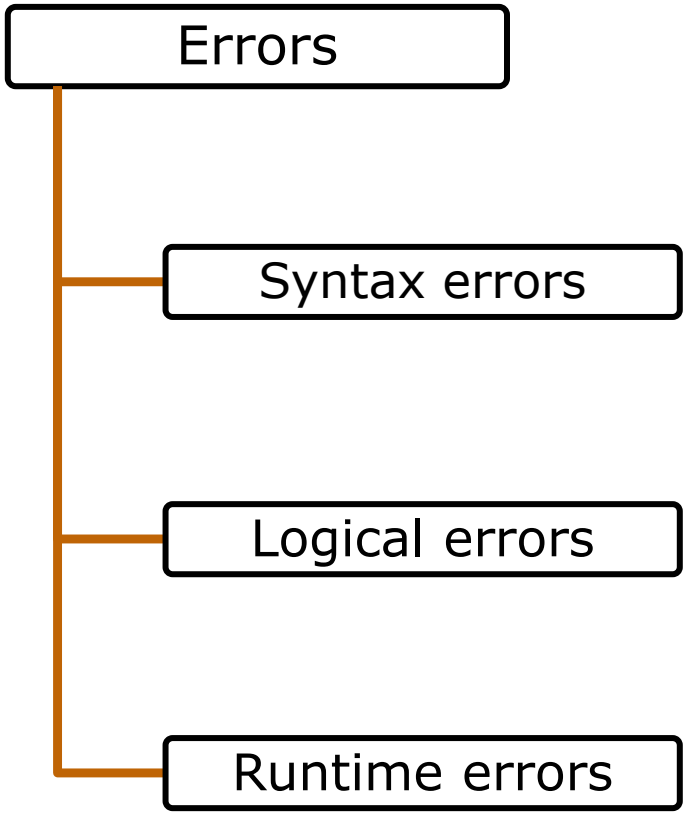
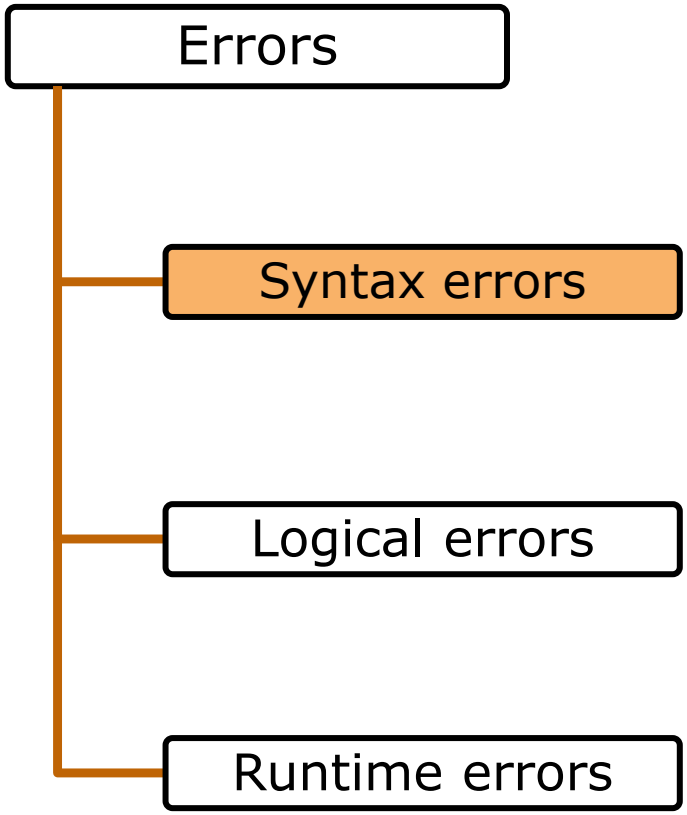
Exception Handling

