Prepared By Abhiyan Dev Rimal
https://github.com/abhirimal/be-software-notes

PAGE :
DATE : / /

Chapter-3 (1Q) (7 marks)
Exception, Stream and I/O

Exception

1) Exception Handling. Why is it important. What are its usage. Define with example.

Ans: An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal form of the program's instructions. Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc. The core advantage of exception handling is to maintain the normal ~~of~~ flow of the application. An exception ~~handling~~ normally disrupts the normal flow of the application that is why we use exception handling.

Types of Java Exceptions

→ There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception.

1. Checked Exception

: The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions eg: IOException, SQLException etc. Checked exceptions are checked

Scanned with CamScanner

PAGE :
DATE : / /

at compile time.

2. Unchecked Exception
= The classes which inherit Runtime Exceptions are known as unchecked exception e.g Arithmetic Exception, Null Pointer Exception, Array Index Out Of Bounds Exception etc. Unchecked exceptions are not checked at compile-time but they are checked at runtime.

3. Error
= Error indicates serious problem that a reasonable application should not try to catch. Error is irrecoverable e.g. Out Of Memory Error, Virtual-Machine Error, Assertion Error, etc.

Advantages of Exception Handling
= Exception Handling ensures that the flow of the program doesn't break when an exception occurs. For example, if a program has bunch of statements and an exception occurs mid way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly.
By handling exceptions we make sure ~~all~~ that all the statements execute and the flow of program doesn't break.

Scanned with CamScanner

PAGE :
DATE : / /

Example: Arithmetic Exception.

```
//Program to check Arithmetic Exception

import java.util.Scanner;
public class Test{
    public static void main(String[] args){
        int a,b;
        Scanner ip=new Scanner(System.in);
        try{
            System.out.println("Enter first number");
            a=ip.nextInt();
            System.out.println("Enter second number");
            b=ip.nextInt();

            System.out.println(a/b);
        }
    catch(ArithmeticException e){
        System.out.println("Can't divide by
                                zero");
        }
    finally{
        System.out.println("Good Job");
        }
    }
}
```

Scanned with CamScanner

PAGE :
DATE : / /

2) Explain the user defined Exceptions with suitable java code.

or

What are exceptions. How can you create your own exception in Java. Explain with program.

Ans: An exception is an unwanted or unexpected event which occurs during the execution of a program i.e run time, that disrupts the normal flow of the program's instructions.

If we create our own Exception then that is known as custom exception or user defined exception. This can be done by extending the class Exception. Java custom exceptions are used to customize the exception according to user need. By the help of custom exception, we can have our own exception and message.

Scanned with CamScanner

PAGE :
DATE : / /

Example:

```
class InvalidAgeException extends Exception{
    InvalidAgeException(String s) {
        super(s);
    }
}

class TestCustomException {
    static void validate(int age) throws InvalidAge-
                                        Exception
    {
        if (age < 18)
            throw new InvalidAgeException("not valid");
        else
            System.out.println("welcome to vote");
    }
public static void main(String[] ~~args~~ args) {
    try {
        validate(13);
    }
    catch (Exception m) {
        System.out.println("Exception occurred:" + m);
    }

    System.out.println("rest of the code...");
    }
}
```

Output: Exception occurred: InvalidAgeException: not valid
rest of the code...

Scanned with CamScanner

PAGE :
DATE : / /

3) Difference between
i) Exception Handling & Error Handling

Exception Handling	Error Handling
1. An exception "indicates conditions that a reasonable application might want to catch."	1. An Error "indicates serious problems that a reasonable application should not try to catch."
~~2. Recovering from Error is not possible.~~	~~2. We can recover from~~
2. We can recover from exceptions by either using try-catch block or throwing exceptions back to caller.	2. Recovering from error is not possible.
3. Exceptions include both checked as well as unchecked type.	3. All errors in java are unchecked type.
4. Program itself is responsible for causing exceptions.	4. Errors are mostly caused by the environment in which program is running.
5. All exceptions occurs at run-time but checked exceptions are known to compiler while unchecked exceptions are not known to compiler.	5. Errors occurs at runtime and not known to the compiler.

Scanned with CamScanner

PAGE :
DATE : / /

6. They are defined in java.lang.Exception package.	6. They are defined in java.lang.Error package.
7. Examples: java.lang.StackOverflowError, java.lang.OutOfMemoryError	7. Examples • Checked Exceptions: SQLException, IOException • Unchecked Exceptions: ArrayIndexOutOfBoundException, ArithmeticException.

