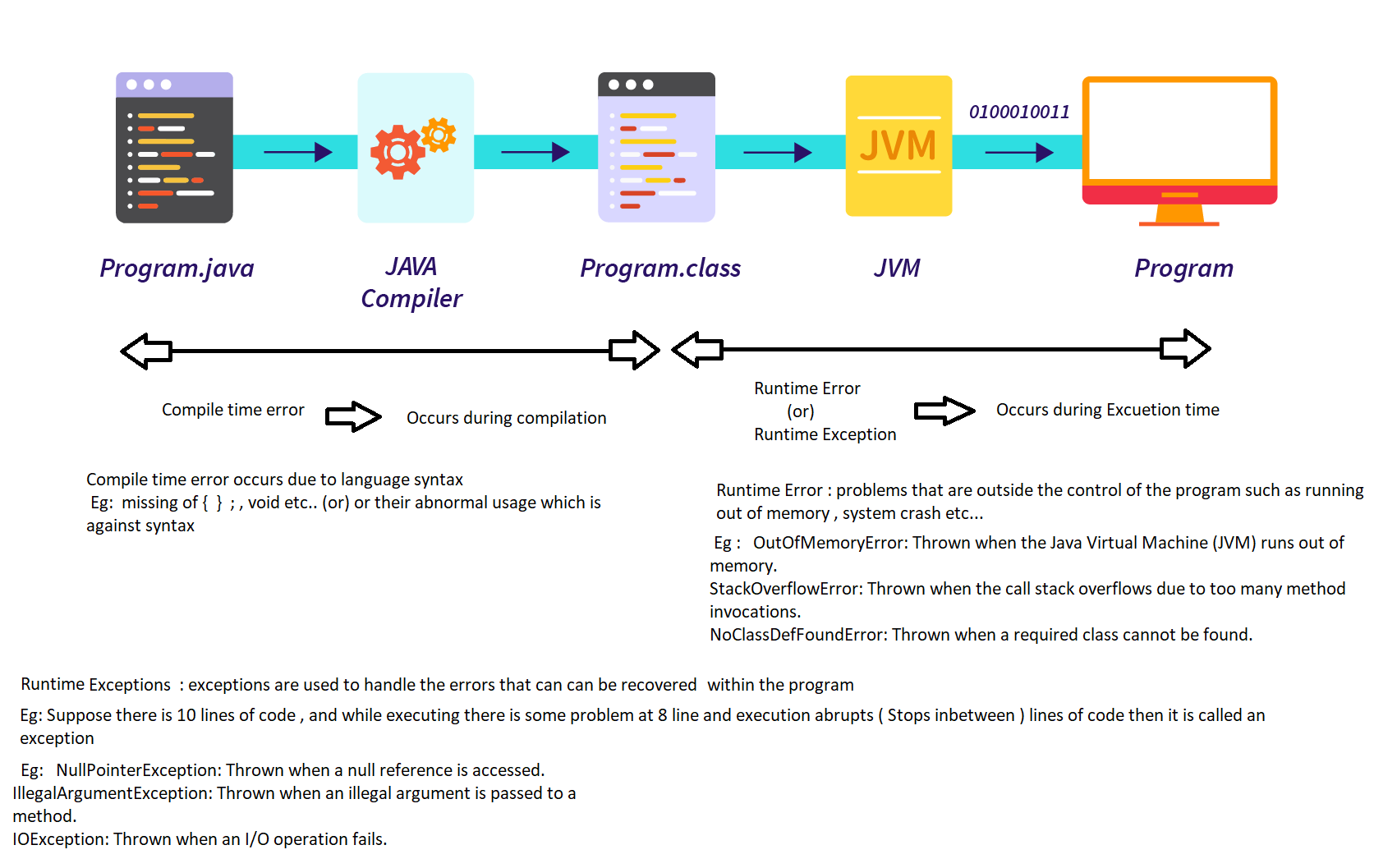
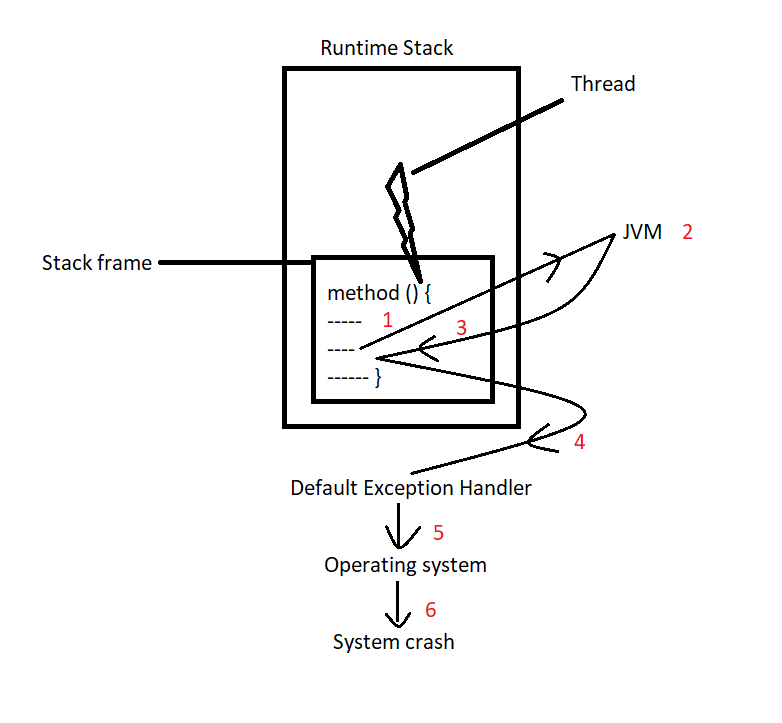
Exceptional Handling