Types of Errors

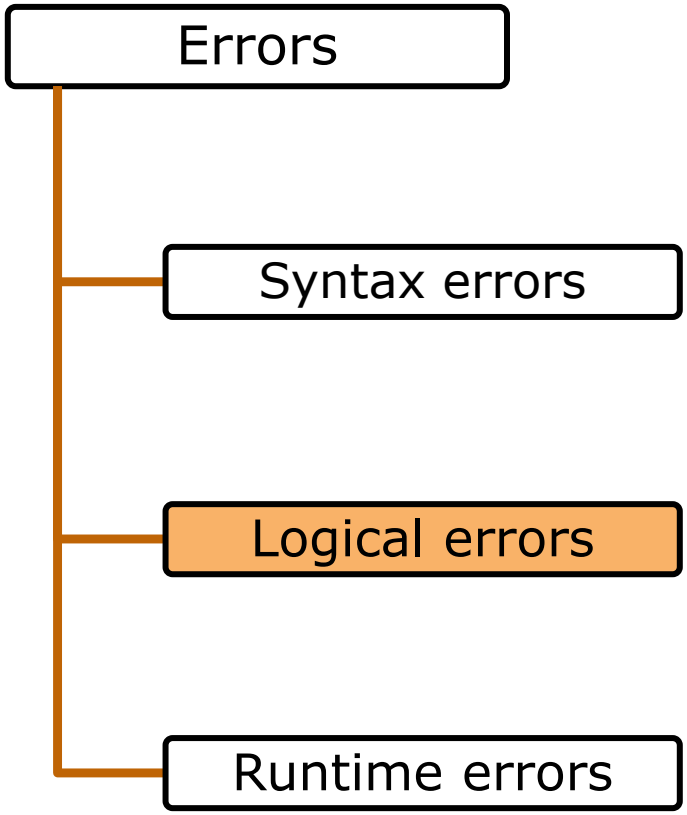


Types of Errors



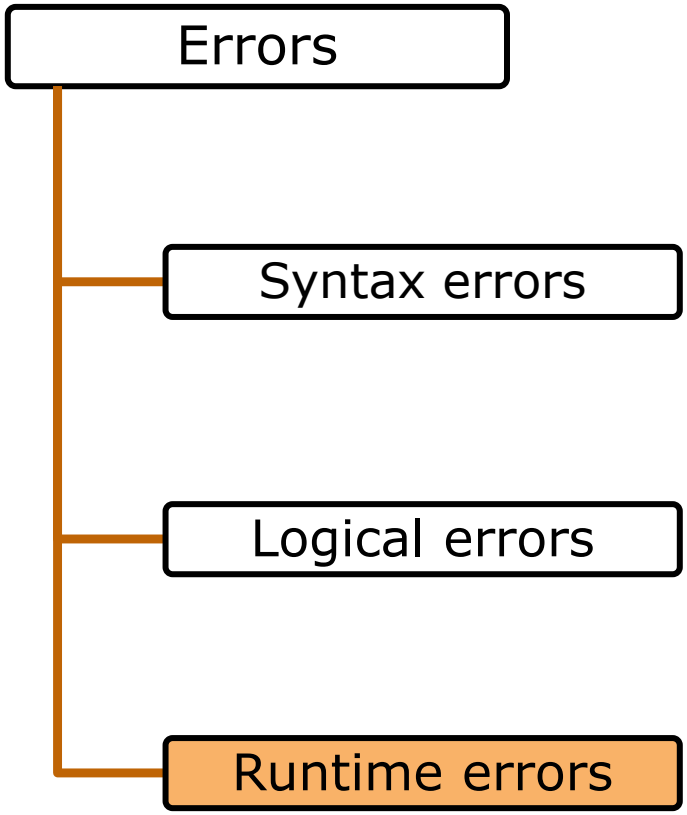
Occurs when statements are not constructed properly, keywords are misspelled, or punctuation is omitted.

Types of Errors



Occurs when an application compiles and runs properly but does not produce the expected results

Types of Errors



Occurs when an application attempts to perform an operation, which is not allowed at runtime.

Exceptions

An exception is an erroneous situation or an abnormal event that occurs during program execution and disrupts the normal flow of instructions.

When the Java interpreter comes across an exception, it creates an exception object and throws it i.e., informs that an exception has occurred.

If the exception is not caught and handled properly, the interpreter will display an error message and will terminate the program.

In order to make the program to continue successfully, then we should try to catch the exception object and handled it properly. This is known as **Exception Handling**.

