EXCEPTION HANDLING IN JAVA

Elizabeth Alexander
Hindustan University

JAVA - EXCEPTIONS

- An exception (or exceptional event) is a problem that arises during the execution of a program.
- When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.
- An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
    - A user has entered an invalid data.
    - A file that needs to be opened cannot be found.
    - A network connection has been lost in the middle of communications or the JVM has run out of memory.
- Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Error vs Exception

Error: An Error indicates serious problem that a reasonable application should not try to catch.

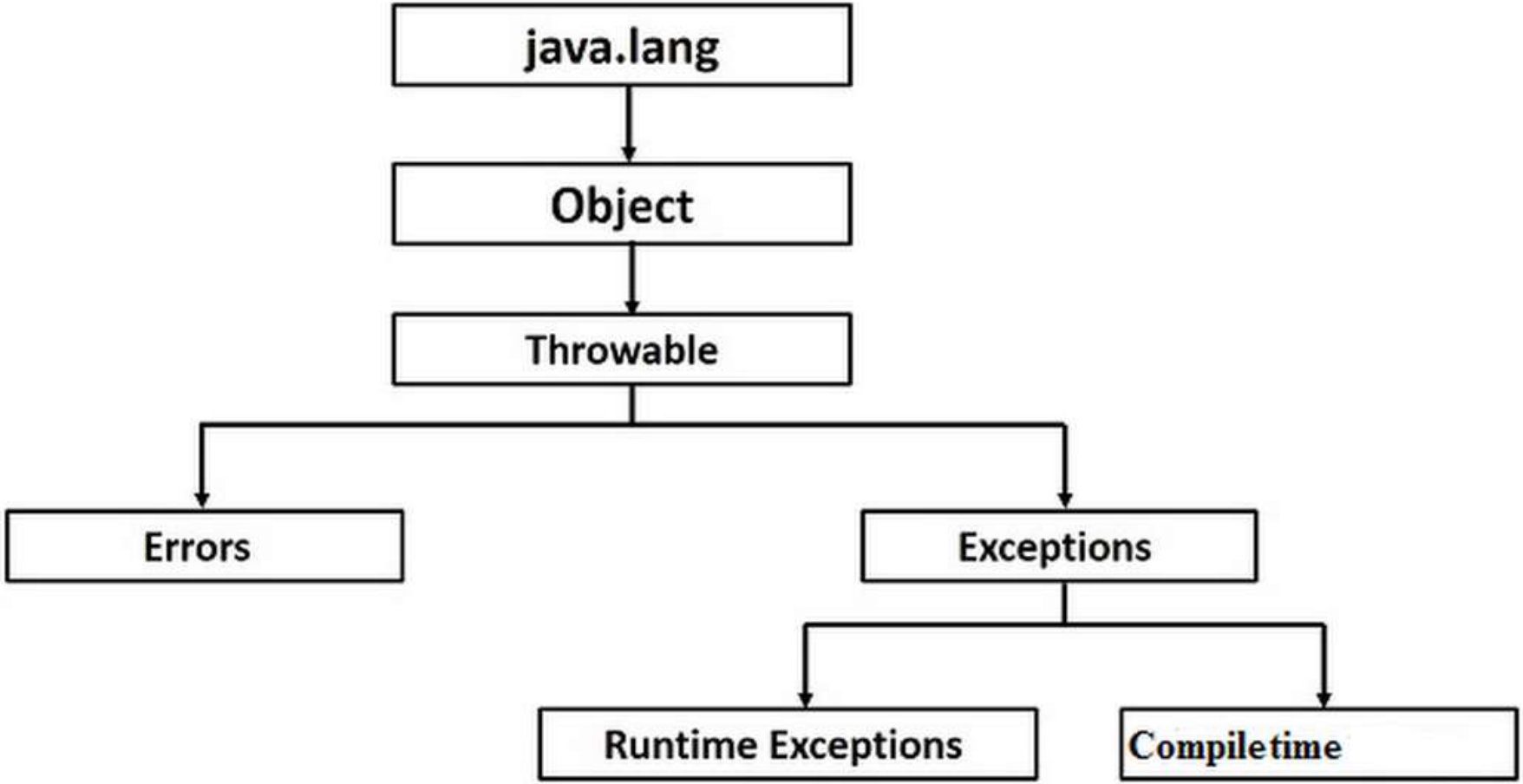
Exception: Exception indicates conditions that a reasonable application might try to catch.

