TED (10)-3069 Reg. No. ………………………….

(REVISION-2010) Signature ……..……………………

FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/ TECHNOLIGY- MARCH 2015

OOP THROUGH JAVA

(Common to CT, CM and IF)

[*Time:* 3 hours

(Maximum marks: 100)

Marks

PART –A

(Maximum marks: 10)

I. Answer all questions in a sentence

1. List any 2 difference of POP and OOP

- POP emphasis on algorithms (procedure), where OOP emphasis on data rather than procedure.
- In POP Large programs are divided into smaller programs known as functions. But in OOP Programs are divided into objects.

2. Define interface

If a class is abstract, one of its subclasses is expected to implement its unimplemented methods. Although if any of the abstract class' subclasses do not implement all interface methods, the subclass itself must be marked again as abstract. Interfaces are commonly used in the Java language for callbacks.

3. List any 2 packages in java

- Language Support Package (java.lang)
- Networking Package (java.net)

4. Define synchronization

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.

5. Compare byte stream and character stream

    - Byte stream class that provide support for handling I/O operations on bytes
    - Character Stream classes that provide support for managing I/O operations on characters.

PART – B

II. Answer *any five* questions. Each question carries 6 marks

1. Explain the advantages of OOP
    - OOP provides a clear modular structure for programs.
    - It is good for defining abstract data types.
    - Implementation details are hidden from other modules and other modules has a clearly defined interface.
    - It is easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
    - objects, methods, instance, message passing, inheritance are some important properties provided by these particular languages
    - Encapsulation, polymorphism, abstraction are also counts in these fundamentals of programming language.
    - It implements real life scenario.
    - In OOP, programmer not only defines data types but also deals with operations applied for data structures.

2. Explain operator overloading. Why it is not supported in java?

    Operator overloading is a technique by which operators used in a programming language are implemented in user-defined types with customized logic that is based on the types of arguments passed. Operator overloading facilitates the specification of user-defined implementation for operations wherein one or both operands are of user-defined class or structure type. This helps user-defined types to behave much like the fundamental primitive data types. Operator overloading is helpful in cases where the operators used for certain types provide semantics related to the domain context and syntactic support as found in the programming language. It is used for syntactical convenience, readability and maintainability. Java does not support operator overloading, except for string concatenation for which it overloads the + operator internally. The reasons are,

- Simplicity and Cleanliness

  Simple and clear design was one of the goals of Java designers. Adding Operator overloading would have definitely made design more complex than without it, and it might have lead to more complex compiler or slows the JVM
- Avoid Programming Errors

  Java doesn't allow user defined operator overloading, because if you allow programmer to do operator overloading, they will come up with multiple meanings for same operator, which will make the learning curve of any developer hard and things more confusing and messy
- JVM Complexity

  From JVM perspective, supporting operator overloading is more difficult, and if the same thing can be achieved, by using method overloading in more intuitive and clean way, it does make sense to not support operator overloading in Java.
- Easy Development of Tools

  This is an additional benefit of not supporting operator overloading in Java. Omission of operator overloading has kept the language easier to handle and process, which in turn makes it easier to develop the tools, that process the language

3. Comment about the visibility controls in java

Public

A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe

```
class Example
{
publicint a;  //public field a
public void show()  //public method show
{Body;}
}
```

Private

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself. Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

```
class Example
{
privateint a;  //private field a
private void show()  //private method show
{Body;}
}
```

Default/friendly access

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. The fields in an interface are implicitly public static final and the methods in an interface are by default public

```
class Example
{
int a;  //default field a
void show()  //default method show
{Body;}
}
```

Protected

Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected. Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

```
class Example
```

```
{
int a;  //protected field a
void show()  //protected method show
{    Body;   }
}
```

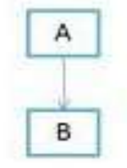
Private protected

Private protected members' visibility lies in between protected and private access. These members are visible in all sub-classes regardless of what package they are in.

