# Day-9

Static Data

The data which is hard coded. Int a = 10; //Hardcoding the data is not advisable.

Dynamic Data

Data that the user can defined as per his requirements

Data from console (Command Line Argument)

Cmd > java FileName arg1 arg2 data1 data2 12

Class ClassName {

Public static void main( String[] args){

SOP(args[0]) // arg1

SOP(args[1]) // arg2

SOP(args[4]) // 12

}

}

Every data from the command line is of String data type. To convert string data type to primitive data type, we use the wrapper class.

Primitive Data type Wrapper Class

Int Integer

float Float

byte Byte

boolean Boolean

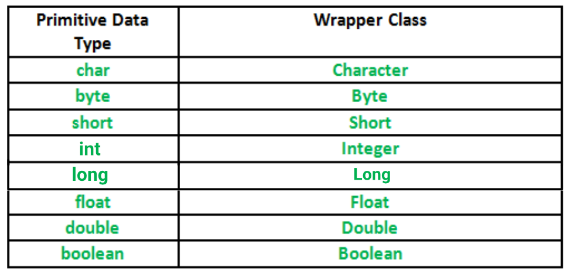
## Wrapper Classes

A Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

**Need of Wrapper Classes**

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/) and [Vector](https://www.geeksforgeeks.org/vector-vs-arraylist-java/), store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

