# Error Handling / Handling Exception

# Exception

- An exception is an object that is generated as the result of an error or an unexpected event.
- To prevent exceptions from crashing your program, you must write code that detects and handles them.
- In Java – each error and exception is also an Object

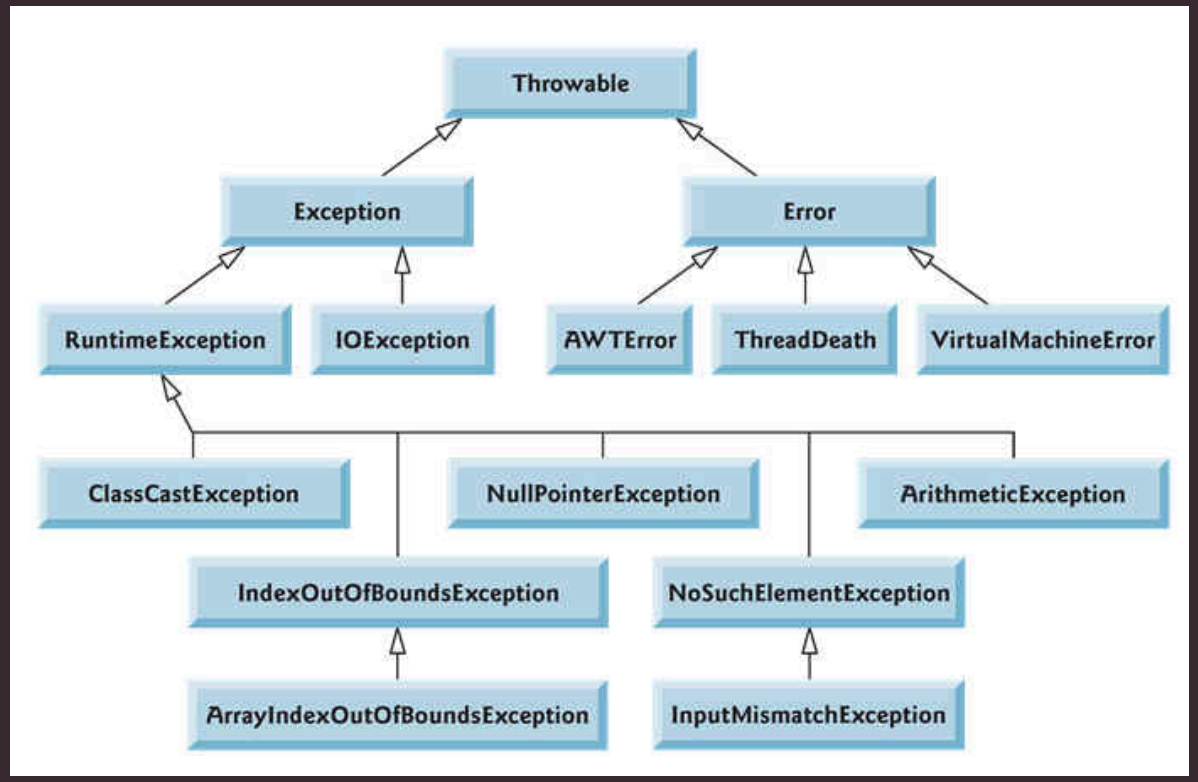
# Source of Errors

- Bugs
- 3rd party library
- Wrong user input
- Network issue
- Hard drive failure

# Exception Classes

- The exception classes are in the Java API. For example FileNotFoundException is in the java.io package.

- When you handle an exception that is not in the java.lang package, you will need the appropriate import statement



# Handling an Exception

- To handle an exception, you use try statement.

```
try{

    //try block statements
    //some code that might throw exception

} catch(Exception e){

    //catch block statements
    //TODO: handle exception

}
```

If this statement throws an exception

Then this statement is skipped

The program jumps to this catch block

```
try{

    file = new File(fileName);
    inputFile = new Scanner(file);
    System.out.println("The file was found");

} catch(Exception e){

    System.out.println("File not found");

}
```

If no exception is thrown in the try block, the program jumps to the statement that immediately follows the try/catch block

```
try{

    file = new File(fileName);
    inputFile = new Scanner(file);
    System.out.println("The file was found");

} catch(Exception e){

    System.out.println("File not found");

}


System.out.println("Done");
```

# Errors in Java

1. Compile errors: Compiler will catch it and will not let code to be compiled successfully. We can not use try/catch block to handle this.

   (Ex: Syntax errors, data types, creating object of interface, etc….)

2. Errors/Exceptions during code execution:
   - Error that takes place during execution(Runtime Error)

     (Ex: StackOverFlow error(Stack memory is full), OutOfMemoryError(Heap memory is full). We do not use try/catch to handle.
   - Runtime Exception that takes place during execution(Runtime Exceptions)

     (Ex: IndexOutOfBounds Exception, NoSuchElementException)

# Exception Object

- When runtime exception happens, java will catch it and assigns to a variable in catch statement.
- After it is successfully caught, we can use the variable and call some methods on the object.
- Popular ones are:
  - printStackTrace(): Prints the exception stack trace
  - getMessage(): Returns only brief description of the exception

# finally block

- The try statement may have an optional finally clause, which must appear after all of the catch clauses.
- The finally block is one or more statements that are always executed after the try block has executed and after any catch blocks have executed if an exception was thrown.
- The statements in the finally block execute whether an exception occurs or not.

