the following result.

This is an inner class.

Advantage of java inner classes: There are basically three advantages of inner classes in java. They are as follows:

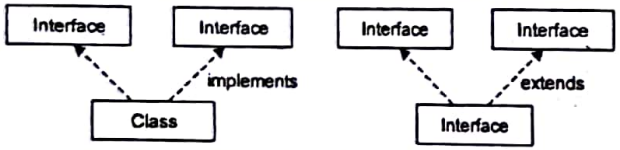
1. Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

2. Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

3. **Code Optimization**: It requires less code to write.

Q. 4. (c) Write a program to explain how multiple inheritance is done in java? **(5)**

Ans. Multiple inheritance in Java: If a class implements multiple interfaces, or an interface extends multiple interfaces *i.e.* known as multiple inheritance.



Multiple Inheritance in Java

To reduce the complexity and simplify the language, multiple inheritance is not supported in java in terms of classes. Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

Java's alternative to multiple inheritance – interfaces

```
1. interface Printable{
2. void print();
3. }
4.
5. interface Showable{
6. void show();
7. }
8.
9. class A7 implements Printable,Showable{
10.
11. public void print(){System.out.println("Hello");}
12. public void show(){System.out.println("Welcome");}
13.
14. public static void main(String args[]){
15. A7 obj = new A7();
16. obj.print();
17. obj.show();
18. }
19. }
```

Output:Hello
Welcome

SECOND TERM EXAMINATION [NOV. 2015]
FIFTH SEMESTER [B.TECH]
JAVA PROGRAMMING [ETCS–307]

Time. 1.5 Hours **M.M. : 30**

Note: Attempt 3 questions in total. Question 1 is compulsory. Attempt any two the remaining 3 questions.

Q.1. (a) What is a Vector class? Explain with exmple. **(5 × 2 = 10)**

Ans. Vector class implements a growable array of objects. Like an array, it contains the components that can be accessed using an integer index. However, the size of a vector can grow or shrink as needed to accommodate adding and removing items after the vector has been created. Each vector tries to optimize storage management by maintaining a capacity and a capacity increment.

eg.

```
import Java. wil.*;
class test vector {
public static void main (strting args [])
{
vector < string> v = new vector <string> ();
vector add ("suresh');
vector add Element ("vinesh");
vector add Element ("kumar");
Enumeration e = Vector elements ();
while (e. has more elements ())
{
System.out.println (e. next element ());
}
}
}
```

Q.1. (b) Explain socket programming. **(2)**

Ans. Socket Programming: Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

The following steps occur when establishing a TCP connection between two computers using sockets:

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.

• On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

An example program illustrating creation of a server socket, waiting for client request, and then responding to a client that requested for connection by greeting it is given below:

```
// SimpleServer.Java: A simple server program.
import java.net.*;
import java.io.* ;
public class SimpleServer {
  public static void main(String args[]) throws IOException {
    // Register service on port 1254
    ServerSocket s = new ServerSocket(1254);
    Socket s1 = s.accept(); // Wait and accept a connection
    // Get a communication stream associated with the socket
    OutputStream slout = s1.getOutputStream();
    DataOutputStream dos = new DataOutputStream (slout);
    // Send a string!
    dos.writeUTF("Hi there");
    // Close the connection, but not the server socket
    dos.close() ;
    slout.close();
    s1.close();
  }
}
// SimpleClient.Java: A simple client program.
import java.net.*;
import java.io.*;
public class SimpleClient {
  public static void main(String args[]) throws IOException {
    // Open your connection to a server, at port 1254
    Socket si = new Socket ("localhost, 1254) ;
    // Get an input file handle from the socket and read the input
    InputStream s1In = s1.getInputStream();
    DataInputStream dis = new DataInputStream(s1In);
    String st = new String (dis.readUTF() ) ;
    System.out.print ln(st);
    // When done, just close the connection and exit
    dis.close();
    s1In.close();
    s1.close();
  }
}
```

Q.1. (c) Explain different types of classes under java.io package. **(2)**

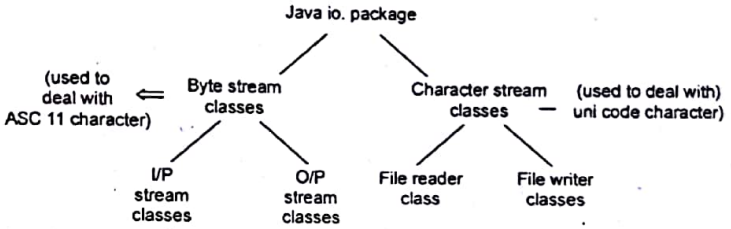
Ans. IO Stream: Java performs I/O through **Streams**. A Stream is linked to a physical layer by java I/O system to make input and output operation in java. In general, a stream means continuous flow of data. Streams are clean way to deal with input/output without having every part of your code understand the physical.

Java encapsulates Stream under **java.io** package. Java defines two types of streams. They are,

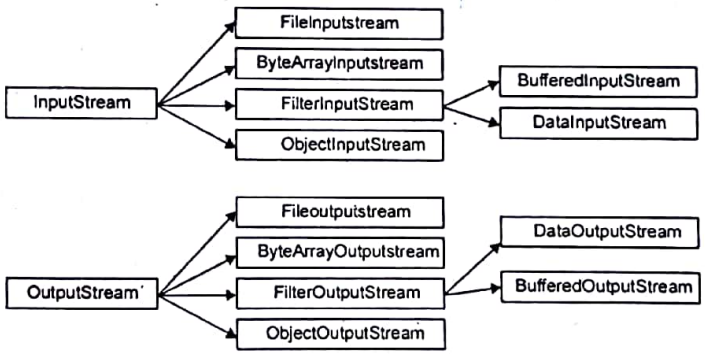
1. Byte Stream : It provides a convenient means for handling input and output of byte.

2. Character Stream : It provides a convenient means for handling input and output of characters. Character stream uses Unicode and therefore can be internationalized.

(c) Different types of classes under java. io. package.



Input stream and output stream classes are further classifies as following:



Q.1. (d) What are iterators? **(2)**

Ans. When the for-each loop is not available, and an explicit Iterator is needed, then iteration over a collection may be done with a while loop or a for loop.

The two styles have different advantages:

• the while loop is considerably more legible

• the for loop minimizes the scope of the Iterator to the loop itself

Q.1. (e) What is the use of synchronized block? **(2)**

Ans. Synrchronized block is used to preforms synchronization on any specific resource of the method. If you put all the codes of the method in the syncronized block, it will work the same as the synchronized method.

Synchronized block is used to lock an object if any shared resource.

Scope of synchronized block is smaller than the method.

Syntax:

```
Synchronized (object reference expression)
{
//code
}
```

Q.2. (a) Differentiate between java.awt and javax.swing package (2)

Ans. Main Differences between Swing and AWT.

Swing components are light-weight as compared to AWT components for the simple reason that they don't make used of the native UI components of the underlying platform.

Swing components have been written completely in java language. Not the same with AWT as it uses native libraries.

Swing components use relatively less number of classes as compared to AWT components not only because it doesn't depend on native code, but also because it has a better design due to its evolution from AWT. Most of the design flaws got fixed which ultimately led to lesser redundancies.

The look and feel of the Swing components is defined only by the Swing extension classes and it's pluggable to have the look and feel of the underlying platform as well whereas the look and feel of the AWT components is mostly governed by the underlying platform. By default, Swing components open up in their own look and feel.

Swing extension is completely based on the MVC (Model View Controller) architecture and hence the inherent benefits of the MVC design pattern add more value to the usage of Swing.

Q. 2. (b) Explain thread life cycle? (3)

Ans. Refer Q.5. of End Term Examination-2015.

Q.2. (c) Write a program using multithreading where in 3 different objects will run in parallel, to print the tables of three nos (5)

Ans. Program:

```
Class Table {
synchronized void print (int n)
{
for (int i = 1; i < = 10; i + s)
{
system.out.printIn (n + i);
try{
Thread.sleep (400);
} catch (Exception e)
{ }
}
}
}
Class Thread 1 extends Thread {
Table t;
Thread 1 (table t)
{
this.t. = t ;
}
Public void run () {
t. print (5);
}
}
```



```
class Thread2 extends Thread {
Table t;
Thread 2 (Table +)
{
this.t = t;
}
public void run (){
t. print (10);
}
}
```

Q.3. (a) Differenatiate between Get request method and Post method? (2)

Ans. The GET request is made by a client to read resources that is HTML documents and images currently residing at the webserver.

In GET, entire form submission can be encapsulated in one url like hyperlink. All data , we are passing to the server will dispalyed in the request String.

Query length is limited to 260 charaters. Not secure faster, quick and easy.

POST, on the forwards sending large amount of information (Possibly mega bytes) to the server only once. Hence a POST request is literally a post.

In POST, your name or data value pairs inside the body of the HTTP request which makes the cleaner url. All data we are passing through the server will be hidden. user cannot see this information.

Q.3. (b) What are the different JSP implicit objects. Explain? (3)

Ans. JSP Implicit Objects are the Java objects that the JSP Container makes available to developers in each page and developer can call them directly without being explicitly declared. JSP Implicit Objects are also called pre-defined variables.

JSP supports nine Implicit Objects which are listed below:

The Request Object: The request object is an instance of a javax. servlet. http. Http Servlet Request object. Each time a client requests a page the JSP engine creates a new object to represent that request.

The request object provides methods to get HTTP header information including form data, cookies, HTTP methods etc.

The Response Object: The response object is an instance of a javax. servlet. http. Http Servlet Response object. Just as the server creates the request object, it also creates an object to represent the response to the client.

The Out Object: The out implicit object is an instance of a javax. servlet. jsp. Jsp Writer object and is used to send content in a response.

The initial Jsp Writer object is instantiated differently depending on whether the page is buffered or not. Buffering can be easily turned off by using the buffered='false' attribute of the page directive.

The Session Object: The session object is an instance of javax. servlet. http. Http Session and behaves exactly the same way that session objects behave under Java Servlets.

The Config Object: The config object is an instantiation of javax. servlet. Servlet Config and is a direct wrapper around the Servlet Config object for the generated servlet.

The Page Object: This object is an actual reference to the instance of the page. It can be thought of as an object that represents the entire JSP page.

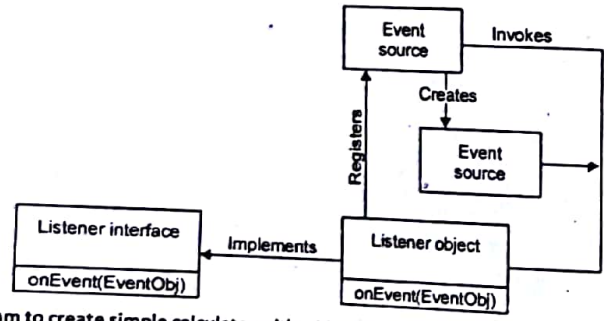
Q.3. (c) What is event delegation model and Write a program to create a simple calculator having four basic operations (Add, Subtract, Multiply, Divide) using swing package and event handing. (5)

Ans. Java's delegation event model

The event model is based on the Event Source and Event Listeners. Event Listener is an object that receives the messages / events. The Event Source is any object which creates the message / event. The Event Delegation model is based on – The Event Classes, The Event Listeners, Event Objects.

There are three participants in event delegation model in Java;

- Event Source – The class which broadcasts the events
- Event Listeners – The classes which receive notifications of events
- Event Object – The class object which describes the event.



Program to create simple calculator with add, sub, mul and div operation:

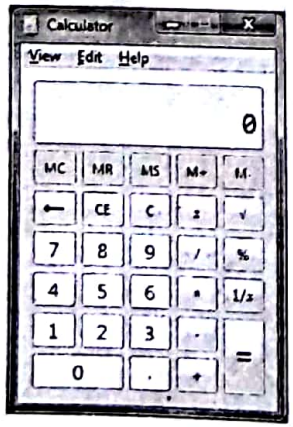
```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class Calculator extends JPanel implements ActionListener {

  private JTextField display = new JTextField("0");
  private double result = 0;
  private String operator = "=";
  private boolean calculating = true;

 public Calculator() {
   setLayout(new BorderLayout());
```



```
   display.setEditable(false);
   add(display, "North");

   JPanel panel = new JPanel();
   panel.setLayout(new GridLayout(4, 4));
   String buttonLabels = "789/456*123– 0.=+";
   for (int i = 0; i < buttonLabels.length(); i++) {
   JButton b = new JButton(buttonLabels.substring(i, i + 1));
   panel.add(b);
   b.addActionListener(this);
   }
   add(panel, "Center");
}

 public void actionPerformed(ActionEvent evt) {
   String cmd = evt.getActionCommand();
   if ('0' <= cmd.charAt(0) && cmd.charAt(0) <= '9' || cmd.equals(".")) {
     if (calculating)
       display.setText(cmd);
     else
     display.setText(display.getText() + cmd);
   calculating = false;
  } else {
   if (calculating) {
     if (cmd.equals("-")) {
       display.setText(cmd);
       calculating = false;
     } else
       operator = cmd;
   } else {
     double x = Double.parseDouble(display.getText());
     calculate(x);
     operator = cmd;
     calculating = true;
   }
  }
}

 private void calculate(double n) {
  if (operator.equals("+"))
    result += n;
  else if (operator.equals("-"))
    result – = n;
  else if (operator.equals("*"))
```

```
        result *= n;
      else if (operator.equals("/"))
        result /= n;
      else if (operator.equals("="))
        result = n;
      display.setText("" + result);
    }

    public static void main(String[] args) {
      JFrame.setDefaultLookAndFeelDecorated(true);
      JFrame frame = new JFrame();
      frame.setTitle("Calculator");
      frame.setSize(200, 200);
      frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
          System.exit(0);
        }
    } ) ;

        Container contentPane = frame.getContentPane();
        contentPane.add(new Calculator());
        frame.show();
      }
    }
```

Output:



Q.4. (a) (i) Layout Manager (3)

Ans. The various Layout manage are the following:

- Flow Layout
- Grid Layout
- Border Layout
- Card Layout

Flow Layout: Flow Layout is the default Layout manager. Flow Layout implements a simple layout style, which is similar to how words flow in a text editor.

Border Layout: It has narrow, fixed-midth components at the edges and one large once in the center. The four sides one referred to as north, south, east and west. The middle area is called the center.

Grid Layout: Grid Layout lays out components in a two-dimensional grid.

Card Layout: The Card Layout class is unique among the other layout managers in that it stores several different layouts.

Q.4. (a) (ii). JDBC (3)

Ans. Refer Q.4. (a) of End Term Examination-2015.

Q.4. (a) (iii). RMI (3)

Ans. The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows on object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the application using two objects stub and skeleton.

RMI uses stub and skeleton object for communication with the remote object.

A remote object is an object whose method can be invoked from another JVM.

Stub: The stub is an object, acts as a gateway for the client slide. All the outgoing requests are routed through it. It resides at the client side and represents the remote object.

Skeleton: The skeleton is an object, act as a gateway for the server side objects. All the incoming requests are routed through it.

Steps to wirte the RMI program

These are 6 steps to wirte the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application.

Q.4. (b) Write a program to read and print contents of file (4)

Ans. In the following program, we read a file called "temp.txt" and output the file line by line on the console.

```
import java.io.*;

public class Test {
  public static void main(String [] args) {

    // The name of the file to open.
    String fileName = "temp.txt";

    // This will reference one line at a time
    String line = null;
```

```
try {
   // FileReader reads text files in the default encoding.
   FileReader fileReader =
     new FileReader(fileName);
   // Always wrap FileReader in BufferedReader.
   BufferedReader bufferedReader =
     new BufferedReader(fileReader);
   while((line = bufferedReader.readLine()) != null) {
     System.out.println(line);
   }
   // Always close files.
   bufferedReader.close();
}
catch(FileNotFoundException ex) {
   System.out.println(
     "Unable to open file '" +
     fileName + "'");
}
catch(IOException ex) {
   System.out.println(
     "Error reading file '"
     + fileName + "'");
   // Or we could just do this:
   // ex.printStackTrace();
   }
  }
}
```

END TERM EXAMINATION [DEC. 2015]
FIFTH SEMESTER [B.TECH]
JAVA PROGRAMMING [ETCS–307]

Time. 3 Hours **M.M. : 75**

Note: *Q No. 1 is compulsory and any five questions from remaining.*

Q.1. (a) Define the characteristics of Java. **(5)**

Ans. Characteristics of java: There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Portable
7. Dynamic
8. Interpreted
9. Multithreaded
10. Distributed

Simple :

- Java is Easy to write and more readable and eye catching.
- Java has a concise, cohesive set of features that makes it easy to learn and use.
- Most of the concepts are drew from C++ thus making Java learning simpler.