4. Explain the different types of inheritance in java

Single Inheritance

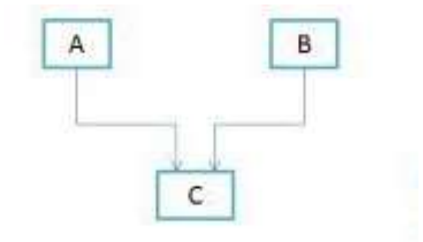
Single inheritance is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.



(a) Single Inheritance

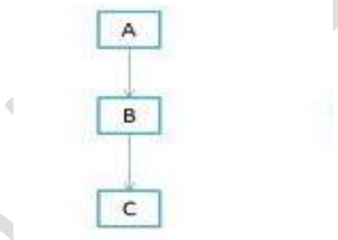
Multiple Inheritances

"Multiple Inheritance**"** refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.


(b) Multiple Inheritance

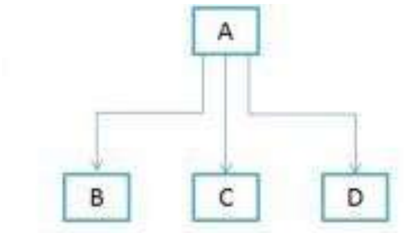
Multilevel Inheritance

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A. For more details and example refer – Multilevel inheritance in Java.


(d) Multilevel Inheritance

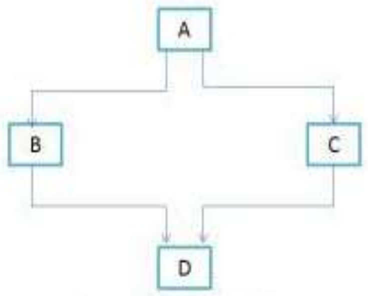
Hierarchical Inheritance

In such kind of inheritance one class is inherited by many **sub classes**. In below example class B,C and D **inherits** the same class A. A is **parent class (or base class)** of B,C & D. Read More at – Hierarchical Inheritance in java with example program.


(c) Hierarchical Inheritance

Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of **Single** and **Multiple** inheritances. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritances can be!! Using interfaces. Yes you heard it right. By using **interfaces** you can have multiple as well as **hybrid inheritance** in Java.



(e) Hybrid Inheritance

5. How is package accessed in a java program?

In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions. This is the general form of the importstatement:

import *pkg1*[.*pkg2*].(*classname*|*);

Here, *pkg1* is the name of a top-level package, and *pkg2* is the name of a subordinate package inside the outer package separated by a dot (.). There is no practical limit on the depth of a package hierarchy, except that imposed by the file system. Finally, you specify either an explicit *classname* or a star (*), which indicates that the Java compiler should import the entire package. This code fragment shows both forms in use:

```
Import java.util.Date;
import java.io.*;
```

Example:

```
import MyPack.*;
class TestBalance
```

```
{
  public static void main(String args[])
  {
  /* Because Balance is public, you may use Balance
  class and call its constructor. */
  Balance test = new Balance("J. J. Jaspers", 99.88);
  test.show(); // you may also call show()
  }
}
```

