Exception Handling in Java

Dr. Partha Pratim Sarangi

School of Computer Engineering

Exception

- Exception is an abnormal condition that arises when executing a program that disrupts the normal flow of the program. Exception is an error, which is possible to detect and handle in order to maintain the normal flow of the application.
- It is recoverable.

- Error is also an abnormal condition that occurs during execution of the program due to lack of system resources. Unfortunate thing for the error is that it cannot be handled or managed by the program.
- It is irrecoverable.

Java programming language provide an efficient syntactic mechanism to detect and handle exceptions based on object-oriented paradigm.

Exception handling

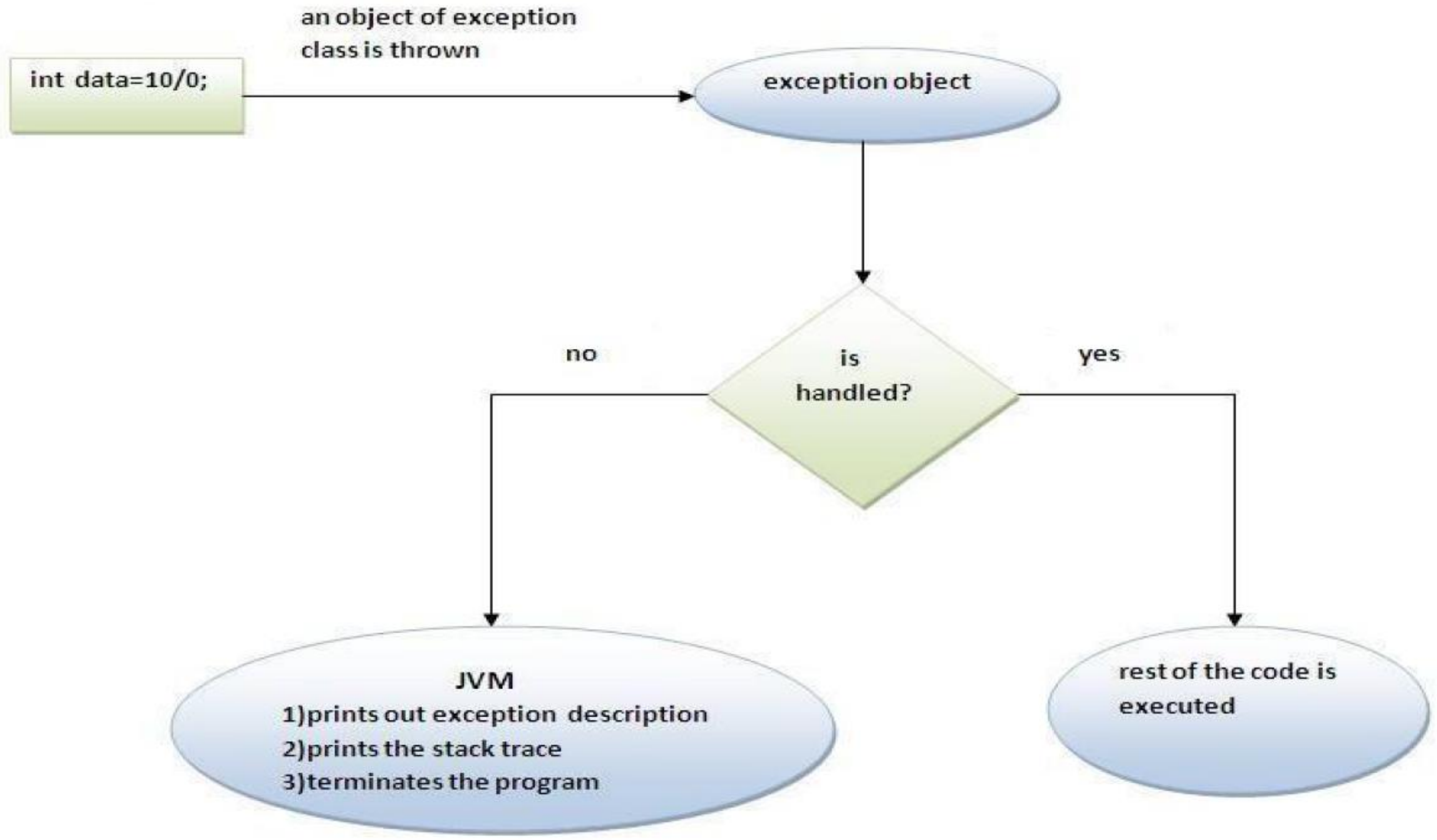
- In Java, an exception is an event (object) that occurs during the execution of a program that disrupts the normal flow of instructions. An exception can be thrown when a runtime error occurs, or when an anticipated result fails to occur.
- Exceptions in Java are objects that are created when an error or an exceptional condition occurs. The object contains information about the error, such as the type of error and the location in the program where the error occurred.
- The object is then thrown by the program, which means that it is passed to the calling method or to the Java Virtual Machine (JVM) to be handled.
- This runtime error management in object-oriented programming is called Exception Handling.

