**Exception Handling**

**Exception**: Any unwanted/undesired event that cause termination of execution of program, is called Exception

Exception handling is not meant as repairing the exception, instead it is the approach of providing alternative way to continue the execution of program normally.

Ex:

Try{

Read data from file locating at London

}Catch( FileNotFoundException e){

Use local file and continue rest of the program normally

}

Here alternative way (providing local copy) is handled if file at London is not present.

**Default Exception:**

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

Name of exception

Description of exception

Location at which exception occurs (Stack trace)

After creating exception object, method hands over that object to the JVM.

JVM will check whether the method contains any exception handling code or not. If the method doesn’t contain exception handling code then JVM terminates that method abnormally and removes the cross-ponding entry from the stack.

Then JVM identifies caller method and checks whether caller method contains any handling code or not. If the caller method doesn’t contain handling code then JVM terminates that caller method also abnormally and removes the cross-ponding entry from the stack. This process will be continued until main method and if the main method also doesn’t contain handling code then JVM terminates main method also abnormally and removes cross-ponding entry from the stack.

Then JVM handover the 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 terminates program abnormally.

Exception in thread “xxx” name of the exception : Description

Stack Trace

Example:

Class test{

P s v main(String[] args){

doStuff();

}

P s v doStuff(){

doMoreStuff();

}

P s v doMoreStuff(){

s.o.println( 10/0 );

}

}

Stack Trace:

Exception in thread main java.lang.arithmetic exception, division by zero

At test.doMoreStuff()

At test.doStuff()

At test.main()

***Note: In a program if atleast one method terminates abnormally then the program termination is said to be abnormal termination. If all methods terminated normally then only program termination is normal termination.***

Throwable class access root for java exception hierarchy.

Throwable class defines two child classes.

Exception

Error

**Exception:** Most of the time exceptions are caused by our program and these are recoverable (by writing try catch block). For example, if our programing req is reading data from remote file locating at London, if at runtime if remote file is not available then we ll get runtime exception saying file not found exception. If file not found exception occurs we can provide local file and continue rest of the program normally.

**Error:** Most of times error are not caused by our program and these are due to lack of system resources. Errors are non-recoverable. For eg. If outOfMemory error occurs, being a programmer we can’t do anything and program will be terminated abnormally. System admin or server admin is responsible to increase heap memory.

**Checked Exception vs Unchecked Exception**

*Checked Exception:* The exception which are checked by compiler for smooth execution of program are called *Checked Exception.* Eg. HallTicketMissingException, penNotWorkingException, fileNotFoundException etc.

In our program if there is a chance of rising checked exception then compulsory we should handle the checked exception (Either by try catch or by throws keyword) otherwise we will get compile time error.

Unchecked Exception: The exceptions which are not checked by compiler whether programmer handling or not such type of exception are called unchecked exception. Eg. ArithmeticException, bombBlastException.

Note:

1. Whether it is checked or unchecked, every exception occurs at runtime only. There is no chance of occurring any exception at compile time.
2. Runtime exception and its child classes, Error and its child classes are unchecked, except these remaining are checked.

**Customized Exception Handling**

Risky code are kept in try and handling code are kept in catch block. Length of try block should be as small as possible.

Within the try block if anywhere exception rised then rest of the try block would not be executed even though we handled that exception, hence within the try block we have to take only risky code and length of try block should be as less as possible.

If try is present with multiple catch blocks then order of catch blocks is very important. We have to take child exception first and then parent, otherwise we will get compile time error saying Exception XXX has already been caught.

**Finally Block**

It is not recommended to maintain clean up code inside try block, because there is no guaranty for the execution of every statement inside try.

It is not recommended to maintain clean up inside catch block, because if there is no exception then catch block would not be executed.

Hence we required some place maintain clean up code which should be executed always irrespective of whether exception rised or not rised or whether handled or not handled, such type of best place is nothing but finally block.

**\*\*\*** The main objective of finally block is to maintain clean up code and specialty of it is that it will be executed always irrespective of anything.

***Note: Even though return statement present inside try or catch block then before the execution of return statement will happen only after execution of finally block.***

***Eg. Class test{***

***P s v main(){***

***Try{***

***s.o.p(“try”);***

***return;***

***}catch(Exception e){***

***e.printstacktrace();***

***}finally{***

***s.o.p(“finally”);***

***}***

***}***

***}***

***O:P-***

***Try***

***Finally***

***Note 2: If in a method return statement is present in try, catch and finally block then return statement of finally block would be considered.***

***Note 3: There is only one situation where finally block won’t be executed that is whenever we are using system.exit(0). Whenever we are using system.exit(0) then jvm itself will be shutdown, in this particular case fanally block won’t get chance to get executed.***

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

*Note: Finally block is responsible to perform clean-up activities related to try block that is whatever resources we opened as part of try block will be closed inside finally block. Whereas finalize method is responsible to perform clean-up activities related to object, that is whatever resources associated with object will be deallocated before destroying an object.*

**Throw Keyword/Class/Statement**

Programmer throws the exception object and jvm catches it.