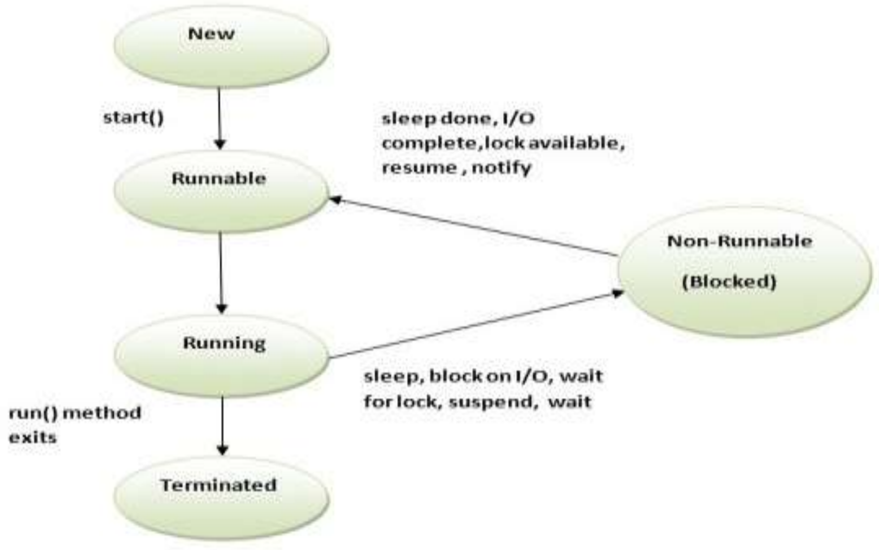
6. Explain the life cycle of a thread

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated


New
start()
Runnable
sleep done, I/O complete, lock available, resume , notify
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

New

When we create a new Thread object using *new* operator, thread state is New Thread. At this point, thread is not alive and it's a state internal to Java programming.

Runnable

When we call start() function on Thread object, it's state is changed to Runnable and the control is given to Thread scheduler to finish it's execution. Whether to run this thread instantly or keep it in runnable thread pool before running it depends on the OS implementation of thread scheduler.

Running

When thread is executing, it's state is changed to Running. Thread scheduler picks one of the thread from the runnable thread pool and change it's state to Running and CPU starts executing this thread. A thread can change state to Runnable, Dead or Blocked from running state depends on time slicing, thread completion of run() method or waiting for some resources.

Blocked/Waiting

A thread can be waiting for other thread to finish using thread join or it can be waiting for some resources to available, for example producer consumer problem or waiter notifier implementation or IO resources, then it's state is changed to Waiting. Once the thread wait state is over, it's state is changed to Runnable and it's moved back to runnable thread pool.

Dead

Once the thread finished executing, it's state is changed to Dead and it's considered to be not alive.

Above are the different **states of thread** and it's good to know them and how thread changes it's state

7. Explain about exception handling

The **exception handling** in java is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained. In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc. The core advantage of exception handling is to maintain the normal flow of the application. Exception normally disrupts the normal flow of the application that is why we use exception handling

Advantages of Exception Handling

- Exception handling allows us to control the normal flow of the program by using exception handling in program.
- It throws an exception whenever a calling method encounters an error providing that the calling method takes care of that error.
- It also gives us the scope of organizing and differentiating between different error types using a separate block of codes. This is done with the help of try-catch blocks.

Tasks incorporated with exception handling

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

(5 x 6 = 30)

PART – C

(Answer *one* full question from each unit. Each question carries 15 marks)

UNIT – I

III.

a) Write the different features of java

8

Simple :

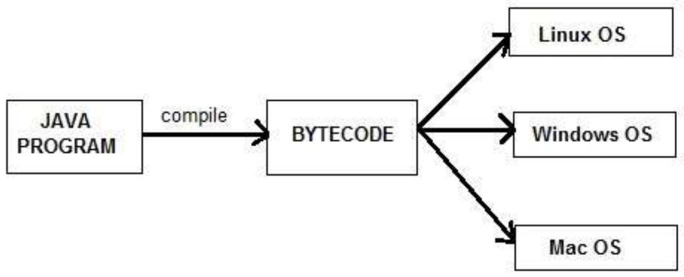
- Java is Easy to write and more readable and eye catching.
- Java has a concise, cohesive set of features that makes it easy to learn and use.
- Most of the concepts are drew from C++ thus making Java learning simpler.