Object-oriented

Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming (OOPs) is a methodology that simplify software development and maintenance by providing some rules.

Object Oriented Programming Language must have the following characteristics.

(1) Encapsulation (2) Polymorphism (3) Inheritance (4) Abstraction

Platform-independent: Java Language is platform-independent due to its hardware and software environment. Java code can be run on multiple platforms e.g. windows, Linux, sun Solaris, Mac/Os etc. Java code is compiled by the compiler and converted into byte code. This byte code is a platform independent code because it can be run on multiple platforms *i.e.* Write Once and Run Anywhere(WORA).

Secure:

- Java program cannot harm other system thus making it secure.
- Java provides a secure means of creating Internet applications.
- Java provides secure way to access web applications.

Java is secured because:

- No explicit pointer
- Programs run inside virtual machine sandbox.

Robust: Java was created as a strongly typed language. Data type issues and problems are resolved at compile-time, and implicit casts of a variable from one type to another are not allowed.

Memory management has been simplified java in two ways. First Java does not support direct pointer manipulation or arithmetic. This make it possible for a java program to overwrite memory or corrupt data.

• Java encourages error-free programming by being strictly typed and performing run-time checks.

Portable

• The feature of java "write once -run any where" make java portable. Many type of computers and operating systems are used for programs. By porting an interpreter for the Java Virtual Machine to any computer hardware/operating system, one is assured that all code compiled for it will run on that system. This forms the basis for Java's portability.

Dynamic

• Because it is interpreted , Java is an extremely dynamic language. At runtime, the java environment can extends itself by linking in classes that may be located on remote servers on a network(for example, the internet).

• At runtime, the java interpreter performs name resolution while linking in the necessary classes.

Interpreted:

• Java supports cross-platform code through the use of Java bytecode.

• Bytecode can be interpreted on any platform by JVM.

Multi-threaded: A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

Distributed: Java facilitates the building of distributed application by a collection of classes for use in networked applications. By using java's URL (Uniform Resource Locator) class, an application can easily access a remote server. Classes also are provided for establishing socket-level connections.

Q.1. (*b*) Differentiate between JAVA and C++ Object Oriented Language. (5)

Ans.

Java	C++
Java does not support pointers, templates, unions, operator overloading, structures etc.	C++ supports structures, unions, templates, operator overloading, pointers and pointer arithmetic.
Java support automatic garbage collection. It does not support destructors as C++ does.	C++ support destructors, which is automatically invoked when the object is destroyed.
Java does not support conditional compilation and inclusion.	Conditional inclusion (#ifdef #ifndef type) is one of the main features of C++.
Java has built in support for threads. In Java, there is a Thread class that you inherit to create a new thread and override the run() method.	C++ has no built in support for threads. C++ relies on non-standard third-party libraries for thread support.
Java doesn't provide multiple inheritance, at least not in the same sense that C++ does.	C++ does support multiple inheritance. The keyword virtual is used to resolve ambiguities during multiple inheritance if there is any.
Java has method overloading, but no operator overloading. The String class does use the + and += operators to concatenate strings and Stringexpressions use automatic type conversion, but that's a special built-in case.	C++ supports both method overloading and operator overloading.

Java has built-in support for documentation comments (/** ... */); therefore, Java source files can contain their own documentation, which is read by a separate tool usually javadoc and reformatted into HTML. This helps keeping documentation maintained in easy way.	C++ does not support documentation comments.
Java is interpreted for the most part and hence platform independent.	C++ generates object code and the same code may not run on different platforms.

Q.1. (*c*) Differentiate between package and Interface with examples. (5)

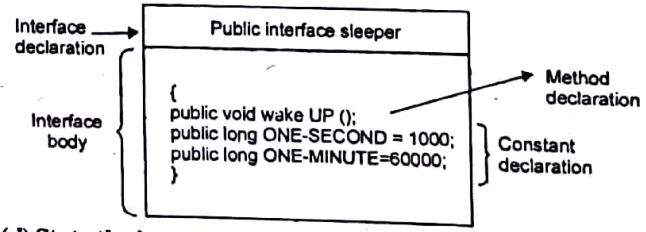
Ans. Packages are Java's way of grouping a variety of classes and/on interfaces together. The grouping is usually done according to functionally. Packages acts as "containers" for classes. Main advantages of storing classes and interface into package are:

(*a*) The classes contained in the packages are easily reserved.

(*b*) Packages provide a way to hide classes thus preventing other programs or packages from accessing classes.

(*c*) In packages, two classes in two different packages can have the same name.

Interface: An interface is a named collection of method declarations (without definitions). An interface can also include constant declarations. The following figure shows that an interface definition has two components: the interface declaration and the interffface body. The interface declaration declares various attributes about the interface such as its name and whether it extends another interface. The interface body contains the constant and method declarations within that interface.

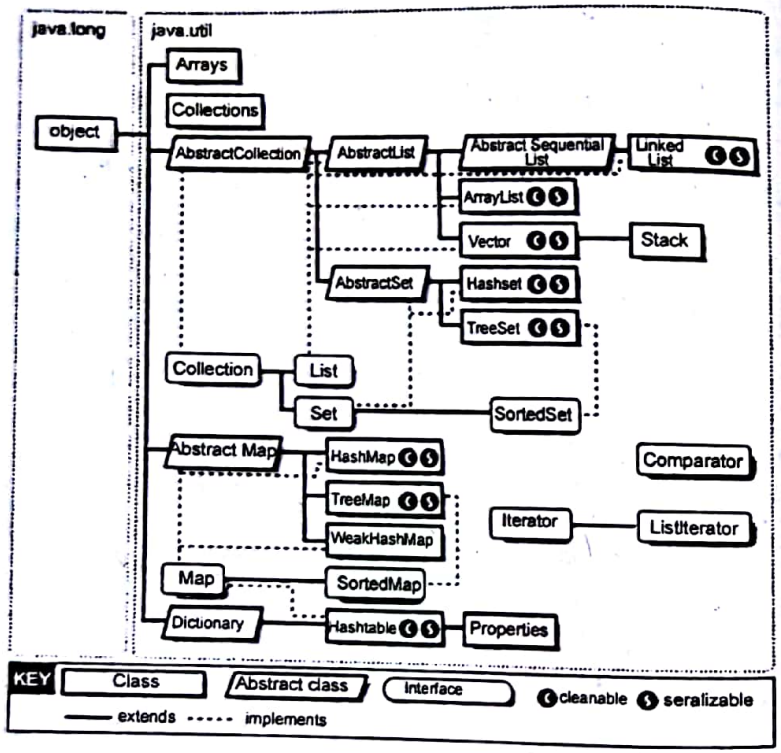


Q.1. (*d*) State the importance of utility package in JAVA. (5)

Ans. The java.util Package: The java.util package defines a number of useful classes, primarily collections classes that are useful for working with groups of objects. This package should not be considered merely a utility package that is separate from the rest of the language; in fact, Java depends directly on several of the classes in this package.

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1.	Public boolean add (Object element)	is used to insert an element in this collection.
2.	Public boolean add All (collection c)	is used to insert the specified collection elements in the invoking collection.
3.	Public boolean remove (Object element)	is used to delete an element from this collection.
4.	Public boolean remove All(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5.	Public Iterator iterator ()	returns an iterator.



Methods of Collection interface

The final Java package, java.util, contains a collection of utility classes.

Data Structure Classes: A variety of useful classes implementing standard computer science data structures: including BitSet, Dictionary, Hashtable, Stack and Vector. The java.util package also defines the Enumeration interface which provides a protocol for classes to count through a set of values.

Date: Use the Date class to create and manipulate calendar dates in a system-independent fashion.

String To kenizer: This class converts a String of text into its tokens.

Properties: This class implements persistent properties. The properties table contains key/value pairs where both the key and the value are Strings. This class is used by the System class to implement System properties.

Observer and Observable: Classes that implement the Observer interface can "watch" Observable objects for state changes. When an Observable object changes it notifies all of its Observers of the change.

Random-Number Generator: The Random class implements a random-number generator.

Enumeration: The Enumeration interface defines a generic programming interface for iterating through a set of values.

Q.1. (e) Explain the life cycle of Applet and its security concerns. (5)

Ans. Following are the methods.

1. init() method
2. start() method
3. paint() method
4. stop() method
5. destroy() method

These methods are known as life cycle methods. These methods are defined in java.applet.Applet class except paint(0 method. The paint() method is defined in java.awt. Component class, an indirect super class of Applet.

The life cycle methods are called as callback methods as they are called implicitly by the browser for the smooth execution of the applet. Browser should provide an environment known as container for the execution of the applet. Following are the responsibilties of the browser.

1. It should call the callback methods at appropriate times for the smmoth execution of the applet.

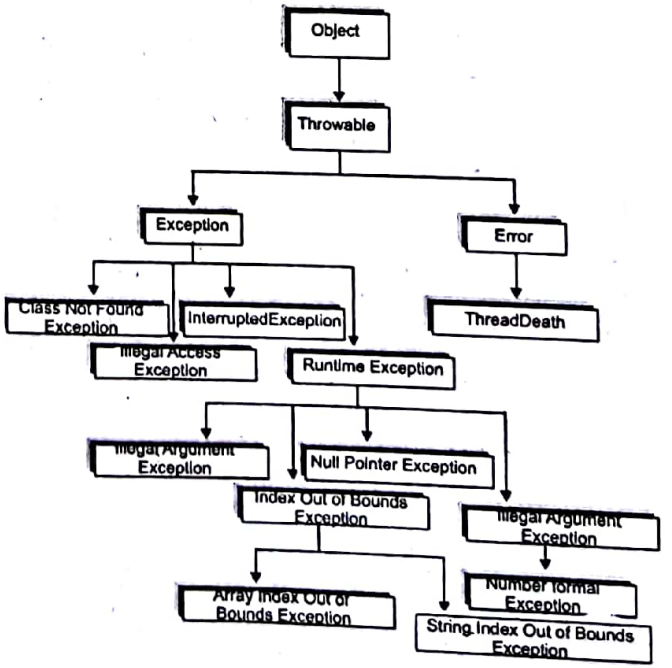
2. It is responsible to maintain to life cycle of the applet.

3. It should have the capability to communicate between applets, applet to JavaScript and HTML, applet to browser etc.

Q.2. (a) Describe the various types of exceptions and illustrate how they are handled. (9)

Ans. Exception: A Java Exception is an object that describes the exception that occurs in a program. When an exceptional events occurs in java, an exception is said to be thrown. The code that's responsible for doing something about the exception is called an **exception handler.**

Exception are categorized into 3 category



• **Checked Exception:** The exception that can be predicted by the programmer. **Example :** File that need to be opened is not found. These type of exceptions must be checked at compile time.

• **Unchecked Exception:** Unchecked exceptions are the class that extends RuntimeException. Unchecked exception are ignored at compile time. **Example :** ArithmeticException, NullPointerException, Array Index out of Bound exception. Unchecked exceptions are checked at runtime.

• **Error:** Errors are typically ignored in code because you can rarely do anything about an error. **Example :** If stack overflow occurs, an error will arise. This type of error is not possible handle in code.

Exception Handling Mechanism

In java, exception handling is done using five keywords,

1. Try 2. Catch 3. Throw 4. Throws 5. Finally

Exception handling is done by transferring the execution of a program to an appropriate exception handler when exception occurs.

• **Using try and catch:** Try is used to guard a block of code in which exception may occur. This block of code is called guarded region. A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in guarded code, the catch block that follows the try is checked, if the type of exception that occurred is listed in the catch block then the exception is handed over to the catch block which then handles it.

• **Example using Try and catch:**

```
class Excp
{
  public static void main(String args[])
  {
      int a,b,c;
      try
      {
      a=0;
      b=10;
      c=b/a;
      System.out.println("This line will not be executed");
      }
      catch(ArithmeticException e)
      {
      System.out.println("Divided by zero");
    }
    System.out.println("After exception is handled");
  }
}
```

output:

Divided by zero

After exception is handled

Throw Keyword: Throw keyword is used to throw an exception explicitly. Only object of Throwable class or its sub classes can be thrown. Program execution stops on encountering **throw** statement, and the closest catch statement is checked for matching type of exception.

Syntax :

throw *ThrowableInstance*

Example demonstrating throw Keyword

```
class Test
{
  static void avg()
  {
    try
    {
    throw new ArithmeticException("demo");
  }
  catch(ArithmeticException e)
  {
  System.out.println("Exception caught");
  }
}
  public static void main(String args[])
  {
    avg();
  }
}
```

In the above example the avg() method throw an instance of ArithmeticException, which is successfully handled using the catch statement.

Q.2. (*b*) Define multithreading with an example. (3.5)

Ans. Java is multithreaded programming language which means we develop multithreaded program using Java. A multithreaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs

By definition multitasking is when multiple processes share common processing resources such as a CPU. Multithreading extends the idea of multitaskig into applicaitions where you can subdivide specific operations wihin a single application into individual threads. Each of the threads can run in paralle. The OS divides processing time not only among different applications, but also among each thread within as application.

Multithreading enables you to write in way where multiple acivities can proceed concurrently in the same program.

Example on creating a thread by deriving from the thread class

```
public class Derived extends Thread
{
    public void run ( )
    {
    /* Your implementation of the
    thread here
    */
    }
}
Derived derivedThread = new Derived();
```

```
/* this call will create a new independent thread
   called derivedThread, and will start its processing
*/
derivedThread.start();
```

Q.3.(a) What is inheritance? How will you call parameterized constructor and override method from parent class in sub class? (9)

Ans. Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

Use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Syntax of Java Inheritance

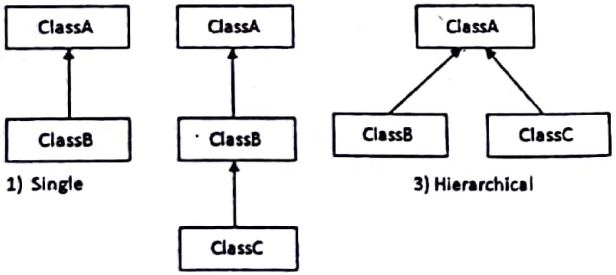
```
1. class Subclass-name extends Superclass-name
2. {
3. //methods and fields
4. }
```

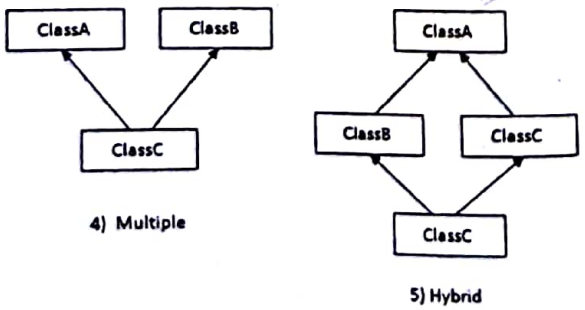
The **extends keyword** indicates that you are making a new class that derives from an existing class.

In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.