Sometimes we can create exception object explicitly and we can handover to jvm manually, for this we have to use throw keyword.

throw new AE(“/ by zero”);

Handover created x object to jvm manually Creation of exception object

Hence the main objective of throw keyword is to handover our created exception object to jvm manually.

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

Eg.

Class test{

P s v withdraw(double amount){

If(amount>balance}

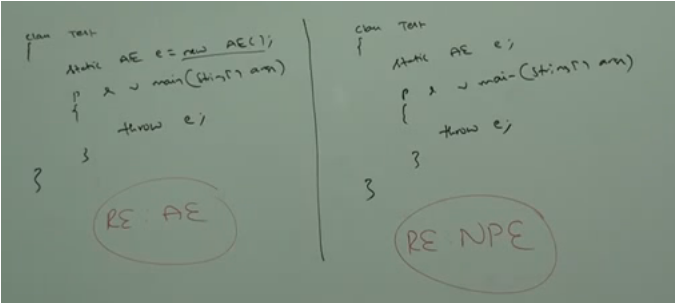
Throw insuficientBalanceException;

}

}

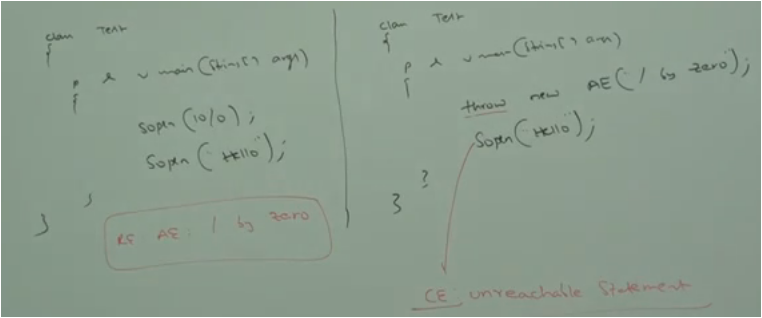
Case 1: throw e;

If e refers null then we will get nullpointerexception.

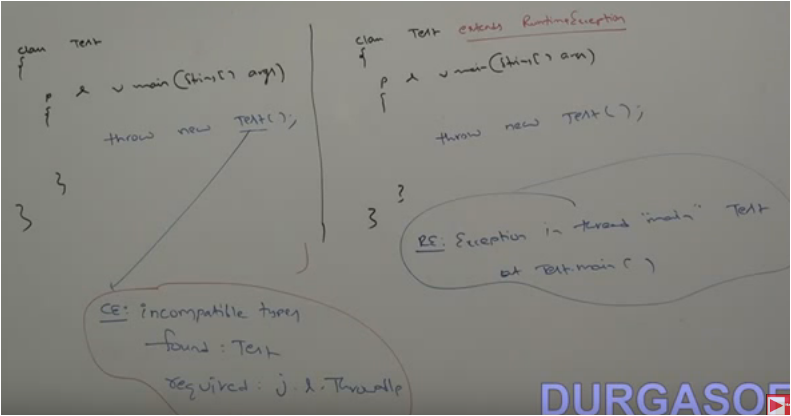


“Static AE e;” is initialized with null implicitly because it is static.

Case 2: After throw statement we are not allowed to write any statement directly, otherwise we ll get compile time error saying unreachable statement.



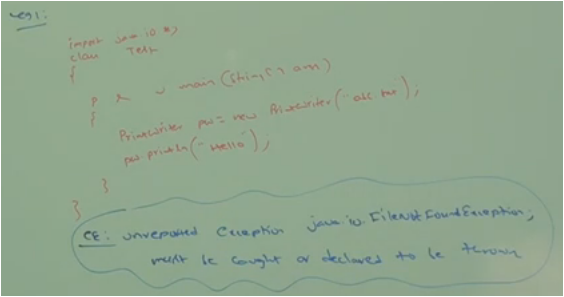
Case 3: We can use throw keyword only for throwable types. If we are trying to use it for normal java object compile time error saying incompatible types.



**Throws Keyword**

In our program if there is a possibility of rising checked exception then compulsory we should handle the checked exception, otherwise we will get compile time error saying unreported exception xxx must be caught or declared to be thrown.

Eg.



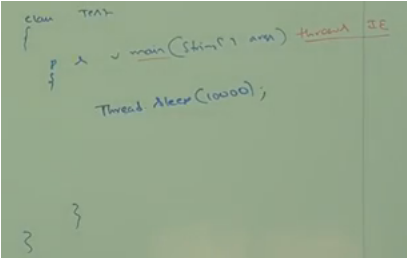
We can handle this compile time error by using the following two ways.

1. By using try catch

Put risky code in try and handling code in catch.

1. By using throws keyword

We can use throws keyword to delegate responsibility of exception handling to the caller (caller may be another method or jvm) then caller method is responsible to handle that exception.



Here jvm is handling the exception throwing thing.

**Points for Throws keyword**

Throws keyword is required only for checked exception and usage throws keyword for unchecked exception there is no impact.

