**Exception Handling:**

An unexpected, unwanted event that disturbs normal flow of program is called Exception.

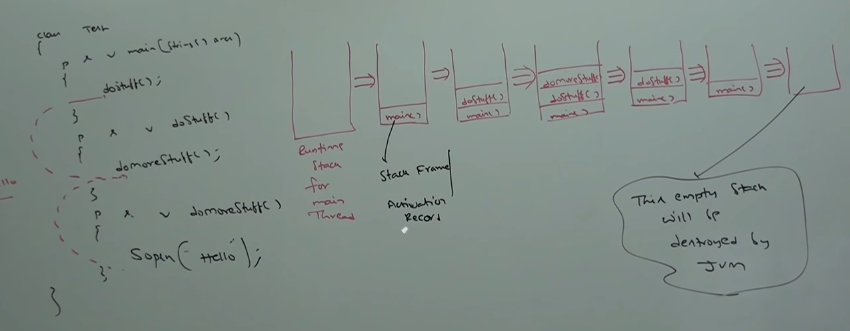
It is highly recommended to handle exceptions. Main Objective of Exception Handling is graceful termination of the program.

Exception handling does not mean repairing an exception. We must provide alternative way to continue rest of the program normally, is the concept of Exception Handling.

**Runtime Stack Mechanism:**

For every thread, JVM will create a runtime stack. Each method called performed by thread will be stored in the corresponding Stack. Each entry in stack is called activation record or Stack Frame.

After completing every method call, the stack will remove the corresponding entry. After completing all method calls the stack will become empty. The empty stack will be destroyed by Garbage collector before terminating the thread.



**Default Exception Handling:**

Inside a method if any exception occurs, the method in which it is raised is responsible to create exception object by including the following information.

1. Name of exception
2. Description of exception
3. Location at which exception occurs (stack trace)

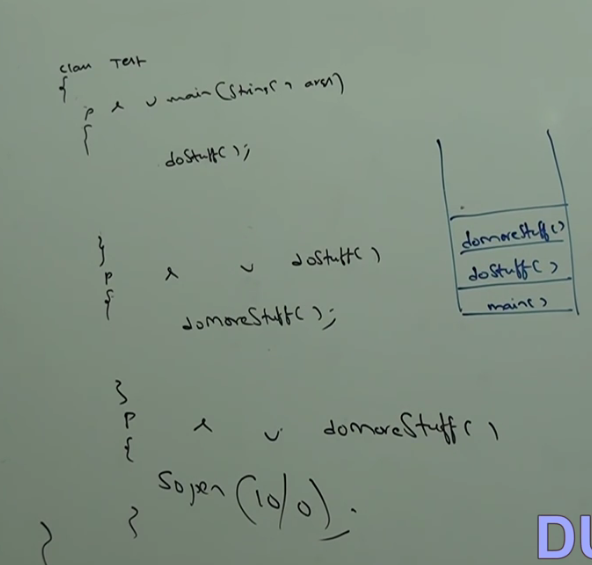
After creating exception object, method handovers that object to the JVM.

JVM will check whether the method contains any exception handling code or not. If the method does not contain exception handling code then JVM terminates that method abnormally and remove the corresponding entry from the stack. Then JVM identifies caller method and checks whether caller method contains any handling code or not.

If the caller method does not contain handling code then JVM terminates that caller method also abnormally and removes corresponding entry from the stack. This process will be continued until main method and if the main method also does not contain handling code then, JVM terminates main method also abnormally and removes corresponding entry from the stack. JVM handovers responsibility of exception handling to default exception handler, which is the part of JVM.

Default exception handler prints exception information in the following format and terminate the program abnormally.

Ex: Exception in thread “xxx” Name of exception: Description Stack trace.



If at least one method terminates abnormally, then program termination is abnormal.

**Exception Hierarchy:**

Throwable class acts as root for java exception hierarchy. It defines 2 child classes.

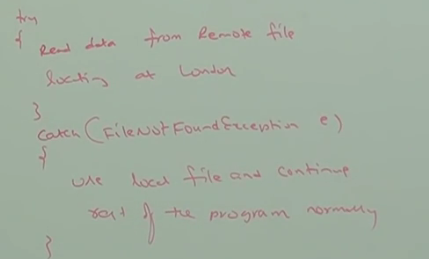
1. Exception
2. Error

**Exception:**

Most of the time Exceptions are caused by our program and these are recoverable.

Ex: If our programming requirement is to read data from remote file locating at London. At run time if remote file is not available then we will get runtime exception saying FileNotFoundException.

