11/10/23

ASSIGNMENT 3

Unit-5&6 Exception Handling and Multithreading & I/O and applets.

Q.1 Differentiate the errors and exceptions in Java.

—↓

Errors	Exceptions
> Recovering from error is not possible	We can recover from exceptions by either using try-catch block or throwing exception back to the caller.
> All errors are unchecked type	Exceptions include both checked as well as unchecked type
> Errors can occur at compile time	unchecked occurs at runtime whereas checked occurs at compile time
> They are defined in java.lang.Error package	They are defined in java.lang.Exception package
> eg: Stack overflow error, Out of memory error.	SQL exception, IO exception, Array index out of Bound exception.

Q.2 What is an exception? Demonstrate how you can handle different types of exceptions separately.

—↓ An Exception is an event that occurs during the execution of a program that disrupts the

normal flow of program. Exceptions are objects that represent these abnormal conditions and are used to handle errors and unexpected situations. A program can deal with an exception in one of three ways:

i) ignore it
ii) handle it where it occurs
iii) handle at another place in the program

→ Types to handle exception separately:

i) try: The 'try' keyword is used to specify a block where we should place exception code

ii) catch: used to handle the exception. It must be preceded by try block which means we can't use catch block alone

iii) finally: used to execute the important code of the program even if the exception is generated or not

iv) throw: used to throw an exception

v) throws: the 'throws' keyword is used to declare exceptions. It doesn't throw an exception.

Q.3 Differentiate the following:

i)	Checked Exception	Unchecked Exception
>	It happens at compile time when the source code is transformed into a executable code	It happen at runtime when the executable program starts running
>	It is checked by the compiler	It is not checked by the compiler.
>	It can be created manually	They can also be created manually.
>	It is counted as a sub-class of the class	It happens in runtime and not included in exception.
>	JVM requires the exception to be caught	JVM does not need exception to be caught.

ii)	Throw	Throws
>	Used to explicitly throw an exception	used to declare an exception.
>	Checked exception cannot be propogated using throw only	Checked exception can be propagated with throws.
>	Followed by an instance	Followed by a class.

> Throw is used within the method	Throws is used with method signature.
> You cannot throw multiple exception	You can declare multiple exception.

Q.4 Explain exception handling in Java. Write an application that generates custom exceptions.

-> ~~For~~ Exception Handling is a mechanism for managing runtime errors and unexpected events that may occure during the execution of a program. It allows you to gracefully handle these exceptions by providing appropriate error messages without causing the program to crash. Java provides keywords like 'try', 'catch', 'finally', 'throw', 'throws' for exception handling.

eg:
```
class CustomException extends Exception {
    public CustomException (String Message) {
        super (message);
    }
}
public class Demo {
    public static void main (String args[]) {
        try {
            int age = 15
            if (age < 18) {
```

```
        throw new CustomException ("You must be
        atleast 18 years old.");
    }
    System.out.println ("You are eligible");
} catch (CustomException e) {
    System.out.println ("Message: " + e.getMessage());
} finally {
    System.out.println ("Closing Resources.");
    }
}
```

Q.5 Explain & Illustrate use of final, finally and finalize by examples.

—↓ i) Final

The 'final' keyword is used to declare a variable, method or class as unchangeable or not extendable. When applied to a variable, then its value cannot be modified once assigned.

ii) Finally

The 'finally' block is used to ensure that a particular block of code always gets executed whether an exception is thrown or not.

iii) Finalize

The 'finalize' method is a method called by the garbage collector when an object is being reclaimed by the garbage collector. or

it is about to be destroyed. This method can be used to override to perform cleanup operations before the object is destroyed.

eg:-
```
public class Demo {
    final int var = 10;
    public void exampleMethod() {
        try {
            int result = 10/0;
        } catch (ArithmeticException e) {
            System.out.println("Error");
        } finally {
            System.out.println("Finally Block");
        }
    }
    protected void finalize() {
        System.out.println("Finalize Method");
    }
    public static void main(String args[]) {
        Demo demo = new Demo();
        System.out.println("Final variable value: " +
        demo.var);
        demo.exampleMethod();
        System.gc();
    }
}
```

Q.6 Define thread. What are the different ways to define / Implement a thread?