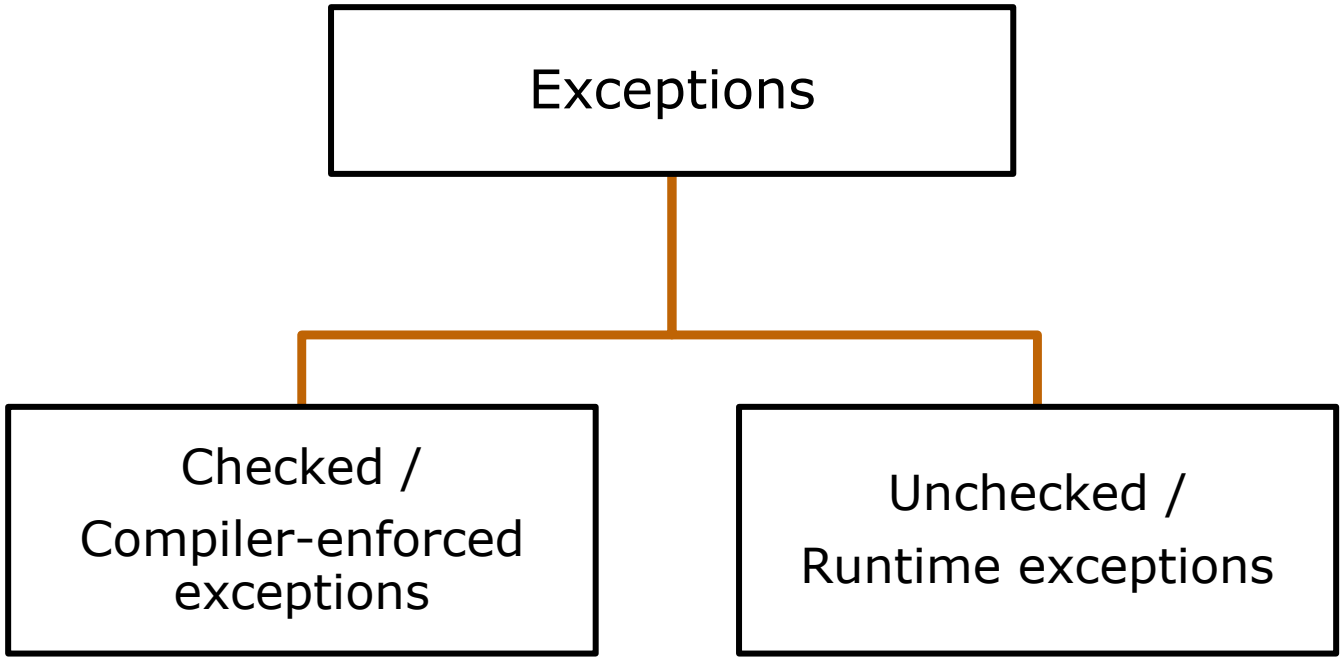
Objective

The purpose of Exception Handling mechanism is to provide a means to detect and report an exceptional circumstance so that appropriate action can be taken.

It performs the following tasks:

- Finds the Problem
- Inform that error has occurred (throws an exception object)
- Receive the error information (catches the exception object)
- Take the corrective actions (handles the exception)

Types of Exceptions



Checked Exceptions

Checked exceptions are those that the programmer is expected to handle in the program, otherwise your program will not be compiled.

There are two ways to handle checked exceptions.

- You may use the **try..catch** block (or)
- At least, use **throws** clause

Checked Exceptions

Exception	Cause of Exception
IOException	This exception is thrown when there is an error in input-output operation.
ClassNotFoundException	Occurs when the Java run time system is unable to find the class referred.
IllegalAccessException	Occurs when you want to refer a class that is not accessible.
InstantiationException	Occurs when you try to create an object of an abstract class or interface.
NoSuchMethodException	Occurs when you call a method that does not exist.

Unchecked Exceptions

Unchecked exceptions might arise from conditions that represent bugs, or situations that are considered generally too difficult for a program to handle reasonably.

They are *unchecked* because you are not required to check for them or do anything about them if they occur.