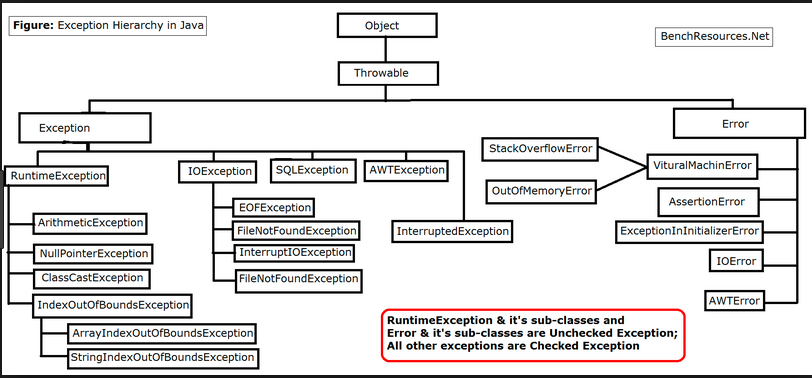
If this exception occurs, we can provide local file and continue rest of the program normally.



**Errors:** Most of the times errors are not caused by our program and these are due to lack of System resources. Errors are non-recoverable.

Ex: If OutOfMemory error occurs being a programmer we can’t do anything and the program will be terminated abnormally.

System/Server admin is responsible to increase heap memory.



**Checked vs Unchecked Exception:**

**The exceptions which are checked by compiler for smooth execution of the program are called checked exception.**

Ex: FileNotFoundException ….

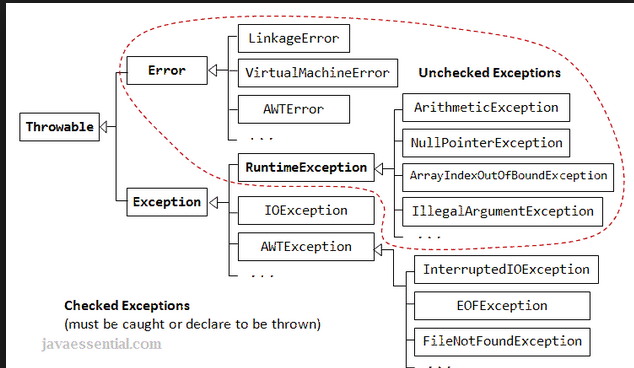
In our program if there is a chance of raising checked exception then compulsorily we should handle that checked exception, either by try catch or by throws keyword. Otherwise, we will get compile time error.

**The exceptions which are not checked by compiler whether programmer handling or not, such type of exceptions are called unchecked exceptions.**

Ex: ArithmeticException, NullPointerException

**Note1\*: Whether it is checked or unchecked every exception occurs at runtime only. There is no chance of occurring any exception at compile time.**

**Note2\*:** RuntimeException and its child classes, Error and its child classes are unchecked. Except this remaining are checked.



**Fully Checked vs partially checked:**

A checked exception is said to be fully checked if and only if all its child classes are checked.

Ex: IOException, InterruptedException.

A checked exception is said to be partially checked if and only if some of its child classes are unchecked.

Ex: Exception, Throwable

Note\*:

Only possible partially checked exception in Java are:

Exception, Throwable

**Try-Catch Block**

Within in the try block if anywhere exception is raised then rest of the code won’t be executed even though we handled the exception. So, within the try block we should take only risky code and length of try block should be as less as possible.

Any statement which is not part of try block and raises exception then it is always abnormal termination.

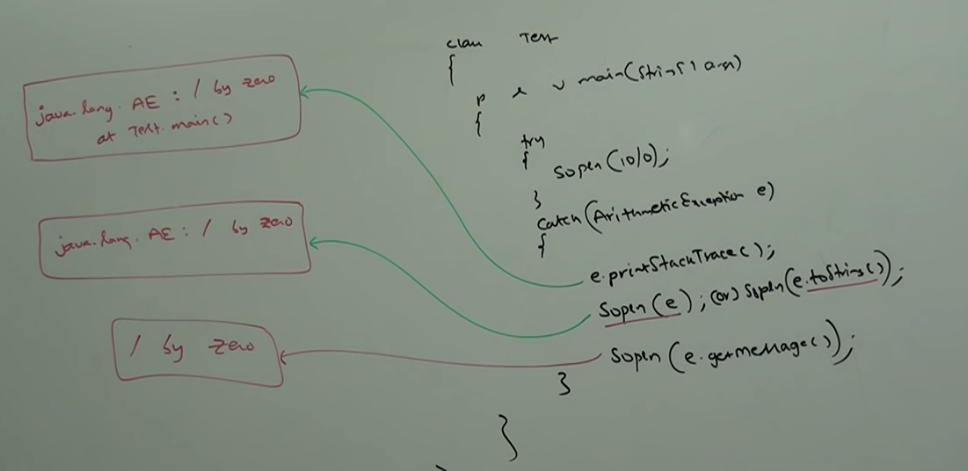
**Methods to print exception information:**

Throwable class defines the following methods to print exception information:

1. printStackTrace()🡪 Name Of Exception: Description

Stack Trace

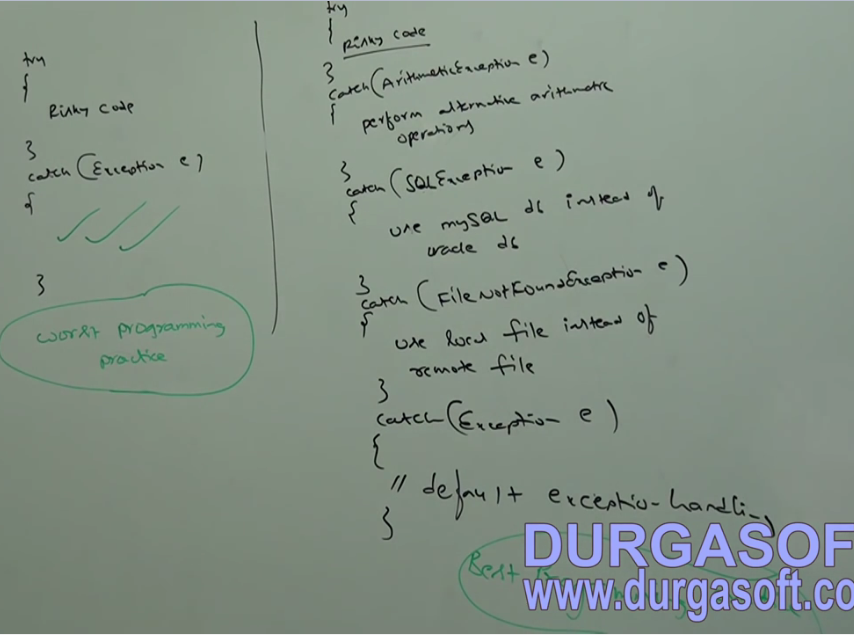
1. toString🡪 Name of Exception: Description
2. getMessage() 🡪 Description



**Note\*:** Internally default exception handler will use printStackTrace method to print exception information to the console.

**Try with Multiple Catch Block:**

Way of handling an exception is varied from exception to exception. Hence, for every exception type it is highly recommended to take separate catch block i.e. try with multiple catch block is always possible and recommended to use.



We can’t declare 2 catch blocks for same exception. Otherwise, we will get compile time error.

**Final:**

Final is the modifier applicable for classes, methods and variables.

If a **class** is declared as final then we can’t extend that class i.e. we can’t create child class for that class🡪 Inheritance is not possible for final classes.

If a **method** is final we can’t override that method in the child class.

If a **variable** declared as final, we can’t perform reassignment for that variable.

**Finally:**

This block is always associated with try catch to maintain clean up code. The specialty of finally block is it will be executed always irrespective of whether exception is raised or not raised. Handled or not handled.

**Finalize ():**

Finalize is a method always invoked by garbage collector just before destroying an object to perform cleanup activities. Once finalize method completes immediately garbage collector destroys that object.

|  |
| --- |
| Finally, block is responsible to perform cleanup activities related to try block i.e. whatever resources we opened as part of try block will be closed by finally block.  Finalize method is responsible to perform cleanup activities related to object i.e. whatever resources associated with object will be deallocated before destroying an object by using finalize method. |

Various possible combinations of try catch finally.

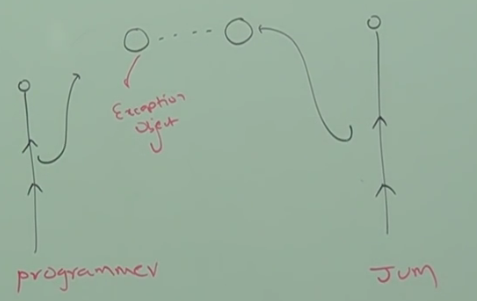
In try catch finally order is important. Whenever we are writing try compulsory we should write either catch or finally. Otherwise, we will get compile time error i.e. try without catch or finally is invalid.

Whenever we are writing catch block compulsory try block must be required i.e. catch without try is invalid.

Whenever we are writing finally block compulsory we should write try block i.e. finally without try is invalid.

In Try catch and finally blocks curly braces are mandatory.

**Throw and throws:**



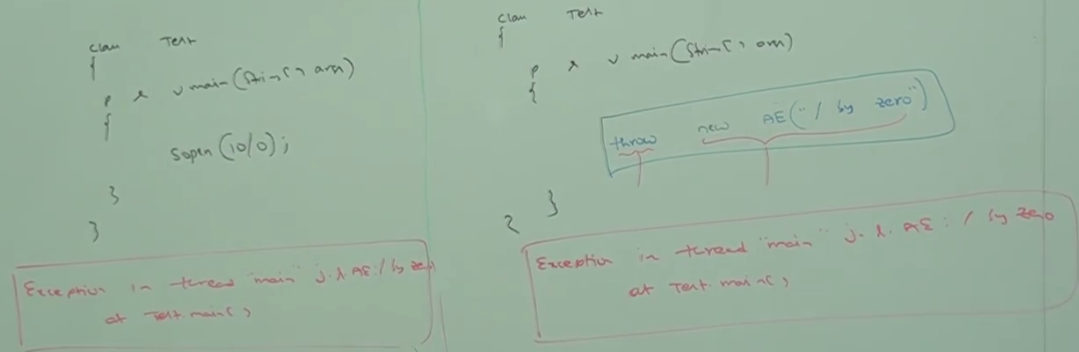
Sometimes we can create exception object explicitly and we can handover to the JVM manually. For this we can use throw keyword.

