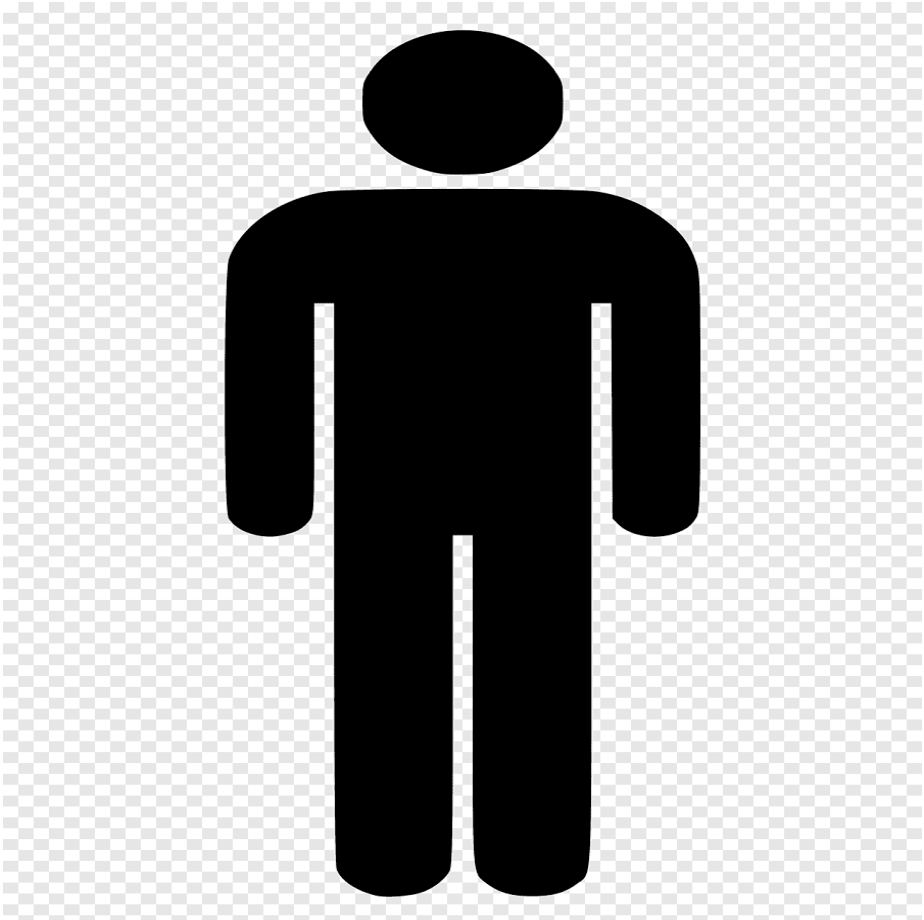
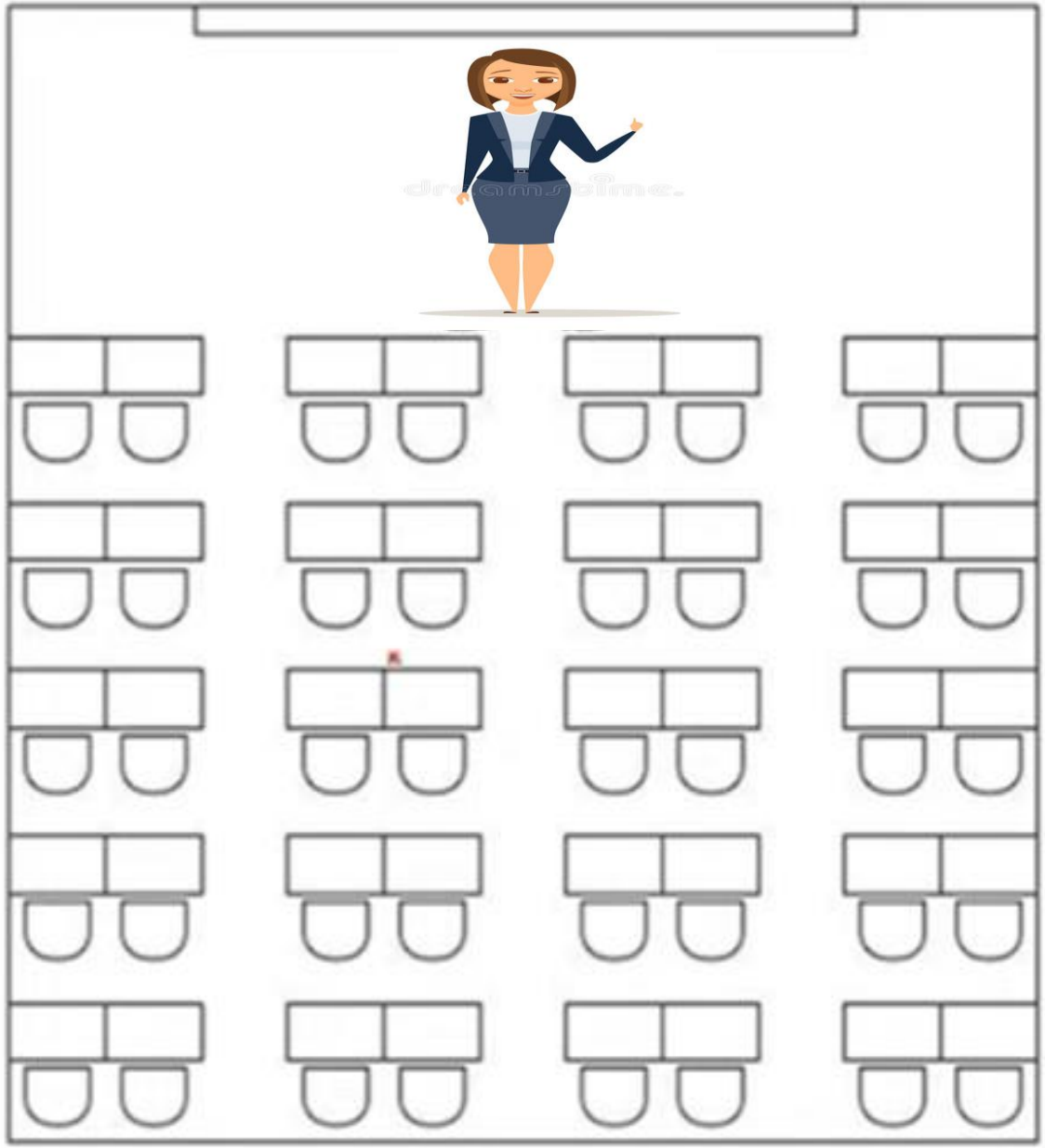
EXCEPTION HANDLING