A mistake / unwanted event that will occur during runtime which leads Abnormal / abrupt termination

Handling exception means abnormal termination should not happen

Process of handling the exception :



1. In which ever method the unwanted event / exception happens at particular line , when thread is executing that method line by line in a stack frame .
2. That method stack frame will create a exception object , which will be thrown to JVM .
3. jvm internally consists of runtime system , checks whether the exception is handled in the code or not .
4. if it is not handled means , then it will be sent to default Exceptional handler .
5. if that is also not happened, that is when OS ( operating systems ) will come into picture.
6. if OS also failed that is when system crash will happen

Eg: Exception\_Eg1

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Exception\_Eg1.main(Exception\_Eg1.java:10)

here the main method stack frame encountered an exception called java.lang.ArithmeticException: / by zero . jvm checks whether it is handled in the code . since it is not handled . Default Exceptional handler has taken the control and abruptly terminated the program by showing the message java.lang.ArithmeticException: / by zero has occurred while the thread is executing main method stack frame at line 10 .

Handling the exception

1. Handling the exception means the abrupt termination should not happen .
2. Use a try , catch block to handle the exception
3. Analyse the risky code which can cause the abrupt termination , and keep that code in the try block .
4. Whenever a try block is written a catch block should also be written.
5. Because when an exception occurs in the code
6. The inbetween code from where the exception occurs and catch block starts will not be executed .
7. The control comes directly to the catch block. Catch block and Code after catch block will be executed.
8. And in the catch block the type of exception should be mentioned along with reference variable .
9. When a exception occurs whatever points you want to execute write them in the catch block
10. catch block will execute if and only if an exception occurs , if exception does not occur

1. Then catch block will not be executed and execution flow will be normal (i.e all the code excluding the catch block will be executed) .

Eg: Exceptional\_Handling\_Try\_Catch

//go through the code

If a programmer will has handled the exception then the control will the control go to the default exceptional handler ?

No

* In the catch block usually write Exception class name followed by reference variable . “Exception” is the parent of all the exception classes .
* The Exception class handles all the exceptions , but it is not the preferrable way to handle all the exceptions with the parent Exception class .
* For every exception there are different exception classes are present

for example to handle divide by zero we use AirthmaticException like this there are different exception classes like NegativeArraySizeException , ArrayIndexOutOfBoundsException etc… every exception class handles different types of exception but for all them “Exception” is the parent .

