Exception Handling in Java

Errors and Exceptions in Java

- Errors are constant companions while programming.
- With practice, we become better at identifying and correcting them.
- Three types of errors in Java

1. Syntax errors : When compiler finds something wrong (missing semicolon, variable not declared)
2. Logical errors: Program does not work in the intended way.
3. Runtime errors : Error encountered when the program is under execution. These are called exceptions.

Syntax errors and logical errors are encountered by the programmers, whereas Run-time errors are encountered by the users.

Exceptions

- An exception is an event that occurs when a program is executed dissented the normal flow of instructions.
- Common exceptions
  - Null pointer exceptions
  - Arithmetic exceptions.
  - Array Index out of Bound exception

Try-catch block

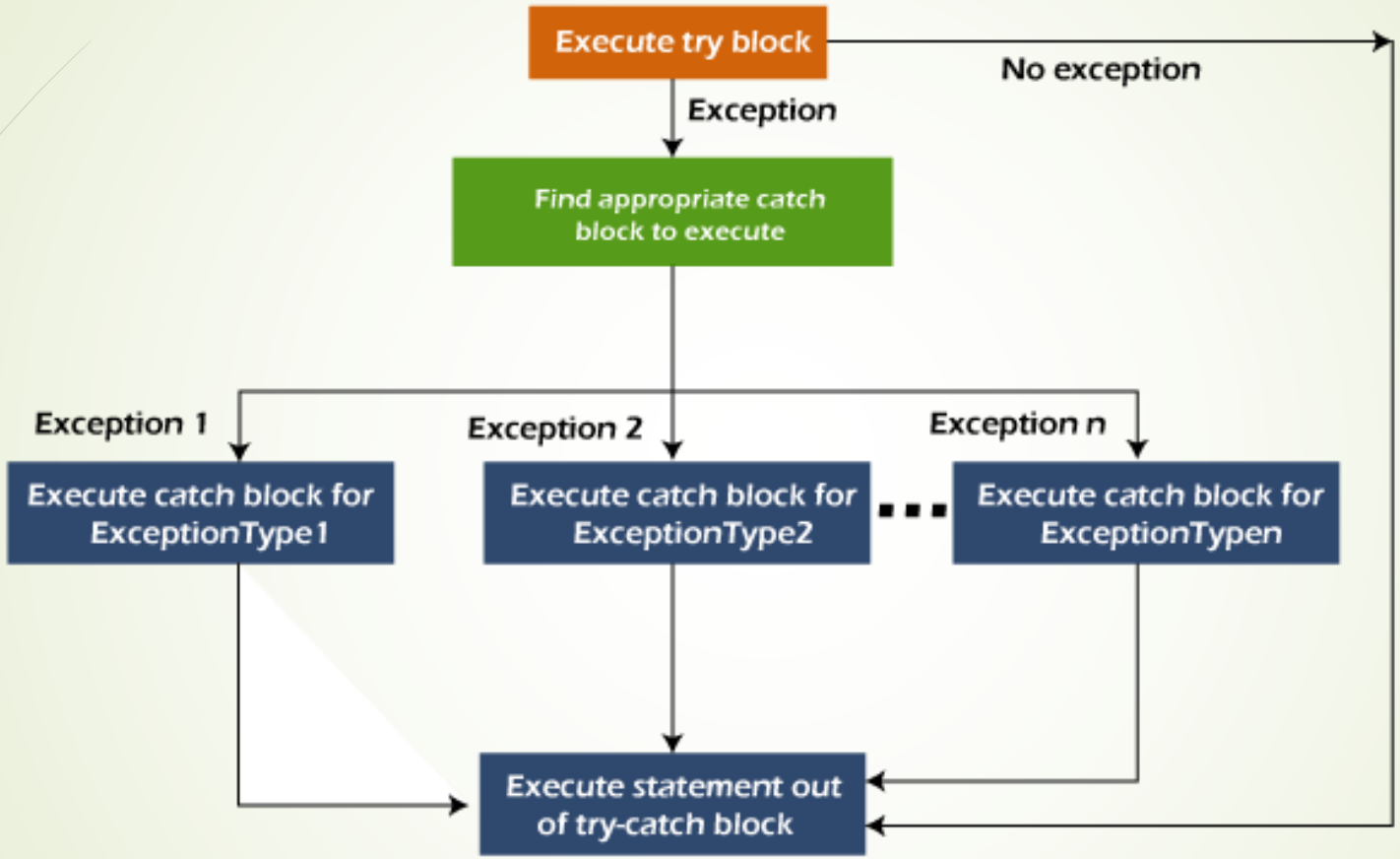
- The statements that might raise exceptions are kept inside try block.
- Once exceptions occurs inside try block, control comes to the corresponding catch block (catch block with matching exception type).
- Remaining catch blocks are skipped.

```
try{
            put your  logic
}
catch (Exception e){

            ...code to handle exception....
}
```

Multi-catch block

- A try block can be followed by one or more catch blocks.
- Each catch block contains a different exception handler.
- If we need to perform different tasks at the occurrence of different exceptions, use java muti-catch block.
- At a time, only one exception occurs and the corresponding catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e., catch for ArithmaticException must appear before Exception.