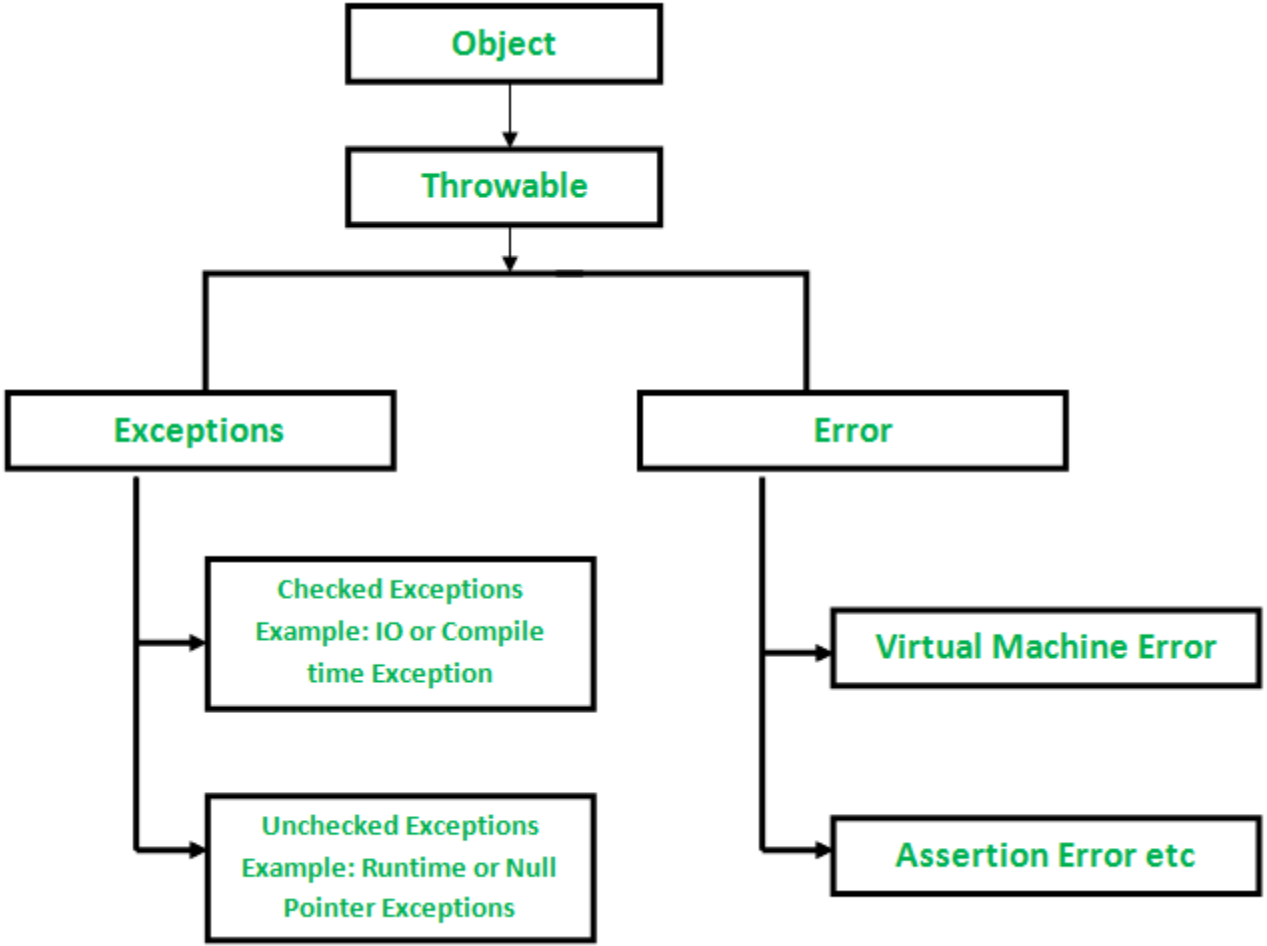
Exception Hierarchy

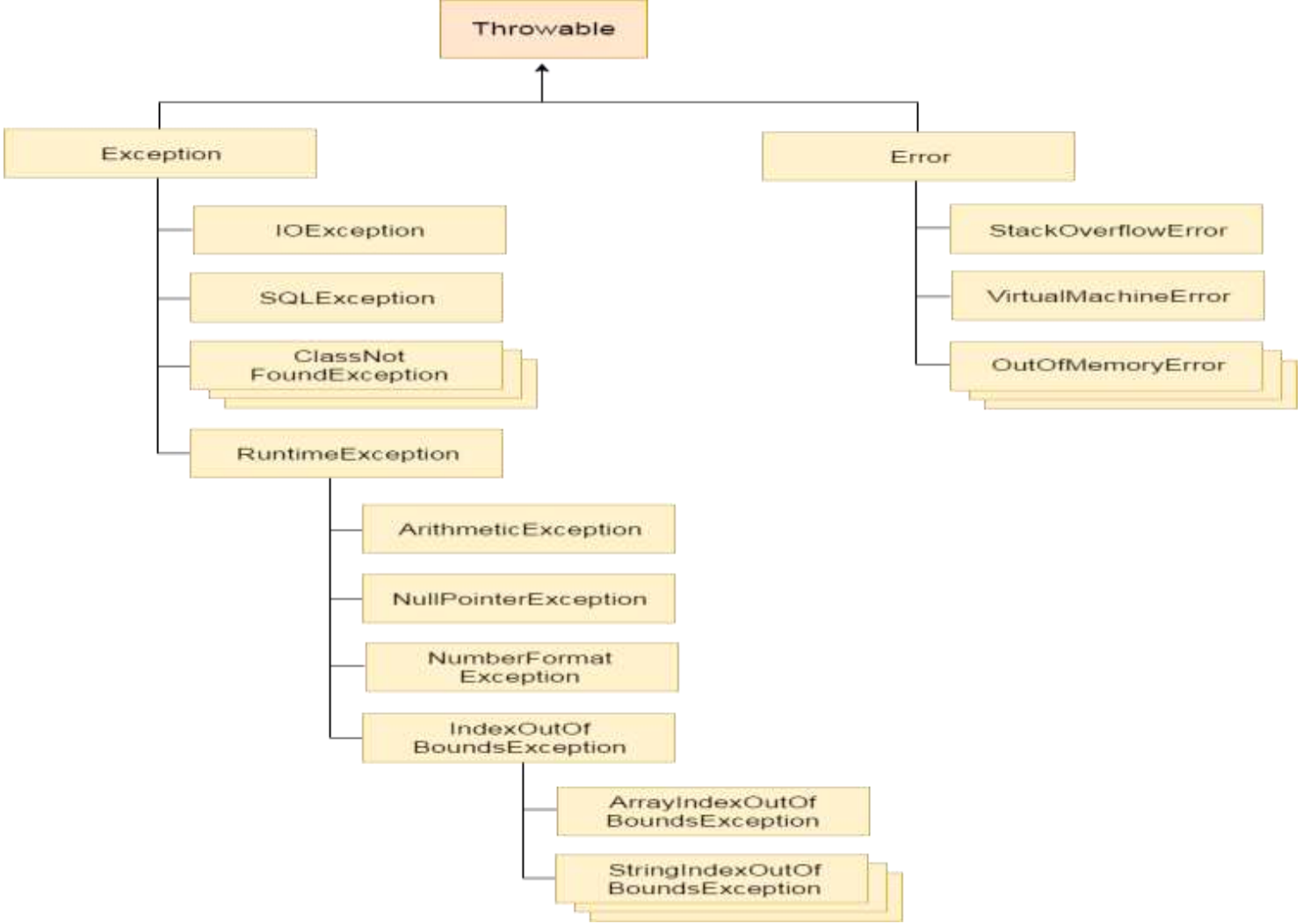
- All exception and errors types are subclasses of class **Throwable**, which is base class of hierarchy.
- One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception.
- There is an important subclass of Exception, called RuntimeException.Exceptions of this type are automatically defined for the programs that you write and include things such as division by zero and invalid array indexing.

