**What is an Exception?**

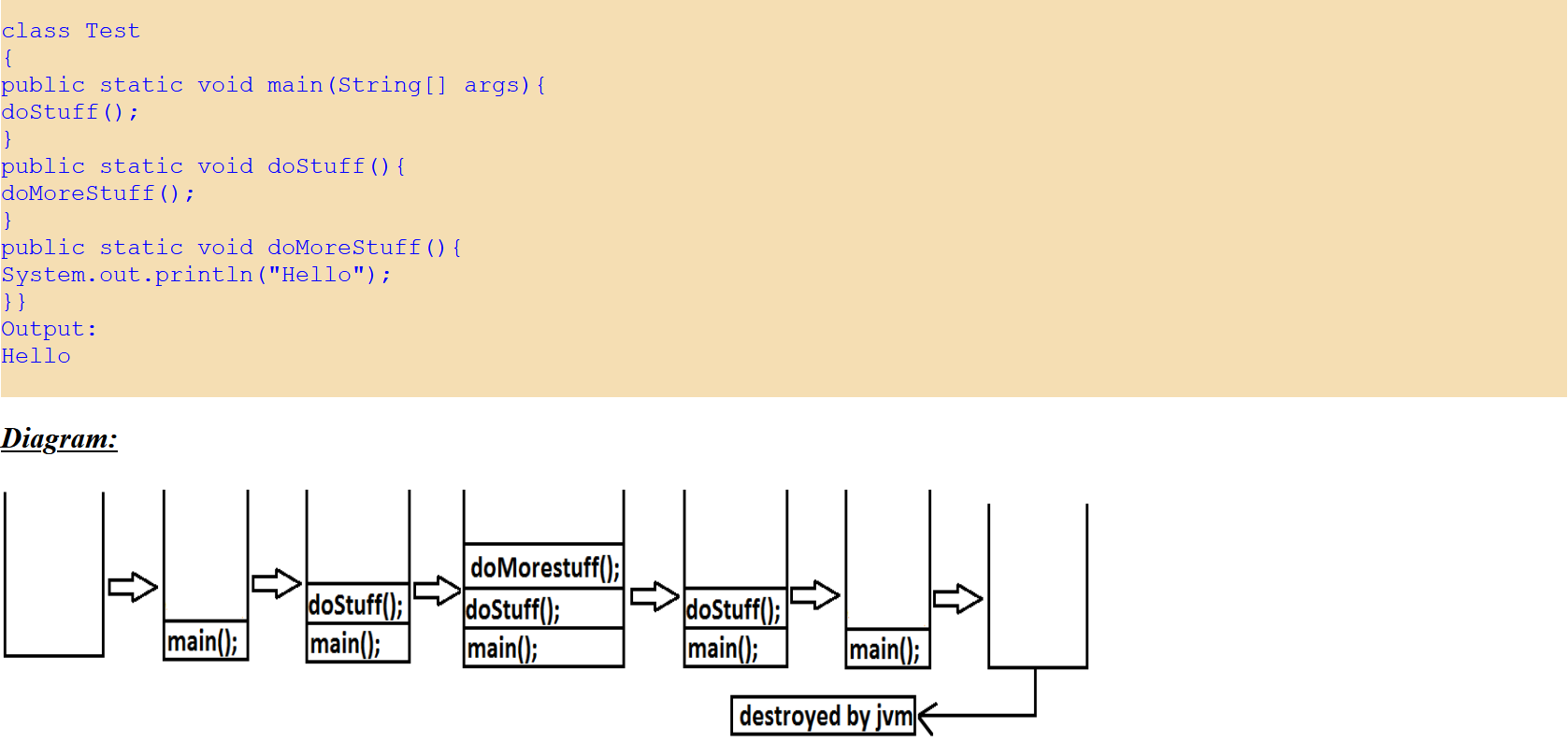
An unwanted unexpected event that disturbs normal flow of the program is called exception.

