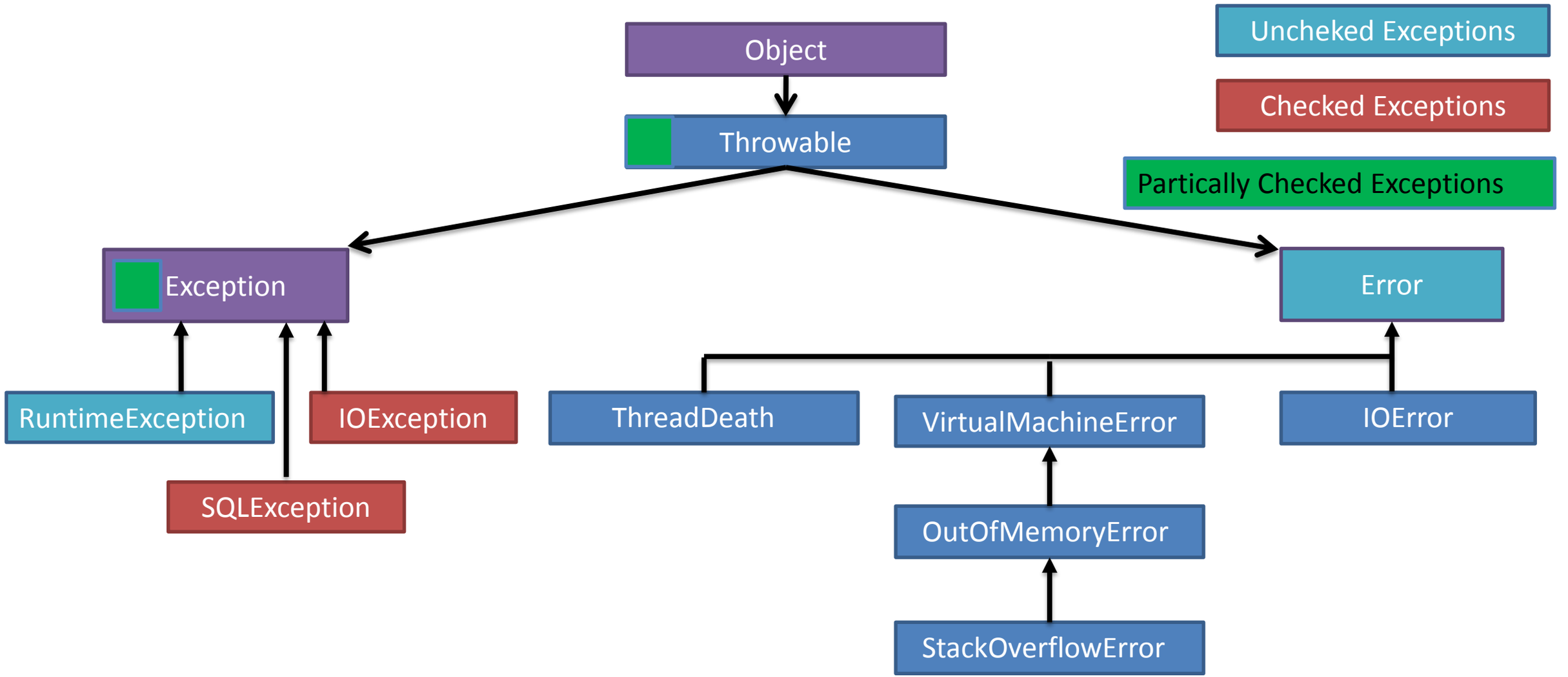
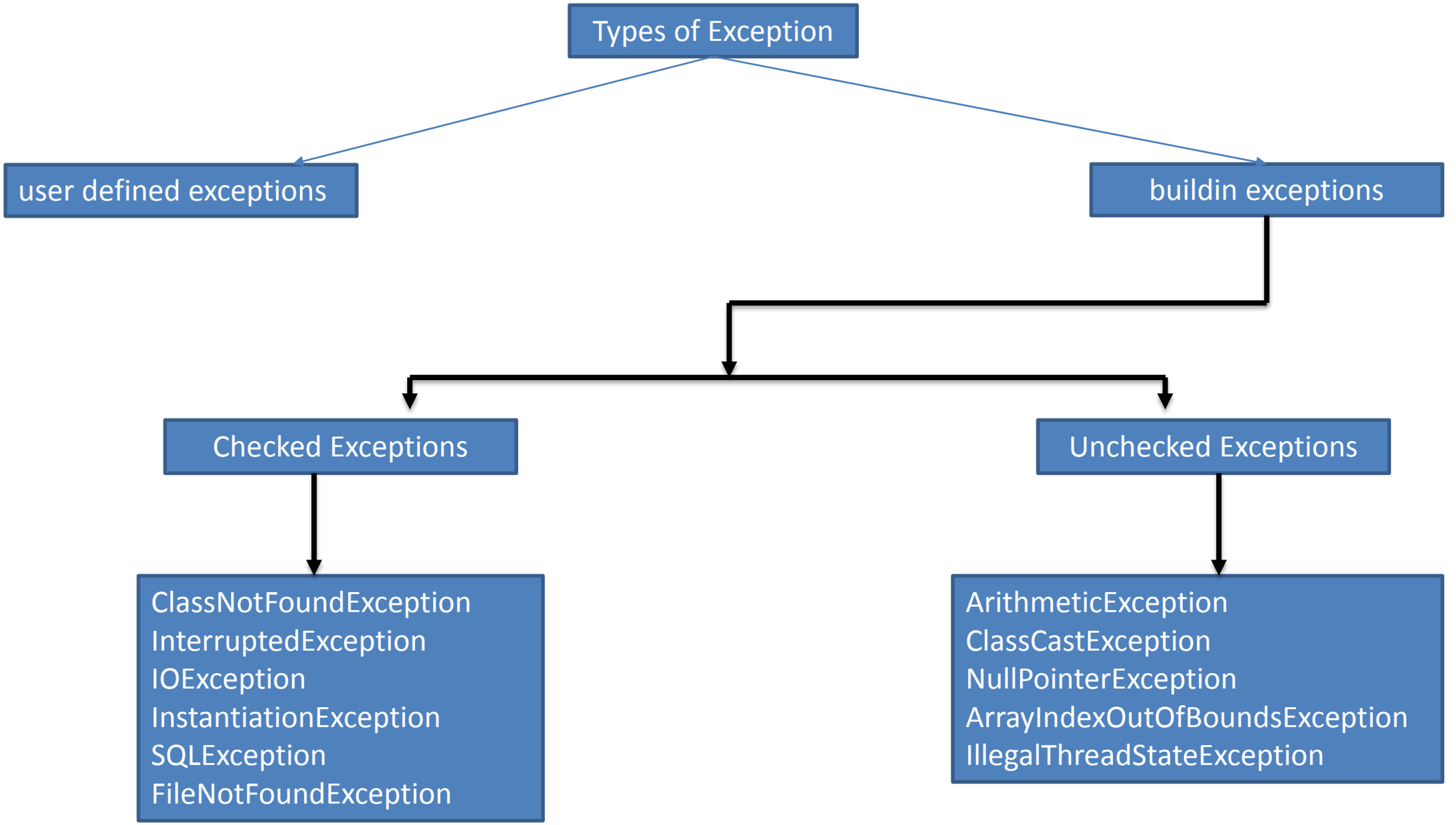
Exception and Exception Hierarchy:

Exception: An unwanted exception, that **disturbs** the normal flow of the program is called **Exception**.




Types of Exception
user defined exceptions
buildin exceptions
Checked Exceptions
Unchecked Exceptions
ClassNotFoundException
InterruptedException
IOException
InstantiationException
SQLException
FileNotFoundException
ArithmeticException
ClassCastException
NullPointerException
ArrayIndexOutOfBoundsException
IllegalThreadStateException

Throwable is root for **Exception Hierarchy**.
Throwable class contains the following **two child classes.**

Exception: Most of the cases Exceptions are caused by our programmers.and these are recoverable.
Ex:FileNotFoundException

Error: Most of the cases **errors** are not caused by programmers these are due to lack of system resources.
Ex:OutOfMemoryError, AssertionError, ExceptionInInitializer, StackOverflowError etc…

Checked Exception and Unchecked Exception

Checked Exception:

The **Exceptions** which are **checked by the compiler** is called **Checked Exception**.

Ex:FileNotFoundException, SQL Exception, Interrupted Exception, No Such Method Exception etc

Unchecked Exception:

The Exceptions which are **not checked by compiler** is called **Unchecked Exception.**
An **UnChecked Exception** is an **exception** that occurs at the **runtime**
These are also called as **Runtime Exceptions.**

Ex:ArithematicException, ArrayIndexOutOfBoundsException, ClassCastException, NullPointerException, NumberFormatException

Partially Checked Exceptions and Fully Checked Exceptions

A checked exception is said to be **Partially checked** if it has both **checked and unchecked** child classes.
Partially Checked Exceptions are **Throwable and Exception.**

Fully Checked Exceptions

A checked exception is said to be fully **checked if and only if all the child classes also checked.**
IOException and **SQLException** are called Fully Checked Exception, Because all their child classes are also checked exceptions

Two different ways to handle exception are explained below:

#1) Using try/catch:

The code is surrounded by **try block.**

If an exception occurs, then it is caught by the **catch block** which is followed by the **try block.**

#2) By declaring throws keyword:

At the end of the method, we can declare the exception using **throws keyword**.