Execute try block
No exception
Exception
Find appropriate catch block to execute
Exception 1
Exception 2
Exception n
Execute catch block for ExceptionType1
Execute catch block for ExceptionType2
Execute catch block for ExceptionTypen
Execute statement out of try-catch block

throw keyword

- It is used to throw an exception explicitly by the programmer.
- The throw needs to be surrounded by try-catch block.

```
if ( b===0 ) {

        try{
                throw new ArithmeticException();
        }
        catch(Exception e){
                System.out.println(e);
        }
}
else{
         return a/b ;
}
```

Creating custom exception

- Custom exception can be created by extending the Exception class in java.lang.* package.
- Users create exceptions as per their own requirements.
- Example:
  1. Negative dimension exception can be created to throw exception when user inputs negative radius for a circle.
- Custom exceptions should be surrounded by try-catch blocks.
- Needs to override methods toString(), getMessage() and printStackTrace()

throws keyword

- It is usually used with a method.
- It gives indication to the programmer that the method might raise an exception.
- Therefore, the caller of the method must be prepare to handle an exception a try-catch logic block.

```
public void calculate (int a, int b) throws IOException { // code }
```

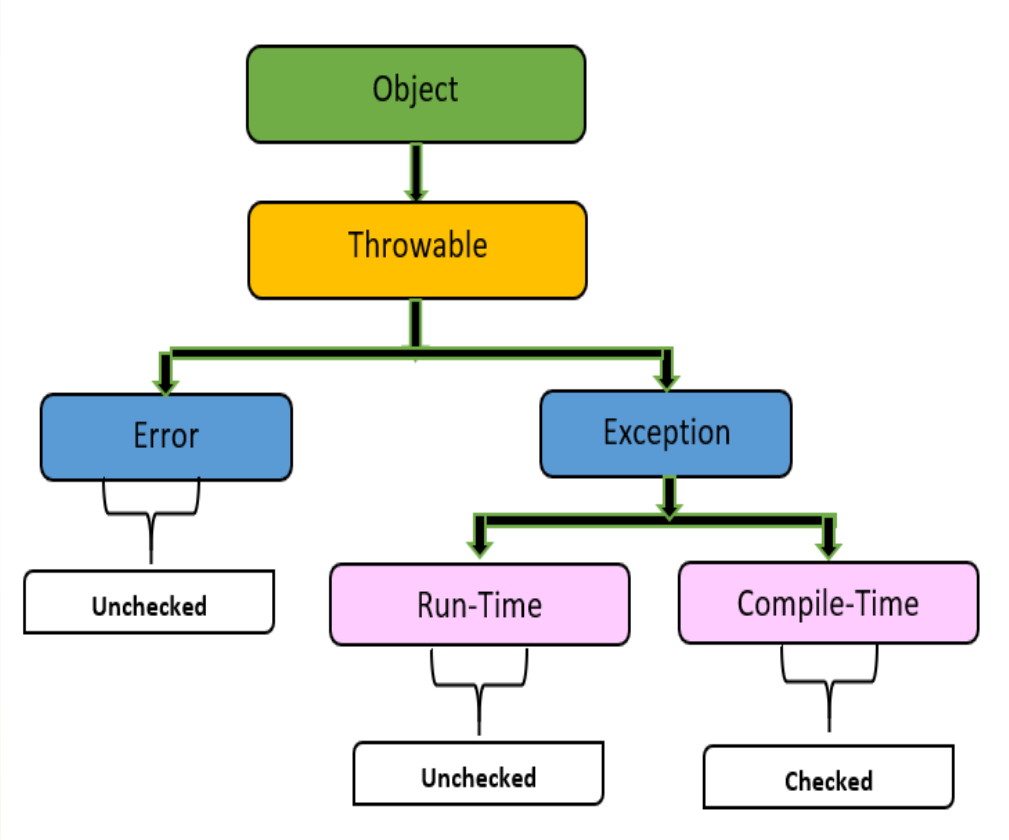
Checked vs Unchecked exceptions

Checked Exceptions

- Detected at compile-time
- If a code within a method throws checked exceptions, then use try-catch or use throws keyword.
- Example : FileNotFoundException under IOException.

Unchecked Exception

- Not detected during compile-time.
- Not forced by the compiler to either handle or specify the exception (using throws).
- It compiles, but shows exceptions during runtime if undesirable events occur.
- Example : ArithmaticException, ArrayOutofBoundException



finally block in Java

- Finally block contains the code which is always executed whether the exception is handled or not.
- Finally block can be used to put cleanup code such as closing a file, closing a connection etc.
- The important statements to be printed can be placed in the finally block.

Case 2 : When an exception does not occur

Check the below program where the Java program does not throw any exception, and the finally block is executed after the try block.

```
public static void main (String args[]){
        try{
                int data=25/5;
                SOP(data);
        }
        catch ((NullPointerException e){
                 SOP(e);
        }
        finally{
                SOP("finally block always gets executed");
        }

}
```

Case 2 : When an exception occurs, but not handled by catch block

Check the below program where the Java program throws an arithmetic exception, but corresponding catch block is not available. Still, finally block will be executed.

```
public static void main (String args[]){
        try{
                int data=25/0;
                SOP(data);
        }
        catch ((NullPointerException e){
                 SOP(e);
        }
        finally{
                SOP("finally block always gets executed");
        }

}
```

Case 3 : When an exception occurs and handled by catch block

Check the below program where the Java program throws an arithmetic exception, which is handled by catch block. Still, finally block will be executed.

```
public static void main (String args[]){
        try{
                int data=25/0;
                SOP(data);
        }
        catch (ArithmaticException e){
                 SOP(e);
        }
        finally{
                SOP("finally block always gets executed");
        }

}
```