**What is Exception Handling?**

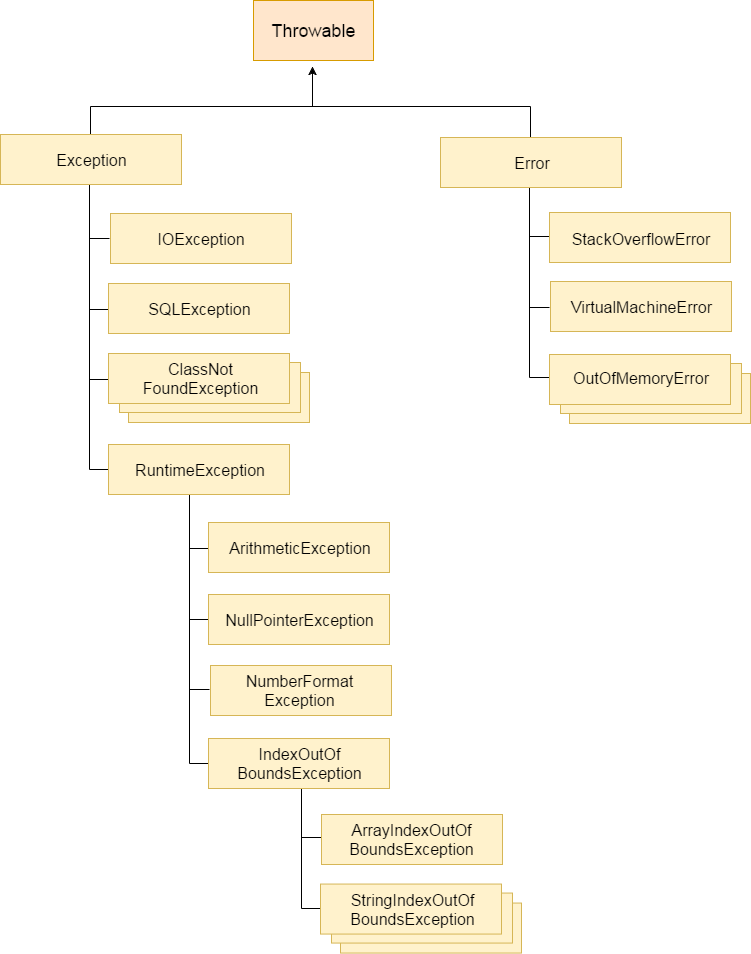
The exception handling in java is a mechanism to handle the runtime errors so that normal flow of the application can be maintained.

**Runtime Stack Mechanism:**

For every thread JVM will create a separate stack. All method calls performed by the thread will be stored in that stack. Each entry in the stack is called "one activation record" (or) "stack frame". After completing every method call JVM removes the corresponding entry from the stack. After completing all method calls JVM destroys the empty stack and terminates the program normally.



**Hierarchy of Java Exception classes:**



**Types of Exception:**

There are mainly two types of exceptions: checked and unchecked, where error is considered as unchecked exception.

1. Checked Exception
2. Unchecked Exception
3. Error

**Difference between checked and unchecked exceptions**

**1) Checked Exceptions:** The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc.

Checked exceptions are checked at compile-time. Java makes sure that you handle these exceptions or else it won’t compile the program.

**2) Unchecked Exceptions:** The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Unchecked exceptions are not checked at compile-time rather they are checked at runtime by JVM.

**3) Error:** Errors are fatal or irrecoverable state of program. e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

**Java Exception Handling Keywords:**

There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws

**Try-Catch-Finally**

It can consist of 3 steps:

* a *try* block that encloses the code section which might throw an exception.
* one or more *catch* blocks that handle the exception and
* a *finally* block which gets executed after the *try* block was successfully executed or a thrown exception was handled.

**Note:** If try with multiple catch blocks presents then order of catch blocks is very important it should be from child to parent by mistake if we are taking from parent to child then we will get Compile time error saying "exception xxx has already been caught".

In try-catch-finally order is important.

**Syntax of java try-catch**

1. try{
2. //code that may throw exception
3. }catch(NullPointerException e){
4. //Code we want to execute if exception has occurred in above block.
5. sysout(e);//Internally uses toString = detailed error msg
6. sysout(e.getMessage());// = only error msg
7. e.printStackTrace();// = detailed error msg+reason+location
8. }

**Syntax for valid try-catch**

1. try{
2. //code that may throw exception
3. }catch(NullPointerException e){...}
4. catch(ArithmeticException | ArrayIndexOutOfBoundsException e){...}
5. catch(Exception e){...}// catch-all block

**Syntax of java try-catch-finally**

1. try{
2. //code that may throw exception
3. }catch(Exception\_class\_Name ref){}
4. finally{System.out.println(“This Finally block always gets executed.”);}

Note: When an exception is encountered by JVM it starts looking for a matching catch block. And if JVM doesn’t come across any then the finally block will also not execute.

