PAGE NO.:
DATE: / /

☆ Generics in java and commonly used classes:

Q. Discuss the benifit of generic over non-generic types.

Ans: Code that uses generics has many benifits over non-generic code:

- Stronger type checks at compile time.

A java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.

- Elimination of casts:

-> Code snippest without generic requiring casting:

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

-> When re-written to use generics, the code

does not requiring casting:

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0); // no cast.
```

→ Enabling programmers to implement generic algorithms:

By using generics, programmer can implement generic algorithms that work on collection of different types, can be customized and are type safe and easier to read.

Q. What is erasure with reference to generics.

Ans. Generics are implemented in java is in order.

An important constraint that governed ~~by~~ the way generics were added to java was the need for compatibility with previous versions of java. Simply put: generic code had to be compiled with preexisting, nongeneric code. Thus any changes to the syntax of java language, or to the JVM had to avoid breaking older

code. The way java implements generics while satisfying this constraints is through the use of erasure.

In general, here is how erasure works, When your java code is compiled, all generic type information is removed. This means replacing type parameters with their bound type, which is Object if no explits bound is specified and then applying the appropriate casts to maintain type compatibility with the types specified by the type arguments. The compiler also enforces this type compatibility. This approach to generics means that no type parameters exist at run time.

They are simply a source-code mechanism.

Q. write shortnotes on:

(1) Generic Constructors:

A constructor can be generic, even if its class is not. for eg. in the following program, the class summation is not generic, but its constructor is.

eg

```
class summation {
    private int sum;
    // generic constructor.
    <T extends number> Summation (T arg) {
        sum = 0;
        for (int i=0; i<= arg.intValue(); i++)
            sum += i;
    }
    int getsum() {
        return sum;
    }
}
class gencondemo {
    public static void main (String args[]) {
        Summation ob = new Summation (4.0);
        System.out.println("Summation of 4.0 is "
                + ob.getsum());
    }
}
```

The summation class computes and encapsulate the summation of the numeric value passed to its constructor. Recall that the summation of N is the sum of all the whole numbers between 0 and N. Because summation's specifys a type parameter that is bounded by Number, a summation object cannot be constructed by using any numeric type, including Integer, Float or Double. No matter what numeric type is used, its value is converted to Integer by calling intValue() and the summation is computed. Therefore is not necessary for the class Summation to be Generic; only a generics constructor is needed.

(ii) Generic Interface:

PAGE NO.:
DATE: / /

Generic Interface:

Generics also work with interface. Thus, you can also have generic interface. Generic interfaces are specified just like generic classes for example.

```
interface containment<T> {   // generic interface.
    boolean contains(T o);
}

class MyClass<T> implements Containment<T>{
    T [] arrayRef;
    MyClass(T[] o) {
        arrayRef = o;
    }
    // implement contains.

    public boolean contains(T o) {
        for (T x : arrayRef)
            if(x.equals(o)) return true;
        return false;
    }
}

class GenIfDemo {
    public static void main(String args[]) {
        Integer x = {1, 2, 3};
        MyClass<Integer> ob = new MyClass<Integer>(x);
```

```
if (ob.contains(2))
    System.out.println("2 is in ob");
else
    System.out.println("2 is NOT in ob");

if (ob.contains(5))
    System.out.println("5 is in ob");
else
    System.out.println("5 is NOT in ob");

    }
}

// output:

2 is in ob.
5 is NOT in ob.
```

☆ Generic Functional Interface.

A lambda expression can't specify type parameters. So it's not generic. However, functional interface associated with lambda expression is generic. In this case, the target type of lambda expression is determined by the type of argument(s) specified when functional interface reference is used

PAGE NO.:
DATE: / /

Syntax:

```
interface SomeFunc {

    T func(T t);
}
```

eg

```
interface MyGeneric<T> {
    T compute(T t);
}

public class GenericInterface {
    public static void main(String args[])
    {
        MyGeneric<String> reverse = (str) -> {
            // lambda expression
            String result = "";
            for(int i = str.length()-1; i>=0; i--)
                result += str.charAt(i);
            return result;
        };

        MyGeneric<Integer> factorial = (Integer n) -> {
            // lambda expression
            int result = 1;
            for(int i=1; i<=n; i++)
                result = i * result;
            return result;
        };
```

PAGE NO.:
DATE: / /

```
System.out.println(reverse.compute("lambuda Generic functional interface"));
System.out.println(factorial.compute(7));

}
}
```

// output.

ecafretni lanoitcnuf cireneG adabmal
5040.

(3) Generic Methods:

Generic methods are methods that introduce their own type parameters. This is similar to declaring a generic type, but the type parameter's scope is limited to the method where it is declared. Static and non-static generic methods are allowed, as well as generic class constructor.

The syntax for a generic method includes a list of type parameters, inside angle brackets, which appears before the method's return type. For static generic methods, the type parameter section must appear before the method's return type.

The util class includes a generic method compare which compares two pair objects.