Calling parameterized constructor of parent class:

The super keyword can also be used to invoke the parent class constructor as given below:

```
Public class Student{
String name;
String studentId;
Public Student(String name,String studentId)
{
This.name=name;
This.studentId=studentId;
}
Public void printStudent()
{
System.out.println("name of the student is:" +name);
System.out.println("Id of the student is:" +studentId);
}
}
Public class GraduateStudent
{
Int noofYears_Course;
Public GraduateStudent(String name,String studentId, int noofYears_Course)
{
Super(name,studentId);
This.noofYears_Course=no of Years_Course;
}
}
```

The super class constructor is triggerd using super followed by brackets. Within the brackets, the parameters to the super class constructor have to be passed.

Calling parent class method:

Super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used in case subclass contains the same method as parent class as in the example given below:

```
1. class Person{
2. void message(){System.out.println("welcome");}
3. }
4.
5. class Student16 extends Person{
6. void message(){System.out.println("welcome to java");}
7.
8. void display(){
```

```
9. message();//will invoke current class message() method
10. super.message();//will invoke parent class message() method
11. }
12.
13. public static void main(String args[]){
14. Student16 s=new Student16();
15. s.display();
16. }
17. }
```

Output:welcome to java
welcome

Q.3. (b) What is an inner class? Give an example for it. **(3.5)**

Ans. Refer to Q.4(b) of First Term Examination 2015.

Q.4. (a) List out the steps to connect JDBC with MS-Access. **(6.25)**

Ans. JDBC connection to Microsoft Access: We make a connection using JDBC to a Microsoft Access database. This connection is made with the help of a JdbcOdbc driver. You need to use the following steps for making the connection to the data base.

Creating a Database

Step 1 : Open Microsoft Access and select **Blank data base** option and give the data base name as**File name** option

Step 2 : Create a table and insert your data into the table.

Step 3 : Save the table with the desired name; in this article we save the following records with the table name student.

Now Creating DSN of your data base

Step 4 : Open your **Control Panel** and than select **Administrative Tools.**

Step 5 : Click on **Data Source(ODBC)—>System DSN.**

Step 6 : Now click on **add** option for making a new DSN.select **Microsoft Access Driver (*.mdb. *.accdb)** and than click on **Finish**

Step 7 : Make your desired **Data Source Name** and then click on the **Select** option,as MYDSN

Step 8 : Now you select your data source file for storing it and then click **ok** and then click on**Create** and **Finish.**

Step 9 : Java program code

The main important steps in any one the database connecting program is: **Step 1:** specify the drivers which you are going to use in the program. For this invoke a method called as **forName()** which is present in the class **"Class".**

```
Class. forName ("sun.jdbc.odbc.JdbcOdbcDriver");
```

Step 2 : Start the connection to the Database. For this method **getConnection()** which is present in the class **DriverManager.** We need to pass the protocol,user id and password for the database. The url consists of three parts protocol, subprotocol, sourcename.

```
1. Class. for Name("sun.jdbc.odbc.JdbcOdbcDriver");
2. Connection con;
```



```
3. con = DriverManager.getConnection(url,userid,pswd);
```

Step 3 : Create an statement to execute the queries. For this method create Statement () which is present in the Connection class and we use the con object to call the method, Which returns an object of statement class.

```
1. Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
2. Connection con;
3. con = DriverManager.getConnection(url,userid,pswd);
4. Statement st = con.createStatement();
```

Step 4 : Execute the queries on the table. We use the methodexecute Query () which is present in the class Statement and we use the object st. It returns the set of results of the operation and we need to store them in an object of ResultSet.

```
1. Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
2. Connection con;
con = DriverManager.getConnection(url,userid,pswd);
3. Statement st = con.createStatement();
4. Resultset res = st.executeQuery("select * from emp");
```

Q.4. (b) Create and application for online shopping with JDBC. **(6.25)**

Ans.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ShoppingCartViewerCookie extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
res.setContentType("text/html");
PrintWriter out = res.getWriter();
// Get the current session ID by searching the received cookies
String sessionid = null;
Cookie[] cookies = req.getCookies();
if (cookies != null) {
for (int i = 0; i < cookies.length; i++) {
if (cookies[i].getName().equals("sessionid")) {
sessionid = cookies[i].getValue();
break;
}
}
}
// If the session ID wasn't sent, generate one
// Then be sure to send it to the client with the response
if (sessionid == null) {
sessionid = generateSessionId();
Cookie c = new Cookie("sessionid", sessionid);
```

```
res.addCookie(c);
}
out.println("<HEAD><TITLE>Current Shopping Cart Items</TITLE></HEAD>");
out.println("<BODY>");
// Cart items are associated with the session ID
String[] items = getItemsFromCart(sessionid);
// Print the current cart items.
out.println("You currently have the following items in your cart:<BR>");
if (items == null) {
out.println("<B>None</B>");
}
else {
out.println("<UL>");
for (int i = 0; i < items.length; i++) {
out.println("<LI>" + items[i]);
}
out.println("</UL>");
}
// Ask if they want to add more items or check out.
out.println("<FORM ACTION=\"/servlet/ShoppingCart\" METHOD=POST>");
out.println("Would you like to<BR>");
out.println("<INPUT TYPE=submit VALUE=\" Add More Items \">");
out.println("<INPUT TYPE=submit VALUE=\" Check Out \">");
out.println("</FORM>");
// Offer a help page.
out.println("For help, click <A HREF=\"/servlet/Help" +
"?topic=ShoppingCartViewerCookie\">here</A>");
out.println("</BODY></HTML>");
}
private static String generateSessionId() {
String uid = new java.rmi.server.UID().toString(); // guaranteed unique
return java.net.URLEncoder.encode(uid); // encode any special chars
}
private static String[] getItemsFromCart(String sessionid) {
// Not implemented
}
}
```

Q.5. Explain life cycle of a thread with help of a diagram. How do the wait and notify All/notify methods enable cooperation between threads. (12.5)

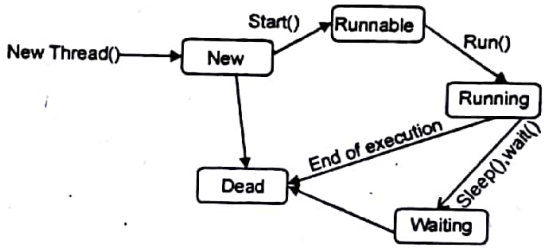
Ans. Life cycle of a Thread (Thread States):

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.

• Thread having below states.

1. New State
2. Ready State
3. Running State
4. Dead State
5. Non Runnable States

Life cycle of thread in java with diagram



1. New State:

• A thread has been created but not started yet. A thread will be started by calling its start() method.

2. Runnable State:

• This state is also called ready to run stage also called queue. A thread starts in runnable state by calling start() method.

• The Thread scheduler decides which thread runs and how long.

3. Running State:

• If a Thread is executing that means Thread is in Running stage.

4. Dead State:

• Once a Thread reached dead state it can not run again.

5. Non runnable or waiting States:

• A Running Thread transit to one of the non runnable states, depending upon the circumstances.

• A Thread remains non runnable until a special transition occurs.

• A Thread does not go directly to the running state from non runnable state.

• But transits first to runnable state.

1. Sleeping: The Threads sleeps for specified amount of time.

2. Blocked for I/O: The Thread waits for a blocking operation to complete.

3. Blocked for join completion: The Thread waits for completion of another Thread.

4. Waiting for notification: The Thread waits for notification another Thread.

5. Blocked for lock acquisition: The Thread waits to acquire the lock of an object.

Cooperation between threads: There are simply three methods and a little trick which makes thread communication possible. First let's see all the three methods listed below:

S.N.	Methods with Description
1.	**public void wait()** Causes the current thread to wait until another thread invokes the notify().
2.	**public void notify()** Wakes up a single thread that is waiting on this object's monitor.
3.	**public void notifyAll()** Wakes up all the threads that called wait( ) on the same object.

These methods have been implemented as **final** methods in Object, so they are available in all the classes. All three methods can be called only from within a **synchronized** context.

This example shows how two thread can communicate using **wait()** and**notify()** method. You can create a complex system using the same concept.

```
class Chat {
    boolean flag = false;

    public synchronized void Question(String msg) {
      if (flag) {
        try {
          wait();
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
      System.out.println(msg);
      flag = true;
      notify();
    }

    public synchronized void Answer(String msg) {
      if (!flag) {
        try {
          wait();
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }

      System.out.println(msg);
      flag = false;
      notify();
```



```
    }
}

class T1 implements Runnable {
    Chat m;
    String[] s1 = { "Hi", "How are you ?", "I am also doing fine!" };

    public T1(Chat m1) {
    this.m = m1;
    new Thread(this, "Question").start();
  }

  public void run() {
    for (int i = 0; i < s1.length; i++) {
      m.Question(s1[i]);
    }
  }
}

class T2 implements Runnable {
    Chat m;
    String[] s2 = { "Hi", "I am good, what about you?", "Great!" };

  public T2(Chat m2) {
    this.m = m2;
    new Thread(this, "Answer").start();
  }

  public void run() {
    for (int i = 0; i < s2.length; i++) {
      m.Answer(s2[i]);
    }
  }
}
public class TestThread {
  public static void main(String[] args) {
    Chat m = new Chat();
    new T1(m);
    new T2(m);
  }
```

```
}
```

When above program is complied and executed, it produces following result:

```
Hi
Hi
How are you ?
I am good, what about you?
I am also doing fine!
Great!
```

Q.6. Given two different array of size M and N in sorted asceeding order, write a JAVA program to merge them into a single sorted array with the condition that merged array should be in ascending order. **(12.5)**

Ans. Steps to merge two sorted arrays:

1. If the first array is empty (*i.e.* a1.length <=0), then copy the elements of the array arr 2 to the result array return the result

2. ru2. If the second array is empty (*i.e* arr2.length < = 0), then copy all the elements of the array arr1 to the result array. return the result

3. Compare the element in the arr1 with element in the arr2 and copy the small element to result ie. if the arr1[i]<arr2[j], copy arr1[i] to result, and increase **i** , otherwise copy arr2[j] to result, and increase **j**

4. Copy the **remaining** elements either in arr1 OR in arr2 to result, return the result

```
package com.javaonline;

class Merge2SortedArrays
{
public static void main(String[] args) {
int arr1[]={4,6, 9, 20, 56};
int arr2[]={1, 7, 25, 45, 70};
int[] result=merge(arr1, arr2);
for (int j=0; j<result.length;j++)
System.out.print(result[j]+ " ");
}

static int[] merge(int[] arr1, int[] arr2)
{
Int m=arr1.length;
int arr1_Length=m;
int n=arr2.length;
int arr2_Length=n;
int[] result = new int[arr1_Length + arr2_Length];
int i=0, j = 0;

for(int k = 0 ; k< (arr1_Length + arr2_Length);k++)
{
//If arr1 is empty, copy the elements of the array arr2 to the result array .
```



```
//When i equals the length of array arr1 , copy the remaining elements in the array arr2 to result
if ( i >= arr1_Length ) //for step1 & 4
{
result[k] = arr2[j];
j ++;
}
//If arr2 is empty, copy the elements of the array arr1 to the result array .
//When j equals the length of array arr2 , copy the remaining elements in the array arr1 to result
else if ( j >= arr2_Length) // For step 2 &amp;amp; 4
{
result[k] = arr1[i];
i ++;
}
else
{

if ( arr1[i] < arr2[j]) // step 3
{
result[k] = arr1[i];
i ++;
}
else
{
result[k] = arr2[j];
j ++;
}
}
}
return result;
}
}
```

Output:

```
1 4 6 7 9 20 25 45 56 70
```

Q.7. How to declare and initialize a string in JAVA and also explain the difference string handling functions with suitable example. **(12.5)**

Ans. String is probably the most commonly used class in java library. String class is encapsulated under Java.lang package. In java, every string that you create is actually an object of type **String**.

One important thing to notice about string object is that string objects are immutable that means once a string object is created it cannot be altered.

An array of characters works same as java string. For example:

1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};

2. String s=new String(ch);

How to create String object?

There are two ways to create String object:

1. By string literal

2. By new keyword

(1) String Literal

Java String literal is created by using double quotes. For Example:

1. String s="welcome"; Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";

2. String s2="Welcome";//will not create new instance

2. By new keyword

1. String s=new String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal(non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable's will refer to the object in heap(non pool).

Java String class methods

String Length:

The length() method, which returns the number of characters contained in the string object.

charAt()

charAt() function returns the character located at the specified index.

String str = "studytonight";

System.out.println(str.charAt(2));

Output : u

substring()

substring() method returns a part of the string. **substring()** method has two forms,

public String substring(int begin);

public String substring(int begin, int end);

The first argument represents the starting point of the substring. If the substring() method is called with only one argument, the substring returned, will contain characters from specified starting point to the end of original string.

toLowerCase() method returns string with all uppercase characters converted to lowercase.

String str = "ABCDEF";

System.out.println(str.toLowerCase());

Output : abcdef

valueOf()

Overloaded version of valueOf() method is present in String class for all primitive data types and for type Object.

NOTE : valueOf() function is used to convert **primitive data types** into Strings.

But for objects, valueOf() method calls **toString()** function.

toString()

toString() method returns the string representation of the object used to invoke this method. **toString()** is used to represent any Java Object into a meaningful string representation. It is declared in the **Object class**, hence can be overrided by any java class. (Object class is super class of all java classes.)

Q.8. Write short note on any two: **(6.25×2 = 12.5)**

(a) Development of client server application.

Ans. For this type of application we employ the use of sockets. A socket is the one end-point of a two way communication link between two program running over the network. One can run the two programs on the same computer. To distinguish different services a numbering convention uses integers numbers called port numbers are used. A socket is a complex data structure that contains an internet address and a port number.

A network connection is initiated by a client program when it creates a socket for the communication with the server.

1. When the server receives a connection request on its specific server post, it create a new socket for it and binds a port number to it.
2. It sends the new post number to client to inform it the connection is established.
3. The server goes on new by listening on two ports:
   - it waits for new incoming connection requests on its specific port, and
   - it reads and writes messages on established connection (on new port) with the accepted client.

The server communicates with the client by needing from and writing to the new port.

A simple client/server application.

The Client

This is a simple client which reads a line from the standard input and stands it to the echo server. The client keeps then reading from the socket fill it receives the message "OK" from the server, once it receives the "OK" message then it breaks the server, once it receives the "OK" message then it breaks.

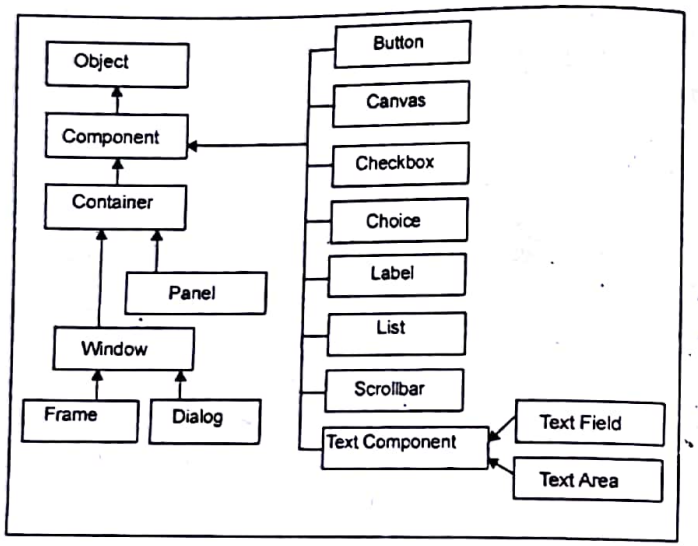
Q. 8. (b) AWT component.

Ans. Java AWT Components:

Java AWT (Abstract Windowing Toolkit) is *an API to develop GUI or window-based application in java.*

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components uses the resources of system.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.



Container: The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window: The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel: The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame: The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Java AWT Example: To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

Simple example of AWT by inheritance

```
1. import java.awt.*;
2. class First extends Frame{
3. First(){
4. Button b=new Button("click me");
5. b.setBounds(30,100,80,30);// setting button position
6.
7. add(b);//adding button into frame
8. setSize(300,300);//frame size 300 width and 300 height
9. setLayout(null);//no layout manager
10. setVisible(true);//now frame will be visible, by default not visible
11. }
12. public static void main(String args[]){
13. First f=new First();
14. }}
```

Q.8. (c) Organization of JVM.

Ans. Refer Q.2. (*b*) First Term Examination 2015.

FIRST TERM EXAMINATION
FIFTH SEMESTER B.TECH. [SEPTEMBER 2016]
JAVA PROGRAMMING [ETCS-307]

Time : 1.5 hrs. **M.M. : 30**

Note: *Attempt any three questions including Q. No. 1 which is compulsory.*

Q.1. (*a*) Explain all possible uses of final keyword in java.

Ans. The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

(*a*) Variable

(*b*) Method

(*c*) Class

The final keyword can be applied with the variables, a final variable that have no value is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

Q.1. (*b*) Explain the difference between interface and abstract class.

Ans.

OOPS interface vs Abstract class

Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does not contains data member	Abstract class contains Data member
Interface does not contains constructors	Abstract class contains constructors
An interface only contain incomplete memeber (signature of member)	An abstract class contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifier for the subs, functions, properties
Member of interface can not be static	Only complete Member of abstract class can be static

Q.1. (*c*) What are access specifiers and explain access scope of a protected method?

Ans: There are two types of modifiers in java: access modifiers and non-access modifiers.The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

(*a*) Private

(*b*) Default

(*c*) Protected

(*d*) Public

Protected access modifier

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
1. //save by A.java
2. package pack;
3. public class A{
4. protected void msg(){System.out.println("Hello");}
5. }
1. //save by B.java
2. package mypack;
3. import pack.*;
4.
5.      class B extends A{
6. public static void main(String args[]){
7.          B obj = new B();
8.                  obj.msg();
9.                          }
10.                         }
```

Output:Hello

Q.1. (d) List java's primitive datatypes along with their size and default values.