Exception Hierarchy

- Another branch,**Error** which defines exceptions that are not expected to be caught under normal circumstances by your program ,
- Errors are used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.


java.lang
Object
Throwable
Errors
Exceptions
Runtime Exceptions
Compiletime


Object
Throwable
Exceptions
Error
Checked Exceptions
Example: IO or Compile time Exception
Unchecked Exceptions
Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc


Throwable
Exception
Error
IOException
SQLException
ClassNot FoundException
RuntimeException
StackOverflowError
VirtualMachineError
OutOfMemoryError
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException

Types of Exceptions

- Java's exceptions can be categorized into two types:

  Checked exceptions

  Unchecked exceptions

1. **Checked exceptions −**
   - A checked exception is an exception that occurs at the compile time.
   - These are also called as **compile time exceptions.**
   - These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.
   - checked exceptions are subject to the catch or specify a requirement, which means they require catching or declaration. This requirement is optional for unchecked exceptions.
   - Code that uses a checked exception will not compile if the catch or specify rule is not followed.

Types of Exceptions Cont...

- All Java exceptions are checked exceptions except those of the Error and RuntimeException classes and their subclasses.
    - For example, if you use **FileReader** class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a *FileNotFoundException* occurs, and the compiler prompts the programmer to handle the exception.
    - Since the methods **read()** and **close()** of FileReader class throws IOException, you can observe that the compiler notifies to handle IOException, along with FileNotFoundException.

Types of Exceptions Cont...

2. Unchecked exceptions − An unchecked exception is an exception that occurs at the time of execution.

- These are also called as **Runtime Exceptions**.
- These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.
- For example, if you have declared an array of size 5 in your program, and trying to call the 6^{th} element of the array then an *ArrayIndexOutOfBoundsExceptionexception* occurs.

3. Errors − These are not exceptions at all, but problems that arise beyond the control of the user or the programmer.

- Errors are typically ignored in your code because you can rarely do anything about an error.
- For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Exception Handling