```
public class util {
    public static <K, V> boolean compare(Pair<K, V>
                                    p1, Pair<K, V> p2) {
        return p1.getKey().equals(p2.getKey())
            && p1.getValue().equals(p2.getValue());
    }
}
```

```
public class pair <K, V> {
    private K key;
    private V value;
}

    public Pair (K key, V value) {
        this.key = key;
        this.value = value;
    }
    public void setkey (K key) { this.key = key; }
    public void setValue (V Value) {
        this.value = value; }
    public K getkey () {
        return key; }
    public V getValue () {
        return value; }.
    }
```

-> The complete syntax for invoking this method would be:

```
Pair <Integer, String> P1 = new Pair <> (1, "Apple");
Pair <Integer, String> P1 = new Pair <> <2, "Pear">;
boolean same = Util. <Integer, String> compare
                                    (P1, P2);
```

Unit - 6

Q.1 Discuss Comparator & Comparable.

→ Comparable :

→ Comparable is an interface which defines a way to compare an object with other objects of the same type. It helps to sort the objects that have self-tendency to sort themselves.

→ The object must know how to order themselves.

→ E.g.. Roll NO. , age, salary, ..

→ This Interface is found in java.lang package & it contains only one method i.e... , CompareTO( ).

→ Comparable is not capable of sorting the objects on its own, but the interface defines a method int CompareTo( ) which is responsible for sorting.

EXCEL™

Comparator :

→ A Comparator interface is used to order the objects of a specific class. This interface is found in java.util.package.

→ It contains two methods:

- Compare (Object obj1, Object obj2)
- Equals (Object element)

→ The first method, Compare (Object obj1, Object obj2) compares its two input arguments & showcase the output. It returns a negative integer, zero, or a positive integer to state whether the first argument is less than, equal to, or greater than the second.

→ The second method, equals (Object element), requires an Object as a parameter & shows if the input object is equal to the Comparator. The method will return true, only if the mentioned object is also a Comparator. The order remains the same as that of the Comparator.

Q.2 List at least 5 predefined Functional Interface.

⇒ Functional Interface :

→ In java is an interface that Contain only one Single abstract method

→ A functional Interface can Contain default & Static methods which do have an implementation.

Predefined Functional Interface :

→ In many Cases, however you won't need to define your own functional Interface because JDK 8 adds a new package called java.util.function that provides several predefined ones.

Interfaces that are predefined :

1. Unary Operator <T>

→ Apply a unary operation to an object of type T & return the result, which is also of type T Its method is called apply().

2. Binary Operator <T> :

→ Apply an operation of 2 objects of type T & return the result. Its method is called apply().

3. Consumer <T> :

→ Apply an operation on an object of type T. Its method is called accept().

4. Supplier <T> :

→ Return an object type T, Its method is called get().

5. Function <T, R> :