Eg: Exceptional\_Handling\_Eg2

// go through the code

Here we have written single catch block for multiple type of exception , in the above program if the user gives incorrect input ( like division by zero , and negative array size (or) size that is out of range ) the message is same for different types of exception .

What exactly the mistake we are not able to say , common message for all mistakes

If the message is same for all the mistakes , the user cant identify the specific problem where it is done any why

To avoid this provide specific catch block for every particular exception .

Eg: Exceptionl\_Handling\_With\_Default\_Exceptional\_Handler

Even if we doesn’t write code for exceptional handling , the default exceptional handler is handling the exception, but the code in abruptly terminated .

To avoid abrupt termination we use exceptional handling ( using try catch block ) .And in the exceptional handling If the same catch block is used for multiple exceptions , the user cant identify the specific problem where the mistake is made and why .

To avoid this provide specific catch block for every particular exception

Eg: Exceptional\_Handling\_With\_Different\_Catch\_Blocks

Here we have written different catch blocks for different types of exception

* A catch block will execute only when a exception is encountered
* here suppose a catch block with NegativeArraySizeException is executed only when negative array size exception occurs ( code after negative array size catch block will also be executed) , the remaining catch blocks will not be executed.
* From this we can say that in java a single try block can have multiple catch blocks.
* Even though we have provided multiple catch blocks for handling different types of Exceptions , there might be some case where we have not anticipated the upcoming Exceptions , so to solve this we use generic “Exception” in catch block ( generic exception means parent of all the exceptions .).
* generally it should used when we write different catch blocks to handle the exceptions , so that if unanticipated exception arises the generic exception will take care of it .

Eg: Exceptional\_Handling\_With\_Specific\_Catch\_Blocks\_And\_Generic\_Exception\_Catch\_Block

* Generic catch block must be present at the last ( after all the catch blocks with different Exceptions ) . if generic catch block is in present at first ( or) inbetween other catch blocks, it leads to compiler error.

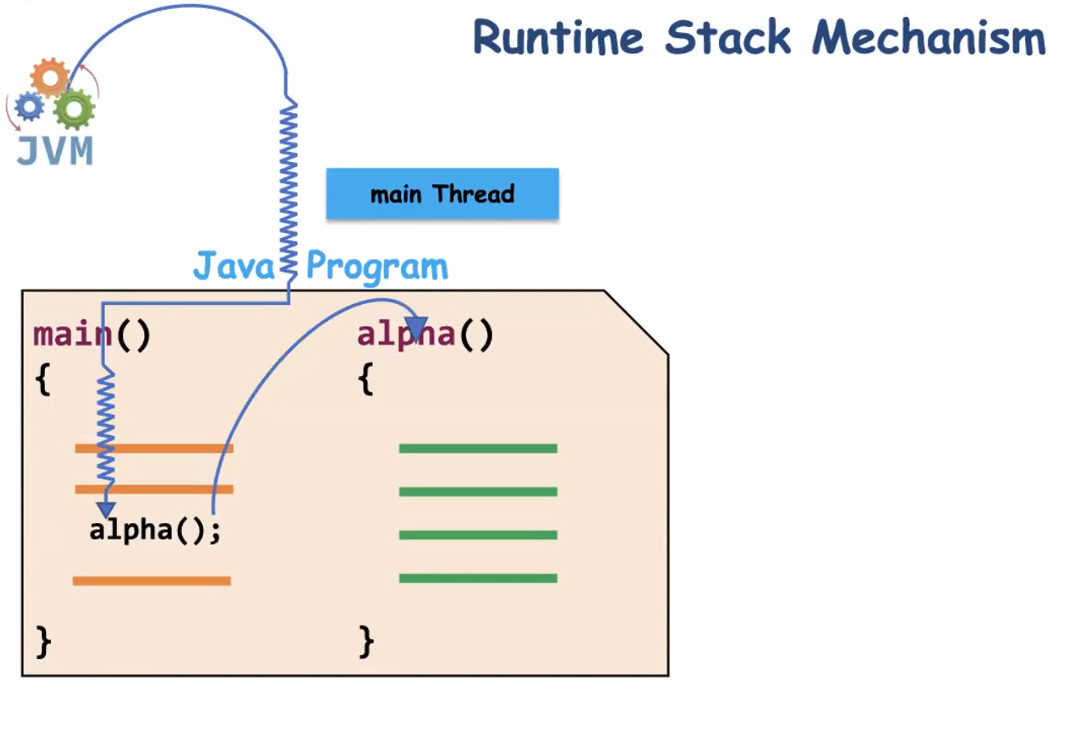
Eg: Exceptional\_handling\_With\_Generic\_Catch\_Block\_InBetween\_Other\_Catch\_Blocks