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

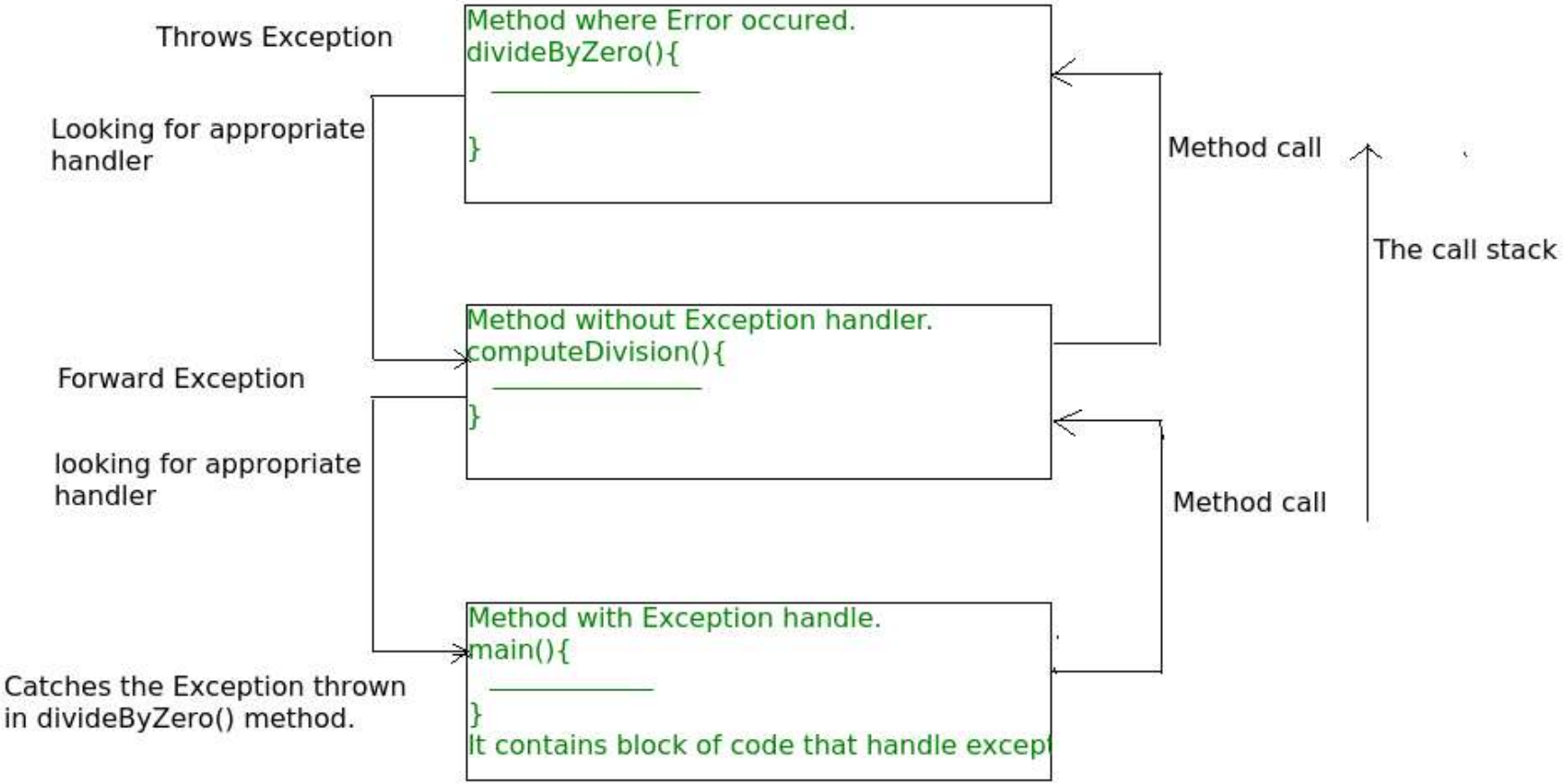
How JVM handle an Exception?

Default Exception Handling : Whenever inside a method, if an exception has occurred, the method creates an Object known as **Exception Object** and hands it off to the run-time system(JVM).

- The exception object contains name and description of the exception, and current state of the program where exception has occurred.
- Creating the Exception Object and handling it to the run-time system is called **throwing an Exception**.
- There might be the list of the methods that had been called to get to the method where exception was occurred. This ordered list of the methods is called **Call Stack**.
- Now the following procedure will happen.
    - The run-time system searches the call stack to find the method that contains block of code that can handle the occurred exception. The block of the code is called **Exception handler**.

Exception Handling

- The run-time system starts searching from the method in which exception occurred, proceeds through call stack in the reverse order in which methods were called.
- If it finds appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.
- If run-time system searches all the methods on call stack and couldn’t have found the appropriate handler then run-time system handover the Exception Object to **default exception handler** , which is part of run-time system. This handler prints the exception information in the following format and terminates program **abnormally**.



The call stack and searching the call stack for exception handler.

How Programmer handles an exception?

Customized Exception Handling :

- Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**.
- Program statements that can raise exceptions are contained within a try block.
- If an exception occurs within the try block, it is thrown.
- System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword throw.
- Any exception that is thrown out of a method must be specified as such by a throws clause.
- Any code that absolutely must be executed after a try block completes is put in a finally block.

Advantage of Exception Handling

- **Maintain the normal flow of the application.**: Exception normally disrupts the normal flow of the application that is why we use exception handling.
- **Separating Error-Handling Code from "Regular" Code :** Exceptions provide the means to separate the details of what to do when something out of the ordinary happens from the main logic of a program.
- **Propagating Errors Up the Call Stack :** Another advantage of exceptions is the ability to propagate error reporting up the call stack of methods.
    - Suppose that the readFile method is the fourth method in a series of nested method calls made by the main program: method1 calls method2, which calls method3, which finally calls readFile.
    - Suppose also that method1 is the only method interested in the errors that might occur within readFile.