Importance of 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 a program and in the statement 5, there occurs an exception then, 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 code will be executed. That is why exception handling plays an important roll in Java.

Events of exception handling


int data=10/0;
an object of exception class is thrown
exception object
is handled?
no
yes
JVM
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

Exception in Java

- In Java, Exception is a predefined class present in java.lang package. The unexpected situations that may occur during program execution are as follows:
- The array index crosses the range
- A number is divided by zero
- Inability to find files
- Problems in network connectivity

In Java, Exception is handled by five keywords such as try, catch, finally, throw, and throws.

Exception handling mechanism in Java

- The Java code that may produce an Exception is placed inside try block.
- catch block is used to handle the Exception.
- If some necessary code is to be executed after try block then it is placed inside finally block.
- By the help of throw keyword programmer generates the Exception.
- When an Exception is thrown out of a method throws keyword is used.

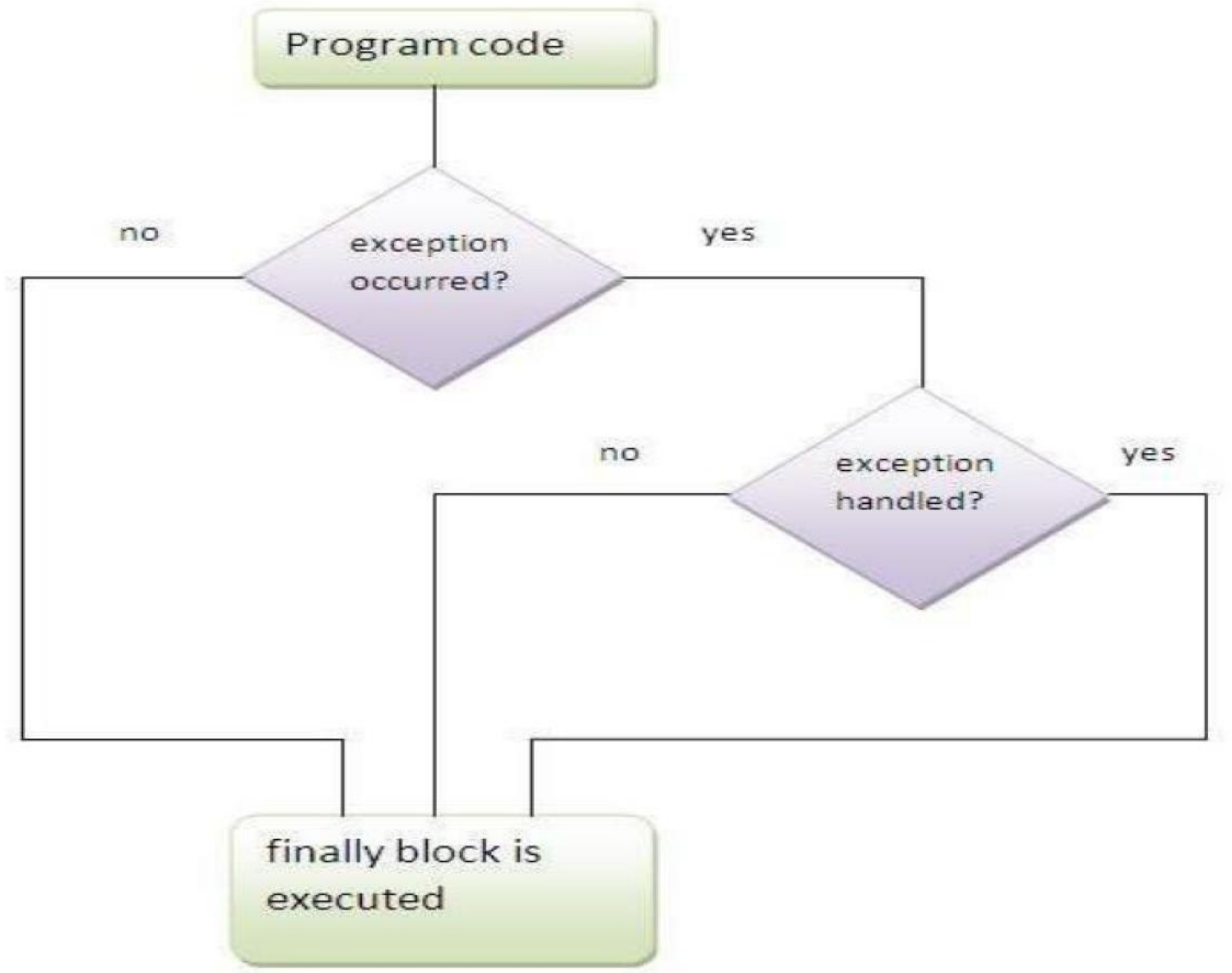
Syntax of Exception Handling

```
try {
            // block of codes expected to produce exception
       }
catch (Exception ex)
{
    // codes to handle exception
}
finally
{
    // codes that have to be executed if exception occurs
}
```

