Page No.
Date

Devanshu Surana
PC-23, 1032210755

OOCCJ Theory Assignment

Q1) Differentiate between error and exception in Java.

→ In Java, an error and an exception are two different concepts that describe different kinds of problems that can arise in a program. An error is a serious issue that usually occurs due to a problem with the system or hardware. Errors are typically unrecoverable and cannot be handled by the program. Example: OutofMemory Error, StackOverflowError, and VirtualMachine Error.

Exception is a less serious issue that occurs within the program's logic and can be handled by the program. Exceptions are typically caused by incorrect user input, incorrect coding, or unexpected conditions that arise during program execution. Example: NullPointer Exception, Arithmetic Exception, and IO Exception.

Exceptions in Java are categorized into two types: checked & unchecked exceptions. Checked Exceptions are those that the compiler checks for at ~~att~~ compile time and require the program to handle them using try-catch blocks or declaring them in the method signature. Example for checked exceptions include IO Exception and SQL Exception.
Unchecked Exceptions are those that the compiler does not check for at compile time and can occur at runtime.
Ex: NullPointer Exception and ArrayIndexOutofBoundsException.

Teacher's Sign.: ____________

Q2) What is the use of BufferedWriter and BufferedReader in Java?

Ans → The BufferedWriter and BufferedReader classes in J are used for handling character streams.

"The BufferedWriter class provides buffering capabilitie write characters to a file, a stream, or a writer. By buffering the output, performance is improved, as it the number of writes to the stream or file. To write characters to a file using "BufferedWriter", th "write()" method is called to write characters to buffer.

Ex:

```
    import java.io.filewriter;
    import java.io.BufferedWriter;
    public class test {
     public static void main (String args[]) {
        String a = " something";
     try {
       FileWriter fw= new fileWriter("file.txt");
       BufferedWriter bw = new BufferedWriter (fw);
       bw. write (a);
       System.out.println("Data stored successfully.
       bw.close();
     }
       catch (Exception e) {
           e.printStack Trace();
       }
```

The "BufferedReader"
to read characters fro
By buffering the inpu
reduces the number o
To read characters from
"read()" method is
buffer, and then the
the reader.

Ex:

```
        import java.io
        import java.i
        import java.
         public class
           public sta

           Buffered R

       System.out.print
        inputStream.c
        }
      }
```

) Explain briefly th
Collection framewo
interfaces for
of collection frame
1. Collection Interfa
which defines th
on a collection
checking for p

The "BufferedReader" class provides buffering capabilities to read characters from a file, a stream, or a reader. By buffering the input, performance is improved, as it reduces the number of reads from the stream or file. To read characters from a file using a "BufferedReader" the "read()" method is called to read characters from the buffer, and then the 'close()' method is called to close the reader.

Ex:

```
    Import java.io. FileReader;
    import java.io. BufferedReader;
    import java.io. IOException;
    public class Chartest {
       public static void main (String args []) throws
                                          IO Exception {
       BufferedReader = new BufferedReader (new fileReader
                                         ("output.txt"));
System.out.println (inputStream.readline ());
inputStream.close ();
   }
}
```

3) Explain briefly the hierarchy of collection framework in Java.

Collection framework in Java provides a set of classes and interfaces for managing collections of objects. The hierarchy of collection framework in Java is as follows:

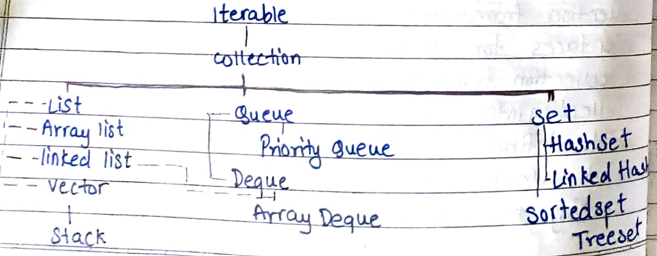
1. Collection Interface : Root interface of the collection hierarchy which defines the basic operations that can be performed on a collection of objects, such as adding, removing, and checking for presence of elements.

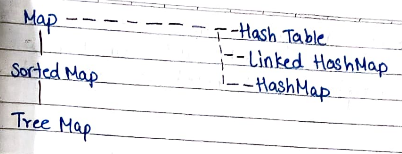
The 'collection' interface has been extended by 2 ~~Sub~~ sub-interfaces:
a. 'List' Interface: This sub interface extends the 'c interface and represents an ordered collection of elem that can contain duplicates.
b. 'Set' Interface: Represents a collection of elements cannot contain duplicates.

2. Queue Interface: This interface extends the 'Collec interface and represents a collection that supports elements insertion and removal at both ends.
a. 'Deque' Interface: This sub-interface extends the 'Queue' Interface and represents a double-ended which supports insertion and removal at both ends.

3. Map Interface: Represents mapping between a set of and their associated values. Each key can map to at one value.

4. Iterator Interface: This interface provides a way to access the elements of a collection one at a time and the caller to remove elements from the underlying collection.





) Write a Java program to convert LinkedList to ArrayList.

```
import java.util.LinkedList;
import java.util.ArrayList;
public class LinkedList To ArrayList {
    public static void main (String [] args) {
     LinkedList <string> linkedlist = new LinkedList <> ();
      linkedlist.add (" apple");
      linkedlist.add (" banana");
      linkedlist.add ( "grapes");
      linked list.add ("berry");

// convert the linkedlist to an arraylist.
     ArrayList <string> arrayList = new ArrayList <> (linkedlist);

 // print the elements of the array list
     for ( String fruit : array List) {
          System.out.println (fruit);
     }
   }
}
```

In this program, we first create a LinkedList and add some elements to it. Then, we create an ArrayList and pass the LinkedList as a parameter to the ArrayList constructor.

This creates a new 'ArrayList' that contains the elements as the original 'linkedlist'. Finally, we through the 'ArrayList' and print. its elements