Ans. Primitive Data Types

There are eight primitive datatypes supported by Java. Primitive datatypes are predefined by the language and named by a keyword.

Type	Size in Bytes	Range
Byte	1 byte	-128 to 127
Short	2 bytes	-32,768 to 32,767
Int	4 bytes	-2,147,483,648 to 2,147,483, 647
Long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) *Java implements IEEE 754 standard*
Double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)
Char	2 byte	0 to 65,536 (unsigned)

Boolean not precisely defined* true or false

Q.1. (e) Explain the use of this and super in java classes.

Ans: In java, this is a **reference variable** that refers to the current object.

Usage of java this keyword

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of java super Keyword

1.super can be used to refer immediate parent class instance variable.

2.super can be used to invoke immediate parent class method.

3.super() can be used to invoke immediate parent class constructor.

Q.2. (a) What is a package and list five java packages along with two classes each.

Ans: 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.

Simple example of java package

The **package keyword** is used to create a package in java.

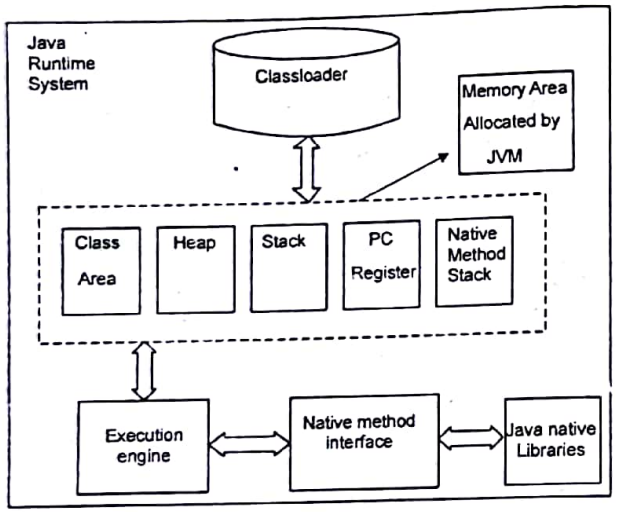
```
1. //save as Simple.java
2. package mypack;
3. public class Simple{
4.  public static void main(String args[]){
5.    System.out.println("Welcome to package");
6.  }
7. }
```

five java packages along with two classes:

1. Lang : Number ,Object
2. Awt : TextField,TextArea
3. Net: Inet Address,Socket
4. Io: DataInputStream and Buffered Reader
5. Util: Date,Scanner

Q.2. (b) Explain JVM architecture.

Ans: JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).



1. Classloader; Classloader is a subsystem of JVM that is used to load class files.

2. Class(Method) Area: Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3. Heap: It is the runtime data area in which objects are allocated.

4. Stack: Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5. Program Counter Register: PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

6. Native Method Stack: It contains all the native methods used in the application.

7. Execution Engine

It contains:

1. A virtual processor

2. Interpreter: Read bytecode stream then execute the instructions.

3. Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term compiler refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

Q.2. (c) Write aprogram toshow how to create user defined exception by extending Exception Class.

Ans:

```
class MyException extends Exception {
  String s1;
  MyException(String s2) {
    s1 = s2;
  }
  @Override
  public String toString() {
    return ("Output String =" +s1);
  }
}
public class NewClass {
  public static void main(String args[]) {
    try {
      throw new MyException("Custom message");
    } catch(MyException exp) {
      System.out.println(exp);
    }
  }
}
```

Q.3. (a) What is sandbox model in java.

Ans: A security measure in the Java development environment. The sandbox is a set of rules that are used when creating an applet that prevents certain functions when the applet is sent as part of a Web page. When a browser requests a Web page with applets, the applets are sent automatically and can be executed as soon as the page arrives in the browser. If the applet is allowed unlimited access to memory and operating system resources, it can do harm in the hands of someone with malicious intent. The sandbox creates an environment in which there are strict limitations on what system resources the applet can request or access. Sandboxes are used when executable code comes from unknown or untrusted sources and allow the user to run untrusted code safely.

The Java sandbox relies on a three-tiered defense. If any one of these three elements fails, the security model is completely compromised and vulnerable to attack:

- **Byte code verifier** — This is one way that Java automatically checks untrusted outside code before it is allowed to run. When a Java source program is compiled, it compiles down to platform-independent Java byte code, which is verified before it can run. This helps to establish a base set of security guarantees.
- **Applet class loader** — All Java objects belong to classes, and the applet class loader determines when and how an applet can add classes to a running Java environment. The applet class loader ensures that important elements of the Java run-time environment are not replaced by code that an applet tries to install.
- **Security manager** — The security manager is consulted by code in the Java library whenever a dangerous operation is about to be carried out. The security manager has the option to veto the operation by generating a security exception.

Q.3. (b) Explain difference between StringBuffer and String Tokenizer with example?

Ans: The **java.lang.StringBuffer** class is a thread-safe, mutable sequence of characters. Following are the important points about StringBuffer "

- A string buffer is like a String, but can be modified.
- It contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.
- They are safe for use by multiple threads.
- Every string buffer has a capacity.

Class Declaration

Following is the declaration for **java.lang.StringBuffer** class—

```
public final class StringBuffer
  extends Object
    implements Serializable, CharSequence
```

Class constructors

S.N.	Constructor and Description
1.	**StringBuffer()** This constructs a string buffer with no characters in it and an initial capacity of 16 characters.
2.	**StringBuffer(CharSequence seq.)** This constructs a string buffer that contains the same characters as the specified CharSequence.
3.	**StringBuffer(int capacity)** This constructs a string buffer with no characters in it and the specified initial capacity.
4.	**StringBuffer(String str)** This constructs a string buffer initialized to the contents of the specified string.

The **java.util.StringTokenizer** class allows an application to break a string into tokens.

- This class is a legacy class that is retained for compatibility reasons although its use is discouraged in new code.
- Its methods do not distinguish among identifiers, numbers, and quoted strings.
- This class methods do not even recognize and skip comments.

Class declaration

Following is the declaration for **java.util.StringTokenizer** class:

```
publicclassStringTokenizer
extendsObject
implementsEnumeration<Object>
```


Class constructors

S.N.	Constructor and Description
1.	**StringTokenizer(String str)** This constructor a string tokenizer for the specified string.
2.	**StringTokenizer(String str, String delim)** This constructor constructs string tokenizer for the specified string.
3.	**StringTokenizer(String str, String delim, boolean returnDelims)** This constructor constructs a string tokenizer for the specified string.

Q.3. (c) Explain applet life cycle and write a program to create an applet with moving banner.

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

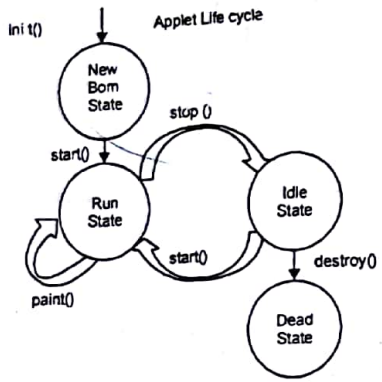
Life Cycle of an Applet

Four methods in the Applet class are:

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

A "Hello, World" Applet

```
import java.applet.*;
import java.awt.*;
public classHelloWorldAppletextendsApplet{
public void paint (Graphics g){
   g.drawString ("Hello World",25,50);
}
}
import java.awt.*;
import java.applet.*;
public classSampleBanner extendsApplet implements Runnable{
String str = "This is a simple Banner";
Thread t ;
boolean b;
publicvoid init(){
```




```
    setBackground(Color.gray);
    setForeground(Color.yellow);
}
publicvoid start(){
    t =newThread(this);
    b =false;
    t.start();
}
publicvoid run (){
char ch;
for(;;){
try{
      repaint();
Thread.sleep(250);
      ch = str.charAt(0);
      str = str.substring(1, str.length());
      str = str + ch;
}
catch(InterruptedException e){}
}
}
publicvoid paint(Graphics g){
   g.drawRect(1,1,300,150);
   g.setColor(Color.yellow);
   g.fillRect(1,1,300,150);
   g.setColor(Color.red);
   g.drawString(str,1,150);
```



```
}
}
```

Q.4. (a) Explain ways of doing multithreading in java.

Ans. Refer Q.2. (b) of End Term Examination 2015.

Q.4. (*b*) What is need of synchronization and how it is done in java?

Ans. Synchronization in java is the capability *to control the access of multiple threads to any shared resource.*

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
   1. Synchronized method.
   2. Synchronized block.
   3. static synchronization.
2. Cooperation (Inter-thread communication in java)

Q.4. (*c*) Explain concept of inner class and anonymous inner class with example?

Ans. Java Anonymous inner class:

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

Java anonymous inner class example using class

```
1. abstract class Person{
2. abstract void eat();
3. }
4.  class TestAnonymousInner{
5.  public static void main(String args[]){
6.   Person p=new Person(){
```

```
7.  void eat(){System.out.println("nice fruits");}
8.  };
9.  p.eat();
10. }
11. }
```

Java Inner Class contains

1. Java Inner classes
2. Advantage of Inner class
3. Difference between nested class and inner class

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

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

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

Syntax of Inner class

```
1. class Java_Outer_class{
2.  //code
3.  class Java_Inner_class{
4.   //code
5.  }
6. }
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

(1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

(2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

(3) **Code Optimization**: It requires less code to write.

END TERM EXAMINATION [DECEMBER 2016]
FIFTH SEMESTER [B.TECH]
JAVA PROGRAMMING [ETCS–307]

Time : 3 hrs. **M.M. : 75**

Note: *Attempt any five questions including Q.No.1 which is compulsory*

Q.1. (a) Define Constructor.Write various type of constructors.

Ans. Constructor in java is a *special type of method* that is used to initialize the object.

Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are three types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor
3. Copy constructor

1. Default Constructor: Default Constructor is also called as Empty Constructor which has no arguments and It is Automatically called when we creates the object of class but Remember name of Constructor is same as name of class and Constructor never declared with the help of Return Type. means we cant Declare a Constructor with the help of void Return Type, if we never Pass or Declare any Arguments then this called as the Copy Constructors.

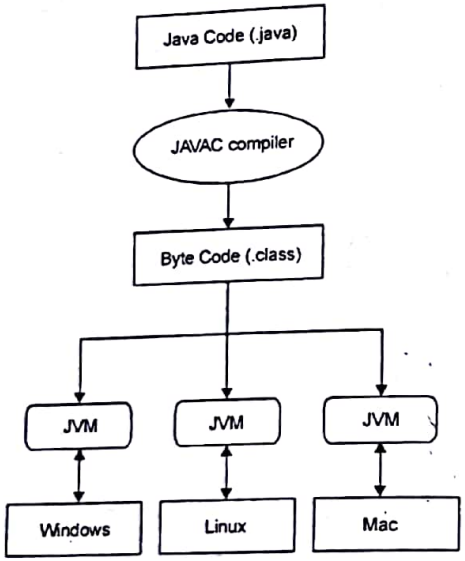
2. Parameterized Constructor: This is another type Constructor which has some arguments and same name as class name but it uses some arguments So For this We have to create object of Class by passing some arguments at the time of creating object with the name of class. When we pass some arguments to the Constructor then this will automatically pass theaArguments to the Constructor and the values will retrieve by the Respective Data Members of the Class.

3. Copy Constructor: This is also another type of Constructor. In this Constructor we pass the object of class into the another Object of Same Class. As name Suggests you Copy, means Copy the values of one Object into the another Object of Class .This is used for Copying the values of class object into an another object of class So we call them as Copy Constructor and For Copying the values We have to pass the name of object whose values we wants to Copying and When we are using or passing an Object to a Constructor then we must have to use the and Ampersand or Address Operator.

Q.1. (b) What is meant bytecode in java ?

Ans. By tecode is nothing but the intermediate representation of Java source code which is produced by the Java compiler by compiling that source code. This byte code is an machine independent code. It is not an completely a compiled code but it is an intermediate code somewhere in the middle which is later interpreted and executed by

JVM.Bytecode is a machine code for JVM.But the machine code is platform specific whereas bytecode is platform independent that is the main difference between them.It is stored in .class file which is created after compiling the source code.Most developers although have never seen byte code (nor have ever wanted to see it!) One way to view the byte code is to compile your class and then open the .class file in a hex editor and translate the bytecodes by referring to the virtual machine specification. A much easier way is to utilize the command-line utility javap. The Java SDK from Sun includes the javap disassembler, that will convert the byte codes into human-readable mnemonics.



Q.1. (c) Explain about creation of classes and objects in java with example.

Ans: Object in Java

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.

An object has three characteristics:

State: represents data (value) of an object.

Behavior: represents the behavior (functionality) of an object such as deposit, withdraw etc.

Identity: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.

Object is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

1. fields
2. methods
3. constructors
4. blocks
5. nested class and interface

Syntax to declare a class:

```
class<class_name>{
field;
method;
}
```

Q.1. (d) Program on thread.

Ans. Java Thread Example by extending Thread class

```
1. class Multi extends Thread{
2. public void run(){
3. System.out.println("thread is running...");
4. }
5. public static void main(String args[]){
6. Multi t1=new Multi();
7. t1.start();
8. }
9. }
```

Output:thread is running...

Java Thread Example by implementing Runnable interface

```
1. Class Multi3 implements Runnable{
2. Public void run(){
3. System.out.println("thread is running...");
4.                                    }
5.
6. Public static void main(String args[]){
7. Multi3 m1 = now Multi3();
8. Thread t1 = new Thread(m1);
9. t1.start();
10.                                   }
11.                                   }
```

Output:thread is running...

Q.1. (e) Write short note on buffered Streams.

Ans. Buffered input streams read data from a memory area known as a *buffer*; the native input API is called only when the buffer is empty. Similarly, buffered output streams write data to a buffer, and the native output API is called only when the buffer is full.

A program can convert an unbuffered stream into a buffered stream using the wrapping idiom we've used several times now, where the unbuffered stream object is passed to the constructor for a buffered stream class. Here's how you might modify the constructor invocations in the CopyCharacters example to use buffered I/O:

inputStream = new BufferedReader(new FileReader("xanadu.txt"));

outputStream = new BufferedWriter(new FileWriter("characteroutput.txt"));

There are four buffered stream classes used to wrap unbuffered streams: Buffered Input Stream and BufferedOutputStream create buffered byte streams, while Buffered Reader and BufferedWriter create buffered character streams.

Q.2. (a) Write a program to create vector with seven elements****

Ans: Below example shows how to copy or create a vector with another collection object. In the code we have created an ArrayList and by using addAll() method, we can copy another collection object.

Code:

```
package com.java2novice.vector;
importjava.util.ArrayList;
importjava.util.List;
importjava.util.Vector;
public class MyVectorNewCollection {
public static void main(String a[]){
    Vector<String>vct = new Vector<String>();
    //adding elements to the end
vct.add("First");
vct.add("Second");
vct.add("Third");
vct.add("Random");
System.out.println("Actual vector:"+vct);
    List<String> list = new ArrayList<String>();
list.add("one");
list.add("two");
vct.addAll(list);
System.out.println("After Copy:" +vct);
  }
}
```

Output:

Actual vector: [First, Second, Third, Random]

After Copy: [First, Second, Third, Random, one, two]

Q.2. (b) Explain the following methods related to thread(i) wait()(ii) sleep()

Ans. Sleep(): It is a static method on Thread class. It makes the current thread into the "Not Runnable" state for specified amount of time. During this time, the thread keeps the lock (monitors) it has acquired.

Wait(): It is a method on Object class. It makes the current thread into the "Not Runnable" state. Wait is called on a object, not a thread. Before calling wait() method, the object should be synchronized, means the object should be inside synchronized block. The call to wait() releases the acquired lock.

	Sleep	Wait
1. Class belongs	java.lang.Thread	java.lang.Object
2. Context	Called from any context	Only synchronized context
2. Locking	Does not release the lockfor specified time or until interrupt.	Releases the lock
4. Wake up Condition	When time expires or due to interruption	Awake by call to notify ()or notifyAll() method
5. Execution	Execution of sleep will pause the current running thread not the object on which it is called.	Thread wait() continues till a specific condition holds true

Q.3. (a) Describe the levels of access protection available for packages.

Ans. Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors.

The four access levels are:

1. Visible to the package, the default. No modifiers are needed.
2. Visible to the class only (private).
3. Visible to the world (public).
4. Visible to the package and all subclasses (protected).

• Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc., A variable or method declared without any access control modifier is available to any other class in the same package.

• If you want only objects in the same class to be able to get or set the value of a field or invoke a method, declare it private.

• If you want any object at all to be able to call a method or change a field, declare it public.

• If you want access restricted to subclasses and members of the same package, declare it protected.

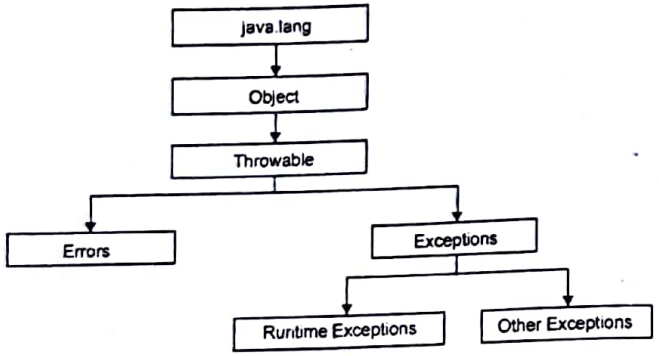
Q.3. (b) What do you mean by Exception handling?Explain the use of various keywords used for exception handling with suitable example.

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

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

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

The Exception class has two main subclasses: IOException class and Runtime Exception Class.



Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Syntax of java try-catch

```
1. try{
2. //code that may throw exception
3. }catch(Exception_class_Name ref){}
```

Syntax of try-finally block

```
1. try{
2. //code that may throw exception
3. }finally{}
```

Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only.

The Throws/Throw Keywords

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

Difference between throws and throw keywords, *throws* is used to postpone the handling of a checked exception and *throw* is used to invoke an exception explicitly.

The following method declares that it throws a RemoteException.

Example

```
import java.io.*;
publicclassclassName{
publicvoid deposit(double amount)throwsRemoteException{
// Method implementation
thrownewRemoteException();
}
// Remainder of class definition
}
```

Q.4. (a) Explain different methods for changing priorities for threads.

Ans. Java Thread Priority :

1. Logically we can say that threads run simultaneously but practically its not true, only one Thread can run at a time in such a ways that user feels that concurrent environment.

2. Fixed priority scheduling algorithm is used to select one thread for execution based on priority.

Example #1 : Default Thread Priority

get Priority() method is used to get the priority of the thread.

```
package com.c4learn.thread;
publicclassThreadPriorityextends Thread {
publicvoid run(){
System.out.println(Thread.currentThread().getPriority());
}
publicstaticvoid main(String[]args)
throwsInterrupted Exception{
ThreadPriority t1 = newThreadPriority();
ThreadPriority t2 = newThreadPriority();
t1.start();
t2.start();
}
}
```

Output :

```
5
5
```

default priority of the thread is 5.

Each thread has norma , iority at the time of creation. We can change or modify the thread priority in the following example 2.

Example #2 : Setting Priority

```
package com.c4learn.thread;
publicclassThreadPriorityextends Thread {
publicvoid run(){
StringtName=Thread.currentThread().getName();
IntegertPrio=Thread.currentThread().getPriority();
System.out.println(tName+ "has priority" +tPrio);
}
publicstaticvoid main(String[]args)
throwsInterruptedException{
ThreadPriority t0 =newThreadPriority();
ThreadPriority t1 =newThreadPriority();
ThreadPriority t2 =newThreadPriority();
t1.setPriority(Thread.MAX_PRIORITY);
t0.setPriority(Thread.MIN_PRIORITY);
t2.setPriority(Thread.NORM_PRIORITY);
t0.start();
t1.start();
t2.start();
}
}
```

Output :

Thread-0 has priority 1

Thread-2 has priority 5

Thread-1 has priority 10

Explanation of Thread Priority :

1. We can modify the thread priority using the **setPriority()** method.
2. Thread can have integer priority between 1 to 10
3. Java Thread class defines following constants –
4. At a time many thread can be ready for execution but the thread with highest priority is selected for execution
5. Thread have default priority equal to 5.

Thread : Priority and Constant

Thread Priority	Constant
MIN_PRIORITY	1
MAX_PRIORITY	10
NORM_PRIORITY	5

Q.4. (b) Write a code for creating a closeable window using AWT classes to receive following inputs.

UserName,Sex,Qualification,Address,Save Button

Ans.

```
import java.awt.*;
importjava.applet.*;
importjava.awt.event.*;
public class student extends Frame implements ActionListener
{String msg;
 Button b1 = new Button("save");
 Label l11 = new Label("Student details",Label.CENTER);
 Label l1 = new Label("Name:",Label.LEFT);
 Label l2 = new Label("age:",Label.LEFT);
 Label l3 = new Label("Sex(M/F):",Label.LEFT);
 Label l4 = new Label("Address:",Label.LEFT);
 Label l5 = new Label("Course:",Label.LEFT);
 Label l6 = new Label("Semester:",Label.LEFT);
 Label l7 = new Label("",Label.RIGHT);
 TextField t1 = new TextField();
 Choice c1 = new Choice();
 CheckboxGroupcbg = new CheckboxGroup();
 Checkbox ck1 = new Checkbox("Male",false,cbg);
 Checkbox ck2 = new Checkbox("Female",false,cbg);
Text Area t2 = new TextArea ("",180,90, TextArea. SCROLLBARS_ VERTICAL_
ONLY);
 Choice course = new Choice();
 Choice sem = new Choice();
 Choice age = new Choice();
public student()
{addWindowListener(new myWindowAdapter());
 setBackground(Color.cyan);
 setForeground(Color.black);
 setLayout(null);
 add(l11);
 add(l1);
 add(l2);
 add(l3);
 add(l4);
 add(l5);
 add(l6);
 add(l7);
```

```
add(t1);
add(t2);
add(ck1);
add(ck2);
add(course);
add(sem);
add(age);
add(b1);
b1.addActionListener(this);
add(b1);
course.add("BSc c.s");
course.add("BSc maths");
course.add("BSc physics");
course.add("BA English");
course.add("BCOM");
sem.add("1");
sem.add("2");
sem.add("3");
sem.add("4");
sem.add("5");
sem.add("6");
age.add("17");
age.add("18");
age.add("19");
age.add("20");
age.add("21");
l1.setBounds(25,65,90,20);
l2.setBounds(25,90,90,20);
l3.setBounds(25,120,90,20);
l4.setBounds(25,185,90,20);
l5.setBounds(25,260,90,20);
l6.setBounds(25,290,90,20);
l7.setBounds(25,260,90,20);
l11.setBounds(10,40,280,20);
t1.setBounds(120,65,170,20);
t2.setBounds(120,185,170,60);
ck1.setBounds(120,120,50,20);
ck2.setBounds(170,120,60,20);
course.setBounds(120,260,100,20);
sem.setBounds(120,290,50,20);
```



```
 age.setBounds(120,90,50,20);
 b1.setBounds(120,350,50,30);
}
public void paint(Graphics g)
{g.drawString(msg,200,450);}
public void actionPerformed(ActionEventae)
{if(ae.getActionCommand().equals("save"))
 {msg="Student details saved!";
 setForeground(Color.red);}
}
public static void main(String g[])
{studentstu=new student();
 stu.setSize(new Dimension(500,500));
 stu.setTitle("student registration");
 stu.setVisible(true);
}
}
 classmyWindowAdapter extends WindowAdapter
 {public void windowClosing(WindowEvent we)
  {
   System.exit(0);
  }
 }
```

Output

Student details

Name	nikhil
age	19
Ssx(MF)	Male Female
Address	Type adress hfhgg
Course	BSc. c.s.
Semester	5
	Save

Q.5. (a) What is 'this' Pointer?How does 'this'pointer can be used in java program? Explain with example.

Ans: this keyword in java

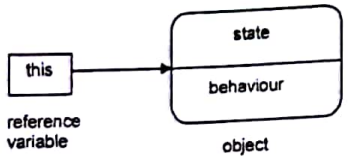
There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1.this can be used to refer current class instance variable.

2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.



this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Calling default constructor from parameterized constructor:

```
1.  class A{
2.   A(){System.out.println("hello a");}
3.   A(int x){
4.   this();
5.   System.out.println(x);
6.  }
7.  }
8.  class TestThis5{
9.  public static void main(String args[]){
10. A a=new A(10);
11. }}
```

Test it Now

Output:

hello a

10

Q.5. (b) Write the syntax of iteration statement of java.

Ans: Iterative Statements in Java

While loop

While loop is used for iteration. If we don't know how many elements are stored in a list to iterate in advance then it is recommended to use while loop is a best practice for this kind of situation.

Syntax

```
While(-) // boolean type
  {...}
```

Curly braces are optional but we can write only one statement under while it should not be declarative statement.

do-while

do-while wil execute loop **atleast once.**

Syntax

```
do{
statement;
  }
while(x);
```

If conditions is false still do-while will execute **atleast once.**

For loop

If we know the number of iteration in advance then we should go for loop .It is widely used for in java.

Syntax

```
for (initialization; conditional-exp; incrementation){
// body
}
```

Initalization

In this section we write inital value from iteration will be stored . It will be executed only once. We can declaration any number of while but same type.

Example

```
int i=10;
   String str="java";  //(invalid)
```

We can write any valid java statement.

Example

```
for(System.out.println("hello java");i<2;i++)
        {
System.out.println("hello java");
      }
```

Conditional Expression

In conditional expression we take any valid java expression which should be Boolean type. This optional if we are not using it then compiler will always place true.

Example

```
int i=0;
for(System.out.println("hello java"); i++)
        {
System.out.println("hello java")
            }
```

Increment /Decrement

In this section if condition is true then it will increment/decrement based on given expressions. We can also written**System.out.println** statement in this section.

Program

```
classFor_Example {
public static void main(String[] args) {
                                int i = 0;
for (System.out.println("hello java"); i < 3; System.out.println("hello"))
{
                                i++;
                                System.out.println("heHo java");
                        }
        }
}
```

Output

hello java
hello java
hello
hello java
hello
hello java
hello

for-each loop(Enhanced for loop)

This loop is used to retrieve elements from collections or arrays.

Example

```
String[] str = f.list();
for(String s : str){
System.out.println(s);
}
```

Q.5. (c) Explain about overloading of methods in java.

Ans. Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading *increases the readability of the program.*

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

(1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
1.  class Adder{
2.  static int add(int a,int b){return a+b;}
3.  static int add(int a,int b,int c){return a+b+c;}
4.  }
5.  class TestOverloading1{
6.  public static void main(String[] args){
7.  System.out.println(Adder.add(11,11));
8.  System.out.println(Adder.add(11,11,11));
9.  }}
```

Test it Now

Output:

22
33

(2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
1.  class Adder{
2.  static int add(int a, int b){return a+b;}
3.  static double add(double a, double b){return a+b;}
4.  }
5.  class TestOverloading2{
6.  public static void main(String[] args){
7.  System.out.println(Adder.add(11,11));
8.  System.out.println(Adder.add(12.3,12.6));
9.  }}
```

Test it Now

Output:

22
24.9

By changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
1.  class Adder{
2.  static int add(int a,int b){return a+b;}
3.  static double add(int a,int b){return a+b;}
4.  }
5.  class TestOverloading3{
6.  public static void main(String[] args){
7.  System.out.println(Adder.add(11,11));//ambiguity
8.  }}
```

Test it Now

Output:

Compile Time Error: method add(int,int) is already defined in class Adder

Q.6. (a)Define Inheritance. Give Various forms of inheritance and write a java program on single inheritance.

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

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

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the IS-A relationship, also known as parent-child relationship.

Why use inheritance in java

1. For Method Overriding (so runtime polymorphism can be achieved).
2. For Code Reusability.

Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

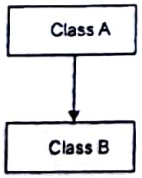
Various forms of inheritance:

- Single Inheritance
- Multiple Inheritance **(Through Interface)**
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance **(Through Interface)**

Lets see about each one of them one by one.

Single Inheritance in Java

Single Inheritance is the simple inheritance of all, When a class extends another class(Only one class) then we call it as **Single inheritance.** The below diagram represents the single inheritance in java where **Class B** extends only one class **Class A.** Here **Class B** will be the **Sub class** and **Class A** will be one and only **Super class.**



Single Inheritance

```
1.  class Animal{
2.  void eat(){System.out.println("eating...");}
3.  }
4.  class Dog extends Animal{
5.  void bark(){System.out.println("barking...");}
6.  }
```



```
7.  class TestInheritance{
8.  public static void main(String args[]){
9.  Dog d=new Dog();
10. d.bark();
11. d.eat();
12. }}
```

Output:

barking...

eating...

Q.6. (b) Define an Applet? What is the use of an Applet?Describe the life cycle of an applet.

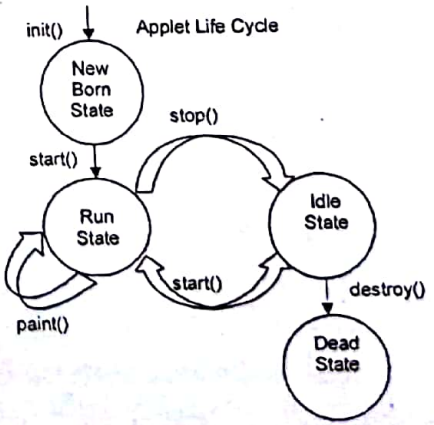
Ans: An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

Java Applets are usually **used** to add small, interactive components or enhancements to a webpage. These may consist of buttons, scrolling text, or stock tickers, but they can also be **used** to display larger programs like word processors or games

Life Cycle of an Applet

Four methods in the Applet class are:

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



A "Hello, World" Applet

```
import java.applet.*;
import java.awt.*;
publicclassHelloWorldAppletextendsApplet{
publicvoid paint (Graphics g){
g.drawString("Hello World",25,50);
}
}
```

Q.7. (a) Define Continer Class. Give a Su:table Example.