finally block

- finally block is a block that is used to execute important code such as closing connection, stream etc.
- finally block is always executed whether exception is handled or not.
- finally block must be followed by try or catch block.
- Rule: For each try block there can be zero or more catch blocks, but only one finally block.
- Note: If you don't handle exception, before terminating the program, JVM executes finally block(if any).
- Note: The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

Program execution sequence for exception handling



Examples of exceptions in Java:

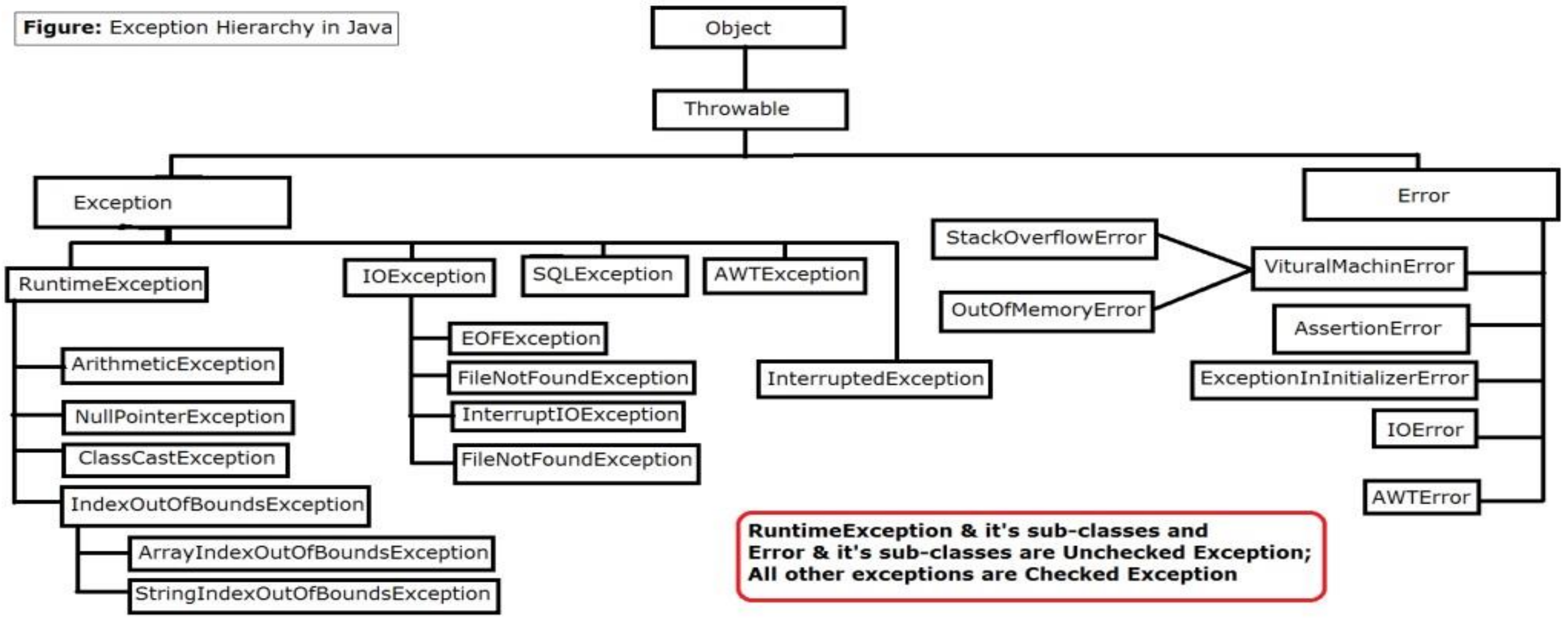
- ArithmeticException - This exception is thrown when an arithmetic operation fails to execute properly, such as dividing by zero.
- NullPointerException - This exception is thrown when a program attempts to use a null reference, which means that the program is trying to access an object that does not exist.
- ArrayIndexOutOfBoundsException - This exception is thrown when a program attempts to access an array with an invalid index, such as an index that is negative or greater than the length of the array.
- ClassCastException - This exception is thrown when a program attempts to cast an object to a type that it is not compatible with.