Ex: throw new AE (“/by Zero”);

Throw🡪 Hand-over our created exception object to JVM manually.

new AE (“…”) 🡪 creation of AE object explicitly.

Main objective of throw keyword is to handover our created exception object to the JVM manually.



Left: Main method creates exception object and handover to JVM

Right: Programmer creating exception object explicitly and handover to JVM manually.

**Note\*:** Best use of throw keyword is for user defined exception or customized exceptions.

**Throws:**

In our program if there is a possibility of raising checked exception then we should handle that checked exception compulsorily. Otherwise, we will get compile time error saying:

Unreported exception XXX: must be caught or declared to be thrown.

This compile time error can be handled in following 2 ways:

1. Using Try Catch
2. Using throws

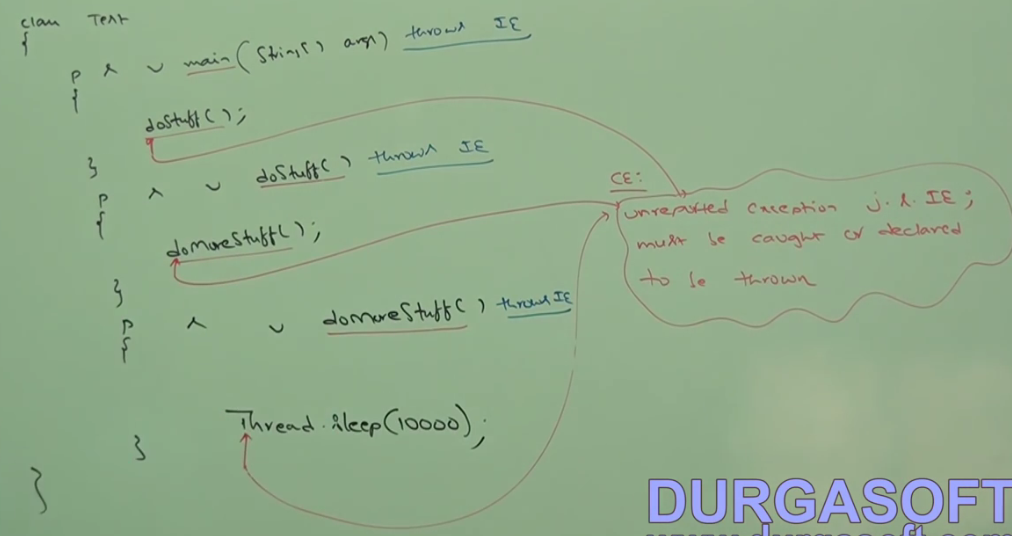
🡪We can use throws keyword to delegate responsibilities of exception handling to the caller (It may be another method or JVM). Caller method is responsible to handle that exception.

🡪Throws keyword required only for checked exception. Usage of throws keyword for unchecked exception-> there is no use or impact.

🡪Throws keyword required only to convince compiler. Usage of throws keyword does not prevent abnormal termination of program.



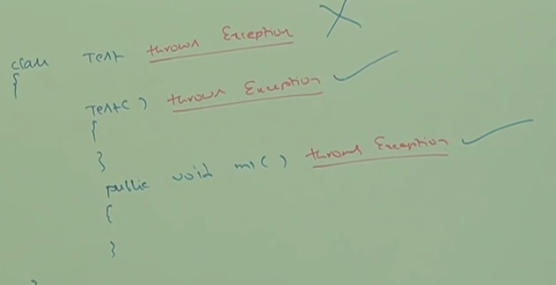
In below program if we remove at least one throws statement then the code won’t compile.



It is recommended to use try catch over throws keyword.

**Case1**:

We can use throws keyword for methods and constructors but not for classes.



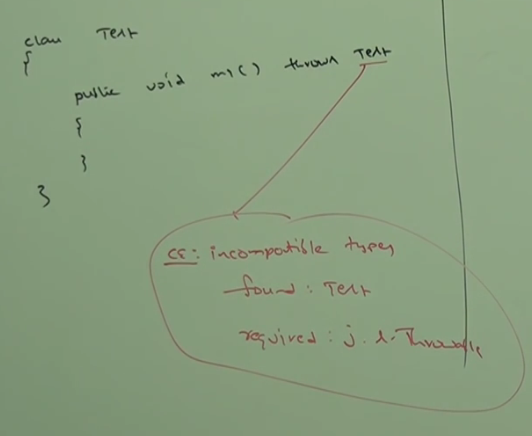
**Case2:**

We can use throws keyword only for throwable types. If we are trying to use for normal java classes then we will get compile time error saying:

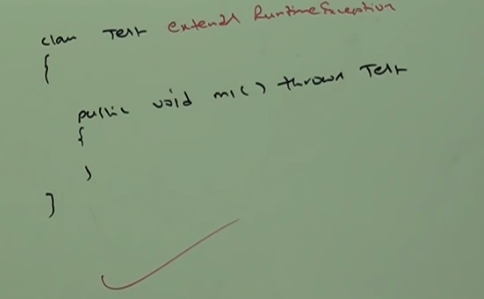
Incompatible types

Found: Test

Required java.lang,Throwable



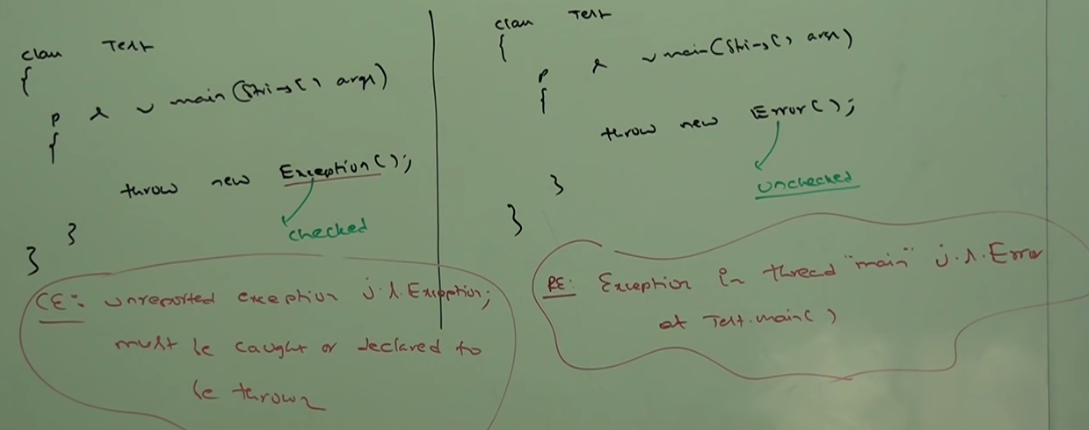
**Case3:** If the class extends RuntimeException then we can use that in throws



**Case4:**

Exception is checked exception

Error is unchecked exception

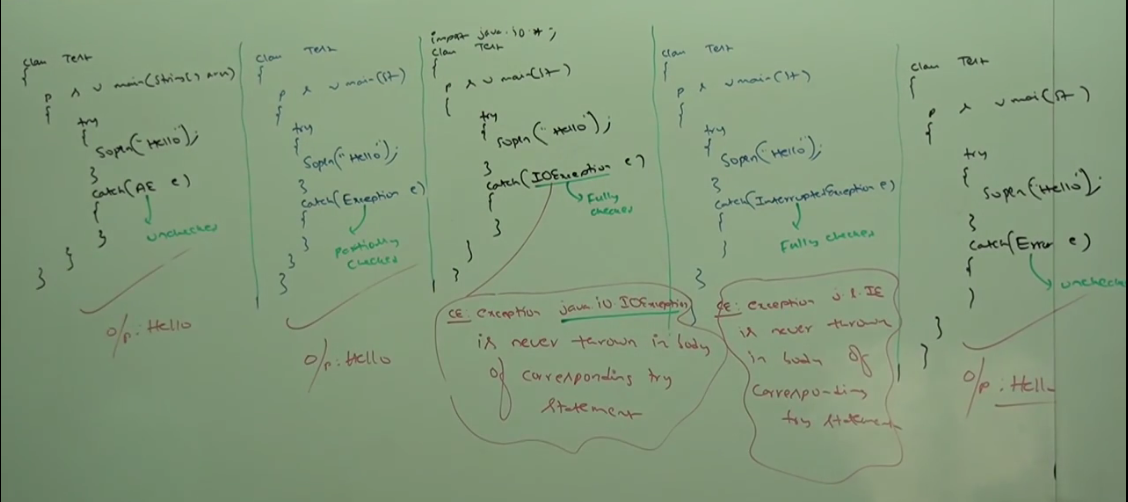


**Case5:**

Within the try block if there is no chance of raising an exception then we can’t write catch block for that exception. Otherwise, we will get compile time error as follows:

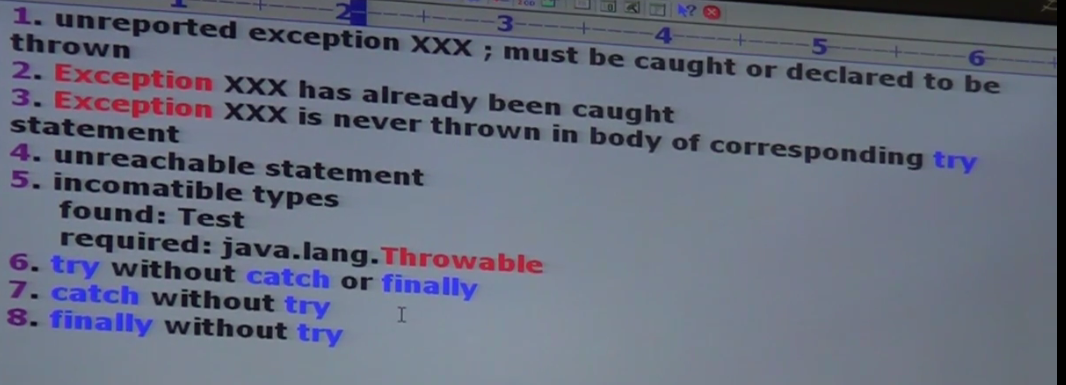
Exception xxx is never thrown in body of corresponding try statement.

This rule is applicable only for **fully** **checked** exceptions.



|  |  |
| --- | --- |
| **try** | To maintain risky code |
| **catch** | To maintain exception handling code |
| **finally,** | To maintain clean up code |
| **throw** | To handover our created exception object to JVM manually |
| **throws** | To delegate responsibilities of exception handling to the caller |

**Various possible compile time errors in exception handling**

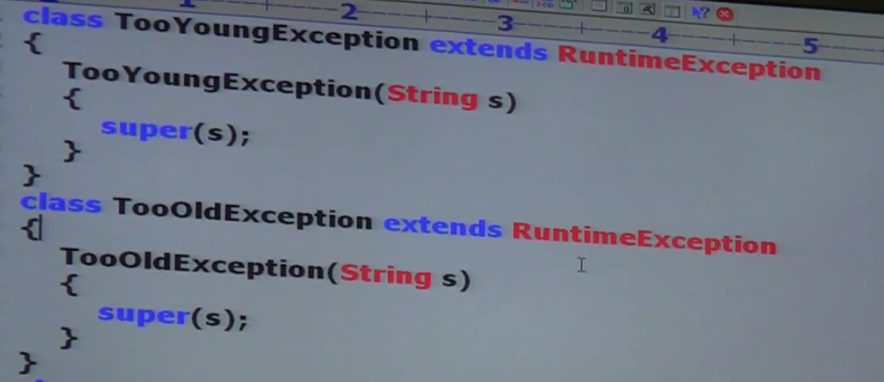


**Customized or user defined exception:**

To meet programming requirements, we can define our own exceptions such type of exceptions is called customized/user defined exception.

1. Throw keyword is best suitable for user defined or customized exception. But not for predefined exceptions.
2. It is highly recommended to define customized exception as unchecked i.e. we should run RuntimeException but not Exception.
3. In the below program super(s) is used to make description available to default exception handler.

Ex:



**Top 10 Exceptions:**

Based on the person who is raising an exception all exceptions are divided into 2 categories:

1. **JVM exceptions**
2. **Programmatic exceptions**

**JVM exceptions:**

The exceptions which are raised automatically by JVM, whenever a event occurs are called JVM exceptions.

Ex: ArithmeticException, NullPointerException

**Programmatic exceptions:**

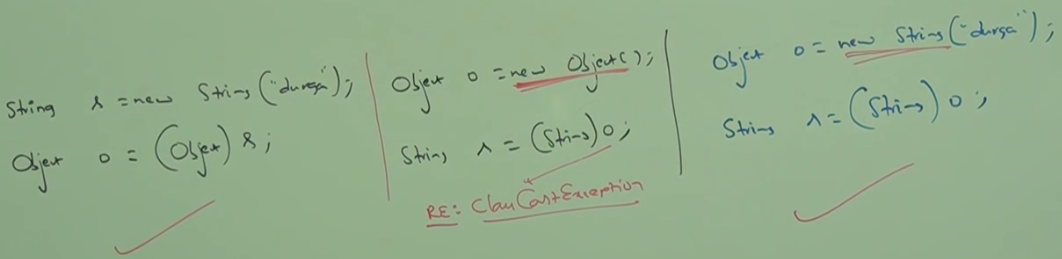
Exceptions raised explicitly either by programmer or API developer to indicate that something goes wrong are called programmatic exceptions.