Ans. A **container class** is a data type that is capable of holding a collection of items.Containers hold and organize your Components, but they also contain code for event handling and many 'niceties' such as controlling the cursor image and the application's icon.

Containers:

- Frame
- Window
- Dialog
- Panel

Example:

public class **Containerextends** Component

Container Methods

Constructors The abstract Container class contains a single constructor to be called by its children. Prior to Java 1.1, the constructor was package private.

protected Container()*

The constructor for Container creates a new component without a native peer. Since you no longer have a native peer, you must rely on your container to provide a display area. This allows you to create containers that require fewer system resources.

For example, if you are creating panels purely for layout management, you might consider creating a LightweightPanel class to let you assign a layout manager to a component group. Using LightweightPanel will speed things up since events do not have to propagate through the panel and you do not have to get a peer from the native environment.

The following code creates the LightweightPanel class:

importjava.awt.*;

public class LightweightPanel extends Container

{ LightweightPanel ()

|| LightweightPanel (LayoutManager lm)

{ setLayout(lm); } }

Grouping A Container holds a set of objects within itself. This set of methods describes how to examine and add components to the set.

publicintgetComponentCount ()*

publicintcountComponents ()*

The getComponentCount() method returns the number of components within the container at this level. getComponentCount() does not count components in any child Container (i.e., containers within the current container). countComponents() is the Java 1.0 name for this method.

public Component getComponent (int position)

The getComponent() method returns the component at the specific position within it. If position is invalid, this method throws the run-time exception Array Index Out of Bounds Exception.

Q.7. (b) write short note on any Two of the following:

(i) Security Promises of java

Ans. Java's security model is focused on protecting users from hostile programs downloaded from untrusted sources across a network. To accomplish this goal, Java

provides a customizable "sandbox" in which Java programs run. A Java program must play only inside its sandbox. It can do anything within the boundaries of its sandbox, but it can't take any action outside those boundaries. The sandbox for untrusted Java applets, for example, prohibits many activities, including:

- Reading or writing to the local disk
- Making a network connection to any host, except the host from which the applet came
- Creating a new process
- Loading a new dynamic library and directly calling a native method

By making it impossible for downloaded code to perform certain actions, Java's security model protects the user from the threat of hostile code.

- The language defines all primitives with a specific size and all operations are defined to be in a specific order of execution. Thus, the code executed in different JVMs will not differ from the specified order of execution.
- The language provides access-control functionality on variables and methods in the object by defining name space management for type and procedure names. This secures the program by restricting access to its critical objects from untrusted code. For example, access is restricted by qualifying the type members as public, protected, private, package, etc.
- The Java language does not allow defining or dereferencing pointers, which means that programmers cannot forge a pointer to the memory or create code defining offset points to memory. All references to methods and instance variables in the class file are done via symbolic names. The elimination of pointers helps to prevent malicious programs like computer viruses and misuse of pointers such as accessing private methods directly by using a pointer starting from the object's pointer, or running off the end of an array.
- The Java object encapsulation supports "programming by contract," which allows the reuse of code that has already been tested.
- The Java language is a strongly typed language. During compile time, the Java compiler does extensive type checking for type mismatches. This mechanism guarantees that the runtime data type variables are compatible and consistent with the compile time information.
- The language allows declaring classes or methods as final. Any classes or methods that are declared as final cannot be overridden. This helps to protect the code from malicious attacks such as creating a subclass and substituting it for the original class and override methods.
- The Java Garbage Collection mechanism contributes to secure Java programs by providing a transparent storage allocation and recovering unused memory instead of deallocating the memory using manual intervention. This ensures program integrity during execution and prevents programmatic access to accidental and incorrect freeing of memory resulting in a JVM crash.

(ii) Wrapper classes

Ans. Normally, when we work with Numbers, we use primitive data types such as byte, int, long, double, etc.

Example:

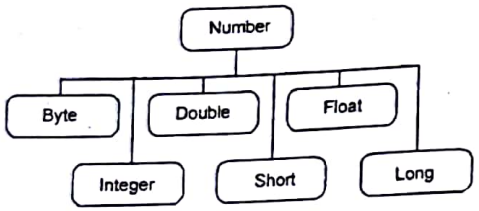
int i = 5000;

floatgpa = 13.65;

double mask = 0xaf;

However, in development, we come across situations where we need to use objects instead of primitive data types. In order to achieve this, Java provides **wrapper classes.**

All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number.



The object of the wrapper class contains or wraps its respective primitive data type. Converting primitive data types into object is called **boxing**, and this is taken care by the compiler. Therefore, while using a wrapper class you just need to pass the value of the primitive data type to the constructor of the Wrapper class.

And the Wrapper object will be converted back to a primitive data type, and this process is called unboxing. The **Number** class is part of the java.lang package.

Wrapper class Example: Primitive to Wrapper

```
1. public class WrapperExample1{
2. public static void main(String args[]){
3. //Converting int into Integer
4.                  int a=20;
5.Integer i=Integer.valueOf(a);//converting int into Integer
6.Integer j=a;//autoboxing, now compiler will write Integer valueOf(a) internally
7.
8.System.out.println(a+" "+i+" "+j);
9.                              }}
```

Output:

20 20 20

Wrapper class Example: Wrapper to Primitive

```
1. public class WrapperExample2{
2. public static void main(String args[]){
3. //Converting Integer to int
4. Integer a=new Integer(3);
5. int i=a.intValue();//converting Integer to int
6. int j=a;//unboxing, now compiler will write a.intValue() internally
7.
8. System.out.println(a+ " "+i+ " "+j);
9.                              }}
```

Output:

3 3 3

Primitive Type	Wrapper class
Boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

***(iii)* Awt components**

Ans: Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java. Java AWT components are platform-dependent *i.e.* components are displayed according to the view of operating system. AWT is heavyweight *i.e.* its components are using the resources of OS.

The java. awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class

Method Description

public void add(Component c)inserts a component on this component.

public void setSize(intwidth,int height) sets the size (width and height) of the component.

public void setLayout(LayoutManager m) defines the layout manager for the component.

public void setVisible(boolean status) changes the visibility of the component, by default false.

Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

By extending Frame class (inheritance)

By creating the object of Frame class (association)

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
importjava.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
```

```
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}
```

(iv) Remote Method invocation

Ans. The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

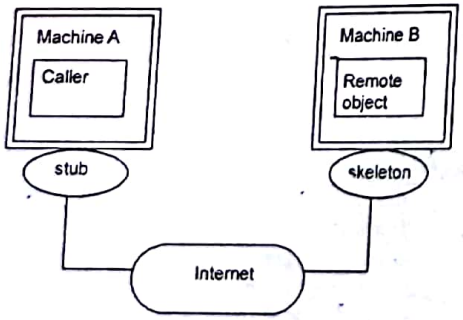
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.



FIRST TERM EXAMINATION [SEPT. 2017]
FIFTH SEMESTER [B.TECH]
JAVA PROGRAMMING [ETCS–307]

Time : 1½ hrs. **M.M. : 30**

***Note:** Attempt Q.no. 1 which is compulsory and any two more questions from remaining.*

Q.1. (a) If we want to execute any statement before the call of main() method then how it will be done? **(1)**

Ans. It will be executed in static block.

Static block is executed when the class which contain static block is loaded in to memory and static method is executed when it is called. Mostly static block is used for Initialization of static members.

Q.1.(b) Explain Autoboxing and Unboxing with help of example. **(1)**

Ans. Autoboxing: Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

```
class BoxingExample1{
public static void main(String args[]){
int a=50;
Integer a2=new Integer(a);//Boxing
Integer a3=5;//Boxing
System.out.println(a2+" "+a3);
}
}
```

Output:50 5

Unboxing: It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.

```
class UnboxingExample1{
public static void main(String args[]){
Integer i=new Integer(50);
int a=i;
System.out.println(a);
}
}
```

Output:50

Q.1.(c) What will be output of following program. **(1)**

```
String s1 = "Java"
String s2 = " Java"
If (s1. equals(s2))
(
System. Out. Println ("hello")
```

Else

System. Out.Println ("Bye")

}

Ans. Hello

The String equals() method compares the original content of the string, It compares values of string for equality.

Q.1.(d). What will be ouput of the program.

String s1 = "Java"

String s2 = " Java"

If (s1 = =(s2))

(

System. Out. Print ln ("hello")

Else

System. Out.Println (Bye)"

}

Ans. Hello

The = = operator compares references not values.

Q.1.(e). What is difference between checked and unchecked exceptions. **(1)**

Ans. The differences are as follows:

Checked: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword

Unchecked: are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.

In Java exceptions under *Error* and *Runtime Exception* classes are unchecked exceptions, everything else under throwable is checked.

Q.1.(f). Explain Anonymous inner class with help of example. **(1)**

Ans. Refer Q.4.(c) First Term September-2016 [Page no- 2016-9].

Q.1.(g) What is difference between abstract class and interface in java. (1)

Ans. Refer Q.1.(b) End Term September-2016 [Page no- 1-2016].

Q.1.(h) Explain applet Tag.

Ans. The <APPLET> Tag

Applets are embedded in HTML documents with the <APPLET> tag. The<APPLET> tag resembles the HTML <IMG> image tag. It contains attributes that identify the applet to be displayed and, optionally, give the web browser hints about how it should be displayed

<APPLET *attribute*

attribute ... >

<PARAM *parameter*>

<PARAM *parameter*>

...

</APPLET>

Q.1.(i) Explain the use of "final" keyword with help of exmple.

Ans. 1. Using final to define constants: If you want to make a local variable, class variable (static field), or instance variable (non-static field) constant, declare it final. A final variable may only be assigned to once and its value will not change and can help avoid programming errors.

Once a final variable has been assigned, it always contains the same value.

```
1. class Bike9(
2. final int speedlimit=90;//final variable
3. void run()( .4. speedlimit=400;
5. )
6. public static void main(String args[])(
7. Bike9 obj=new Bike9();
8. obj.runO;
9.)
10.)//endofclass
```

Output:Compile Time Error

2. Using final to prevent inheritance: If you find a class's definition is complete and you don't want it to be sub-classed, declare it final. A final class cannot be inherited, therefore, it will be a compile-time error if the name of a final class appears in the extends clause of another class declaration; this implies that a final class'cannot have any subclasses.

It is a compile-time error if a class is declared both final and abstract, because the nplementation of such a class could never be completed..

```
final class XYZ(
  )
class ABC extends XYZ(
void demo ()(
System.outprintln("My Method")
  )
public static void main(String args[])(
  )
)
```

Output:

The type ABC cannot subclass the final class XYZ

3. Using final to prevent overriding: When a class is extended by other classes, its methods can be overridden for reuse. There may be circumstances when you want to

prevent a particular method from being overridden, in that case, declare that method final. Methods declared as final cannot be overridden.

```
classXYZ{
  final void demo(){
    System.out.printlnC'XYZ Class Method");
  }
}
class ABC extends XYZ{
  void demo(){
    System.out.println("ABC Class Method");
  }
  public static void main( String args[]){
    ABC obj= new ABC();
    obj.demo();
  }
}
```

The above program would throw a compilation error.

Q.1.(j) What do you mean by Dynamic Method Dispatch.

Ans. Refer Q.4.(c) End Term December-2017

Q.2. Write down program to create user defined Exception. Explain following with help of example. (a) Throw (b) Throws (c) Finally.

Ans. This example shows how to create user defined exception by extending Exception Class.

```
classMyExceptionextendsException{
String s1;
MyException(String s2){
    s1 = s2;
}
publicStringtoString(){
return("Output String =" +s1);
}
}
publicclassNewClass{
publicstaticvoid main(Stringargs[]){
try{
thrownewMyException("Custom message");
}catch(MyExceptionexp){
System.out.println(exp);
}
}
```


The above code sample will produce the following result.

Output String = Custom message

(a) Throw Keyword: The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

Syntax:

throwInstance

Example:

```
throw new ArithmeticException("/ by zero");
classThrowExcep
{
    staticvoidfun()
    {
      try
      {
          thrownewNullPointerException("demo")
      }
      catch(NullPointerException e)
      {
          System.out.println("Caught inside fun().");
          throwe; // rethrowing the exception
      }
    }
    publicstaticvoidmain(String args[])
      {
      try
      {
          fun();
      }
      catch(NullPointerException e)
      {
          System.out.println("Caught in main.");
      }
    }
}
```

Output:

Caught inside fun().

Caught in main.

(b) Throws Keyword: Throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

Syntax:

typemethod_name(parameters) throws exception_list

exception_list is a comma separated list of all the exceptions which a method might throw.

```
publicclassExample1{
int division(int a, int b) throwsArithmeticException{
        int t = a/b;
        return t;
    }
publicstaticvoid main(Stringargs[]){
        Example1obj = newExample1();
        try{
        System.out.println(obj.division(15,0));
        }
        catch(ArithmeticException e){
        System.out.println("You shouldn't divide number by zero");
        }
    }
}
```

Output:

You shouldn't divide number by zero

(c) Finally Keyword: Finally block is a block that is used *to execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