Contd…

- IOException - This exception is thrown when an input/output operation fails, such as when a file cannot be read or written to.
- FileNotFoundException - This exception is thrown when a program attempts to open a file that does not exist.
- IllegalArgumentException - This exception is thrown when a method is passed an invalid argument.
- IllegalStateException - This exception is thrown when a program is in an illegal or inappropriate state to perform a certain operation.

Hierarchy of Exception

- All exception classes are subtypes of the java.lang.Exception class. The Exception class is basically a subclass of the Throwable class. Apart from Exception class, another subclass named as Error is derived from the Throwable class.



RuntimeException & it's sub-classes and Error & it's sub-classes are Unchecked Exception; All other exceptions are Checked Exception

Checked Exception

- Exception which is checked at compile-time during compilation known as Checked Exception.
- As the name suggests checked exceptions are checked by the compiler.
- If a line of code could possibly through an exception in the method, then compiler enforces to handle that exception or declare that exception in the same method, then that type of exception is called checked exception.
- For example, accessing a file from remote location could possibly throw file not found exception.
- It is the programmer’s responsibility to handle the checked exception for successful compilation.
- **Note: if checked exception isn’t handled then program will throw compile-time error**

Unchecked exception

- Exception which is not checked at compile-time known as Unchecked Exception.
- A line of code that could possibly throw exception at runtime is said to be unchecked exception.
- Unchecked exception are because of programming-error.
- Hence, it is programmer’s responsibility to handle unchecked exception, otherwise, program will terminate abnormally at runtime.
- Runtime exception & its child classes and error & its child classes are examples of Unchecked Exception.
- **The compiler does not enforce the programmer either handle the exception or declare the exception in the method.**

Misconception about checked and unchecked exception:

- Sometimes, checked exception are also referred as compile-time exception and unchecked exception are referred as runtime exception.
- But this is **mis-leading** because every exception (whether it is checked or unchecked) occurs/raised only at the runtime i.e.; during program execution only.
- Reason: during compilation; checked exception are checked and raises compile-time error, due to which programmer has to handle the exception by providing either try-catch blocks or using throws keyword declaration in the method where exception occurred.
- Whereas unchecked exception aren’t checked during compilation, rather it raises exception during execution because of programming error. Hence, programmer must handle the exception, otherwise program execution will abnormally terminated at runtime.