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

b) How is a constructor implemented in java

A **java constructor** has the same name as the name of the class to which it belongs. Constructor's syntax does not include a return type, since constructors never return a value. Constructors may include parameters of various types. When the constructor is invoked using the new operator, the types must match those that are specified in the constructor definition. Java provides a default constructor which takes no arguments and performs no special actions or initializations, when no explicit constructors are provided. The only action taken by the implicit default constructor is to call the superclass constructor using the super() call. Constructor arguments provide you with a way to provide parameters for the initialization of an object.

```
class Programming
{       //constructor method
        Programming()
        {
          System.out.println("Constructor method called.");
        }
        public static void main(String[] args)
        {
          Programming object = new Programming(); //creating object
        }
}
```

OR

IV.

a) Comment about the classes and object declaration

8

```
class MyClass {
    // field, constructor, and
    // method declarations
}
```

This is a *class declaration*. The *class body* (the area between the braces) contains all the code that provides for the life cycle of the objects created from the class: constructors for initializing new objects, declarations for the fields that provide the state of the class and its objects, and methods to implement the behavior of the class and its objects.

The preceding class declaration is a minimal one. It contains only those components of a class declaration that are required. You can provide more information about the class, such as the name of its superclass, whether it implements any interfaces, and so on, at the start of the class declaration. For example,

```
class MyClass extends MySuperClass implements YourInterface {
    // field, constructor, and
    // method declarations
}
```

While the declaration of an object is not a necessary part of object creation, object declarations often appear on the same line as the creation of an object. Like other variable declarations, object declarations can appear alone like this:

```
        Date today;
```

Either way, declaring a variable to hold an object is just like declaring a variable to hold a value of primitive type:

```
        type name
```

where *type* is the data type of the object and *name* is the name to be used for the object. In Java, classes and interfaces are just like a data type. So *type* can be the name of a class such as the Date class or the name of an interface

b) Explain function overriding with an example

7

- In Method Overriding, sub class has the same method with same name and exactly the same number and type of parameters and same return type as a super class.
- Method Overriding means method of base class is re-defined in the derived class having same signature.
- Method Overriding is to “Change” existing behavior of method.
- It is a **run time polymorphism.**
- It always requires inheritance in Method Overriding.
- In Method Overriding, methods must have **same signature.**
- In Method Overriding, relationship is there between methods of super class and sub class.

- In Method Overriding, methods have same name and same signature but in the different class.Method Overriding requires at least two classes for overriding.

```
Eg: Class A          // Super Class
{
void display(intnum)
  {
printnum ;
   }
}                    //Class B inherits Class A
Class B              //Sub Class
{
void display(intnum)
  {
printnum ;
   }
}
```

UNIT – II

V.

a) Explain the private, public and protected access specifiers in java

9

Public

A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe

```
class Example
{
publicint a;  //public field a
public void show()  //public method show
{Body;}
}
```

Private

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself. Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

```
class Example
{
privateint a;  //private field a
private void show()  //private method show
{Body;}
}
```

Protected

Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected. Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

```
class Example
{
int a;  //protected field a
void show()  //protected method show
{   Body;   }
}
```

b) How is an interface defined and the methods in it called

6

An interface declaration consists of modifiers, the keyword interface, the interface name, a comma-separated list of parent interfaces (if any), and the interface body. For example:

```
public interface GroupedInterface extends Interface1, Interface2, Interface3
{
   // constant declarations
   // base of natural logarithms
   double E = 2.718282;
   // method signatures
   void doSomething (int i, double x);
   int doSomethingElse(String s);
}
```

The public access specifier indicates that the interface can be used by any class in any package. If you do not specify that the interface is public, then your interface is accessible only to classes defined in the same package as the interface. An interface can extend other interfaces, just as a class subclass or extend another class. However, whereas a class can extend only one other class, an interface can extend any number of interfaces. The interface declaration includes a comma-separated list of all the interfaces that it extends.

Default methods can be provided to an *interface* without affecting implementing classes as it includes an implementation. If each added method in an interface defined with implementation then no implementing class is affected. An implementing class can override the *default implementation* provided by the interface.