Throws keyword is required only to convinced compiler and usage of throws keyword doesn’t necessarily prevent abnormal termination of the program.

**Top 10 Exception**

Based on the person who is rising an exception, all exceptions are divided in to 2 categories:

JVM Exceptions

Programmatic Exceptions

**JVM Exception**: The exceptions which are raised automatically by JVM whenever a particular event occurs are called JVM exceptions.

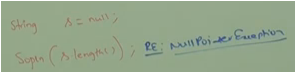
Eg: Arithmetic Exceptions, Null Pointer Exceptions etc.

**Programmatic Exceptions**: the exceptions which are raised explicitly either by programmer or by api developer to indicate that something gone wrong are called programmatic exception.

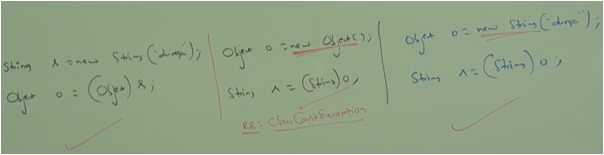
Eg: tooOld exception, Illegal Argument exception etc.

Important Exceptions:

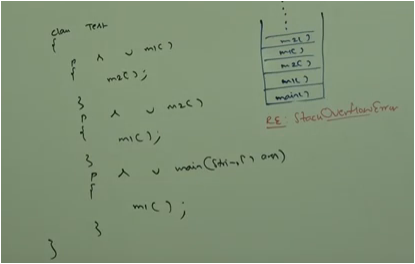
1. ArrayIndexOutOfBoundException: It is the child class of RuntimeException and hence it is unchecked. It is raised automatically by jvm whenever we try to access array element with out of range index.
2. NullPointerException: It is the child class of RuntimeException and hence it is unchecked. It is raised automatically by jvm whenever we try to perform any operation on **null.**



1. ClassCastException: It is the child class of RuntimeException and hence it is unchecked. It is raised automatically by jvm whenever we try to type cast parent object to child type.



1. StackOverflowError: It is the child class of Error and hence it is unchecked. It is raised automatically by jvm whenever we try to perform recursive method call.

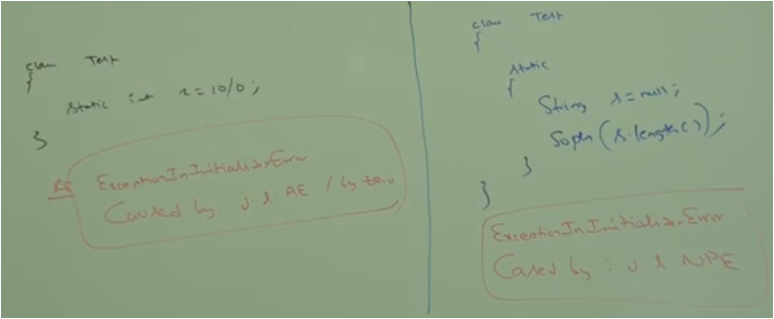


1. NoClassDefFoundError: It is child class of error and hence it unchecked. It is raised automatically by jvm whenever jvm is unable to find required .class file.

Eg. Java test

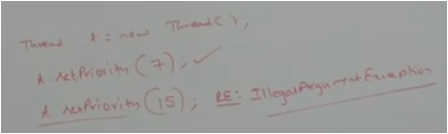
If test.class file is not available then we would get runtime exception saying NoClassDefFoundError: test.

1. ExceptionInInitializerError: It is child class of error and hence it unchecked. It is raised automatically by jvm if any exception occurs while executing static variable assignments and static blocks.

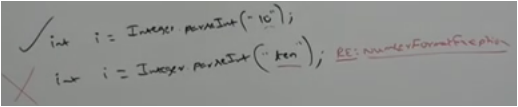


1. IllegalArgumentException: It is the child class of RuntimeException and hence it is unchecked. It is raised explicitly either by programmer or by api developer to indicate that a method has been invoked with illegal argument.

Eg. The valid range thread priorities is 1 to 10. If we try to set the priority with other than 1 to 10 then we will get runtime exception saying IllegalArgumentException.

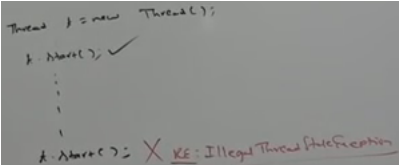


1. NumberFormatException: It is the direct child class of IllegalArgumentException which is the child class of RuntimeException and hence it is unchecked. It is raised explicitly either by programmer or api developer to indicate that we are trying to convert string to number and the string is not properly formatted.



1. IllegalStateException: It is the child class of RuntimeException hence it is unchecked. It is raised explicitly either by programmer or by api developer to indicate that a method has been invoked at wrong time.

Eg: After starting of s thread we are not allowed to restart the same thread once again otherwise we will get RuntimeException saying IllegalThreadStateException.



1. AssertionError: It is the child class of Error and hence it is unchecked. It is raised explicitly by the programmer or by api developer to indicate that assert statement fails.

Eg: Assume x is lesser than 10.