We can also write a try-finally block without having a catch block.

class Main {

public static void main(String[] args) {

try {

System.out.println("Inside try block..");

} finally {

System.out.println("Inside finally..");

}

}

}

Output

Inside try block..

Inside finally..

**Why use java finally?**

‘Finally’ block in java can be used to put "cleanup" code such as closing a file, closing connection etc. But modern features like ‘try-with-resources’ block can manage closing connections.

Java finally block is always executed whether any exception has occurred or not or not.

**Note: If you don't handle exceptions, before terminating the program, JVM still executes finally block (if any).**

**Rule: For every try block there can be zero or more catch blocks, but only one finally block.**

**Note: The finally block will not be executed if the program terminates (either by calling System.exit(0) or by causing a fatal error that causes the process to abort).**

**Note:** **System.exit(0);**   
Insteadof zero, we can take any integer value. Zero means normal termination , non-zero means abnormal termination.   
This status code internally used by JVM, whether it is zero or non-zero there is no change in the result and effect is same.

Even though return statement is present in try or catch blocks, first finally will be executed and after that only return statement will be considered that is finally block dominates return statement.

If a return statement is present in try, catch and finally blocks then finally block return statement will be considered.

**Try With Resources block**

It is used to declare one or more object, which must be closed after the program is finished with it.

Any type of object which implements “java.lang.AutoCloseable” interface can be included in try-with-resources block.

Traditional approach:

FileReader fr = new FileReader(path);

BufferedReader br = new BufferedReader(fr);

try {

return br.readLine();

} finally {

br.close();

fr.close();

}

Modern approach:

try (FileReader fr = new FileReader(path);

BufferedReader br = new BufferedReader(fr)) {

return br.readLine();

}

**Java Throw Keyword:**

The Java throw keyword is used to explicitly throw an exception at any line of the program.

Syntax: throw new Exception (“Exception message.”);

We can throw either checked or unchecked exceptions using the throw keyword.

Note: The throw keyword is mainly used to throw custom exceptions.

Example:

1. public class TestThrow1{
2. static void validate(int age) throws ArithmeticException{
3. if(age<18)
4. throw new ArithmeticException("not valid");
5. else
6. System.out.println("welcome to vote");
7. }
8. public static void main(String args[]){
9. try{
10. validate(13);
11. }catch(ArithmeticException e){
12. e.printStackTrace();
13. }
14. System.out.println("rest of the code...");
15. }
16. }

Throws keyword is used to handle checked exceptions only (generally the exceptions raised by throw keyword).

Throws is a keyword in Java which is used in the signature of a method to indicate that this method might throw one of the listed type exceptions. The caller of these method has to handle the exception using a try-catch block or add another throws declaration.

Example 1:

class Testthrows1{

void m() throws IOException{

throw new IOException("device error");//checked exception

}

void n()throws IOException{

m();

}

void p(){

try{

n();

}

catch(Exception e){System.out.println("exception handled");}

}

public static void main(String args[]){

Testthrows1 obj=new Testthrows1();

obj.p();

System.out.println("normal flow...");

}

}

Example 2:

class M{

void method()throws IOException{

throw new IOException("device error");

}

}

public class Testthrows2{

public static void main(String args[]){

try{

M m=new M();

m.method();

}

catch(Exception e){System.out.println("exception handled");}

System.out.println("normal flow continues...");

}

}

**Custom Exceptions in Java:**

In Java, we can create our own exceptions that are derived classes of the Exception class.

Reasons to use custom exceptions:

1. To catch and provide specific treatment to a subset of existing Java exceptions.
2. To address the  exceptions related to our business logic and workflow.

Syntax:

1. import java.io.Exception;
2. class MyException extends Exception{
3. MyException (String msg){
4. super(msg);
5. }
6. }

Note: Any overriding form of method cannot throw any new/wider checked exception.