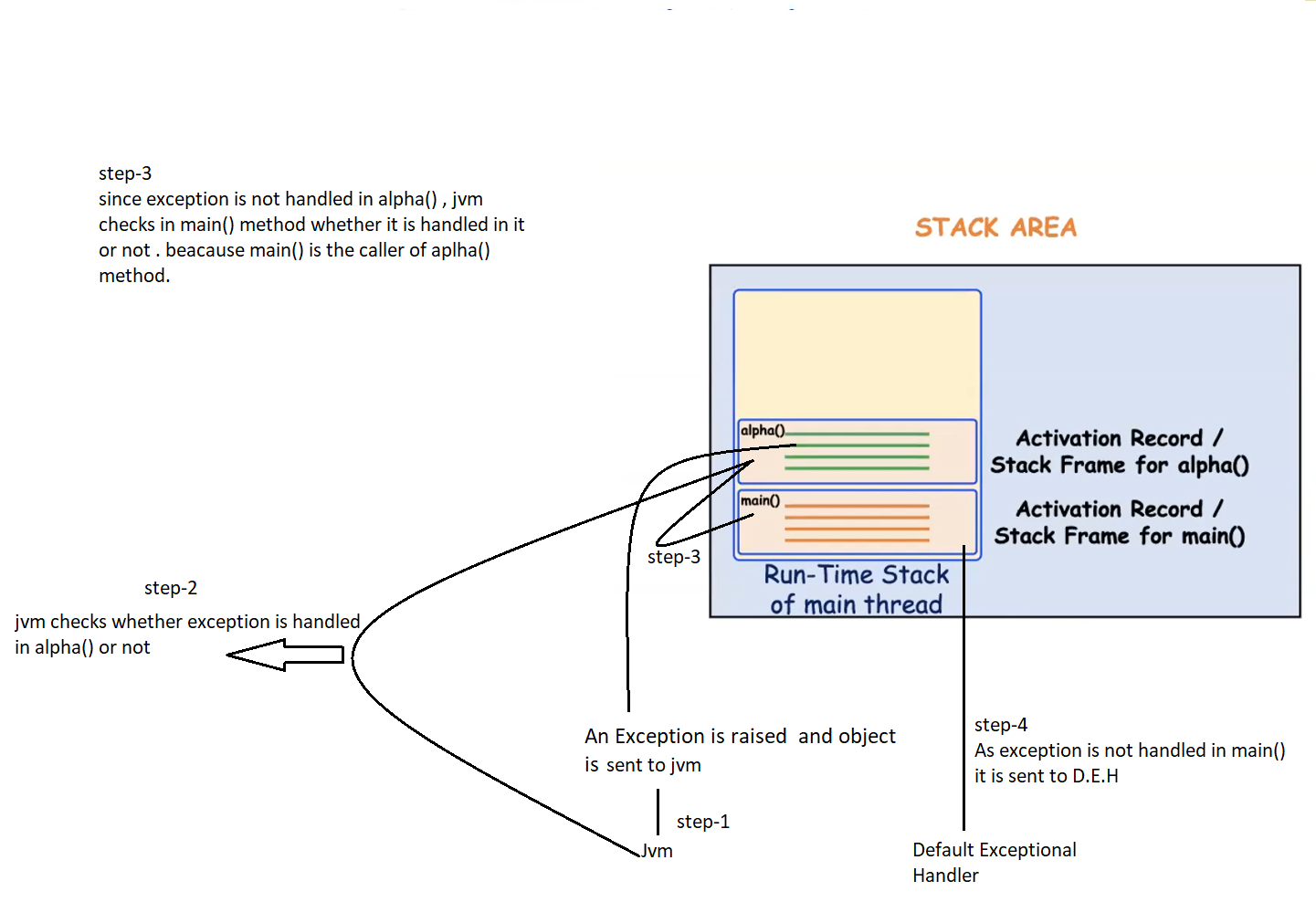
Industry recommended

Analyse the code that might lead to abrupt termination , and keep them in try and just after it write the catch block, continue with the remaining code after the catch block.

Eg: Recommended\_Syntax\_For\_Try\_Catch







// similar to **process of exceptional handling**  as above

Whenever an exception is occurred we can do three things they are

1. Handle the exception ( try , catch block )
2. Duck the exception ( throws )
3. Re-throwing an Exception ( throw, throws , try , catch , finally )

Ducking the exception :

Means if the method contains some risky code that might result in the exception and that is not handled , caller method should be informed that its code might result in the exception using throws keyword followed by exception name .

Here we are not actually handling the exception but informing the caller method that its code might result in exception using “throws” keyword and exception name.

Here exception is not handled instead it is escaping by informing caller method about the exception. predicting that caller method to handle it . this is called ducking the exception.

Eg: Ducking\_Exceptional\_Handling

In the above program alpha() is not actually handling the exception, but informing the caller method beta() about the exception using “throws” keyword and exception name

Similarly beta() is not handling the exception , but informing the caller method main() about the exception using “throws” keyword and exception name

The exception is handled in the main() method.

Unchecked Exception : compiler will not identify the risky code that might lead to exception and force us to handle the exception

checked exception : compiler will identify the risky code that might lead to exception and force us to handle that exception