```
class TestFinallyBlock{
 public static void main(String args[]){
 try{
  int data=25/5;
  System.out.println(data);
 }
 catch(NullPointerException e){System.out.println(e);}
 finally{System.out.println("finally block is always executed");}
 System.out.println("rest of the code...");
 }
}
```

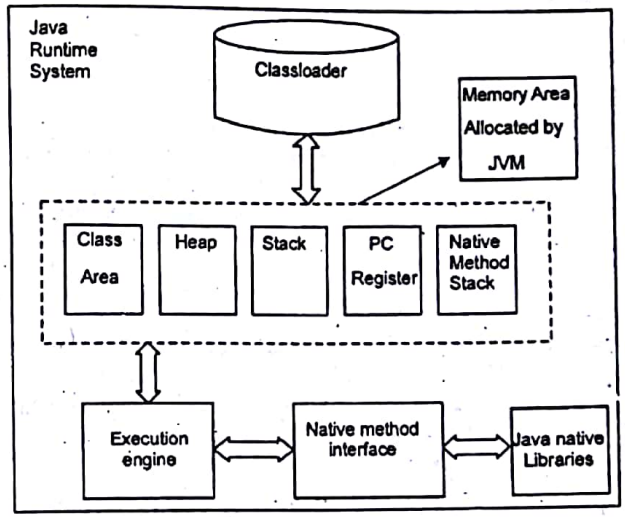

Output:5

finally block is always executed

rest of the code...

Q.3. Explain the internal working of java virtual machine and also explain the class file format in detail.

Ans. JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which Java bytecode can be executed.JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).



1. Classloader: Classloader is a subsystem of JVM that is used to load class files.

2. Class (Method) Area: Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3. Heap: It is the runtime data area in which objects are allocated.

4. Stack: Java Stack stores frames.It holds local'variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5. Program Counter Register: PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

6. Native Method Stack: It contains all the. native methods used in the application.

7. Execution Engine **It contains:**

1. A virtual processor

2. Interpreter: Read bytecode stream then execute the instructions.

3. Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term compiler refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

Java class file format: Compiled Java is typically distributed in a very compact format called Class files. It is the assembler code for a virtual stack-oriented

Java-friendly CPU (Central Processing Unit) chip. Such chips do exist, but most of the time they are simulated with interpreters, JIT (Just In Time), Hotspots or AOT (Ahead Of Time) compilation on the target machines which have quite different architectures.

Class files are usually compressed and bundled into jar files.

Each class file is marked with a class file format version, which lets you know which JDK (Java Development Kit) it was compiled for. At offset 6 in the class file is a big-endian short that gives the major version number. You can display for the classes is in a jar with JarCheck.

A Java class file is a file (with the .class filename extension) containing Java bytecode that can be executed on the Java Virtual Machine (JVM). A Java class file is produced by a Java compiler from Java programming language source files (Java files) containing Java classes. If a source file has more than one class, each class is compiled into a separate class file.

JVMs are available for many platforms, and a class file compiled on one platform will execute on a JVM of another platform. This makes Java platform-independent.

Q.4. How Garbage Collection in java works? Explain Sandbox model in detail JVM as an Emulator comment on it.

Ans. Java Garbage Collection

In Java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in Java it is performed automatically. So, Java provides better memory management.

Advantage of Garbage Collection

It makes Java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

public static void gc(){[.

sandbox model in java.

A security measure in the Java development environment. The sandbox is a set of rules that are used when creating an applet that prevents certain functions when the applet is sent as part of a Web page. When a browser requests a Web page with applets,

the applets are sent automatically and can be executed as soon as the page arrives in the browser. If the applet is allowed unlimited access to memory and operating system resources, it can do harm in the hands of someone with malicious intent. The sandbox creates an environment in which there are strict limitations on what system resources the applet can request or access. Sandboxes are used when executable code comes from unknown or untrusted sources and allow the user to run untrusted code safely.

The Java sandbox relies on a three-tiered defense. If any one of these three elements fails, the security model is completely compromised and vulnerable to attack:

- **Byte code verifier** — This is one way that Java automatically checks untrusted outside code before it is allowed to run. When a Java source program is compiled, it compiles down to platform-independent Java byte code, which is verified before it can run. This helps to establish a base set of security guarantees.
- **Applet class loader** — All Java objects belong to classes, and the applet class loader determines when and how an applet can add classes to a running Java environment. The applet class loader ensures that important elements of the Java run-time environment are not replaced by code that an applet tries to install.
- **Security manager** — The security manager is consulted by code in the Java library whenever a dangerous operation is about to be carried out. The security manager has the option to veto the operation by generating a security exception.

Java typically interpret instead of compile? The main advantage of compilation is that you end up with raw machine language code that can be efficiently executed on your machine. However, it can only be executed on one type of machine architecture (Intel Pentium, PowerPC). A primary advantage of a compiling to an intermediate language like Java bytecode and then interpreting is that you can achieve *platform independence:* you can interpret the same class file on differently types of machine architectures. However, interpreting the bytecode is typically slower than executing pre-compiled machine language code. A second advantage of using the Java bytecode is that it acts as a buffer between your computer and the program. This enables you to download an untrusted program from the Internet and execute it on your machine with some assurances. Since you are running the Java interpreter (and not raw machine language code), you are protected by a layer of security which guards against malicious programs. It is the combination of Java and the Java bytecode that yield a platform-independent and secure environment, while still embracing a full set of modern programming abstractions.

END TERM EXAMINATION [DEC. 2017]
FIFTH SEMESTER [B.TECH]
JAVA PROGRAMMING [ETCS–307]

Time : 3 hrs. **M.M. : 75**

Note: *Attempt any five questions including Q.no. 1 which is compulsory.*

Q.1. Answer the following:

Q.1. (a) Give a brief overview on Java packages. Write necessary code snippets. **(5)**

Ans. Refer Q.2.(a). First Term September-2016 [Page no. 2016-3].

Q.1.(b). State the difference between inner class and anonymous class? **(5)**

Ans. Java Inner Classes

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

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

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

Syntax of Inner class

```
class Java_Outer_class{
//code
class Java_Inner_class{
//code
}
}
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1. Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.

2. Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.

3. Code Optimization: It requires less code to write.

Anonymous class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

Example :

```
abstract class Person{
abstract void eat();
}
```



```
class TestAnonymousInner{
public static void main(String args[]){
Person p=new Person(){
void eat(){System.out.println("nice fruits");}
};
p.eat();
}
}
```

Output:

nice fruits

Q.1. (c) What are access specifiers? Discuss them in the context of Java. **(5)**

Ans. Access Specifiers In Java: Java Access Specifiers (also known as Visibility Specifiers) regulate access to classes, fields and methods in Java.These Specifiers determine whether a field or method in a class, can be used or invoked by another method in another class or sub-class. Access Specifiers can be used to restrict access

Types Of Access Specifiers :

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

1. Public 2. Private
3. Protected 4. Default(no specifier)

We look at these Access Specifiers in more detail:

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

Public Specifiers : Public Specifiers achieves the highest level of accessibility. Classes, methods, and fields declared as public can be accessed from any class in the Java program, whether these classes are in the same package or in another package.

Example :

```
public class Demo { // public class
public x, y, size; // public instance variables
}
```

Private specifiers : Private Specifiers achieves the lowest level of accessibility. private methods and fields can only be accessed within the same class to which the methods and fields belong. private methods and fields are not visible within subclasses and are not inherited by subclasses.

Example :

```
public class Demo {  // public class
private double x, y;  // private (encapsulated) instance variables
public set(int x, int y) {  // setting values of private fields
this.x = x;
this.y = y;
}
public get() {  // setting values of private fields
return Point(x, y);
}
}
```

Protected specifiers : Methods and fields declared as protected can only be accessed by the subclasses in other package or any class within the package of the protected members' class. The protected access specifier cannot be applied to class and interfaces.

Default(no specifier): When you don't set access specifier for the element, it will follow the default accessibility level. There is no default specifier keyword. Classes, variables, and methods can be default accessed.Using default specifier we can access class, method, or field which belongs to same package,but not from outside this package.

Example :

```
class Demo
{
inti; (Default)
}
```

Q.1.(d) What is interface and mention its use. **(5)**

Ans. An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in java is **a mechanism to achieve abstraction.** There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship.**It cannot be instantiated just like abstract class.

Use of interface:

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Declare interface: Interface is declared by using interface keyword. It provides total abstraction; means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default. A class that implement interface must implement all the methods declared in the interface.

Syntax:

```
1. interface <interface_name>{
2. // declare constant fields
3. // declare methods that abstract
4. // by default.
5. }
```


Q.1.(e) What is the difference between vector and array? Give suitable example. **(5)**

Ans. Differences between a Vector and an Array:

A vector is a dynamic array, whose size can be increased, where as an array size can not be changed.

- Reserve space can be given for vector, where as for arrays can not.
- A vector is a class where as an array is not.
- Vectors can store any type of objects, where as an array can store only homogeneous values.

Advantages of Arrays:

- Arrays supports efficient random access to the members.
- It is easy to sort an array.
- They are more appropriate for storing fixed number of elements

Disadvantages of Arrays:

- Elements can not be deleted
- Dynamic creation of arrays is not possible
- Multiple data types can not be stored

Advantages of Vector:

- Size of the vector can be changed
- Multiple objects can be stored
- Elements can be deleted from a vector

Disadvantages of Vector:

- A vector is an object, memory consumption is more.

Java program that uses Vector

```
importjava.util.Vector;
public class Program {
public static void main(String[] args) {
    // Add two values to Vector.
Vector<Integer> v = new Vector<>();
v.add(100);
v.add(1000);
    // Display values.
for (int value : v) {
System.out.println(value);
    }
  }
}
```

Output

```
100
1000
```

Java program that converts Vector, array

```
importjava.util.Vector;
public class Program {
public static void main(String[] args) {
    // Add ArrayList to Vector.
Vector<String> v = new Vector<>();
v.add("one");
v.add("two");
v.add("three");
    // ... Convert Vector into String array.
String[] array = {};
String[] result = v.toArray(array);
for (String value : result) {
System.out.println(value);
    }
  }
}
```

Output

```
one
two
three
```

Q.2.(a) Write a program to create two threads so one thread will print multiples of 5 where as second thread will print multiples of 7 between 1 to 70 numbers. **(8)**

Ans. When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
//example of java synchronized method
class Mul{
synchronized void printMul(int n){//synchronized method
for(int i=1;i<=70;i++){
If((i%n)==0)
System.out.println(i);
try{
Thread.sleep(400);
}catch(Exception e)
{System.out.println(e);}
}
}
}
class MyThread1 extends Thread{
Mul t;
MyThread1(Mul t){
this.t=t;
```



```
}
public void run(){
t.printMul(5);
}
}
class MyThread2 extends Thread{
Mul t;
MyThread2(Mul t){
this.t=t;
}
public void run(){
t.printMul(7);
}
}
public class TestSynchronization2{
public static void main(String args[]){
Mul obj = new Mul();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

```
Output: 5
10
15
20
25
30
35
40
45
50
55
60
65
70
7
14
21
28
35
```

42
49
56
63
70

Q.2.(b). How is synchronization achieved in Java, **(4.5)**

Ans. Synchronization in java is achived by following ways:

1. Synchronized method.
2. Synchronized block.

Example of Synchronized Method in Java

Using **synchronized keyword** along with method is easy just apply synchronized keyword in front of the method. What we need to take care is that static synchronized method locked on class object lock and nonstatic synchronized method locks on current object (this). So it's possible that both static and nonstatic java synchronized method running in parallel. This is the common mistake a naive developer do while **writing Java synchronized code.**

```
publicclassCounter{

privatestaticint count = 0;

publicstaticsynchronizedintgetCount(){
return count;
}

publicsynchoronizedsetCount(int count){
this.count = count;
}

}
```

In this example of *Java, the synchronization code is not properly synchronized* because both getCount() and setCount() are not getting locked on the same object and can run in parallel which may result in the incorrect count. Here getCount() will lock in Counter.class object while setCount() will lock on current object (this). To make this code properly synchronized in Java *you need to either make both method static or nonstatic or use java synchronized block instead of java synchronized method.*

Example of Synchronized Block in Java

Using synchronized block in java is also similar to using synchronized keyword in methods. Only important thing to note here is that if object used to lock synchronized block of code, Singleton.class in below example is null then Java synchronized block will throw a NullPointerExcepuon.

```
publicclassSingleton{

privatestaticvolatile Singleton _instance;
```



```
publicstatic Singleton getInstance(){
if(_instance == null){
synchronized(Singleton.class){
if(_instance == null)
        _instance = newSingleton();
    }
}
return _instance;
}
```

This is a classic example of double checked locking in Singleton. In this **example of Java synchronized code,** we have made the only critical section (part of the code which is creating an instance of singleton) synchronized and saved some performance.

Q.3.(a) Distinguish between: **(6)**

(i) Inputstream and Reader classes

Ans. InputStreams are used to read bytes from a **stream** . It grabs the data byte by byte without performing any kind of translation. So they are useful for **binary datasuch** as images, video and serialized objects.

Readers on the other hand are **character streams** so they are best used to read character data. If the information you are reading is all text, then the **Reader** will take care of the character decoding for you and give you **unicode characters** from the raw input stream. If you are reading any type of text, this is the stream to use.

Let's see an example of *How to use FileInputStream and FileReader in Java.* You can provide either a File object or a String, containing location of file to start reading character data from File.

```
import java.awt.Color;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.IOException;

public class HowToReadFileInJava {
public static void main(String args[]) {
// Example 1 - Reading File's content using FileInputStream
try (FileInputStreamfis = new FileInputStream("data.txt")) {
int data = fis.read();
        while (data != -1) {
System.out.print(Integer.toHexString(data)); data = fis.read();
        }
    } catch (IOException e) {
System.out.println("Failed to read binary data from File");
e.printStackTrace();
        }
```

```
// Example 2 - Reading File data using FileReader in Java
try (FileReader reader = new FileReader("data.txt")) {
int character = reader.read();
while (character != -1) {
System.out.print((char) character);
character = reader.read();
        }
        } catch (IOExceptionio) {
System.out.println("Failed to read character data from File");
io.printStackTrace();
            }
    }
}
```

Output:

4157532d416d617a6f6e2057656220536572766963656da474f4f472d476f6f676c65da4150504c2d4170706c65da47532d476f6c646d616e205361636873

AWS-Amazon Web Service

GOOG-Google

APPL-Apple

GS-Goldman Sachs

Q.3.(a)(ii) Outputstream and Writer classes

Ans. The OutputStream class is an abstract class that defines methods to write a stream of bytes sequentially. Java provides subclasses of the OutputStream class for writing to files and byte arrays, among other things. It is also easy to define a subclass of OutputStream that writes to another kind of destination.

The Writer class is an abstract class that defines methods to write to a stream of characters sequentially.

OutputStream: The OutputStream class is the abstract superclass of all other byte output stream classes. It defines three write() methods for writing to a raw stream of bytes:

write(int b)

write(byte[] b)

write(byte[] b, int off, intlen)

Some OutputStream subclasses may implement buffering to increase efficiency. OutputStream provides a method, flush(), that tells the OutputStream to write any buffered output to the underlying device, which may be a disk drive or a network.

Because the OutputStream class is abstract, you cannot create a "pure" OutputStream. However, the various subclasses of OutputStream can be used interchangeably. For example, methods often take OutputStream parameters. This means that such a method accepts any subclass of OutputStream as an argument.

Writer: The Writer class is the abstract parent class of all other character output stream classes. It defines nearly the same methods as OutputStream, except that the write() methods deal with characters instead of bytes:

write(int c)

write(char[] cbuf)

write(char[] cbuf, int off, intlen)

write(String str)

write(String str, int off, intlen)

Writer also includes a flush() method that forces any buffered data to be written to the stream.

Writer is designed so that write(int) and write(char[]) both call write(char[], int, int). Thus, when you subclass Writer, you only need to define the write(char[], int,int) method.

Q.3.(b) Write a program to copy the content of one file to another in Java using buffered streams. **(6.5)**

Ans. This example shows how to copy contents of one file into another file using read & write methods of BufferedWriter class.

```
import java.io.*;

publicclassMain{
publicstaticvoid main(String[]args)throwsException{
BufferedWriter out1 =newBufferedWriter(newFileWriter("srcfile"));
out1.write("string to be copied\n");
out1.close();
InputStreamin=newFileInputStream(newFile("srcfile"));
OutputStreamout=newFileOutputStream(newFile("destnfile"));
byte[]buf=newbyte[1024];
intlen;

while((len=in.read(buf))>0){
out.write(buf,0,len);
}
in.close();
out.close();
BufferedReader in1 =newBufferedReader(newFileReader("destnfile"));
Stringstr;
while((str= in1.readLine())!=null){
System.out.println(str);
}
in.close();
}
```

```
}
```

Result

The above code sample will produce the following result.

string to be copied

Q.4.(a). How is event handling done in Java? Define an exception called "StringNotFoundException" that is thrown when a string is not equal to "JavaProgram". (6.5)

Ans. Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events. Let's have a brief introduction to this model.

Steps to handle events:

1.Implement appropriate interface in the class.

2.Register the component with the listener.

Example of Event Handling

```
importjava.awt.*;
importjava.awt.event.*;
importjava.applet.*;
importjava.applet.*;
importjava.awt.event.*;
importjava.awt.*;
publicclassTestextendsAppletimplementsKeyListener
{
 String msg="";
publicvoidinit()
{
addKeyListener(this);
}
publicvoidkeyPressed(KeyEvent k)
{
showStatus("KeyPressed");
}
publicvoidkeyReleased(KeyEvent k)
{
showStatus("KeyRealesed");
}
publicvoidkeyTyped(KeyEvent k)
{
```



```
msg=msg+k.getKeyChar();
repaint();
}
publicvoidpaint(Graphics g)
{
g.drawString(msg,20,40);
}
}
```

HTML code :

```
<applet code= "Test" width=300, height=100 >
```

String Not Found Exception:

```
public class StringNotFoundException extends Exception
public StringNotFoundException(String message) {
    super(message);
    }
}
```

And the following example shows the way a custom exception is used is nothing different than built-in exception:

```
1.    public class StringManager {
2.
3.      public Student find(String stringID) throws StringNotFoundException {
4.        if (stringID.equals("123456")) {
5.          return new String();
6.        } else {
7.          throw new StringNotFoundException(
8.            "Could not find student with ID" + studentID);
9.        }
10.     }
11.   }
```

And the following test program handles that exception:

```
1.    public class StringTest {
2.      public static void main(String[] args) {
3.        StringManager manager = new StringManager();
4.
5.        try {
6.
7.          String s = manager.find("0000001");
8.
9.        } catch (StringNotFoundException ex) {
10.         System.err.print(ex);
11.       }
```

12. }

13. }

Run this program and you will see this output:

1StringNotFoundException: Could not find student with ID 0000001

Q.4.(b) What is meant by- bytecode in Java and class file format. **(3)**

Ans. Java class file formate: Compiled Java is typically distributed in a very compact format called Class files. It is the assembler code for a virtual stack-oriented Java-friendly CPU (Cental Processing Unit) chip. Such chips do exist, but most of the time they are simulated with interpreters, JIT (Just In Time), Hotspots or AOT (Ahead Of Time) compilation on the target machines which have quite diffeent architectures.