EXCEPTION HANDLING

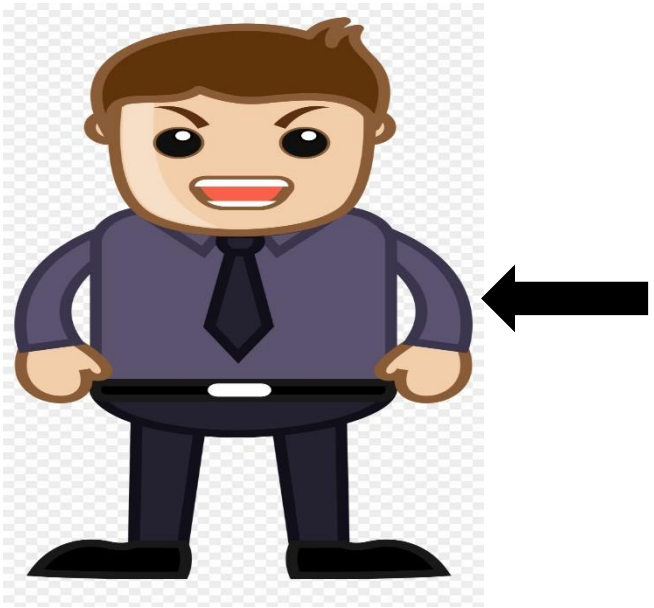
Exception is anything that does not follow the rule