→ A Thread is the smallest unit of a program's

execution and it represents single flow of control within a program. Threads enable concurrent execution, allowing a program to perform multiple tasks simultaneously.

→ There are two main ways to implement a thread:

i) By extending the Thread class

eg:
```
class MyThread extends Thread {
    public void run() {
        System.out.println("Hello"); }
}
```

→
```
// Create and start a thread
MyThread thread = new MyThread();
thread.start();
```

ii) By implementing the runnable interface

eg:
```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Hello"); }
}
```

Q.7 Differentiate multiprocessing and multi-tasking. How to give priorities to threads? Write a program to demonstrate the same.

	Multiprocessing	Multitasking
>	Running multiple processes on multiple CPUs.	Running multiple tasks on a single CPU
>	Each process has its own set of resources.	Resources are shared among task
>	Each process can have its own scheduling ~~alog~~ algorithm	uses priority-based scheduling to allocate CPU time to tasks.
>	Each process has its own memory spaces.	Each task has its own memory space
>	uses inter-process communication mechanism	uses message passing or shared memory

—> Thread Priorities is a number assigned to a thread that is used by Thread scheduler to decide which thread should be allowed to execute.

To give priorities to threads, you can use the 'setPriority' method. By default, all threads have a priority of (5).

i) MIN_PRIORITY = 1

ii) NORM_PRIORITY = 5

iii) MAX_PRIORITY = 10.

eg:
```
public class Demo {
    public static void main (String args[]) {
        Thread highpriority = new Thread (new
            Worker ("High Priority"), "HighPriority Thread");
        highpriority.setPriority (Thread.MAX-PRIORITY);
        highpriority.start();
    }
    static class Worker implements Runnable {
        private String name;
        public Worker (String name) {
            this.name = name;
        }
        public void run() {
            for(int i=0; i<5; i++) {
                System.out.println (name + ", iteration" + i);
            }
        }
    }
}
```

Q.8 Describe the life cycle of a thread with a diagram

→ The life cycle of a thread can be divided into five states and it transition between these states as it executes.

i) New State: A Thread begins its life cycle here, where it is created but not yet started.

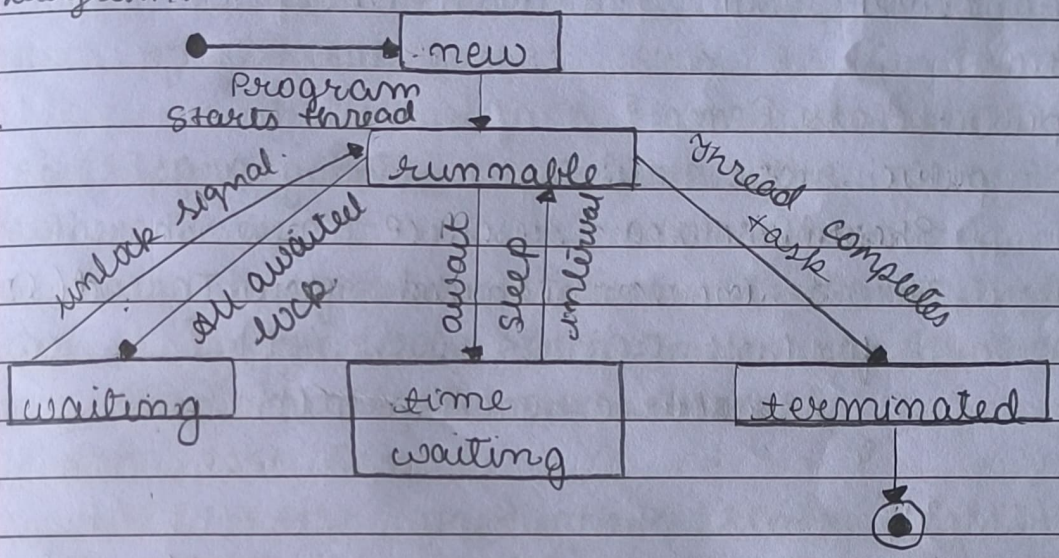
ii) Runnable State: Here, a thread is ready to run but has not been selected by the scheduler.

iii) Waiting State: Sometimes a thread transition to the waiting state while the thread waits for another thread to perform a task

iv) Time Waiting State: A Runnable thread can enter this state for a specified ~~th~~ interval of time.

v) Terminated State: When a thread has completed its execution or explicitly terminated.

