Exceptions

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

- Advantages of Exceptions
- java.lang.Throwable
- Types of exceptions
- Catching Exceptions (examples)

Advantages

- Separating Error-Handling Code from "Regular" Code
- Propagating Errors Up the Call Stack
- Grouping and Differentiating Error Types

Separating Error-Handling Code from "Regular" Code

Exceptions provide the means to separate the details of what to do when something out of the ordinary happens from the main logic of a program.

Applications can be created without exceptions, but they will look like spaghetti.

For example, consider the pseudocode method here that reads an entire file into memory.

```
readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

It looks fine, but it ignores many possible errors like:

- What happens if the file can't be opened?
- What happens if the length of the file can't be determined?
- What happens if the read fails?
- What happens if the file can't be closed?

To handle all possible errors by “normal” way we need to change to code to:

```
errorCodeType readFile {
   initialize errorCode = 0;
   open the file;
   if (theFileIsOpen) {
      determine the length of the file;
      if (gotTheFileLength) {
         allocate that much memory;
         if (gotEnoughMemory) {
            read the file into memory;
            if (readFailed) {
               errorCode = -1;
            }
         } else {
            errorCode = -2;
         }
      } else {
         errorCode = -3;
      }
      close the file;
      if (theFileDidntClose && errorCode == 0) {
         errorCode = -4;
      } else {
         errorCode = errorCode and -4;
      }
   } else {
      errorCode = -5;
   }
   return errorCode;
}
```

And the same with exceptions

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

Propagating Errors Up the Call Stack

An example of code with out ability to throws exception in the signature of a method

```
method1 {
   errorCodeType error;
   error = call method2;
   if (error)
      doErrorProcessing;
   else
      proceed;
}

errorCodeType method2 {
   errorCodeType error;
   error = call method3;
   if (error)
      return error;
   else
      proceed;
}

errorCodeType method3 {
   errorCodeType error;
   error = call readFile;
   if (error)
      return error;
   else
      proceed;
}
```

With the ability to automatically re-throw the exception:

```
method1 {
    try {
        call method2;
    } catch (exception e) {
        doErrorProcessing;
    }
}

method2 throws exception {
    call method3;
}

method3 throws exception {
    call readFile;
}
```

Grouping and Differentiating Error Types

```
catch (FileNotFoundException e) {
    ...
}

catch (IOException e) {
    ...
}

catch (Exception e) {
    ...
}
```

java.lang.Throwable

public String getMessage()

Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.

public Throwable getCause()

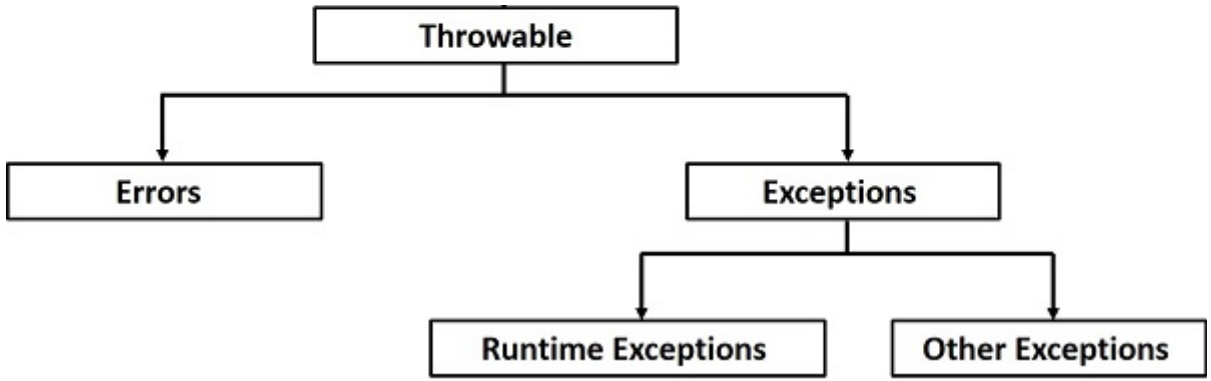
Returns the cause of the exception as represented by a Throwable object.

public void printStackTrace()

Prints the result of toString() along with the stack trace to System.err, the error output stream.

public StackTraceElement [] getStackTrace()

Returns an array containing each element on the stack trace. T


Throwable
Errors
Exceptions
Runtime Exceptions
Other Exceptions

Types of exceptions

- java.lang.Error
- java.lang.Exception
- java.lang.RuntimeException

java.lang.Error

An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions.

- NoClassDefFoundError
- OutOfMemoryError
- StackOverflowError

java.lang.Exception

The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.

The class Exception and any subclasses that are not also subclasses of RuntimeException are checked exceptions. Checked exceptions need to be declared in a method or constructor's throws clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary.

java.lang.RuntimeException

RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

RuntimeException and its subclasses are unchecked exceptions. Unchecked exceptions do not need to be declared in a method or constructor's throws clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary.

- ArithmeticException
- NullPointerException
- IndexOutOfBoundsException
- ClassCastException