Thrower

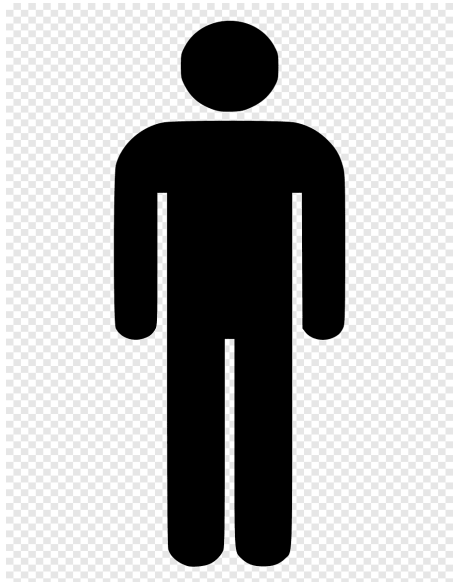


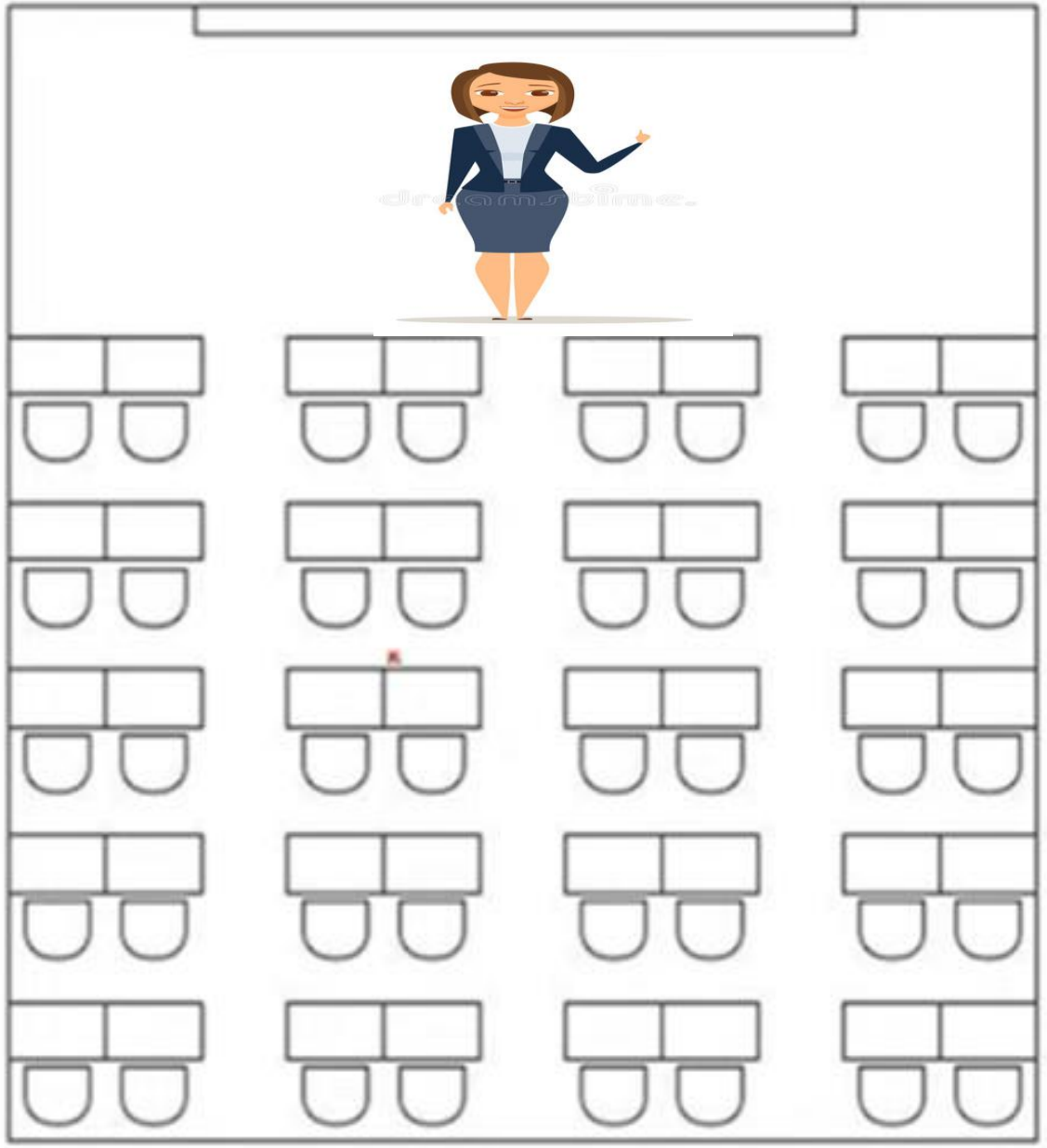


Mobile Phone Catcher

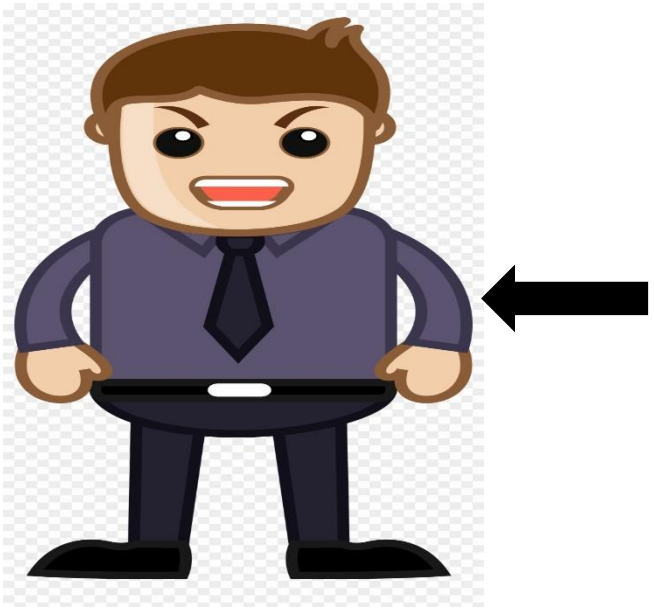


Thrower

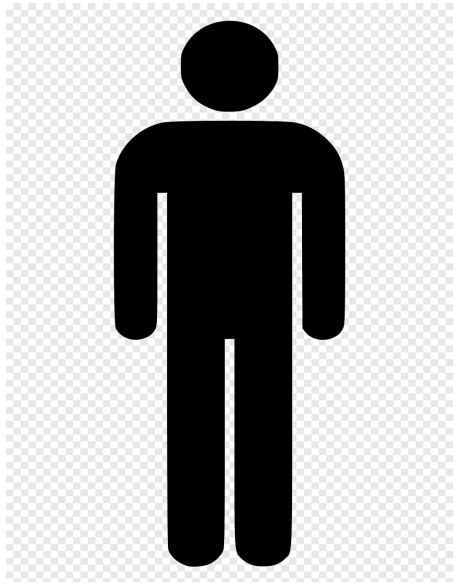


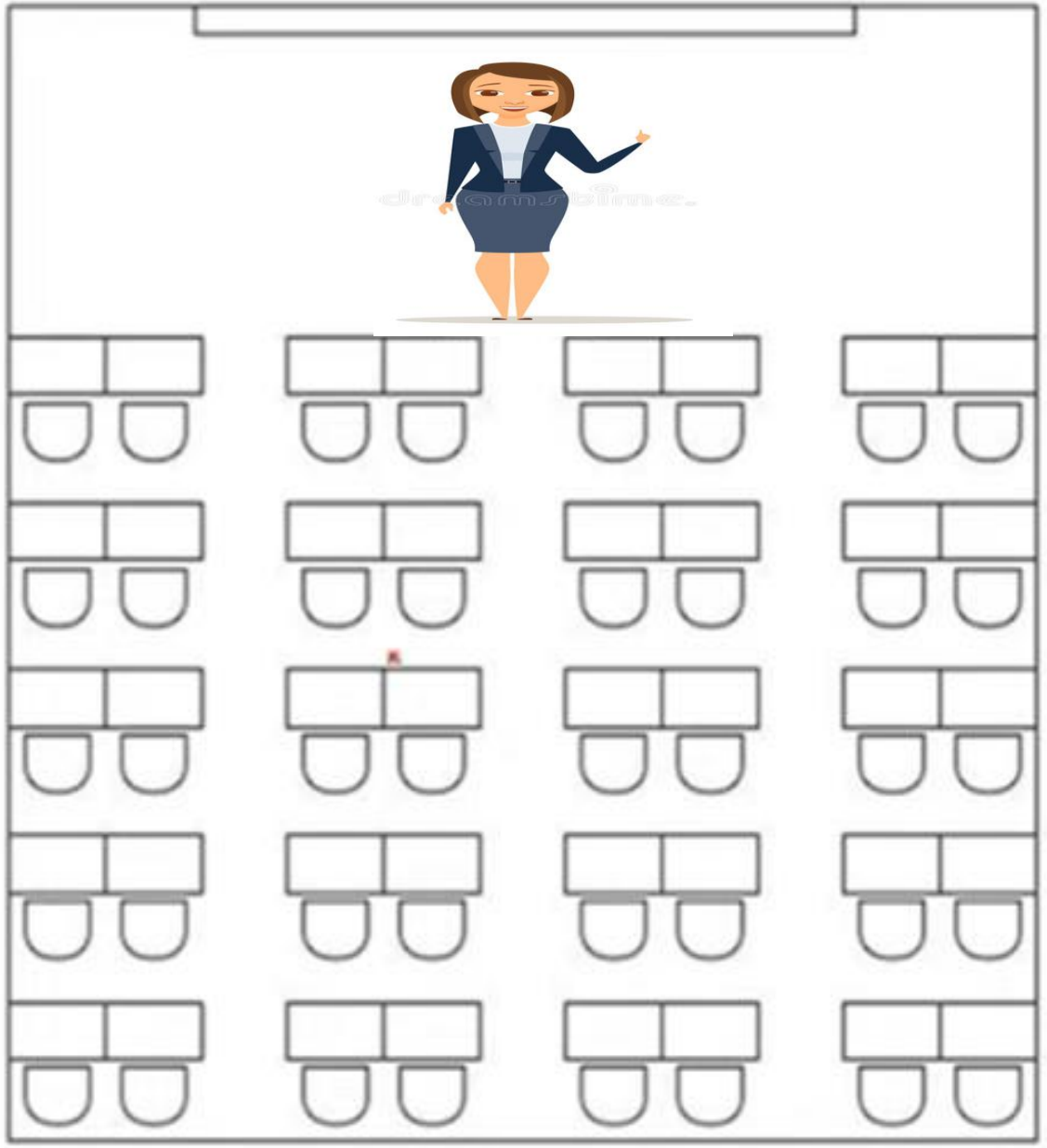


Mobile Phone Catcher

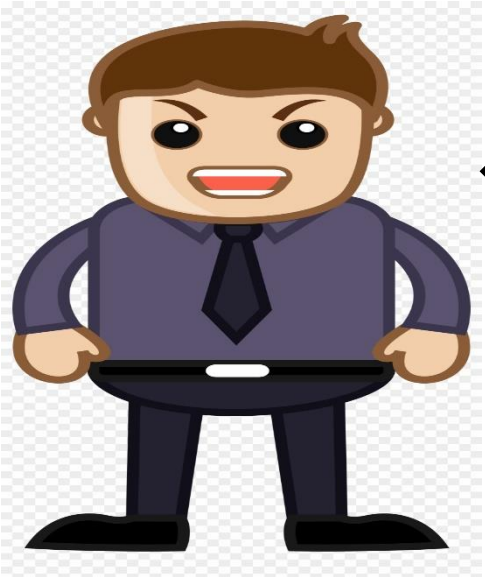


Thrower