→ Diagram:-



Q.9 Explain thread Synchronization with an example.

→ Thread Synchronization is a technique in multithreading that ensures that multiple threads to modify an object in a sequence. It is the ability to regulate multiple thread's access to a shared resources. It is used to avoid race conditions, where multiple threads try to modify shared data concurrently, leading to unpredictable outcomes. Types of

Synchronization are Process and Thread.

eg:
```
class SharedResource {
    private int sharedData = 0;
    public synchronized void increment() {
        sharedData++;
    }
    public synchronized int getValue() {
        return sharedData;
    }
}
public class Demo {
    public static void main(String args[]) {
        SharedResource resource = new SharedResource();
        Thread incrementThread = new Thread(() -> {
            for(int i=0; i<10000; i++) {
                resource.increment();
            }
        });
        Thread readThread = new Thread(() -> {
            for(int i=0; i<10000; i++) {
                int value = resource.getValue();
                System.out.println("Read: " + value);
            }
        });
        incrementThread.start();
        readThread.start();
        System.out.println("Final Value: " + resource.
                                        getValue());
    }
}
```

Q.10 Define User defined exceptions. Support your answer with an example.

→ User-defined exceptions, also known as custom exceptions, are exceptions that you the user creates in the program to handle specific error conditions that are not adequately represented by the built-in exception classes. To define a user-defined exception, you typically create a new class that extends either the 'Exception' class or one of its subclasses, such as 'RunTimeException'

* Example given in Question (4).

Q.11 Write a program to demonstrate multiple catch for a single try block. In what situation do we need multiple catch blocks?

→
```
public class Demo {
    public static void main (String args[]) {
        try {
            String str = null;
            int[] arr = new int[3];
            int result = 10/0;
            System.out.println(str.length());
            System.out.println(arr[5]);
        } catch (ArithmeticException e) {
            System.out.println("E-1: " + e.getMessage());
        } catch (NullPointerException e) {
            System.out.println("E-2: " + e.getMessage());
        } catch (ArrayIndexOutOfBoundsException e) {
```

```
        System.out.println("E-3: " + e.getMessage());
    }
}
}
```

→ You can have multiple 'catch' blocks to handle different type of exceptions for a single 'try' block. Having multiple 'catch' blocks allows you to provide ~~specific~~ specific error-handling logic for each exception type.

Q.12 Describe how a to implement a runnable interface, for creating and starting threads.

→ Implementing 'Runnable' is a common and flexible way to create threads because it allows you to separate the thread's code from its execution context.

i) Create a class that implements the "Runnable" interface. and then it should be override by the run() method which contains logic of the thread.

```
class MyRunnable implements Runnable {
    public void run() {
        for(int i=0; i<5; i++) {
            System.out.println(Thread.currentThread().
                getName() + ": " + i);    }
    }
}
```

iii) Now, instantiate the 'Runnable' object.
```
MyRunnable runnable = new MyRunnable();
```

iii) Create and start the Thread.
```
Thread thread1 = new Thread(runnable);
Thread thread2 = new Thread(runnable);
thread1.start();
thread2.start();
```

Q.13 Explain isAlive() and join() Method in multithreading.

-> i) 'isAlive()' Method These methods are used to manage and query the status of threads.

i) 'isAlive()' Method

This method is used to check whether a thread is alive or has terminated. The method returns a boolean value, 'true' if the thread is alive and 'false' if it has terminated. It is commonly used to check if a thread has completed its execution before moving on to another task or making decisions based on a thread's status.

eg: myThread.isAlive()

ii) 'join()' Method.

This method is used to wait for a thread to complete its execution. When you call this method on a thread, the current thread will block and wait until the specified thread has

finished. It is often used to ensure that one thread complete its tasks before another thread continues.

eg: thread1.join();

Q.14 List out the exception handling keywords in Java. Describe each keyword with an example.