Unchecked and checked exceptions both are runtime exceptions the only difference is if compiler identifies the risky code ( warning that an exception might arise ) it is checked , not identified means unchecked.

Industry recommended :

For unchecked exceptions it is highly recommended to handle the exception , don’t duck it .

Means if it contains some risky code handle there itself , don’t duck it .

Ducking the exception is usually recommended for checked exception.

Is ducking the exception is equal to handling the exception ?

No, ducking the exception is not actually handling the exception, we are just ignoring it.

Ducking the exception means ignoring , only tray and catch will handle the exception.

Rethrowing the Exception :

what if there is need to propagate the exception object to caller even though the exception is handled there itself.

Example : in an ATM if a person used a wrong pin for 3 times the account will be blocked . even if he uses different ATM machines the account remains to be blocked .

Here whenever wrong input is given in ATM the exception is handled. And it is propagated to the higher hierarchy . Due to some code the Account is blocked . even if he tries to use the account in the other ATM it gets blocked.

Rethrowing an Exception : means throwing / propagating already handled exception to the caller.

To rethrow an exception :

* use “throw” keyword in the catch block with reference
* since we are propagating the Exception to the caller it is recommended to use “throws” keyword in the method signature , along with the exception name.
* code below the “throw” will not be executed . so to execute the code below “throw” keyword use finally block. So that code below “throw” will be executed before propagating the exceptional object to the caller.
* Handle the exception in the called method also , since the exception will be propagated it.