→ Apply an operation to an object of type T and return the result as an object of type R, its method is called apply().

Unit 10 The java.io Package

Q.1 Difference between Byte streams & Character streams.

Byte stream	Character stream
→ Byte streams provide a convenient means for handling input & output of Bytes.	→ Character streams are designed for handling the input and output of characters.
→ They are used for when reading or writing binary data.	→ They use Unicode and, therefore, can be internationalized.
→ All byte streams are descended from InputStream & OutputStream.	→ All character stream classes are descended from Reader & writer.
→ preforms input & output operations of 8 bit bytes.	→ Performs input & output operations of 16 bit Unicode.
→ e.g. images, sounds, etc..	→ e.g. plain text files, web pages, user keyboard input. etc..

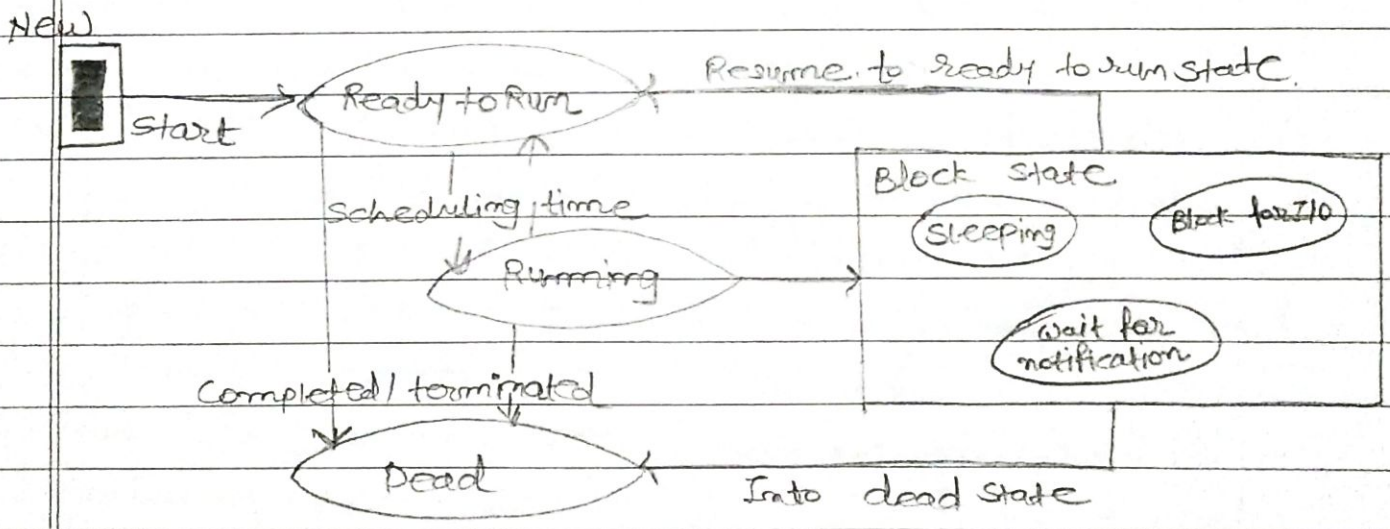
Unit-II Threads

Q.1 Explain Thread life-cycle with diagram.

→ A thread passes through several states throughout its life time. & after completing its task, it is dead.

→ The different states of a thread are as follows:

1) New
2) Ready to run (Runnable)
3) Running
4) Block (waiting)
5) Dead. (terminated)



1) New : At the time of thread creation, it is in the New state. By calling the start() method, the thread will start its execution.

2) ready to run : After the start() method, the thread is in the ready to run state. It means that now the thread is ready for CPU allocation but still the CPU is not allocated it. So, we can say that the thread is runnable but not running.

3) Running : By calling the Run() method, the CPU time is allocated to the thread and it enters the running state.

→ In the running state, the thread executes its task for which the thread was created. After the CPU time slice expires, the thread may go back to the ready to run state if its task is uncompleted or may go to the dead state if its tasks is completed or may go to the block state.

4) Block : A thread may enter the block state for several reasons. For example, the thread itself or some

XCEL™

other threads may invoke the sleep() method to wait for an I/O operation to finish.

→ A block thread may enter the dead or the ready to run state after its block state is Completed.

5) dead : A thread is dead if it Completes the execution of its run() method or if its stop() method is invoked.

EXCEL™ Excellence Driven By Quality

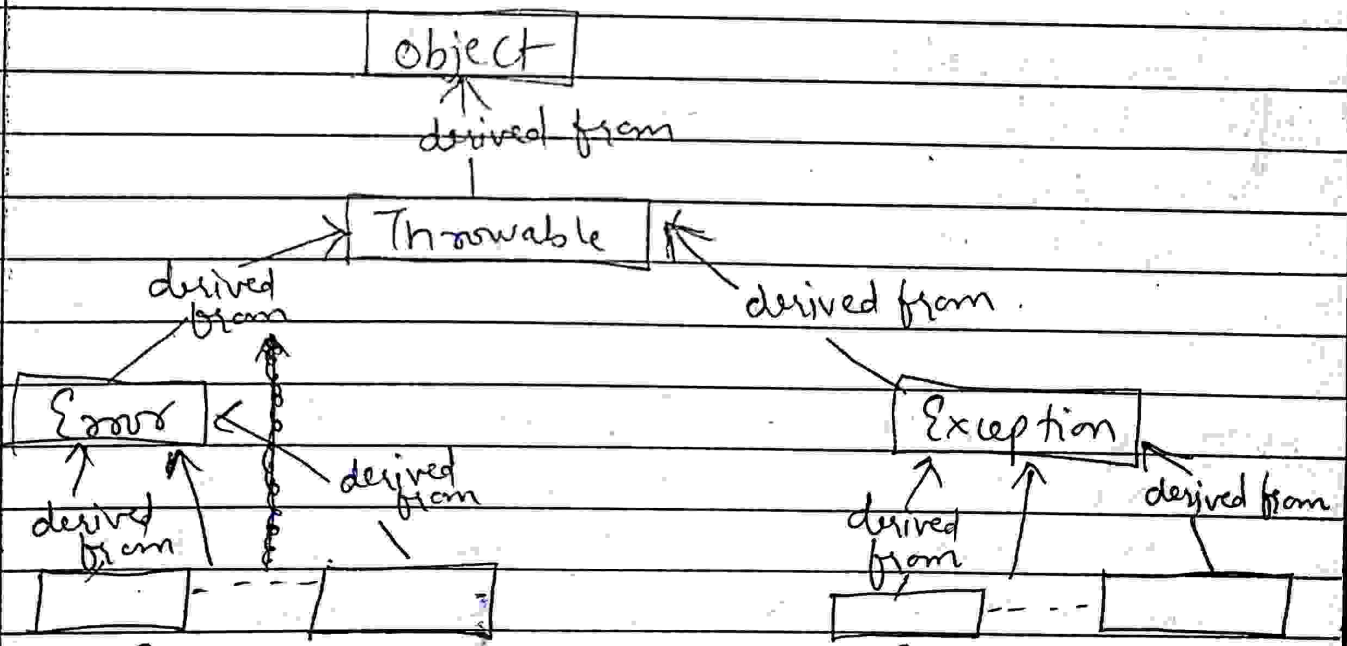
PAGE NO :-
DATE

Unit 7 (Exceptions):

1. Discuss Exception & types of Exceptions in Java.

Ans Two direct subclasses of class Throwable - the class Error & class Exception - cover all the standard Exception.

Fig Hierarchy to which these classes belong.



Error Exceptions

→ Error has three direct subclasses -

PAGE NO :-
DATE

ThreadDeath, LinkageError & VirtualMachineError.

- A ThreadDeath exception is thrown whenever an executing thread is deliberately stopped & for the thread to be destroyed properly, you should not catch this exception.

- The LinkageError exception class has subclasses that record serious errors with the classes in program.