→ Java provides several keywords and constructs for exception handling

i) 'Try': used to enclose a block of code where exceptions may occur. It is followed by one or more 'catch' blocks and an optional 'finally' block.

eg:
```
try {
    int result = 10/0;
} catch (ArithmeticException e) {
    System.out.println("Exception: "+e.getMessage);
} finally {
    System.out.println("End.");
}
```

ii) 'catch': used to catch and handle exception that are thrown within the corresponding 'Try' block. It specifies the type of exception it can catch and the code to execute when that occurs.

iii) 'finally': used to define a block of code that executes regardless of whether an

exception is thrown or not. Typically used for cleanup operations.

iii) 'throw': Used to explicitly throw exceptions within a method or code block. You provide an instance of an exception class that you want to throw.

eg: throw new CustomException("Exception");

iv) 'throws': Used in a method declaration to indicate that the method may throw one or more exception.

eg: public void myMethod() throws SQLException {
    // Method code that may throw SQLException
}

Q.15 Write a Java Program that generates and handle ArrayIndexOutOfBoundException and ArithmeticException using multiple try-catch block.

```
public class Demo {
    public static void main(String args[]) {
        int[] num = [1,2,3];
        int divisor = 0;
        try {
            int result1 = num[4];
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("E-1: " + e.getMessage());
        }
        try {
```

```
        int result2 = num[1] / divisor;
      } catch (ArithmeticException e) {
        System.out.println("E-2: " + e.getMessage());
      }
    }
}
```

Q.16 Write a Java Program to demonstrate the use of FileReader and FileWriter classes.

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class Demo {
    public static void main(String args[]) {
        String input = "input.txt";
        String output = "output.txt";
        try {
            FileReader reader = new FileReader(input);
            FileWriter writer = new FileWriter(output);
            int character;
            while ((character = reader.read()) != -1) {
                writer.write(character);
            }
            reader.close();
            writer.close();
            System.out.println("Read & write.");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Q.17 What are the various methods in a File class?

→ The 'File' class is used for file and directory manipulations which provides several methods for interacting with files and directories.

i) canRead() : Returns 'true' if the file or directory can be read; otherwise, it returns 'false'.

ii) canWrite() : Returns 'true' if the file or directory can be written to; otherwise, it returns 'false'.

iii) length() : Returns the length of the file in bytes.

iv) delete() : Delete the file or directory represented by the 'File' object. Returns 'true' if the deletion is successful and 'false' if the file doesn't exist.

v) createNewFile() : Creates a new, empty file. Returns 'true' if the file is created successfully and 'false' if the file already exists.

vi) renameTo() : Renames file represented by a 'File' object to the specified destination 'File'.

Q.18 Differentiate between InputStream class and Reader class.

→

InputStream Class	Reader Class
> It is a byte-based object that can read and write bytes	It is character Based, it can be used to read or write characters.
> It is used to binary input/output	It is used to character input/output
> It can Serialization and Deserialization can be done and serialized objects can be saved to a file.	It is not used for Serialization and Deserialization, as it reads character not byles.
> Used for reading binary files	used for reading text files in platform default encoding

Q.19 Write a Java program to copy the content of the file "file1.txt" into a new file "file2.txt".

→

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class Demo {
    public static void main(String args[]) {
        String source = "file1.txt";
```

```
String target = "file2.txt";
try (FileInputStream sourceFile = new File
                    InputStream (source);
     FileOutputStream targetFile = new File
                    OutputStream (target)) {
    byte[] buffer = new byte[1024];
    int bytesRead;
    while ((bytesRead = sourceFile.read
                    (buffer) != -1) {
        targetFile.write(buffer, 0, bytesRead);
    }
    System.out.println("File Copied");
} catch (IOException e) {
    System.out.println("Error: " + e.getMessage());
}
    }
}
```

Q.20 What do you mean by BufferedInputStream and BufferedOutputStream ? Explain with relevant Java program.

—ᵛ These are Java classes that provide buffering capabilities for reading from and writing to input and output streams, respectively.

i) BufferedInputStream

It is used to read data from an input stream while efficiently managing the buffering of data. It reads data from the

underlying input stream in larger chunks, making reading more efficient.

ii) BufferedOutputStream

It is used to write data to an output stream with efficient buffering. It accumulates data in a buffer before writing it to the underlying output stream, reducing the number of write operations.

eg:
```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class Demo {
    public static void main(String args[]) {
        String sourceFile = "source.txt";
        String destinationFile = "destination.txt";
        try {
            BufferedInputStream input = new
            BufferedInputStream(new FileInputStream
                                (sourceFile));
            BufferedOutputStream output = new
            BufferedOutputStream(new FileOutputStream
                                (destinationFile));
            byte[] buffer = new byte[1024];
            int bytesRead;
            while((bytesRead = input.read(buffer)) != -1) {
                output.write(buffer, 0, bytesRead);
```