Exception Handling keywords in Java

try block is used for writing business logic if an exception occurs in the **try block,**
it is caught by a **catch block**.
try can be followed either by **catch (or) finally (or) both.**

catch followed by **try block**. **Exceptions are caught here.**

finally is followed either by **try block (or) catch block.**
This block gets executed regardless of an exception.
So **generally clean up codes are provided here.**

throw keyword allows us to throw **checked** or **unchecked** exception

throws is written in method's definition to indicate that method can throw exception.

```
try {
int a = 10;
int b = 0;
System.out.println(a / b);
} catch (ArithmeticException e) {
System.out.println(e);
}
```

```
try {
int [] a = {10,20,30,40,50};
System.out.println(a[5]);
} catch (ArrayIndexOutOfBoundsException e) {
System.out.println(e);
}
```

```
try {
String s1 = "Ten";
int parseInt = Integer.parseInt(s1);
System.out.println(parseInt);
} catch (NumberFormatException e) {
System.out.println(e);
}
```

```
try {
String s1 = null;
System.out.println(s1.length());
} catch (NullPointerException e) {
System.out.println(e);
}
```

```
try {
String s2 = "Hello Java and Hello Python";
System.out.println(s2.charAt(6));
System.out.println(s2.charAt(30));
} catch (StringIndexOutOfBoundsException e) {
System.out.println(e);
}
```

```
package com.dl.one;

class JDBC {

static {
System.out.println("JDBC Class Loaded");
}
}

public class Eg2 {

public static void main(String[] args){

try {
Class.forName("com.dl.one.JDBC");
} catch (ClassNotFoundException e) {
System.out.println(e);
}

}
}
```

```java
//Multiple catch blocks
try {

int a = 10;
int b = 0;
System.out.println(a / b);

int[] i = { 10, 20, 30, 40, 50 };
System.out.println(i[5]);

String s1 = "Ten";
int parseInt = Integer.parseInt(s1);
System.out.println(parseInt);

String s2 = null;
System.out.println(s2.length());

String s3 = "Hello Java and Hello Python";
System.out.println(s3.charAt(6));
System.out.println(s3.charAt(30));

}
```

```java
catch (ArithmeticException e) {
System.out.println(e);
} catch (StringIndexOutOfBoundsException e) {
System.out.println(e);
} catch (IndexOutOfBoundsException e) {
System.out.println(e);
} catch (NumberFormatException e) {
System.out.println(e);
} catch (NullPointerException e) {
System.out.println(e);
}
```

throw keyword allows us to throw **checked** or **unchecked** exception

```
public static void main(String[] args) {
getCustName("Admin");

}

public static void getCustName(String name) {
if (name != "Admin") {
try {
throw new Exception("Execution failed due to Names Mistake");
} catch (Exception e) {
System.out.println(e);
}
} else {
System.out.println(name);
}
}
```

throws is written in method's definition to indicate that method can throw exception.

```
public static void main(String[] args) throws IOException {

File f = new File("one.txt");
f.createNewFile();
System.out.println("File Created");
FileOutputStream stream = new FileOutputStream(f);
String s1 = "Hello Java";
byte[] bytes = s1.getBytes();
stream.write(bytes);
System.out.println("Data Inserted");
stream.close();
}
```

```
//When catch and finally block both use return keyword ,
//method will ultimately return value returned by finally block irrespective of value returned by catch block.

public static void main(String[] args) {
System.out.println(sum());
}

static String sum() {
try {
int i = 10;
System.out.println(i);
} catch (ArithmeticException e) {
return "catch block";
} finally {
return "finally block";
}

}
```

```
//When try and finally block both use return keyword,
//method will ultimately return value returned by finally block irrespective of value returned by try block.

public static void main(String[] args) {

System.out.println(sum());
}

static String sum() {
try {
int i = 10;
System.out.println(i);
return "try block";
} finally {
return "finally block";
}

}
```

```
//Exception chaining
//When Exception is thrown we catch it and throws some other Exception than the concept is called Exception chaining in java.
public static void main(String[] args) {
new Eg5().m1();
}

public void m1() {
m2();
}

public void m2() {
m3();
}

public void m3() {
try {
throw new NullPointerException();
} catch (RuntimeException e) {
throw new RuntimeException("RuntimeException occured", e);
}
}
```

```
//finally is not executed when System.exit is called in java.

try {
System.out.println("in try block");
System.exit(0);
} finally {
System.out.println("finally block executed");
}
```

```
try {
/* Infinite for loop */
for (;;) {
System.out.println("in try block - Infinite for loop");
}
} finally {
System.out.println("finally block executed");
}
```