How do you do exception handling?

- Three mechanism are involved to handle an exception:
  - **Claiming exceptions** - each method needs to specify what exceptions it expects might occur (i.e. what it will throw, and not handle internally).
  - **Throwing an exception** - When an error situation occurs that fits an exception situation, an exception object is created and thrown.
  - **Catching an exception** - Exception handlers (blocks of code) are created to handle the different expected exception types. The appropriate handler catches the thrown exception and performs the code in the block

Claiming Exceptions

- In a Java method claim an exception with the keyword throws. This goes at the end of the method's prototype and before the definition. Multiple exceptions can be claimed, with a comma-separated list. Examples:
  - public void myMethod() throws IOException
  - public void yourMethod() throws IOException, AWTException, SQLException

Throwing Exceptions

- In Java, use the keyword throw, along with the type of exception being thrown. An exception is an object, so it must be created with the new operator. It can be created within the throw statement or before it. Examples:
  - throw new BadHairDayException();
  - MyException m = new MyException();

    throw m;
  - if(amount < 5000)

    throw new MyValidateException("You are not eligible for loan)
- **Note that this is different than the keyword throws, which is used in claiming exceptions.**

Catching Exceptions

- Any group of statements that might throw an exception -- place inside a try block.
    - If an exception occurs (i.e. "is thrown"), execution ends at the "throw point" and will only resume if the exception is "caught“.
- After the try block, there should be one or more catch blocks.
    - Each catch block has a parameter -- the type of exception that this block will handle.
    - There can be multiple catch blocks
    - If an exception is thrown, the first matching catch block is the one that runs.
- An optional finally block can be used
    - A finally block is always executed, no matter how control leaves the try block

What happens if an exception is not caught?

- If an exception is not caught (with a catch block), the runtime system will abort the program (i.e. crash) and an exception message will print to the console. The message typically includes:
  - name of exception type
  - short description of the exception
  - stack trace (provides line number where exception occurred)
- For **a checked exception**, a method must either catch an exception (handle internally) or claim it (declare that it will be thrown)
  - This is enforced by compiler.
  - Claiming an exception is giving information to the caller, so that they will know to catch it!
- For an **unchecked exception** (runtime exception), there's no compiler enforcement.
  - Runtime exceptions could occur anywhere, arising automatically.
  - Usually can be fixed by better coding
  - Examples: Division by zero, array index out of bounds, null pointer exception