Code Example of Exception

```
public class ExceptionExample {
    public static void main(String[] args) {
        int x=100;
        int y=0;
        int z=x/y;
    }
}
```

Output: java.lang.ArithmeticException

P.T.O ⇒

Scanned with CamScanner

PAGE :
DATE : / /

Code Example of Error

```
public class ErrorExample {
    public static void main (String[] args){
        recursiveMethod(10)
    }
public static void recursive(int i) {
    while (i != 0) {
        i = i+1;
        recursiveMethod(i);
    }
}
}
```

Output: java.lang.StackOverflowError

Scanned with CamScanner

PAGE :
DATE : / /

ii) Checked and Unchecked Exception:

Checked Exceptions	Unchecked Exceptions
1. Checked Exceptions occurs at compile time.	1. Unchecked exceptions occurs at runtime.
2. The compiler checks a checked exception.	2. The compiler doesnot check these types of exceptions.
3. These types of exceptions can be handled at the time of compilation.	3. These types of exceptions cannot be catch or handle at the time of compilation, because they get generated by the mistakes in the program.
4. Everything except error and runtime exception classes under throwable is checked exceptions.	4. Error and Runtime Exception classes are unchecked exceptions.
5. Here, the JVM needs the exception to catch and handle.	5. Here, the JVM does not require the exception to catch and handle.
6. Examples. FileNotFoundException, ClassNotFoundException, InterruptedException etc.	6. Examples. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundException, etc.

Scanned with CamScanner

PAGE :
DATE : / /

4) WAP to input an integer from keyboard and print it on the consolde. Give an exception if the input is other than integer using try and catch blocks.

```
//program to check InputMismatchException

import java.util.Scanner;
import java.util.InputMismatchException;
public class Test {
    public static void main(String[] args) {
    int a,b;
    Scanner ip = new Scanner(System.in);

    try {
      System.out.println("Enter first number");
      a = ip.nextInt();
      System.out.println("Enter Second number");
      b = ip.nextInt();
      ~~System.print~~
      System.out.println("Answer is:" + a/b);
    }

    catch (InputMismatchException e) {
      System.out.println("Please enter numbers only");
    }
  }
}
```

Scanned with CamScanner

PAGE :
DATE : / /

5) Java File Handling

Java I/O (Input and Output) is used to process the input and produce the output.
Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations

Stream
A stream is a continuous flow of ~~ta~~ data. In Java, a stream is composed of bytes. Stream represents an input source or an output destination which could be a file, i/o device, other program, etc.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.
1) System.out : standard output stream
2) System.in : standard input stream
3) System.err : standard error stream

Based on the data they handle there are two types of streams
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.

Scanned with CamScanner

PAGE :
DATE : / /

6) List stream classes. Why do we need to serialize an object?

→ The stream classes are:-

1) File Input Stream: Input stream that reads from a file.
2) File Output Stream: output stream that write to a file.
3) Print Stream : Output Stream that contain print() and println() method.
4) Data Input Stream: Contains method for reading java standard datatype.
5) Data Output Stream: An output stream that contain method for writing java standard data type.

→ Serialization allows us to convert the state of an object into a byte stream, which then can be saved into a file on the local disk or sent over the network to any other machine.

We need to serialize an object in java because:-

1) to save state of an object.
2) to travel an object across the network.

Scanned with CamScanner

PAGE :
DATE : / /

```
// Reads character from a file

import java.io.*;

public class file1 {
    public static void main(String[] args) {
    file infile = new file("C:\\Users\\abhis\\
        desktop\\abc.txt");
    ~~file reader~~
    FileReader fr = null;
    try {
        fr = new FileReader(infile);
        int i;
        while ((i = fr.read()) != -1)
        {
            System.out.println((char)i);
        }
    } catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Reads one character at a time.

Scanned with CamScanner

PAGE :
DATE : / /

```
//write character in a file

import java.io.*;

public class file2 {
    public static void main(String[] args) {
        file outfile= new file("C:\\users\\
            abhir\\desktop\\abc.txt");
        filewriter fout= null;

    try {
            fout=new filewriter(outfile);
            fout.write("Hello world");
            fout.close();
        }
        catch(Exception e)
            {
                System.out.println(e);
            }
        System.out.println("Success...");
    }
}
```

Same as 8 no.

Scanned with CamScanner

PAGE :
DATE : / /

3) WAP to read content from file "abc.txt" and store it in "xyz.txt".

```
import java.io.*;

public class FileRW{
    public static void main (String[] args){
  file infile = new file("c:\\users\\abhir\\
        desktop\\abc.txt");
  file outfile = new file("c:\\users\\abhir\\
        desktop\\ xyz.txt");
  ~~file Reader fr = null~~;
  ~~File Reader f~~
  FileReader fr=null;
  FileWriter fw= null;
  try{
        fr=new file Reader(infile);
        fw= new FileWriter (outfile);
        int ch;
  while((ch=fr.read()) != -1)
        {
            fw.write(ch);
        }
  fr.close();
  fw.close();
}  catch (Exception e)
        {
        System.out.println(e);
  }
}
```

Scanned with CamScanner

PAGE :
DATE : / /

iv) Explain about FileReader and BufferedReader class.

Buffered Reader	File Reader
1. Buffered Reader is buffered.	1. File Reader is not buffered.
2. It reads character from another reader (eg. FileReader)	2. It reads character from a file.
3. When BufferReader.read() is called mostly data is read from the buffer. When data is not available in buffer a call is made to read system file and lot of characters are kept in buffer.	3. Every time FileReader.read is called a call is made to read a system file. FileReader.read() reads 2 byte (16-bit) at a time.
4. It enables another Reader to buffer the characters and supports the mark and reset.	4. It obtains characters from a file in a file system. And does not supports mark and reset methods.
5. It is much faster as compared to FileReader.	5. It is slower as compared to Buffered Reader.

Scanned with CamScanner