List of Concepts Involved:

- Different types of Errors in Java
- What is an Exception?
- try catch
- Multiple catch block
- Handling vs Ducking an Exception
- Rethrowing an Exception(throw, throws,finally) and Custom Exception
- Hierarchy of an Exception class
- Control flow of Exception Handling concept
- try with Resources

Different types of Errors in Java
In any programming language we categorise errors into 2 types

1. Syntax Error/CompileTime Mistakes
2. Logical Error/RunTimeMistakes

Syntax error/CompileTime Mistakes
- It refers to the mistakes done by the programmer with respect to syntax.
- These mistakes are identified by the compiler, so we say it as“CompileTimeMistake”.

Logical Error/RunTimeMistakes
- It refers to the mistakes done by the programmer in terms of writing a logic
- These mistakes are identified by jvm during the execution of a program, so we say it as “RunTimeMistake”.

What is an Exception?
- An unwanted/expected event that disturbs the normal flow of execution of a program is called "Exception handling".
- The main objective of Exception handling is to handle the exception.
- It is available for graceful termination of program.

What is the meaning of Exception handling?
- Exception handling means not repairing the exception.
- We have to define alternative ways to continue the rest of the program normally. This way of defining an alternative is nothing but **"Exception handling".**

Example
Suppose our programming requirement is to read data from a file located at one location,At run time if the file is not available then our program should terminate successfully.

Solution: Provide the local file to terminate the program successfully,This way of defining alternatives is nothing but **"Exception handling".**

Example

```
try{
    read data from London file
 } catch(FileNotFoundException e){
     use local file and continue rest of the program normally
}
```

RunTimeStackMechansim

- For every thread in the Java language, jvm creates a separate stack at the time of Thread creation.
- All method calls performed by this thread will be stored in the stack. Every entry in the stack is called **"StackFrame/Activation Record".**
- main() => doStuff() => doMoreStuff()

Example

```
class Demo{
 public static void main(String[] args){
  doStuff();
 }
 public static void doStuff(){
  doMoreStuff();
 }
 public static void doMoreStuff(){
  System.out.println("hello");
 }
}
```

output:: Hello

Syntax of Exception handling

```
try{
 //risky code
}catch(Exception e){
 //handling logic
}
```

Default Exception handling

```
class Demo{
 public static void main(String[] args){
  System.out.println("Entering main");
  doStuff();
  System.out.println("Exiting main");
 }
 public static void doStuff(){
  System.out.println("Entering doStuff");
  doMoreStuff();
                System.out.println("Exiting doStuff");
 }
 public static void doMoreStuff(){
  System.out.println("Entering doMoreStuff");
  System.out.println(10/0);
  System.out.println("Exiting doMoreStuff");
 }
}
```

Output::

```
Entering main
Entering doStuff
Entering doMoreStuff
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at TestApp.doMoreStuff(TestApp.java:14)
	at TestApp.doStuff(TestApp.java:9)
	at TestApp.main(TestApp.java:4)
```

As noticed in the above example in the method called doMoreStuff(), an exception is raised.
When exception is raised inside any method, that method is responsible for creating the Exception object will the following details

Name of the exception::java.lang.ArithmeticException
Description of exception::/ by zero
location/stacktrace::

1. This Exception object will be handed over to jvm, now jvm will check whether the method has the handling code or not,if it is not available then that method will be abnormally terminated.
2. Since it is a method, it will propogate the exception object to caller method.
3. Now jvm will check whether the caller method is having the code of the caller method or not.
4. If it is not available, then that method will be abnormally terminated.
5. Similar way if the exception object is propagated to main(), jvm will check whether the main() is having a code for handling or not, if not then the exception object will be propagated to JVM by terminating the main().
6. JVM now will handover the exception object to "Default exception handler", the duty of "default exception handler" is to just print the exception object details in the following way

```
Exception in thread "main" java.lang.ArithmeticException:/ by zero
		at TestApp.doMoreStuff
		at TestApp.doStuff
		at TestApp.main
```

Example

```
class Demo{
 public static void main(String[] args){
  System.out.println("Entering main");
  doStuff();
  System.out.println("Exiting main");
 }
 public static void doStuff(){
  System.out.println("Entering doStuff");
  doMoreStuff();
  System.out.println(10/0");
                System.out.println("Exiting doStuff");
 }
 public static void doMoreStuff(){
  System.out.println("Entering doMoreStuff");
  System.out.println("hello");
  System.out.println("Exiting doMoreStuff");
 }
}
```

Output::

```
Entering main
Entering doStuff
Entering doMoreStuff
Hello
Exiting doMoreStuff
Exception in thread "main" java.lang.ArithmeticException : / by zero
        at TestApp.doStuff()
          TestApp.main()
```

Example

```
class TestApp{
 public static void main(String[] args){
  System.out.println("Entering main");
  doStuff();
  System.out.println(10/0);
  System.out.println("Exiting main");
 }
 public static void doStuff(){
  System.out.println("Entering doStuff");
  doMoreStuff();
  System.out.println("hiee");
                System.out.println("Exiting doStuff");
 }
 public static void doMoreStuff(){
  System.out.println("Entering doMoreStuff");
  System.out.println("hello");
  System.out.println("Exiting doMoreStuff");
 }
}
```

Output::

```
Entering main
Entering doStuff
Entering doMoreStuff
Hello
Exiting doMoreStuff
hiee
ExitingdoStuff
Exception in thread "main" java.lang.ArithmeticException : / by zero
        at TestApp.main().
```