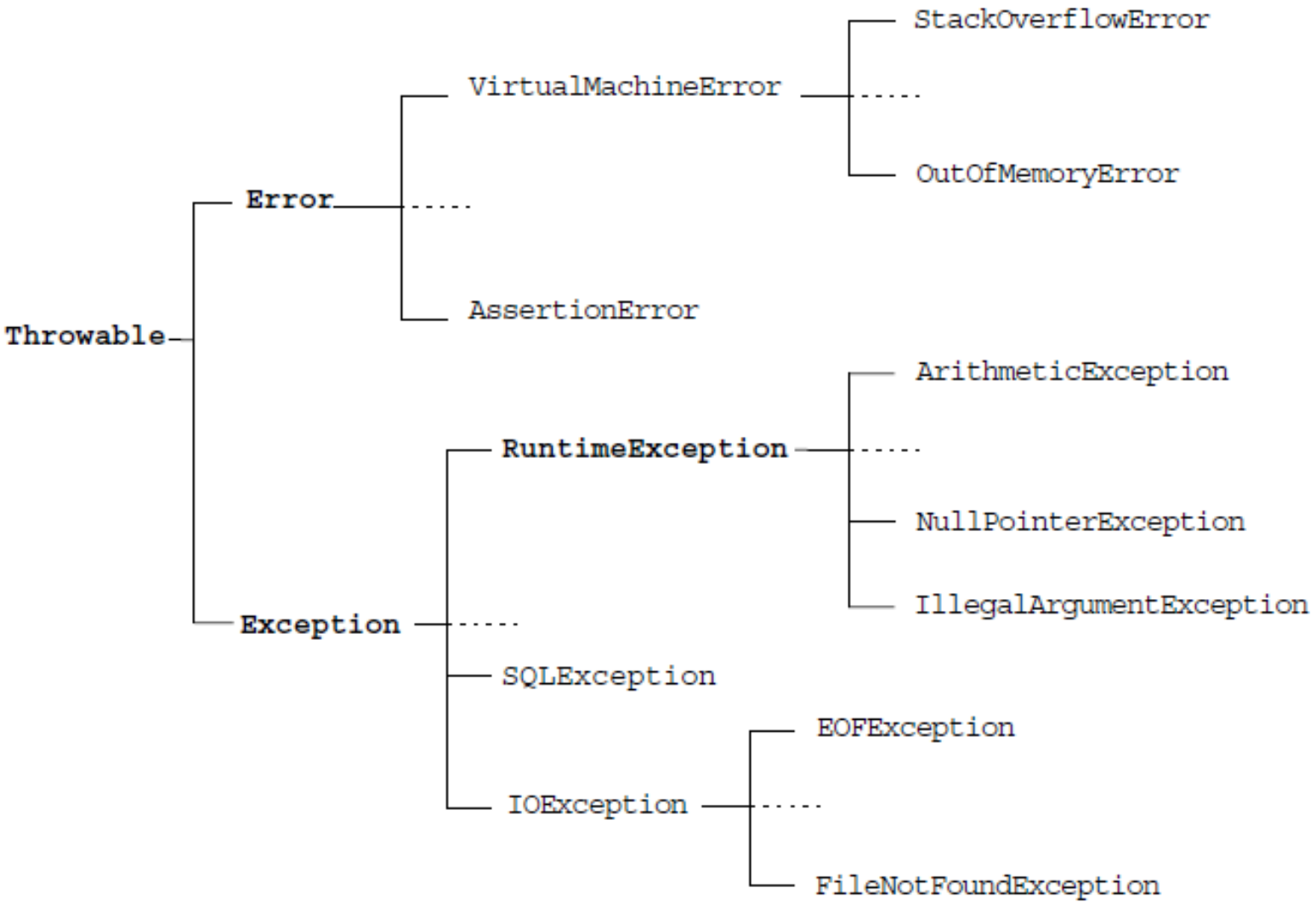
Runtime exceptions

Exceptions that arise from a category of situations that probably represents bugs are called *runtime exceptions*. An example of this is attempting to access beyond the end of an array.

Errors

Exceptions that arise as a result of environmental issues that are rare enough or hard enough to recover from are called *errors;* an example of an error is running out of memory.

Exception Categories



Exception Class

The Exception class is the base class that represents checked and unchecked exceptions. Rather than terminating the program, it is best if you write code to handle the exception and continue.

Error Class

The Error class is the base class used for the unchecked, serious error conditions from which your program is not expected to attempt recovery. In most cases, you should let the program terminate when you encounter this type of error.

Runtime Exception

The Runtime Exceptions class is the base class that is used for the unchecked exceptions that might arise as a result of a program bugs.

Runtime Exceptions

Exception	Reason for Exception
ArithmeticException	When you divide a number by zero
NullPointerException	When you try to implement an application without referencing the object and allocating to a memory
ClassCastException	When you try to assign a reference variable of a class to an incompatible reference variable of another class
ArrayStoreException	When you assign an array which is not compatible with the data type of that array
ArrayIndexOutOfBoundsException	When an attempt is made to access an array element beyond the index of the array.
NumberFormatException	When you try to convert a string variable in an incorrect format to integer (numeric format)
NegativeArraySizeException	When you declare an array of negative size.
SecurityException	Access the local file system, Open a socket to a host that is not the same host that served the applet, etc,