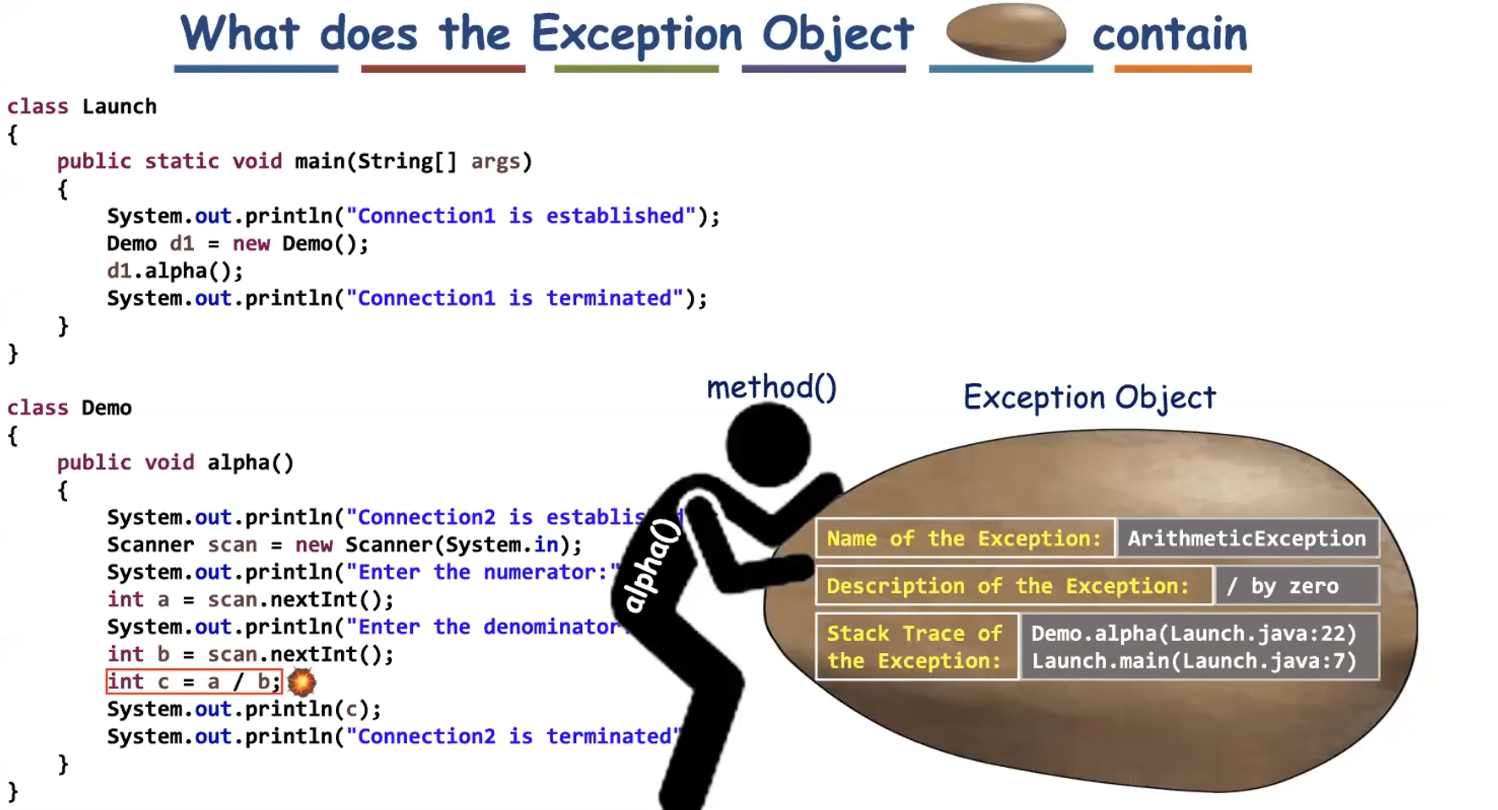
throw vs throws keyword :

* throw is used inside method ( or ) catch block. While throws is used in the method signature.
* throw is used to rethrow an exception . throws is used for ducking an exception.
* Statements below throw will not be executed. Statements below throws will be executed.

Note : finally block will be executed even if there is exception (or) no-exception (or) throw keyword (or) return statement.