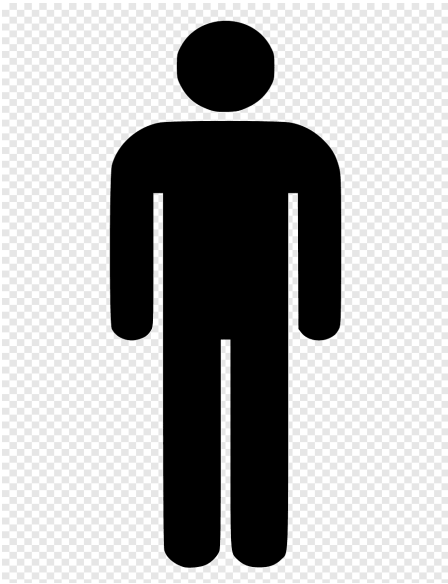


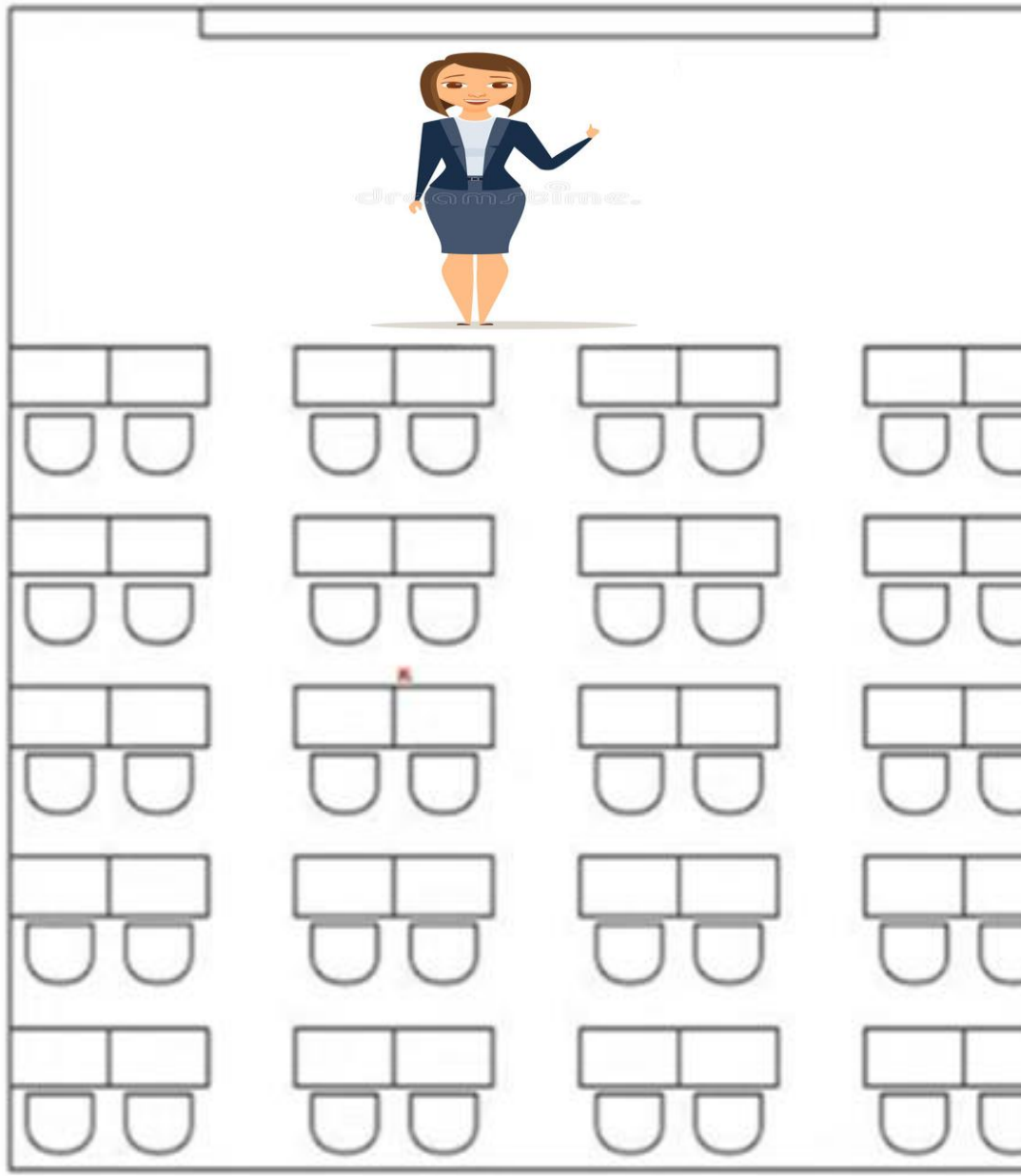


Mobile Phone Catcher



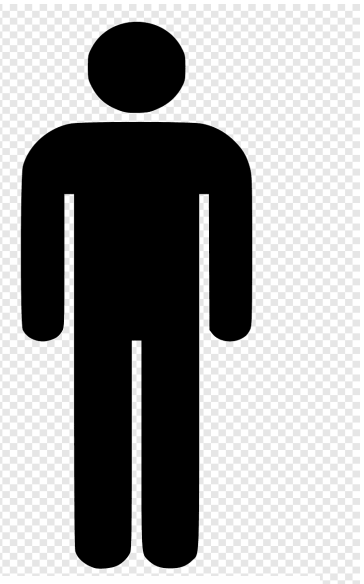
Thrower



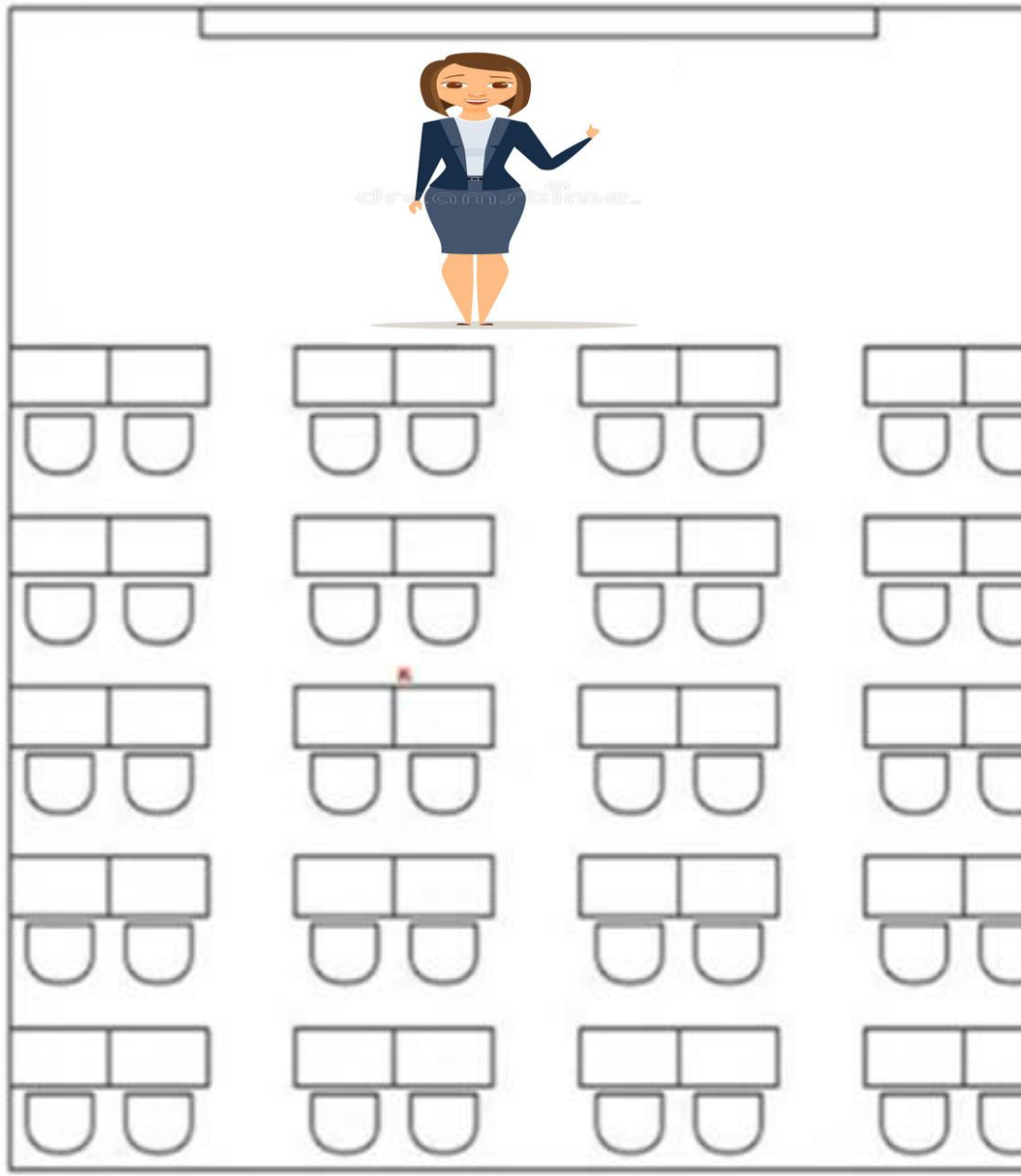


Mobile Phone Catcher

Thrower



Late Comer Catcher



ADVANTAGES OF THE APPROACH

- *Preventing issues from overruling your main stream activities*
- *Separating the main stream activities from issue handling process*
- *Reusing the same exception handlers*

EXCEPTION IN PROGRAMS??

- ***Any problem that may occur during the execution of a program.***
- ***Your program can encounter abnormal situations during run time***

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

What is Exception Handling?

