Day8 Revisit

1. Exception Handling – Exceptions are unexpected conditions while running the program which causes pre-mature closure of the application. If handled properly, we could avoid the pre-mature termination. Recommended approach to handle exception is using try/catch block.
2. Exception Hierarchy – Throwable – Errors & Exceptions – Checked & Un-Checked Exceptions
3. Handling Exceptions using try/catch & throws keyword
4. Annotations (MetaData – that provides more information about the underlying member to JVM – introduces in JAVA5, It starts with @ symbol. @Deprecated, @SupressWarnings, @Override, @FunctionalInterface …(java.lang package)
5. Enums – Array of fixed values (constants) – This will help to assign/select one value from set of available values. Ordinal() –index value. Value – will help to assign either numeric or non-numeric data to the member. Enum is also a derived data type similar to arrays & objects.
6. Handling Exceptions using try/catch/finally block
   1. Try block must have either a catch block or a finally block
   2. Try block alone can’t handle exceptions & it gives compile time error
   3. Try block can have multiple catch block
   4. Try block with catch & finally block also allowed
   5. The code inside the finally block will gets executed always irrespective of exception status (Exception is thrown or not)
   6. Usually, inside finally block, we can add resource releasing code (closing opened files, closing db connection, closing streams etc., - cleaning activities related code)
   7. Try with resource will avoid finally block – parameterized try block
   8. While using try with resource block, finally block is optional
7. Creating custom exceptions can be done with the help of either Exception (Checked-Exception) or RuntimeException (Unchecked Exception) class
8. To throw a custom exception we can use “throw” keyword with “new” keyword.

Agenda

1. File Handling
2. IO Streams
3. Reflection

IO Streams –

Generally, Stream represents flow of water. In java streams represents flow of data.

In Streams, Producer which produces the data – Consumer who consumes the data.

In System class, there are three streams are available to represent 3 different operations.

To represent input stream – in

To represent output stream – out

To Represent error stream – err

In – used to transfer the data from user (keyboard/ip device) to the program

Out – used to transfer the data from the program to the user (console/any op device)

Err – used to transfer the error data from the program to the user (console/any op device)

Two types of stream – (Char [2 bytes] Stream & Byte [1 Byte] Stream )

**package** com.zensar;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.util.Scanner;

**public** **class** StreamsDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

// Getting input from user using StreamReader

BufferedReader stdin = **new** BufferedReader(**new** InputStreamReader(System.***in***));

String str;

System.***out***.print("Enter any Text :");

str = stdin.readLine();

System.***out***.println("Entered Text is : " + str);

**int** a;

System.***out***.print("Please Enter a number :");

a = stdin.read(); // 2 byte unicode character

System.***out***.println("Entered number is :" + (**char**) a);

// Stream readers will store the data in ASCII format

Scanner scan = **new** Scanner(System.***in***);

System.***out***.print("Enter any text :");

str = scan.nextLine();

System.***out***.println("Entered Text is : " + str);

System.***out***.print("Please Enter a number :");

a = scan.nextInt(); // 2 byte unicode character

System.***out***.println("Entered number is :" + a);

}

}

File class in java is used to represent either a file or directory (folder) in the underlying operating system.

We can’t perform read/write operation using file class, but we can create an empty file, check the file isEmpty(), check it is pointing to a folder?, delete file, search for a file.

DataStreams allow to use primitive data. This can be used to perform read/write operation on primitive data types. It support readInt, readFloat, readLong…., writeInt, writeFloat,writeLong …

StreamTokenizer is available in java.io package

StringTokenizer is available in java.util package

Both classes are used to spilt the String using tokens or delimiters.

Serialization – It is a process of storing the state of an object to a permanent storage. (Flat file)

Persistence - It is a process of storing the state of an object to a permanent storage. (Database table)

Deserialization – reverse of serialization (Reading a flat file and constructing object from it)

Static and transient (non-access modifier) members will not be serialized. PAN, Aadhaar, Debit card details.

We generally serialize bean classes. Before serializing it is important to mark bean class as a serializable class using Serializable interface.

Serializable (This is a marker Interface – An Interface without any abstract methods)

There is a sub-interface available for Serializable interface called “Externalizable”

The “Externalizable” is used for customizing the data before storing and retrieving it from flat file.

Eg: Encrypting/decrypting password

There are two method in Externalizable interface readExternal(), writeExternal() these methods will be called automatically doing serialization & deserialization

There is a class in java.io package RandomAccessFile – This is used to perform any operation on a file.

Using this we can read/write data from & to a file.