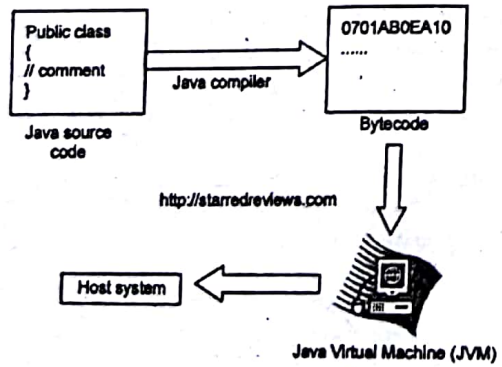
Class files are usually compressed and bundled into jar files.

Each class file is marked with a class file format version, which lets you know which JDK (Java Development Kit) it was compiled for. At offset 6 in the class file is a big-endian short that gives the major version number. You can display.for the classes is in a jar with JarCheck.

A Java class file is a file (with the class filename extension) containing Java bytecode that can be executed on the Virtual Machine (JVM). A Java clss file is produced by a Java compiler from Java programming language sources files (Java files) containing Java classes, if a source file has more than one class, each class is compiled into a separate class file.

JVMs are available for many platforms, and a class file compiled on one platform will execute on a JVM of another platform. This makes JAVA platform-independent.

BYTE CODE: Bytecode is nothing but the intermediate representation of Java source code which is produced by, the Java compiler by compiling that source code. This byte code is an machine independent code. It is not an completely a compiled code but it is an intermediate code somewhere in the middle which is later interpreted and executed by JVM. Bytecode is a machine code for JVM. But the machine code is platform specific whereas bytecode is platform independent that is the main difference between them.It is stored in class file which is created after compiling the source code. Most developers although have never seen byte code (nor have everwanted to see it!) One way to view the byte code is to compile your class and then open the .class file in a hex editor and translate the bytecodes by referring to the virtual machine specification.



Q.4.(c) What is dynamic method dispatch? **(3)**

Ans. Dynamic Method Dispatch: Dynamic method dispatch is a technique by which call to a overridden method is resolved at runtime, rather than compile time.When an overridden method is called by a reference, then which version of overridden method is to be called is decided at runtime according to the type of object it refers.Dynamic method dispatch is performed by JVM not compiler.

Dynamic method dispatch allows java to support overriding of methods and perform runtime polymorphism.It allows subclasses to have common methods and can redefine specific implementation for them.This lets the superclass reference respond differently to same method call depending on which object it is pointing.

```
Baseb=newBase();
Derivedd=newDerived();
b=newDerived();      //Base class reference refer to derived class object  that is upcasting
```

When parent class reference variable refers to a child class object than its called upcasting and using this technique dynamic method dispatch perform.

We can not use derived class reference to refer to a base class object.

For example:

```
class Student {
publicvoid show(){System.out.println("Student details.");
}
}
publicclassCollegeStudentextends Student {
publicvoid show()
System.out.println("College Student details.");
}
publicstaticvoid main(Stringargs[]){
    Student obj=newCollegeStudent();
obj.show();
}
}
```

Output:

College Student details.

School Student details.

Q.5.(a) What is 'this' pointer? How can 'this' pointer be used in Java program to handle instance variable hiding? **(3)**

Ans. There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of this keyword

1. This can be used to refer current class instance variable.
2. This can be used to invoke current class method (implicitly)

3. This() can be used to invoke current class constructor.
4. This can be passed as an argument in the method call.
5. This can be passed as argument in the constructor call.
6. This can be used to return the current class instance from the method.

Using 'this' keyword,handling current class instance variables hiding:

```
class Test
{
  int a;
  int b;
    Test(int a, int b)
  {
    this.a = a;
    this.b = b;
  }
  void display()
  {
      System.out.println("a =" + a + " b = " + b);
}
public static void main(String[] args)
{
    Test object = new Test(10, 20);
    object.display();
  }
}
```

Output:

a = 10 b = 20

Q.5.(b) Discuss atleast three methods available in Component class. (3)

Ans. Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(intwidth,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Example:

```
1. import javax.swing.*;
```



```
2. public class FirstSwingExample {
3. public static void main(String[] args) {
4. JFrame f=new JFrame();//creating instance of JFrame
5. JButton b=new JButton("click");//creating instance of JButton
6. b.setBounds(130,100,100, 40);//x axis, y axis, width, height
7. f.add(b);//adding button in JFrame
8. f.setSize(400,500);//400 width and 500 height
9. f.setLayout(null);//using no layout managers
10. f.setVisible(true);//making the frame visible
11. }
12. }
```

Q.5.(c) How is an applet program different from an application? (6.5)

Ans.

Applet	Application
Small-Program	Large Program
Used to run a program on client Browser	Can be executed on stand alone computer system
Applet is portable and can be executed by supported browser.	Need JDK, JRE, JVM installed on JAVA client machine
Applet applicationss are executed in a Restricted Environment	Application can access all the resources of the computer
Applets are created by extending the Java, applet. Applet	Applications are created by writing public static void main(String[] S) method.
Applet application has 5 methods which will be automatically invoked on occurance of specific event	Application has a single start point which is main methods

Example:

```
import java.awt. *;
import Java, applet. *;
Public class MyClass extends Applet
{
public void init () {}
public void start () {}
public void step Oil
public void destroy () {}
public void paint (Graphics g) {}
}
```

```
Public class Myclass
{
public static void main (string args []) {{
{
```

Q.6.(a) Explain briefly any two layout managers in Java. **(4)**

Ans. The various Layout manager are the following:

Flow Layout • Border Layout

Grid Layout, • Card Layout

Flow Layout: Flow Layout is the default Layout manager. Flow Layout implements a simple layout style, which is similar to how words flow in a text editor.

Border Layout: It has narrow, fixed-midth components at the edges and one large once in the center. The four sides one referred to as north, south, east and west. The middle area is called the center.

Grid Layout: Grid Layout lays out components in a two-dimensional grid.

Card Layout: The Card Layout class is unique among the other layout managers in that it stores several different layouts.

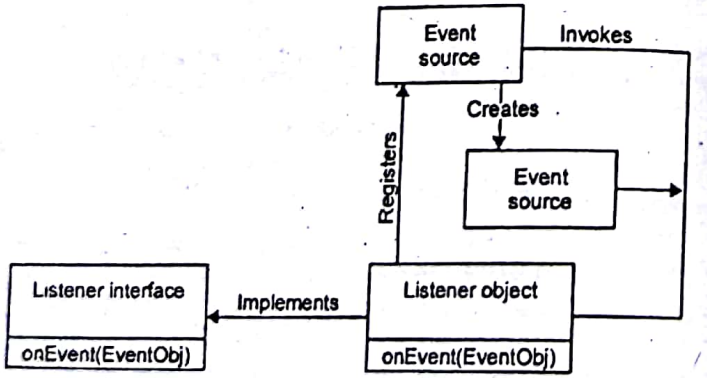
Q.6.(b) What do you understand by delegation event model? Write a program to illustrate key event handling in Java. **(8.5)**

Ans. Java's delegation event model

The event model is based on the Event Source and Event Listeners. Event Listener is an object that receives the messages / events. The Event Source is any object which creates the message / event. The Event Delegation model is based on - The Event Classes, The Event Listeners, Event Objects.

There are three participants in event delegation model in Java;

- Event Source - The class which broadcasts the events
- Event Listeners — The classes which receive notifications of events
- Event Object — The class object which describes the event.



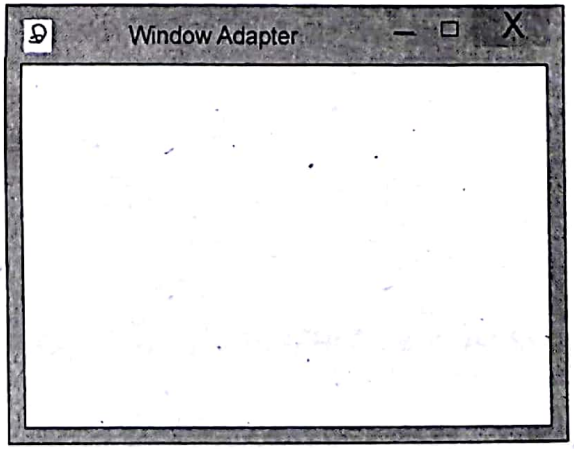
Key Event Handling:

```
importjava.awt.*;
importjava.awt.event.*;
```



```
importjava.applet.*;
publicclassKeyextendsApplet
implementsKeyListener
{
int X=20,Y=30;
Stringmsg= "KeyEvents—>";
publicvoidinit()
{
addKeyListener(this);
}
publicvoidkeyPressed(KeyEvent k)
{showStatus("KeyDown");
repaint();}
publicvoidkeyReleased(KeyEvent k)
{        showStatus("Key Up");}
publicvoidkeyTyped(KeyEvent k)
    {
msg+=k.getKeyChar();
repaint();}
publicvoid paint(Graphics g)
{
g.drawString(msg,X,Y);
    }
}
```

Output:



Q.7.(a) How are adapter classes different from listener interfaces? Support your answer with suitable example. (6)

Ans. Java adapter classes *provide the default implementation of listener interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

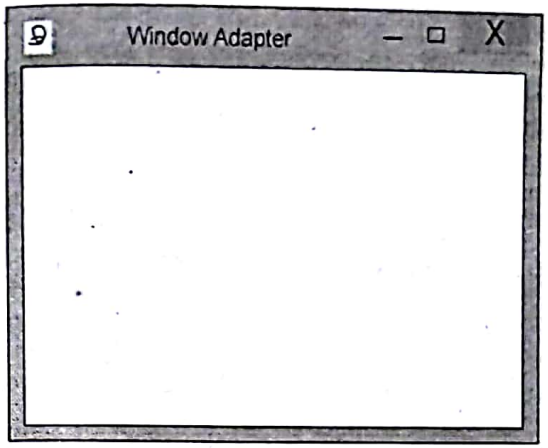
The adapter classes are found in **java.awt.event, java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener

WindowAdapter Example

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class AdapterExample{
4.     Frame f;
5.     AdapterExample(){
6.       f=new Frame("Window Adapter");
7.       f.addWindowListener(new WindowAdapter(){
8.         public void windowClosing(WindowEvent e) {
9.            f.dispose();
10.        }
11.      });
12.      f.setSize(400,400);
13.      f.setLayout(null);
14.      f.setVisible(true);
15.   }
16. public static void main(String[] args) {
17.    new AdapterExample();
18. } }
```


Output:



Q.7.(b) What is JDBC Connection? Explain steps to get Database connection in a simple Java program. (6.5)

Ans. JDBC is an acronym for Java Database Connectivity. It's an advancement for ODBC (Open Database Connectivity). JDBC is an standard API specification developed in order to move data from frontend to backend. This API consists of classes and interfaces written in Java. It basically acts as an interface (not the one we use in Java) or channel between your Java program and databases i.e it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use.

Java Database Connectivity with 5 Steps

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- o Register the driver class
- o Creating connection
- o Creating statement
- o Executing queries
- o Closing connection

1. Register the driver class: The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

1.public static void forName(String className)throws ClassNotFoundException

Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

1.Class.forName("oracle.jdbc.driver.OracleDriver");

2. Create the connection object: The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1. public static Connection getConnection(String url)throws SQLException
2. public static Connection getConnection(String url,String name,String password) throws SQLException

Example to establish connection with the Oracle database

1.Connection con=DriverManager.getConnection(

2."jdbc:oracle:thin:@localhost:1521:xe","system","password");

3. Create the Statement object: The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

1.public Statement createStatement()throws SQLException

Example to create the statement object

1.Statement stmt=con.createStatement();

4. Execute the query: The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

1.public ResultSet executeQuery(String sql)throws SQLException

Example to execute query

```
1. ResultSet rs=stmt.executeQuery("select * from emp");
2.
3. while(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }
```

5. Close the connection object: By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

1.public void close()throws SQLException

Example to close connection

1. con.close();

Q.8. Write note on any two of the following. **(12.5)**

Q.8.(a) Buffered Streams

Ans. In case of 'byte' and 'character' streams every byte or piece of data that is being read or write requires an direct support from underlying OS because of not having an intermediate buffer included. This makes a extensive use of memory and resources. On the other hand Bufferted streams works as a wrapper to hold unbuffered streams and make it possible to store bunch of data or bytes in buffers (memory). The underlying OS

resource are needed only when the buffer is full or empty and not on every instance of read or write.

The native input API is called only when the buffer is empty. Similarly, buffered output streams write data to a buffer, and the native output API is called only when the buffer is full. Unbuffered streams can be converted to buffered streams by wrapping them in a constructor of Buffered Stream class.

Example program converting 'byte Stream' to 'Buffered byte Stream'

```
package com.beingjavaguys.core;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import Java.io.FileOutputStream;
import java.io.IOException;
/**
* ©author Na,gesh Chauhan
*
*/
public class BufferedIOStream {
public static void main(String args[]) throws IOException {
BufferedInputStream bis = null;
BufferedOutputStream bos = null;
try {
 bis = new BufferedInputStream(new FileInputStream(
       "files/source.txt"));
 bos=new
 BufferedOutputStream(new FileOutputStream(
  "files/destination.txt"));
 int data;
 while ((data = bis.readQ) !=-1) {
 System.out.println(data);
 bos.write(data);
 }
 } catch (FileNotFoundException e) {
 e.printStackTrace();
 } finally{
 if (bis != null)
 bis.close();
 if (bos != null)
```

```
bos.close();
}
}
}
```

Q.8.(b) Remote Method Invocation

Ans. Refer Q.7.(b).(iv). End Term Examination December-2016 [Page no -32-2016]

Q.8.(c) Object Serialization

Ans. To *serialize* an object means to convert its state to a byte stream so that the byte stream can be reverted back into a copy of the object. A Java object is *serializable* if its class or any of its superclasses implements either the java.io.Serializable interface or its subinterface, java.io.Externalizable. *Deserialization* is the process of converting the serialized form of an object back into a copy of the object.

For example, the java.awt.Button class implements the Serializable interface, so you can serialize a java.awt.Button object and store that serialized state in a file. Later, you can read back the serialized state and deserialize into a java.awt.Button object.

The Java platform specifies a default way by which serializable objects are serialized. A (Java) class can override this default serialization and define its own way of serializing objects of that class. The Object Serialization Specification describes object serialization in detail.

When an object is serialized, information that identifies its class is recorded in the serialized stream. However, the class's definition ("class file") itself is not recorded. It > is the responsibility of the system that is deserializing the object to determine how to l locate and load the necessary class files

Binding a Serializable Object: You can store a serializable object in the director)' if the underlying service provider supports that action, as does Oracle's LDAP service; provider.

The following example invokes Context.bind to bind an AWT button to the name "cn=Button". To associate attributes with the new binding, you use DirContext.bind. Tb overwrite an existing binding, use Context.rebind and DirContext.rebind.

```
// Create the object to be bound
Button b = new Button("Push me");
// Perform the bind
ctx.bind("cn=Button", b);
```

You can then read the object back using Context.lookup, as follows.

```
// Check that it is bound
Button b2 = (Button)ctx.lookup("cn=Button");
System.out.println(b2);   /
```

Running this example produces the following output.-

```
# javaSerObj
java.awt.Button[button0,0,0,0x0,invalid,label=Push me]
```