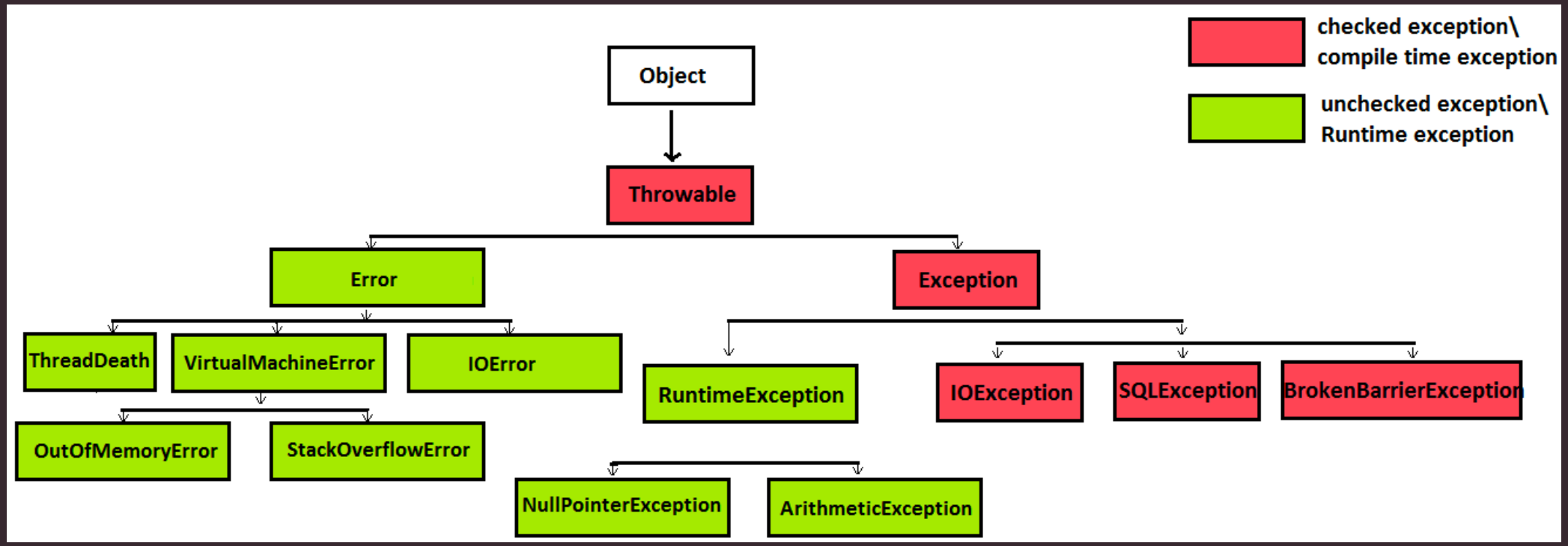
```
try{
    //code
}catch(Exception e){
    //code
}finally{
    //code that runs always
}
```

# Finally block

- Finally block, always runs except in 2 situations:
  - System.exit(0);
  - JVM crash

# Checked and Unchecked Exceptions

- There are two categories of exceptions:
  1. Checked Exception: It is an exception that we must handle for the code to compile. If you do not handle, code will not compile
  2. Unchecked Exception: It is optional to handle, code will compile even if we do not handle


checked exception\
compile time exception
unchecked exception\
Runtime exception
Object
Throwable
Error
Exception
ThreadDeath
VirtualMachineError
IOError
OutOfMemoryError
StackOverflowError
RuntimeException
IOException
SQLException
BrokenBarrierException
NullPointerException
ArithmeticException

# Checked Exceptions

- All of the exceptions that do not inherit from Error or RuntimeException are checked exceptions. These are the exceptions that you should handle in your program.
- For the code to compile we need to either :
  1. Handle: try…catch…finally
  2. Declare: throws declaration

# Unchecked Exceptions

- All of the exceptions that inherit from Error class or RuntimeException class are unchecked exceptions.
- These are the exceptions that you should not handle in your program.
- Code will compile even if we handle or not.
- Happens due to programming mistakes.