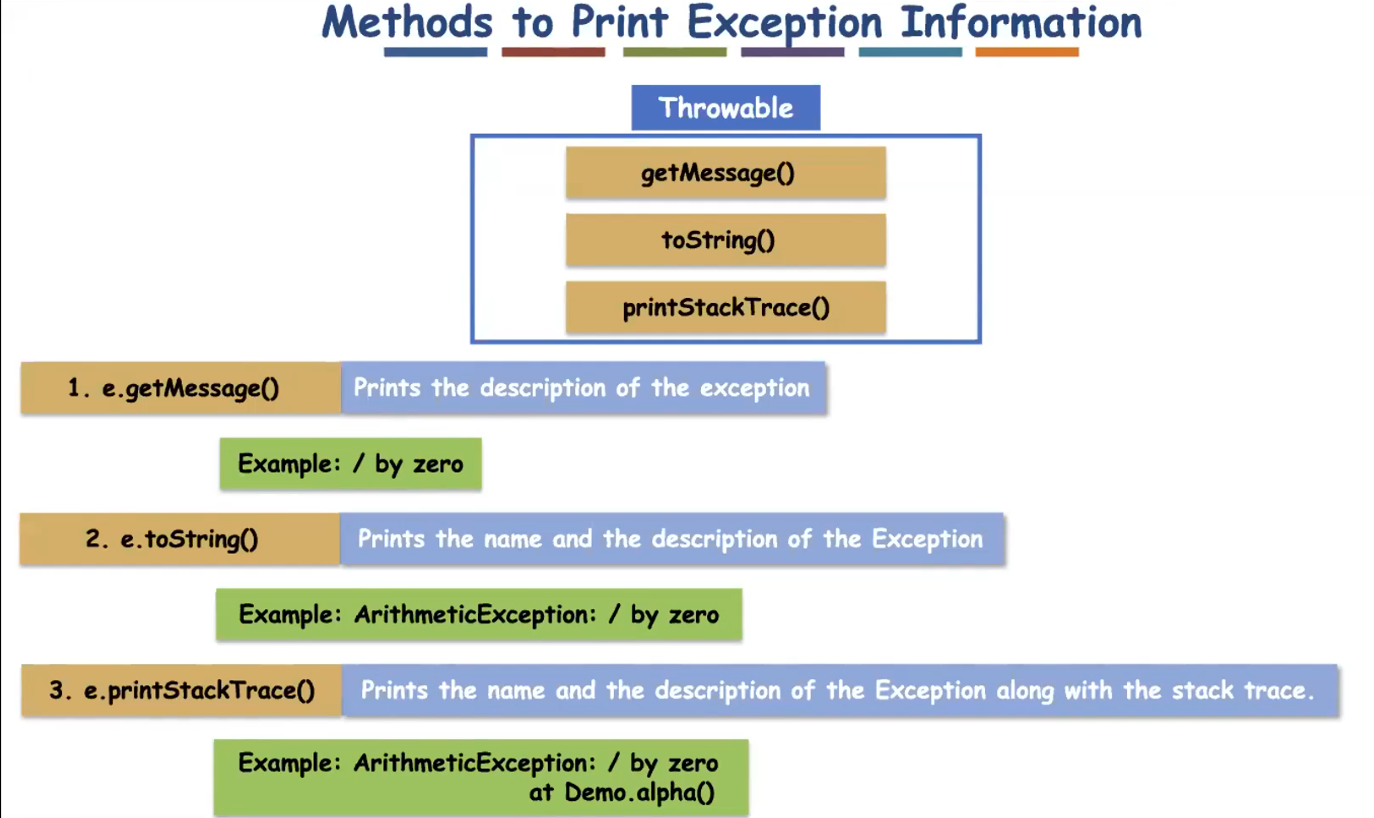
Some important resources are opened during the execution of a program , and that needs to be closed at any cost so we use finally block.

The exception contains the following details



Stack trace means at which stack frame the exception has occurred.

Methods used in the catch bock to print the information are



Eg: Print\_Exception\_Information\_Using\_Methods

// go through the code

finally block cannot be written alone , it should be written along with try, catch ( or) with try block.

finally block will not be executed if System.exit() is used before it. Because exit() will terminate the program. When control comes to it executed.

Eg: Finally\_Block\_Exit\_Method

//go through the code