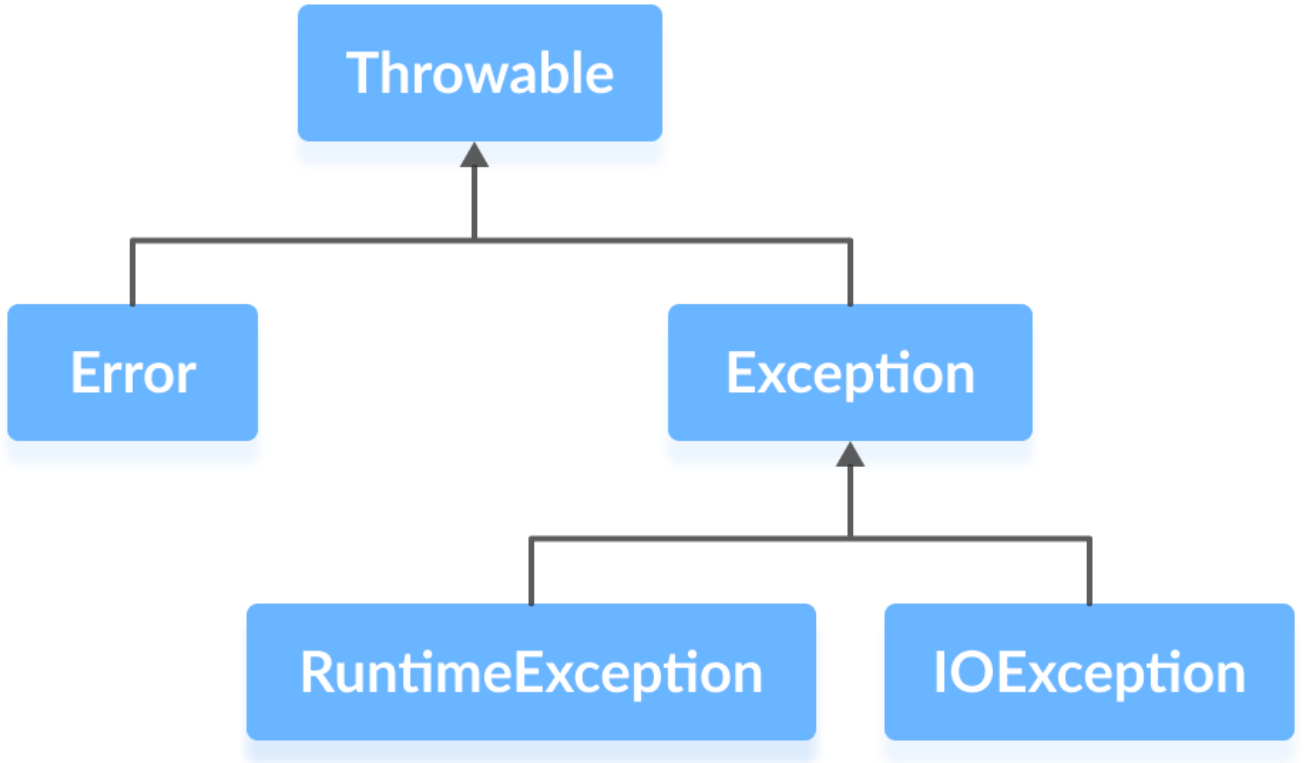
Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

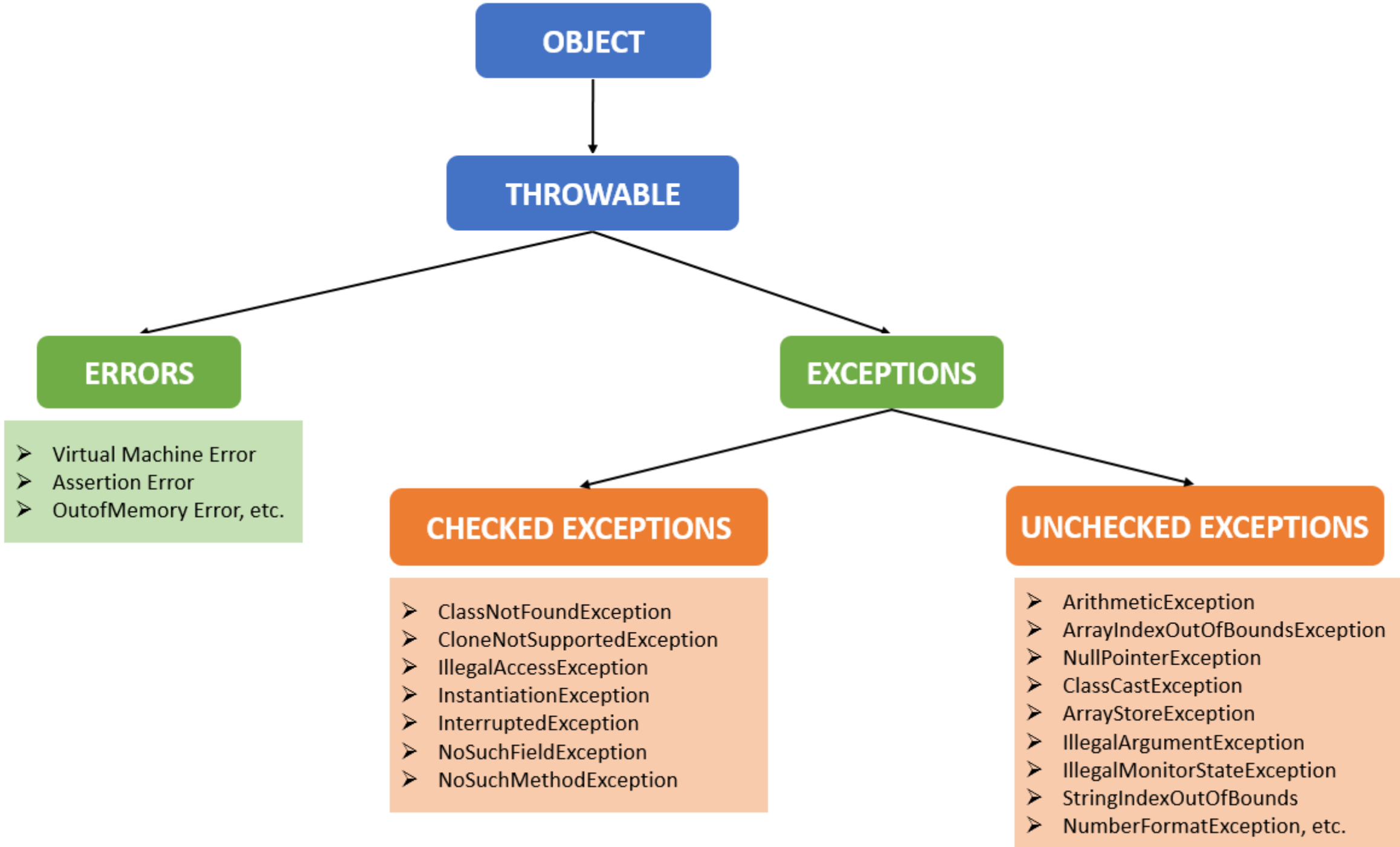
Dictionary Meaning: Exception is an abnormal condition.