# Combinations

```
try{
    //code
}catch(Exception e){
    //code
}
```

1

```
try{
    //code
}catch(Exception e){
    //code
}finally{
    //code
}
```

2

```
try{
    //code
}catch(Exception e){
    //code
}catch(Exception e){
    //code
}catch(Exception e){
    //code
}
```

3

```
try{
    //code
}finally{
    //code
}
```

4

```
try{
    //code
}catch(Exception e){
    //code
}catch(Exception e){
    //code
}catch(Exception e){
    //code
}finally{
    //code
}
```

5

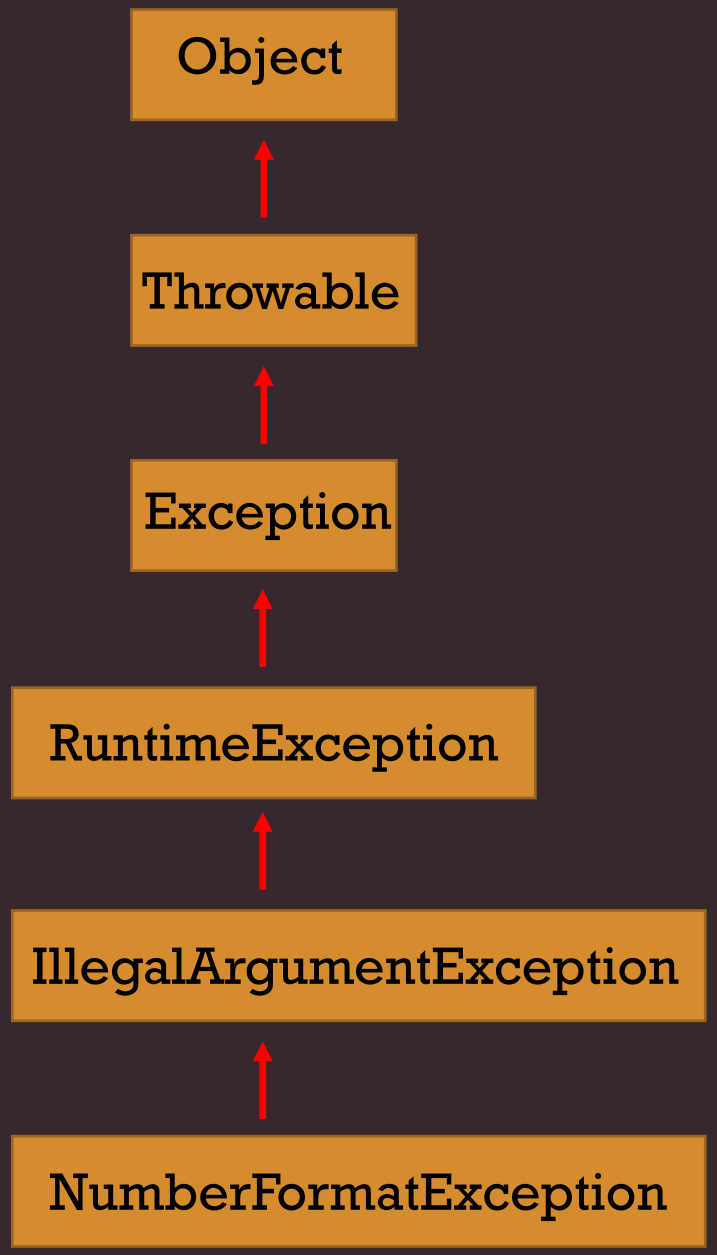
# Multiple catch Blocks

- In many cases, the code in the try block will be capable of throwing more than one type of exception. In such a case, we need to write a catch clause for each type of exception that could potentially thrown.

```
try{

}catch(ArithmeticException e){

    //handle exception

}catch(NullPointerException e){

    //handle exception

}catch(ArrayIndexOutOfBoundsException e){

    //handle exception
}
```

# Rule

- If you are handling multiple exceptions in the same try statement and some of the exceptions are related to each other through inheritance, then you should handle the more specialized exception classes before the more general exception classes.




```java
try{
    number = Integer.parseInt(str);
}catch(NumberFormatException e){
    System.out.println(str + " is not a number");
}catch(IllegalArgumentException e){
    System.out.println("Bad number format");
}
```

# Handling Multiple Exceptions with One catch Block

```
try{
    //code
}catch(NumberFormatException | IOException | InputMismatchException e){
    //code
}
```

# throw clause

- You can write code that throws one of the standard Java exceptions, or an instance of a custom exception class that you have designed.
- You can use throw statement to throw an exception manually. The general format of the throw statement is as follows:

```
throw new ExceptionType(MessageString);
```

# throws clause

- Throws clause informs the compiler that a method throws one or more exceptions.

```
public void sleep(int seconds) throws InterruptedException{

    Thread.sleep(second * 1000);

}
```

# throws clause -RULE

- When you declare a CHECKED exception, whoever CALLS this method is responsible to HANDLE it or DECLARE again

```
public static void sleep(int seconds) throws InterruptedException{

    Thread.sleep(second * 1000);

}

public static void sleep2(int seconds){

    sleep(seconds); //UNHANDLED EXCEPTION

}

public static void sleep3(int seconds){

    sleep2(seconds); //UNHANDLED EXCEPTION
}
```

```java
public static void sleep(int seconds) throws InterruptedException{

    Thread.sleep(second * 1000);

}

public static void sleep2(int seconds) throws InterruptedException{

    sleep(seconds);

}

public static void sleep3(int seconds){

    try{
        sleep2(seconds);
    }catch(InterruptedException e){
        //code
    }
}
```

# Creating Custom Exceptions

- We can create our own exception classes by extending the Exception class or one of its subclass.

```
public class HungryException extends RuntimeException{

}


throw new HungryException();
```