Instance methods in exception objects

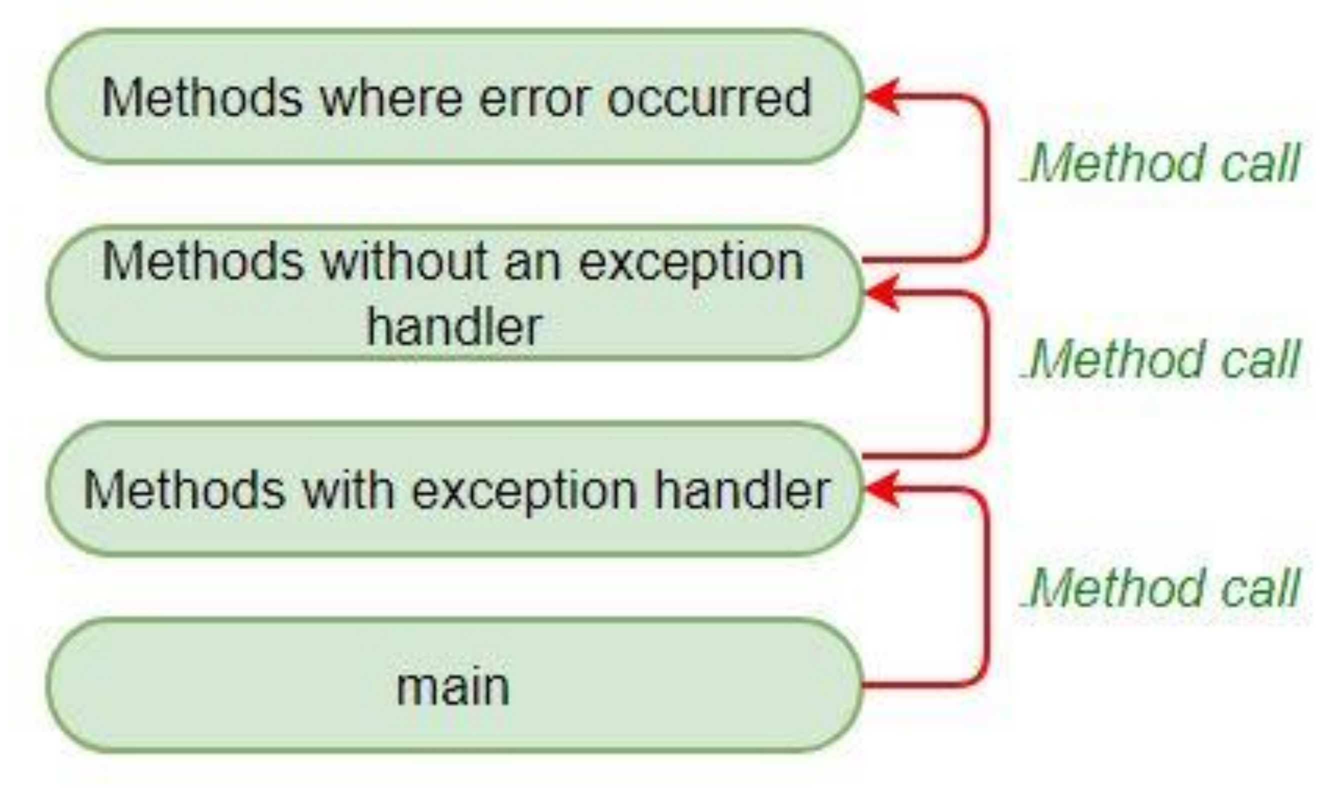
- Exception objects are created from classes, which can have instance methods. There are some special instance methods that all exception objects have (inherited from Throwable):
  - public String getMessage() -- returns a detailed message about the exception
  - public String toString() -- returns a short message describing the exception
  - public void printStackTrace() -- returns all three information (Exception name, Exception short description, Stack trace)

Difference between throw and throws:

throw keyword	throws keyword
1. throw is used to explicitly throw an exception.	throws is used to declare an exception.
2. checked exception can not be propagated with throw.	checked exception can be propagated with throws.
3. throw is followed by an instance using new keyword.	throws is followed by method
4. throw is used within the method.	throws is used with the method signature.
5. You cannot throw multiple exception.	You can declare multiple exception e.g. public void method()throws IOException, SQLException.

Exception Propagation in Java

- An exception is first thrown from the top of the stack and if it is not caught, it drops down **the call stack** to the previous method. If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.
- When an exception happens, Propagation is a process in which the exception is being dropped **from to the top to the bottom of the stack**. If not caught once, the exception again drops down to the previous method and so on until it gets caught or until it reach the very bottom of the call stack.


Methods where error occurred
Methods without an exception handler
Methods with exception handler
main
Method call
Method call
Method call

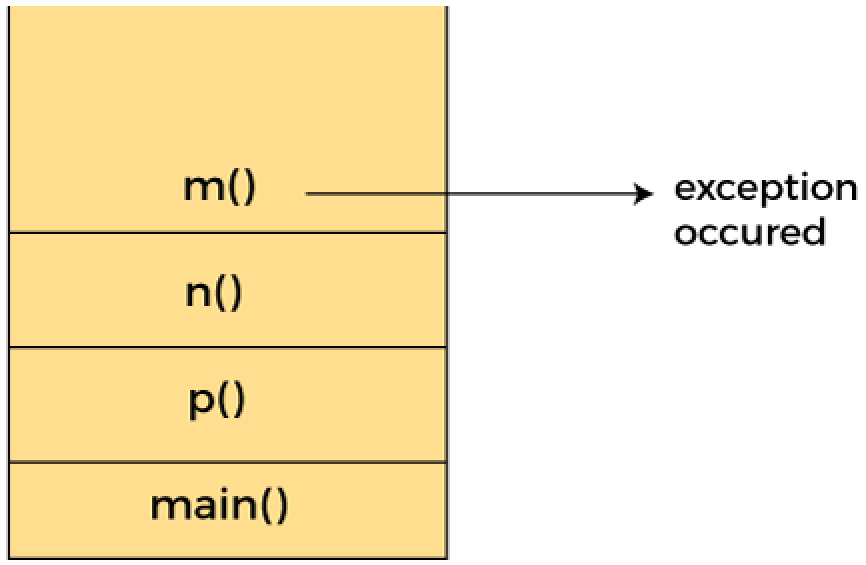
- Exception Propagation is possible in **Unchecked Exceptions**.
- Note: By default Unchecked Exceptions are forwarded in calling chain (propagated).