Advantage of Exception Handling Cont...

- The Java runtime environment searches backward through the call stack to find any methods that are interested in handling a particular exception.
- A method can duck any exceptions thrown within it, thereby allowing a method farther up the call stack to catch it.
- Hence, only the methods that care about errors have to worry about detecting errors.

Grouping and Differentiating Error Types : Because all exceptions thrown within a program are objects, the grouping or categorizing of exceptions is a natural outcome of the class hierarchy.

- An example of a group of related exception classes in the Java platform are those defined in java.io — IOException and its descendants.

- IOException is the most general and represents any type of error that can occur when performing I/O. Its descendants represent more specific errors. For example, FileNotFoundException means that a file could not be located on disk.

Try Catch in Java – Exception handling

Try block

- The try block contains set of statements where an exception can occur.
- A try block is always followed by a catch block, which handles the exception that occurs in associated try block.
- A try block must be followed by catch blocks or finally block or both.

Syntax of try block

```
try{
   //statements that may cause an exception
}
```

- While writing a program, if certain statements in a program can throw a exception, enclosed them in try block and handle that exception

Try Catch in Java- Cont...

Catch block

- A catch block is where handle the exceptions are handled
- This block must follow the try block.
- A single try block can have several catch blocks associated with it.
- We can catch different exceptions in different catch blocks.
- When an exception occurs in try block, the corresponding catch block that handles that particular exception executes.
- For example if an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.

Try Catch in Java- Cont...

Syntax of try catch in java

```
try
{
    //statements that may cause an exception
}
catch (exception(type) e(object))
{
    //error handling code

}
```

Multiple catch blocks in Java

Rules about multiple catch blocks

- a single try block can have any number of catch blocks.
- A generic catch block can handle all the exceptions

```
catch(Exception e){
        //This catch block catches all the exceptions

}
```

- If no exception occurs in try block then the catch blocks are completely ignored.
- Corresponding catch blocks execute for that specific type of exception:
    - catch(ArithmeticException e) is a catch block that can hanlde ArithmeticException
    - catch(NullPointerException e) is a catch block that can handle NullPointerException

- If an exception occurs in try block then the control of execution is passed to the corresponding catch block.
- A single try block can have multiple catch blocks associated with it, you should place the catch blocks in such a way that the generic exception handler catch block is at the last
- The generic exception handler can handle all the exceptions but you should place is at the end
- if you place it at the before all the catch blocks then it will display the generic message. The user should get a meaningful message for each type of exception rather than a generic message.

throws Keyword

- Throws keyword is used for handling checked exceptions
- It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

```
returntype methodname() throws exception_list
{
//method code
}
```

- Any method that is capable of causing exceptions must list all the exceptions possible during its execution, so that anyone calling that method gets a prior knowledge about which exceptions are to be handled. A method can do so by using the **throws** keyword.

throw exception in java

- Throw keyword can also be used for throwing custom/user defined exceptions
- throw keyword is used to throw an exception explicitly.
- Only object of Throwable class or its sub classes can be thrown.
- Program execution stops on encountering **throw** statement, and the closest catch statement is checked for matching type of exception.

Syntax :