**Primitive Data types and their Corresponding Wrapper class**

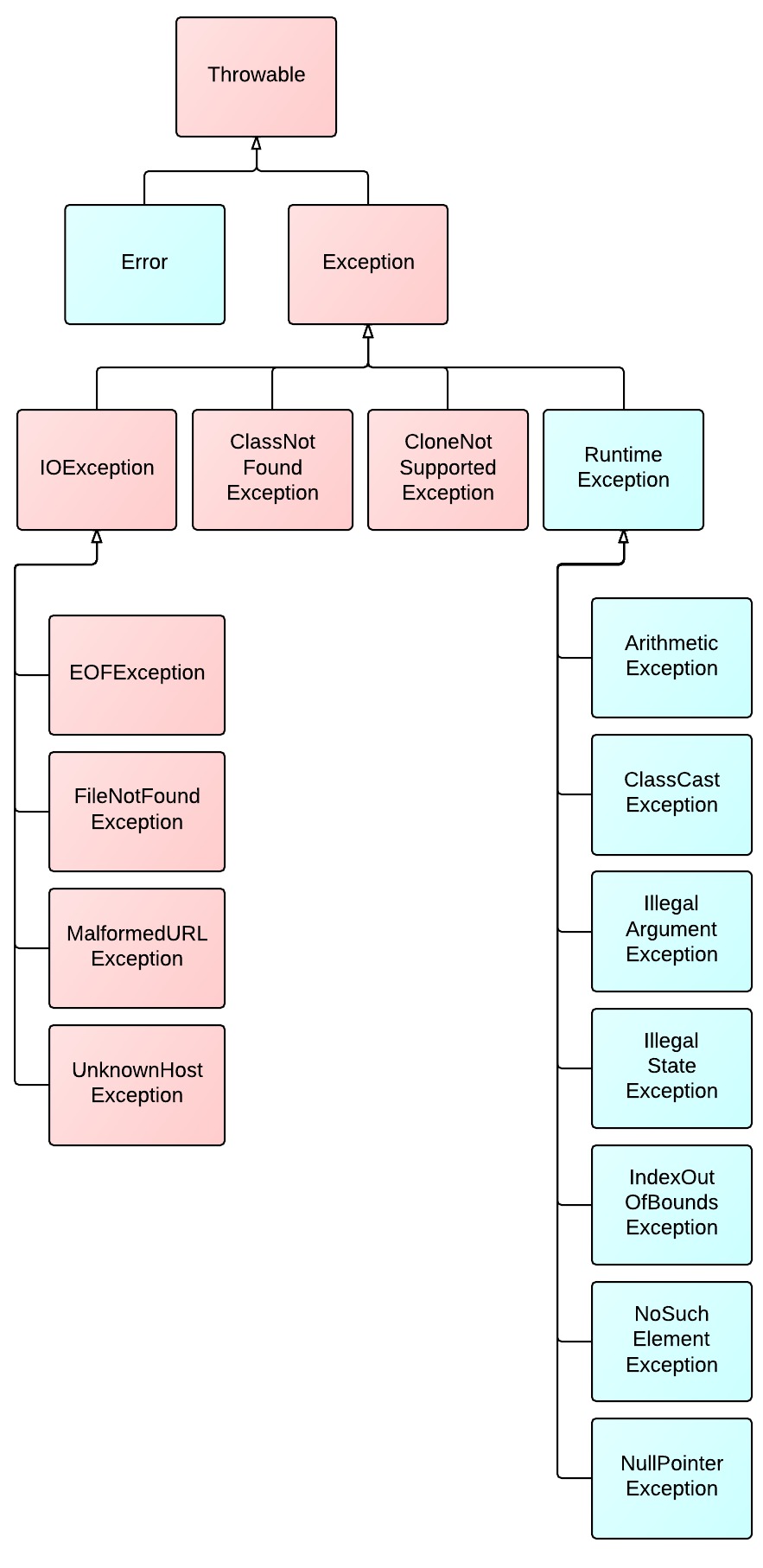


**Autoboxing and Unboxing**

**Autoboxing:** Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

Exceptions

Object



# Exceptions in Java

Exception Handling in Java is one of the effective means to handle the runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

#### What is an Exception?

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program’s instructions. 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 the state of the program when the exception occurred.

**An exception can occur for many reasons. Some of them are:**

* Invalid user input
* Device failure
* Loss of network connection
* Physical limitations (out of disk memory)
* Code errors
* Opening an unavailable file

#### What is an Error?

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.

### Error vs Exception

* **Error:**An Error indicates a serious problem that a reasonable application should not try to catch.
* **Exception:**Exception indicates conditions that a reasonable application might try to catch.

### Exception Hierarchy

All exception and errors types are subclasses of class **Throwable**, which is the base class of the hierarchy. One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception. Another branch, **Error** is used by the Java run-time system([JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.

**Exceptions can be Categorized into 2 Ways:**

1. Built-in Exceptions
   * Checked Exception
   * Unchecked Exception
2. User-Defined Exceptions

**1. Built-in Exceptions:**Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.

* **Checked Exceptions:**Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.
* **Unchecked Exceptions:**The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn’t handle or declare it, the program would not give a compilation error.

***Note:****For checked vs unchecked exception, see*[*Checked vs Unchecked Exceptions*](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/)

**2. User-Defined Exceptions:**Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions which are called ‘user-defined Exceptions’.

*The advantages of Exception Handling in Java are as follows:*

* Provision to Complete Program Execution
* Easy Identification of Program Code and Error-Handling Code
* Propagation of Errors
* Meaningful Error Reporting
* Identifying Error Types

### How does JVM handle an Exception?

**Default Exception Handling:**Whenever inside a method, if an exception has occurred, the method creates an Object known as Exception Object and hands it off to the run-time system(JVM). The exception object contains the name and description of the exception and the current state of the program where the exception has occurred. Creating the Exception Object and handling it to the run-time system is called throwing an Exception. There might be a list of the methods that had been called to get to the method where an exception occurred. This ordered list of the methods is called **Call Stack**. Now the following procedure will happen.

* The run-time system searches the call stack to find the method that contains a block of code that can handle the occurred exception. The block of the code is called an **Exception handler**.
* The run-time system starts searching from the method in which the exception occurred, proceeds through the call stack in the reverse order in which methods were called.
* If it finds an appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.
* If the run-time system searches all the methods on the call stack and couldn’t have found the appropriate handler then the run-time system handover the Exception Object to the **default exception handler**, which is part of the run-time system. This handler prints the exception information in the following format and terminates the program **abnormally**.

Exception in thread "xxx" Name of Exception : Description

... ...... .. // Call Stack