Ex: Customized Exception, IllegalArgumentException

1. **ArrayIndexOuOfBoundException**

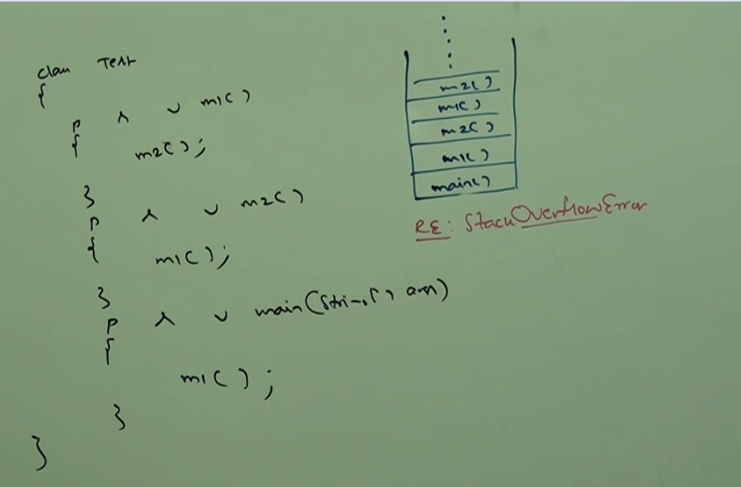
Child class of RuntimeException. Hence it is unchecked.Raised automatically by JVM whenever we are trying to access array element with out of range index.

1. **NullPointerException:** Unchecked**.**
2. **ClassCastException:** Child class of RuntimeException and hence it is unchecked. Raised automatically by JVM whenever we are trying to typecast parent object to child type.



1. **StackOverflowError**:

Child class of Error and hence it is unchecked. Raised automatically by JVM, whenever we are trying to perform recursive method call.



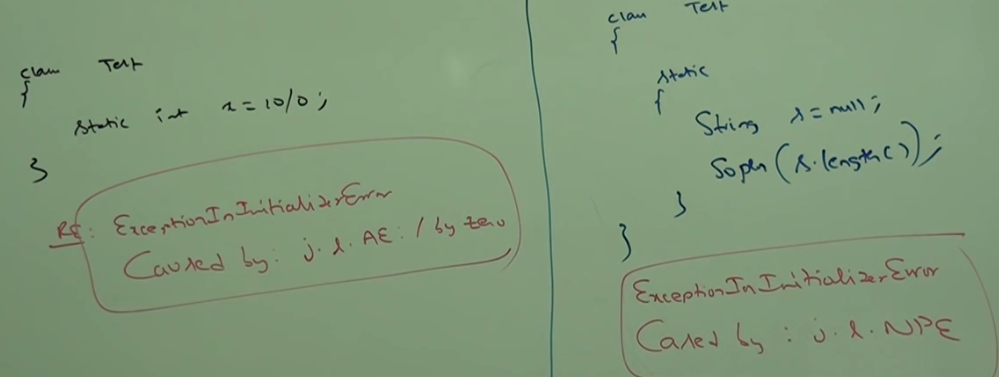
1. **NoClassDefFoundError**: It is the child class of Error and hence it is unchecked. Whenever JVM unable to find required .class file.

Ex: java Test.

If Test.class file is not available then we will get runtime exception saying:

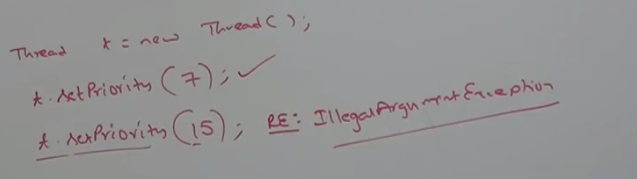
NoClassDefFoundError: Test

1. **ExceptionInInitializerError:** It is the child class of error. Hence, it is unchecked. Raised automatically by JVM. If any exception occurs while executing static variable assignments and static blocks.

Ex: In java 6 version. 7 onwards main method is required to run a java file.

1. **IllegalArgumentException**

Child class of runtime exception. Hence, unchecked. Raised explicitly either by programmer or by API developer to indicate that a method has been invoked with IllegalArgument.



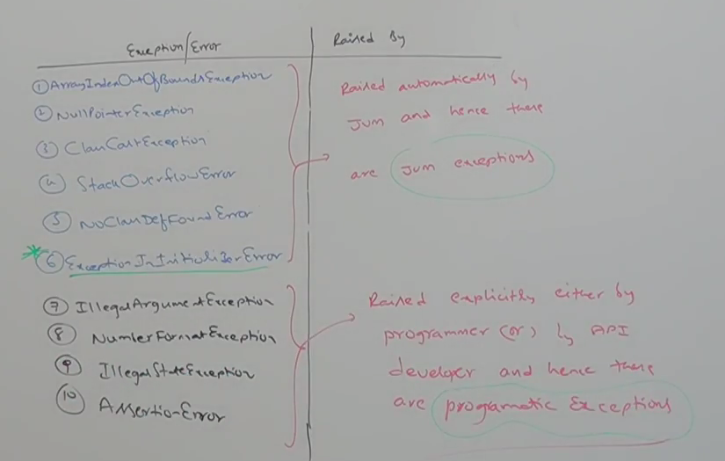
1. **NumberFormatException:** Direct child class of IllegalArgumentException which is child class of runtime exception. Hence it is unchecked. Raised explicitly either by programmer or API developer to indicate that we are trying to convert String to Number and the String is not properly formatted.
2. **IllegalThreadStateException**: Child class of RuntimeException and it is unchecked. Raised explicitly either by programmer or by API developer to indicate that a method has been invoked at wrong time.