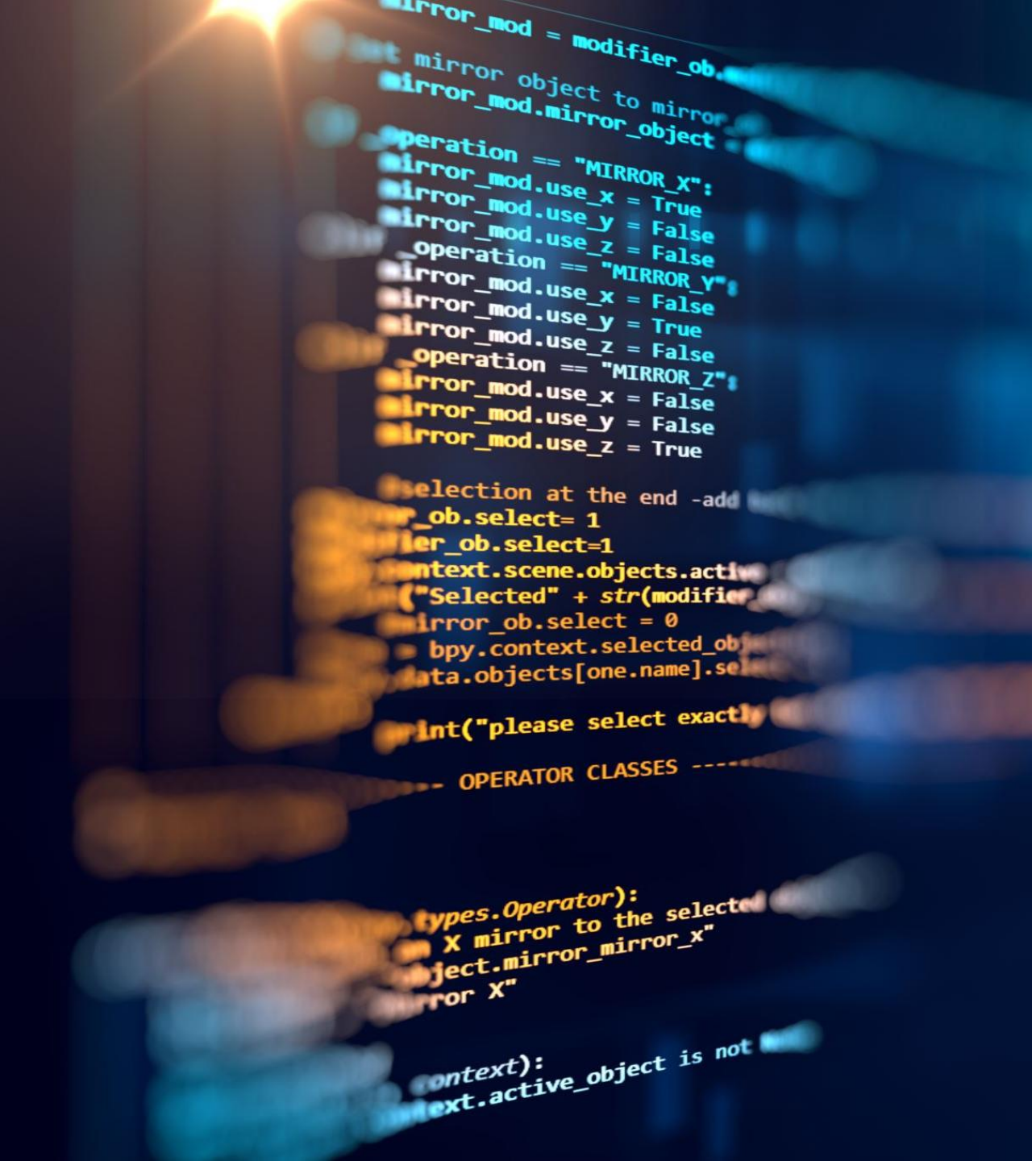
To avoid abnormal termination of a program.

Java Exception hierarchy

Here is a simplified diagram of the exception hierarchy in Java.




OBJECT
THROWABLE
ERRORS
Virtual Machine Error
Assertion Error
OutofMemory Error, etc.
EXCEPTIONS
CHECKED EXCEPTIONS
ClassNotFoundException
CloneNotSupportedException
IllegalAccessException
InstantiationException
InterruptedException
NoSuchFieldException
NoSuchMethodException
UNCHECKED EXCEPTIONS
ArithmeticException
ArrayIndexOutOfBoundsException
NullPointerException
ClassCastException
ArrayStoreException
IllegalArgumentException
IllegalMonitorStateException
StringIndexOutOfBounds
NumberFormatException, etc.



• Errors

• Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc.

• Errors are usually beyond the control of the programmer and we should not try to handle errors.

Exceptions

- Exceptions can be caught and handled by the program.
- When an exception occurs within a method, it creates an object. This object is called the exception object.
- It contains information about the exception such as the name and description of the exception and state of the program when the exception occurred.

Java Exception Types

The exception hierarchy also has two branches: RuntimeException and IOException.

1. RuntimeException

A **runtime exception** happens due to a programming error. They are also known as **unchecked exceptions**.

These exceptions are not checked at compile-time but run-time. Some of the common runtime exceptions are:

- Improper use of an API - `IllegalArgumentException`
- Null pointer access (missing the initialization of a variable) - `NullPointerException`
- Out-of-bounds array access - `ArrayIndexOutOfBoundsException`
- Dividing a number by 0 - `ArithmeticException`

You can think about it in this way. "If it is a runtime exception, it is your fault".

The `NullPointerException` would not have occurred if you had checked whether the variable was initialized or not before using it.

An `ArrayIndexOutOfBoundsException` would not have occurred if you tested the array index against the array bounds.

2. IOException

An `IOException` is also known as a **checked exception**. They are checked by the compiler at the compile-time and the programmer is prompted to handle these exceptions.

Some of the examples of checked exceptions are:

- Trying to open a file that doesn't exist results in `FileNotFoundException`
- Trying to read past the end of a file

We learned about Java exceptions. We know that exceptions abnormally terminate the execution of a program.

This is why it is important to handle exceptions.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions. Let's consider a scenario:

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

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

- List of different approaches to handle exceptions in Java.