Incompatibilities between classes or attempting to create an object of a nonexistent class type are the sort of things that cause these exception to be thrown.

- The VirtualMachineError class has four subclasses that specify exceptions that will be thrown when a catastrophic failure of JVM occurs.

RuntimeException Exceptions

RuntimeException are treated differently because of serious error in your code.

PAGE NO :-
DATE

→ Quite a lot of subclasses of RuntimeException are used to signal problems in various packages in Java class library.

Subclasses of RuntimeException defined in standard package java.lang are:

Class Name	Exception condition Represented.
ArithmeticException	An invalid Arithmetic condition has arisen, such as an attempt to divide an integer value by 0
IndexOutOfBoundException.	To use an index that is outside the bounds of object it is applied to. This may be an array, a string object, or vector object.
NegativeArraySizeException.	To define an array with a negative dimension.
NullPointerException.	An object variable containing null is used, when it should refer to an object for proper operation
ArrayStoreException	To store an array that isn't permitted for the array type.

PAGE NO :-
DATE

2 Explain commonly used methods define by Throwable.

Ans: All exceptions are subclasses of Throwable so all Exceptions support the methods defined by Throwable.

⇒ Commonly used method defined by Throwable

1. Throwable fillnStackTrace()

Returns a Throwable object that contains a completed stack trace. This object can be rethrown.

2 String getLocalizedMessage()

Returns a localized description of Exception

3. String getMessage()

This Returns the contents of message, describing the current exception. This will typically be fully qualified name of the exception class & a

4. void printstackTrace()

This will output the message & the

PAGE NO :-
DATE

Stack trace of Standard error output stream - which is Screen in case of console program.

5 void printStackTrace(PrintStream s)

~~This will output the message & the Stack trace of Standard error output stream - which is screen in case of console program.~~

6 This is same as previous method except that can specify the output stream as an argument. Calling previous method for an Exception object e is equivalent to:

e.printStackTrace(System.err);

6. void printStackTrace(PrintWriter stream)

Sends the stack trace to the specified stream

7 String toString()

Returns a String object containing a complete description of exception. This method is called by println() when

out putting a Throwable object.

Program demonstrats these methods:

```
class ExcTest {
    static void genException() {
        int nums[] = new int[4];

    System.out.println ("Before Exception");

    num[7] = 10;
    System.out.println ("won't be displayed");
}
}

class UseThrowableMethods {
    public Static void main(String args[]) {
        try {
            ExcTest.genException();
        }
        catch (ArrayIndexoutofBaundsException exc)
        {
            System.out.println ("Standerd message is:");
            System.out.println (exc);
            System.out.println ("In Stack trace:");
            exc.printStackTrace ();
        }
        System.out.println ("After catch");
    }
}
```