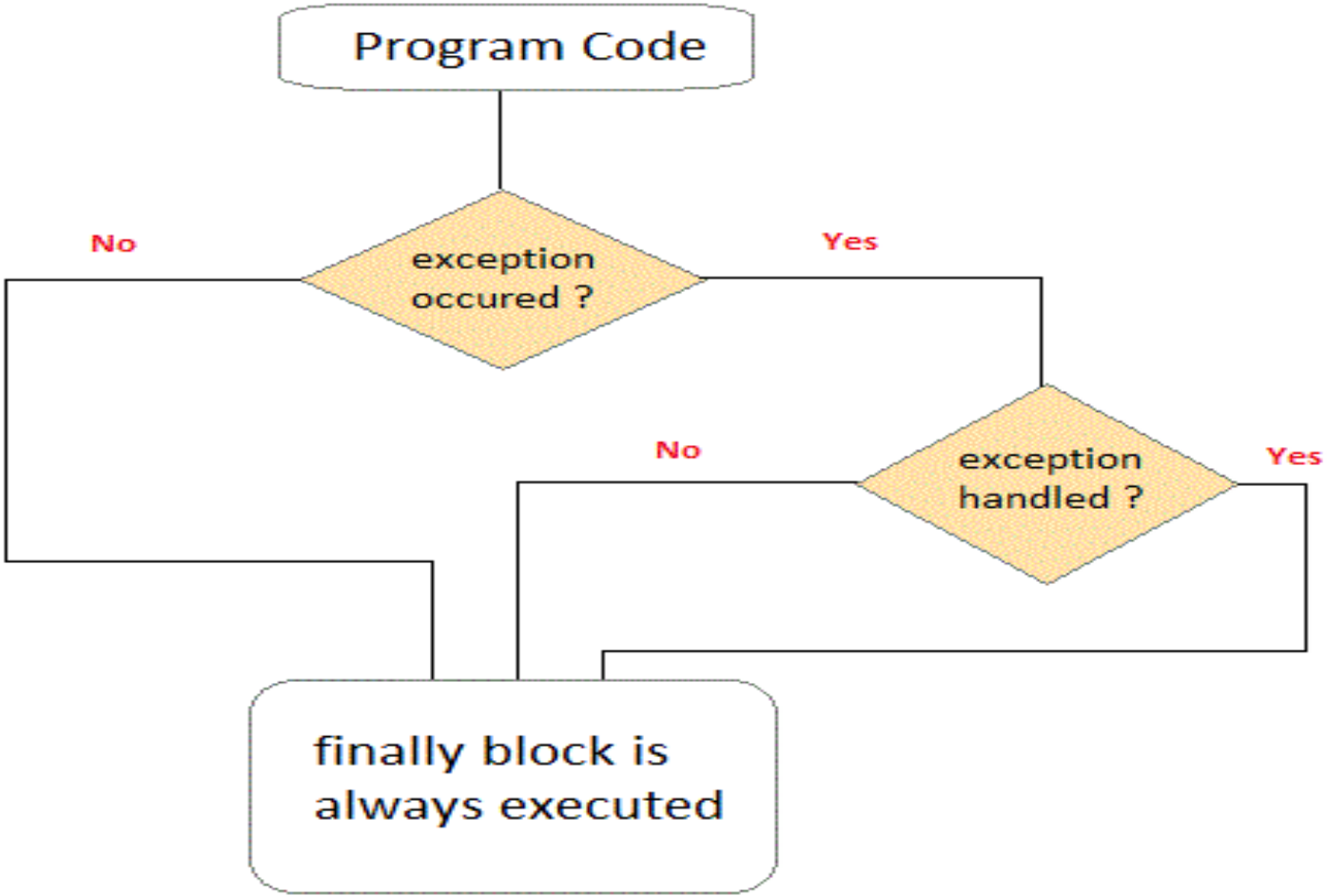
- **throw** *ThrowableInstance*

Difference between throw and throws

throw	throws
throw keyword is used to throw an exception explicitly.	throws keyword is used to declare an exception possible during its execution.
throw keyword is followed by an instance of Throwable class or one of its sub-classes.	throws keyword is followed by one or more Exception class names separated by commas.
throw keyword is declared inside a method body.	throws keyword is used with method signature (method declaration).
We cannot throw multiple exceptions using throw keyword.	We can declare multiple exceptions (separated by commas) using throws keyword.

finally

- A finally keyword is used to create a block of code that follows a try block.
- A finally block of code is always executed whether an exception has occurred or not. Using a finally block, it lets you run any clean-up type statements that you want to execute, no matter what happens in the protected code.
- A finally block appears at the end of catch block.


Program Code
exception occured ?
No
Yes
exception handled ?
No
Yes
finally block is always executed

User defined exception in java

- In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as user-defined or custom exceptions.

Syntax

- class classname extends Exception

Notes:

  - 1. User-defined exception must extend Exception class.
  - 2. The exception is thrown using throw keyword.