- try...catch block
- finally block
- throw and throws keyword

Try

Throw

Catch

Throws

Finally

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

```
Try
{
//code that may throw an exception
}
catch(Exception_class_Name ref)
{
}
```

```
Try
{
//code that may throw an exception
}
catch(Exception 1)
{
}
catch(Exception 2)
{
}
```

Syntax of try-finally block

1.try{

2.//code that may throw an exception

3.}**finally**

4.{

5.}

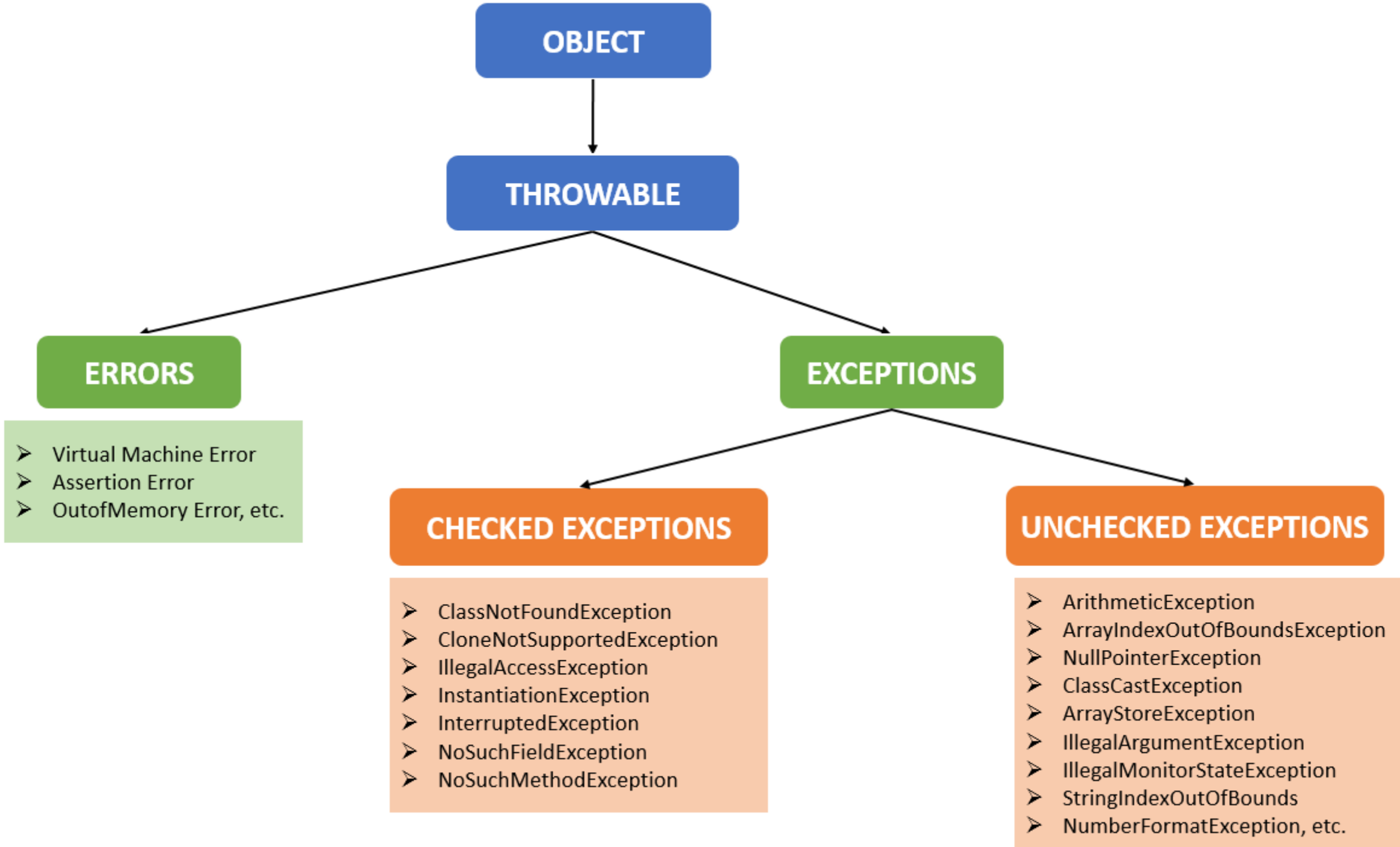
Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1.Checked Exception

2.Unchecked Exception

3.Error


OBJECT
THROWABLE
ERRORS
Virtual Machine Error
Assertion Error
OutofMemory Error, etc.
EXCEPTIONS
CHECKED EXCEPTIONS
ClassNotFoundException
CloneNotSupportedException
IllegalAccessException
InstantiationException
InterruptedException
NoSuchFieldException
NoSuchMethodException
UNCHECKED EXCEPTIONS
ArithmeticException
ArrayIndexOutOfBoundsException
NullPointerException
ClassCastException
ArrayStoreException
IllegalArgumentException
IllegalMonitorStateException
StringIndexOutOfBounds
NumberFormatException, etc.

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions.

For example,

- IOException,
- SQLException,
- Class not found Exception

Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions.

For example,

- ArithmeticException,
- NullPointerException,
- ArrayIndexOutOfBoundsException, etc.

Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

ArithmeticException

1.int a=50/0; //ArithmeticException

```
public class ArithmeticExceptionExample
{
   public static void main(String[] args) {
      try
{
         int result = 10 / 0;
         System.out.println("Result is: " + result);
      }
      catch (ArithmeticException e)
       {
         System.out.println("Caught an arithmetic exception");
         e.printStackTrace();
      }
   } }
```

NullPointerException

If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

```
1.String s=null;
2.System.out.println(s.length());//NullPointerException
```

```
public class NullPointerExceptionExample {
    public static void main(String[] args) {
        // Declare a string variable and initialize it to null.
        String text = null;

        // Try to call the length() method on the string reference.
        try {
            int length = text.length();
            System.out.println("Length of the string: " + length);
        } catch (NullPointerException e) {
            // Catch the NullPointerException and print a message.
            System.out.println("Caught a NullPointerException.");
            e.printStackTrace();
        }
    }
}
```

NumberFormatException

If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a string variable that has characters; converting this variable into digit will cause NumberFormatException.

1.String str=“Java Programming";

2.int i=Integer.parseInt(str);//NumberFormatException

```
public class NumberFormatExceptionExample {
    public static void main(String[] args) {
        String number = "10a"; // This is not a purely numeric string.

        try {
            // This line will throw NumberFormatException because "10a" is
not a valid integer.
            int result = Integer.parseInt(number);
            System.out.println("The number is: " + result);
        } catch (NumberFormatException e) {
            // This block will catch the NumberFormatException.
            System.out.println("NumberFormatException occurred: " +
e.getMessage());
        }
    }
}
```

ArrayIndexOutOfBounds Exception

When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException. Consider the following statements.

```
1.int a[]=new int[5];

2.a[6]=50; //ArrayIndexOutOfBoundsException
```