OR

VI.

a) Write a java program to show the inheritance of bus from a car with the necessary methods

9

```
interface car
     {
       int  speed=90;
       public void distance();
     }
     interface bus
     {
      int distance=100;
      public void speed();
     }
     class vehicle  implements car,bus
     {
     public void distance()
     {
     int  distance=speed*100;
     System.out.println("distance travelled is"+distance);
     }
      public void speed()
     {
      int speed=distance/100;
```

```
    }
   }
     class maindemo
    {
      public static void main(String args[])
     {
      System.out.println("Vehicle");
     Vechicle v1=new Vehicle();
     v1.distance();
     v1.speed();
     }
     }
```

b) What is the use of inheritance in java? Explain with example

6

Inheritance is one of the fundamental features of object-oriented programming. Its main uses are to enable polymorphism and to be able to reuse code for different classes by putting it in a common superclass. The most fundamental element of Java is the class. A class represents an entity and also, defines and implements its functionality. In Java, classes can be **derived** from other classes, in order to create more complex relationships. A class that is derived from another class is called *subclass* and inherits all fields and methods of its *superclass*. In Java, only single inheritance is allowed and thus, every class can have at most one direct superclass. A class can be derived from another class that is derived from another class and so on. Finally, we must mention that each class in Java is implicitly a subclass of the Object class.

Suppose we have declared and implemented a class *A*. In order to declare a class *B* that is derived from *A*, Java offers the *extend* keyword that is used as shown below:

```
class A {
        //Members and methods declarations.
}

class B extends A {
        //Members and methods from A are inherited.
```

```
                //Members and methods declarations of B.
        }
```

UNIT – III

VII.

a) What are the advantages in using packages

7

- Anyone can easily determine which files are related.
- Name space collision is minimized.
- One can allow types in one package to have unrestricted access to one another, still restricting the access for the types outside the package.
- Java packages can be stored in compressed files called JAR files, thus allowing the classes to download faster as a group rather than downloading each one at a time.
- Packages can contain hidden classes that are used by the package but are not visible or accessible outside the package.
- Classes in packages can have fields and methods that are visible by all classes inside the package, but not outside.
- Different packages can have classes with the same name. For example, java.awt.Frame and photo.Frame.

b) Explain the terms:

i. Throw
ii. Throws
iii. Try
iv. Catch

8

throw

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

Throws

The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained

SYNTAX:

```
return_type method_name() throws exception_class_name{
//method code
}
```

Try

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.

Catch

Java catch block is used to handle the Exception. It must be used after the try block only.

SYNTAX:

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

OR

VIII.

a) Write the different java API packages

8

a. Language support packages(java.lang)

java.lang Provides classes that are fundamental to the design of the Java programming language such as String, Math, and basic runtime support for threads and processes.

Example classes: String, Thread. It contain language support classes

b. Networking Packages(java.net)

The java.net package provides a powerful and flexible infrastructure for networking. Many of the classes in this package are part of the networking infrastructure and are not used by normal applications; these complicated classes can make the package a difficult one to understand.

Example classes: InetAddresss, Socket

c. i/o Package(java.io)

Java.io package provides classes for system input and output through data streams, serialization and the file system. The Java I/O package, a.k.a. java.io, provides a set of input streams and a set of output streams used to read and write data to files or other input and output sources. There are three categories of classes in java.io: input streams, output streams and everything else.

Example classes: InputStreamReader, BufferedReader

d. Utilities Package (java.util)

Java.util package contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes.

b) Explain synchronization methods in threads with example

7

The Java synchronized keyword is an essential tool in concurrent programming in Java. Its overall purpose is to only allow one thread at a time into a particular section of code thus allowing us to protect, for example, variables or data from being corrupted by simultaneous modifications from different threads. This article looks at how to use synchronized in Java to produce correctly functioning multithreaded programs. Other articles in this section look at other Java 5 concurrency facilities which have in fact superseded synchronized for certain tasks.

At its simplest level, a block of code that is marked as `synchronized` in Java tells the JVM: *"only let one thread in here at a time"*.