Ex: After starting of a thread we are not allowed to restart the same thread once again. Otherwise, we will get runtime exception saying: illegalThreadStateException

1. **AssertError:** Child class of Error. Hence, it is unchecked. Raised explicitly by the programmer or the API developer to indicate that assert statement fails.

Ex: assert (x>10); If x is not greater than 10 then we will get runtime exception saying:

Assertion error.

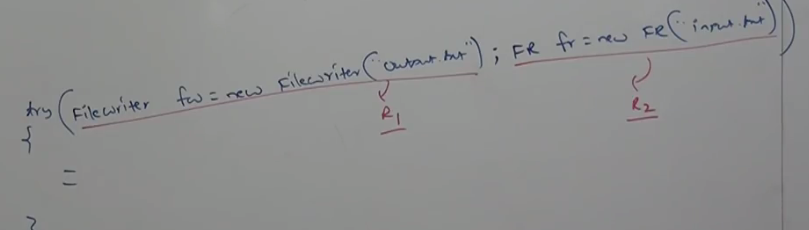


Until Java 1.6 version it is highly recommended to write finally block to close resources which are opened as part of try block. In this approach programmer is required to close the resources inside finally block. It increases complexity of programming. It increases length of code and reduces readability.

In 1.7 version try with resources is introduced. Advantage of try with resources is whatever resources will be opened as part of try block will be closed automatically once control reaches end of try block either normally or abnormally. Hence, we are not required to close explicitly.

As we are not required to write finally block so length of code will be reduced and readability will be improved. We can declare any number of resources. But these resources should be separated with semicolon.

**Syntax:**

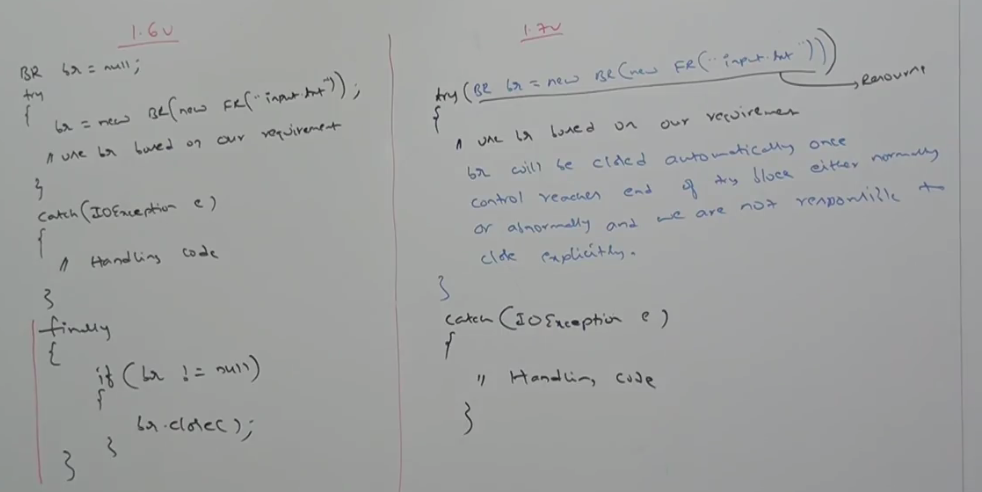


All resource should be Auto Closeable. A resource is auto closeable if the class implements java.lang.AutoCloseable interface. All IO related resources, database related resources and network related resources have already implemented AutoCloseable Interface. Being a programmer, we are not required to do anything.

AutoCloseable interface came in 1.7 version and it contains only one method close ().

Public void close ()

**All resource variable is implicitly final. Within the try block we can’t perform reassignment. Otherwise we will get compile time error**.



Until 1.6 try is associated with catch or finally. From 1.7 onwards we can take only try with resource without catch or finally.

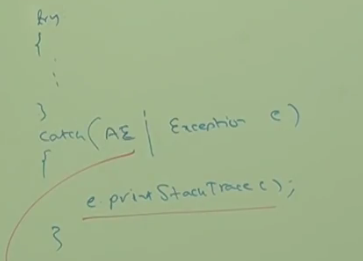
Try(Resource) {====}

Main advantage of try with resources is we are not required to write finally block explicitly. Because we are not required to close resources explicitly. Hence, until 1.6 version finally block is of high importance. However, from 1.7 onwards it is of no use.

Main advantage of try with multiple catch is:

1. Length of code will be reduced.
2. Readability will be improved.

In Multi catch block there should not be any relation between exception types (Either Parent to Child, Child to Parent, Same type). Otherwise, we will get compile time error.



CE: Alternatives in a multi catch statement can’t be related by sub classing.

**Exception Propagation:**

Inside a method if an exception is raised and if we are not handling that exception. Then exception object will be propagated to caller. Then caller method is responsible to handle exception. This process is called exception propagation.

**Rethrowing Exception:**

We can use this approach to convert one exception type to another exception type.