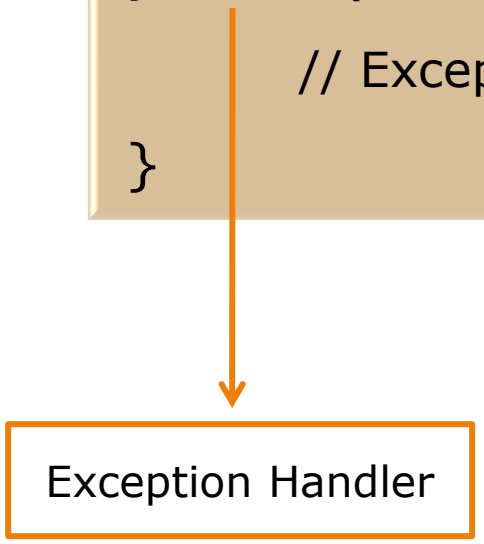
Implementing Exception Handling

You can implement exception-handling in a program by using the following keywords:

- try
- catch
- throw
- throws
- finally

The `try - catch` Statement

```
try {
        // Statements that cause an exception.
} catch ( ExceptionName obj ) {
        // Exception handling code.
}
```



Using Multiple `catch` Clauses

```
try {
        // Statements that cause an exception.
} catch ( MyException e1 ) {
        // code to execute if a MyException is thrown
} catch ( MyOtherException e2 ) {
        // code to execute if a MyOtherException is thrown
} catch ( Exception e3 ) {
        // code to execute if any other exception is thrown
}
```

Points to Remember

- The Exception class is the super-class of all the exception classes.
- So, make sure that the last catch block in multiple catch blocks contain the Exception class object.
- If the first catch block contains the Exception class object then the subsequent catch blocks are never executed and leads to compile-time error.
- This is known as the unreachable code problem

Call Stack Mechanism

If a statement throws an exception, and that exception is not handled in the immediately enclosing method, then that exception is thrown to the calling method.

If the exception is not handled in the calling method, it is thrown to the called of that method. This process continues.

If the exception is still not handled by the time it gets back to the main() method and main() method does not handle it, the exception terminates the program abnormally.

Using `finally` Clause

The finally clause defines a block of code that always executes, no matter whether an exception was caught or not.

```
try {
        // Statements that cause an exception.
} catch ( MyException e1 ) {
        // code to execute if a MyException is thrown
} finally {
        // code that always execute
}
```

Note

- The try clause must have at least one catch block or finally block
- The only situations that prevent the finally clause executing are virtual machine shutdown (for example **System.exit** method is executed)

Using `throw` Clause

Before you can catch an exception, some code somewhere must throw an exception.

If it is a pre-defined exception, then JVM creates an object of that Exception class and throws it automatically.

If it is an User-Defined exception, then you have to create an User-Defined Exception object and throw it.

You can also create your own exception classes (User-Defined Exceptions) to represent problems that can occur within the classes you write.

Using `throw` Clause

```
try {

        // create an Object of User-Defined Exception
        throw obj;

} catch ( User-DefinedException obj ) {
        // Exception handling code.
 }
```

Using `throws` Clause

The throws statement is used by a method to specify the types of exceptions the method throws.

If a method is capable of raising an exception that it does not handle, the method must specify that the exception has to be handled by the calling method.

```
class  Demo {
    public static void main(String args[]) throws IOException{
        // Statements that cause IOException
    }
}
```

The Nested try-catch Block

- You can enclose a try-catch block in an existing try-catch block.
- The enclosed try-catch block is called the inner try-catch block, and the enclosing block is called the outer try-catch block.
- If the inner try block does not contain the catch statement to handle an exception then the catch statement in the outer block is checked for the exception handler.

```
class Demo{
        public static void main(String a[]){
                // Statements
                try{
                        // Statements
                        try{
                                // Statements
                        }catch(ExceptionName obj){
                                // Error handling statements
                        }
                // Statements
                } catch(ExceptionName obj){
                        try{
                                 // Error handling statements
                        }catch(ExceptionName obj){
                                // Error handling statements
                        }
                }
        }
}
```

- The following table lists the various methods defined by the Throwable class

Methods	*Description*
String getMessage()	Returns a description of the exception
String toString()	Returns a string object containing a description of the exception.
Throwable fillInStackTrace()	Returns a Throwable object that contains a stack trace
void printStackTrace()	Prints the stack trace

Assertions in Java

- Assertions are the checks provided by the Java language to ensure that any assumption made at the start of a program is true throughout the program.
- An assertion statement contains a boolean expression that the programmer believes to be true at the time the statement is executed.
- The following syntax shows how to declare an assertion using assert statement in Java:

```
assert  <condition>;
```

- To enable assertion during execution

```
java –ea AssertionDemo
```

- To disable assertion during execution

```
java –da AssertionDemo
```