Exception Propagation Example

```
class TestExceptionPropagation
{
  void m()
  {
    int data=50/0;
  }
 void n()
 {
    m();
  }
  void p()
  {
    try
    {
       n();
    }
  catch(Exception e)
  {
    System.out.println("exception
    handled");
  }
}
public static void main(String args[])
 {
     TestExceptionPropagation obj = new
     TestExceptionPropagation();
     obj.p();
     System.out.println("normal flow...");
  }
}
```

Contd…

- In the above example exception occurs in the m() method where it is not handled, so it is propagated to the previous n() method where it is not handled, again it is propagated to the p() method where exception is handled.
- Exception can be handled in any method in **call stack** either in the main() method, p() method, n() method or m() method.



Call Stack

Is exception propagation possible in Checked Exception?

- Yes, through Java **throws clause** which describes that checked exceptions can be propagated by throws keyword.
- Rule: If we are calling a method that declares an exception, we must either caught or declare the exception.
- In case we declare the exception, if exception does not occur, the code will be executed fine.
- In case we declare the exception and the exception occurs, it will be thrown at runtime because throws does not handle the exception.

Exception Propagation for Checked Exception

```
import java.io.IOException;
class TestExceptionPropagation
{
  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...");
  }
}
```

Exception Handling with Method Overriding in Java

Different cases for exception handling with respect to method overriding:

1. If parent-class method doesn’t declare any exception
2. If parent-class method declares unchecked exception
3. If parent-class method declares checked exception
4. If parent-class method declares both checked & unchecked exceptions

1. If parent-class method doesn’t declare any exception

Rule 1.1: Then child-class overriding-method can declare any type of unchecked exception

Rule 1.2: If child-class overriding-method declares checked-exception, then compiler throws compile-time error

Rule 1.3: Then child-class overriding-method can declare no exception in the overriding-method of child-class

Contd...

2. If parent-class method declares unchecked exception

Rule 2.1: Then child-class overriding-method can declare any type of unchecked exception

Rule 2.2: If child-class overriding-method declares any checked-exception, then compiler throws compile-time error

Rule 2.3: Then child-class overriding-method can declare no exception

3. If parent-class method declares checked exception

Rule 3.1: Then child-class overriding-method can declare any type of unchecked exception

Rule 3.2: Then child-class overriding-method can declare same type of checked exception or one of its sub-class OR sub-type of declared checked exception

Rule 3.3: Then child-class overriding-method can declare no exception in the overriding-method of child-class

Contd…

4. If parent-class method declares both checked & unchecked exceptions

Rule 4.1: Then child-class overriding-method can declare any type of unchecked exception

Rule 4.2: Then child-class overriding-method can declare same type of checked-exception or one of its sub-class or no exception

Rule 4.3: Then child-class overriding-method can declare no exception

Conclusion

- When parent-class method declares no exception, then child-class overriding-method can declare,
  - No exception
  - Any number of unchecked exception
  - But checked exception allowed
- When parent-class method declares unchecked exception, then child-class overriding-method can declare,
  - No exception
  - Any number of unchecked exception
  - But checked exception not allowed
- When parent-class method declares checked exception, then child-class overriding-method can declare,
  - No exception
  - Same checked exception
  - Sub-type of checked exception
  - Any number of unchecked exception
- All above conclusion hold true, even if combination of both checked & unchecked exception is declared in parent-class‘ method