```
        }
        input.close();
        output.close();
        System.out.println("File Copied");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
```

Q.21 What is an applet? Describe the life cycle of an applet with methods.

→ An Applet is a Java program that is designed to be embedded within a web page and run in a web browser. Applets are a way to extend the functionality of web page by adding interactive and dynamic elements that can be executed on the client side.

→ The life cycle of an applet is controlled by the browser and it consists of several key methods that an applet can override to define its behaviour.

i) 'init()' : It is called when the applet is first initialized. It is used to perform one-time initialization such as setting up variables, loading resources.

ii) 'start()': It is called when the applet is about to start executing. It is used to perform actions that need to occur each time the applet is loaded or reloaded in a web page.

iii) 'stop()': It is called when the applet is about to stop executing or when it is no longer visible in the web page.

iv) 'paint()': It is called whenever the applet needs to be redrawn or updated on the screen.

v) 'destroy()': It is called when the applet is about to be removed from the web page or when the web page is closed.

Q.22 Describe ByteArrayInputStream and ByteArrayOutputStream with an example.

→ These are classes in Java that allow you to read from and write to byte arrays.

i) ByteArrayInputStream

It is used to read data from a byte array. It allows you to create an input stream that reads bytes from an existing byte array.

iii) Byte Array Output Stream

It is used to write the data to a byte array. It allows you to create an output stream that writes bytes to an internal byte array.

eg:

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
public class Demo {
    public static void main(String args[]) {
        String data = "Sample Text";
        ByteArrayInputStream input = new
            ByteArrayInputStream(data.getBytes());
        ByteArrayOutputStream output = new
            ByteArrayOutputStream();
        byte [] buffer = new byte [1024];
        int bytesRead;
        try {
            while ((bytesRead = input.read(buffer)) != -1) {
                output.write(buffer, 0, bytesRead);
            }
            byte[] copied = output.toByteArray();
            System.out.println(new String(copied));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Q.23 Discuss byte streams and character streams in Java.

—↓ These streams are used to read from and write to various input and output sources such as files, network connections.

i) Byte Streams.

They are used for reading and writing raw binary data such as images, audio and any data where the actual byte values are important. They are typically used for low-level I/O operations. They often have names ending with 'InputStream' or 'OutputStream' like File, Buffered, Data.

ii) Character Stream

They are used for reading and writing character data. They handle character encoding, which allows you to work with text data in different character set and languages. They are higher-level and more suitable for working with text-based files. They often have names ending with 'Reader' or 'Writer'. like File, Buffered.

Q.24 How to read and write a file in Java? Explain with an example.

—↓ i) Reading a File

To read a file, you typically use classes like 'FileInputStream' for byte-level reading

or 'FileReader' for character-level reading.

ii) Writing to a File:

To write to a file, you typically use classes like 'FileOutputStream' for byte-level writing or 'FileWriter' for character-based writing.

* Example is given in Question (16)

Q.25 Describe FileInput and FileOutput Stream classes within an example.

→ These classes are used for reading and writing raw binary data from and to files. They provide low-level I/O operations for file handling.

i) 'FileInputStream'

It is used for reading data from a file in the form of raw bytes. It reads binary data from a specified file path.

ii) 'FileOutputStream'

It is used for writing data to a file in the form of raw bytes. It allows you to create a new file or overwrite an existing one with binary data.

* Example is given in Question (19)

xxxxxxxxx