Imagine, for example, that we have a counter that needs to be incremented at random points in time by different threads. Ordinarily, there would be a risk that two threads could simultaneously try and update the counter at the same time, and in so doing currpt the value of the counter (or at least, miss an increment, because one thread reads the present value unaware that another thread is just about to write a new, incremented value). But by wrapping the update code in a `synchronized` block, we avoid this risk:

```
public class Counter
{
  private int count = 0;
  public void increment()
  {
    synchronized (this)
   {
      count++;
    }
  }
  public int getCount()
  {
    synchronized (this)
    {
      return count;
     }
  }
}
```

UNIT – IV

IX.

a) What is exception? How it is handled?

8

Exceptions are the customary way in Java to indicate to a calling method that an abnormal condition has occurred.The **exception handling** in java is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained. In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc. The core advantage of exception handling is to maintain the normal flow of the application. Exception normally disrupts the normal flow of the application that is why we use exception handling

Advantages of Exception Handling

- Exception handling allows us to control the normal flow of the program by using exception handling in program.
- It throws an exception whenever a calling method encounters an error providing that the calling method takes care of that error.
- It also gives us the scope of organizing and differentiating between different error types using a separate block of codes. This is done with the help of try-catch blocks.

Tasks incorporated with exception handling

5) Find the problem (Hit the exception)
6) Inform that an error has occurred(Throw the exception)
7) Receive the error information(Catch the exception)
8) Take corrective actions(Handle the exception)

b) What are stream classes?

7

A stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination. InputStream and OutputStream are the basic stream classes in Java.

All the other streams just add capabilities to the basics, like the ability to read a whole chunk of data at once for performance reasons (BufferedInputStream) or convert from one kind of character set to Java's native unicode (Reader), or say where the data is coming from (FileInputStream, SocketInputStream and ByteArrayInputStream, etc.)

Input streams

The class `java.io.InputStream` is the base class for all Java IO input streams. If you are writing a component that needs to read input from a stream, try to make our component depend on an `InputStream`, rather than any of its subclasses (e.g. `FileInputStream`). Doing so makes your code able to work with all types of input streams, instead of only the concrete subclass

Following is the declaration for **Java.io.InputStream** class:

```
public abstract class InputStream
 extends Object
```

```
implements Closeable
```

OutputStream

The class java.io.OutputStream is the base class of all Java IO output streams. If you are writing a component that needs to write output to a stream, try to make sure that component depends on an OutputStream and not one of its subclasses.

Following is the declaration for **Java.io.OutputStream** class:

```
public abstract class OutputStream
  extends Object
    implements Closeable, Flushable
```

OR

X.

a) Explain the different types of exceptions

7

Exception	Description
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
UnsupportedOperationException	An unsupported operation was encountered.

b) Write the different methods in applet. Explain each

8

- The init() method is called exactly once in an applet's life, when the applet is first loaded. It's normally used to read PARAM tags, start downloading any other images or media files you need, and set up the user interface. Most applets have init() methods.
- The start() method is called at least once in an applet's life, when the applet is started or restarted. In some cases it may be called more than once. Many applets you write will not have explicit start()methods and will merely inherit one from their superclass. A start() method is often used to start any threads the applet will need while it runs.
- The stop() method is called at least once in an applet's life, when the browser leaves the page in which the applet is embedded. The applet's start() method will be called if at some later point the browser returns to the page containing the applet. In some cases the stop() method may be called multiple times in an applet's life. Many applets you write will not have explicit stop()methods and will merely inherit one from their superclass. Your applet should use the stop() method to pause any running threads. When your applet is stopped, it *should* not use any CPU cycles.
- The destroy() method is called exactly once in an applet's life, just before the browser unloads the applet. This method is generally used to perform any final clean-up. For example, an applet that stores state on the server might send some data back to the server before it's terminated. many applets will not have explicit destroy() methods and just inherit one from their superclass.