Object Oriented Programming with Java

Zheng Chen

Exception

1. Java Exception Handling
2. Java Exception Class
3. Java Exception Throw
4. Java Custom Exception

TestAlgorithm - Apache NetBeans IDE 12.3

<default conf...

Search (⌘+I)

TestAlgorithm.java

Source | History

```
1
2   class HelloException {
3
4       public static void main(String args[]) {
5           int i = 10, j = 0;
6           System.out.print(i / j);
7       }
8   }
9
```

Projects | Files | Services | Navigator

Output

nlpAlgorithms - /Users/chenzheng/github/nlpAlgorithms | TestAlgorithm (run)

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at HelloException.main(TestAlgorithm.java:6)
/Users/chenzheng/Library/Caches/NetBeans/12.3/executor-snippets/run.xml:111: The following error occurred whi
/Users/chenzheng/Library/Caches/NetBeans/12.3/executor-snippets/run.xml:94: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

Transferring Maven repository index: Central Repository 11% | 9:1 | INS | Unix (LF)

Exception

- Exception is an abnormal condition.

- In Java, an exception **is an event** that disrupts the normal flow of the program. It **is an object** which is thrown at runtime.

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

try-catch Block

- To handle exceptions, place the code in a try block.

```
try  {
    // Codes may throw exception
}
```

- A try block cannot be used just by itself.

- It must be followed by one or more catch blocks, or one finally block, or a combination of both.

try-catch Block

- To handle an exception that might be thrown inside a try block, use a catch block.

```
catch (ExceptionClassName parameterName)   {
    // Exception handling  code
}
```

- When an exception is thrown, the reference of the **exception object is copied to the parameter**. We can use the it to get information from the exception object.

- The parameter **type is the class of the exception** that it is supposed to catch. The parameterName is a user-given name.

General syntax for a try-catch block

```
try  {
    // Your code  that may throw  an  exception
}
catch  (ExceptionClass1 e1){
    // Handle  exception of  ExceptionClass1 type
}
catch  (ExceptionClass2 e2){
    // Handle  exception of  ExceptionClass2 type
}
catch  (ExceptionClass3 e3){
    // Handle  exception of  ExceptionClass3 type
}
```

Example

```
public class Main {
  public static void main(String[] args) {
1   int x = 10, y = 0, z;
    try {
2     z = x / y;
      System.out.println("z = " + z);
    } catch (ArithmeticException e) {
3     String msg = e.getMessage();
4     System.out.println("The error is: " + msg);
    }
5   System.out.println("The end.");
  }
}
```

Finally block

- A try block can also have zero or one finally block. A finally block is always used with a try block.

```
finally  {
    // Code for finally block
}
```

- A try block may be followed by zero or more catch blocks.
- A try block can have a **maximum of one** finally block.
- A try block must have either a catch block, a finally block, or both.

Usage of finally block

- A finally block is guaranteed to be executed no matter what happens in the associated try and/or catch block.

- Typically, we use a finally block to write cleanup code.

```
try  {
    // Obtain   and  use  some resources here
}
finally  {
    // Release the   resources that were  obtained in the   try  block
}
```

```
ObjectOutputStream oos = null;
try {
   oos = new ObjectOutputStream(new FileOutputStream(file));
   oos.writeObject(shapes);
   oos.flush();
} catch (FileNotFoundException ex) {
    // complain to user
} catch (IOException ex) {
    // notify user
} finally {
    if (oos != null) {
        try {
            oos.close();
        } catch (IOException ex) {
            // ignore ... any significant errors should already have been
            // reported via an IOException from the final flush.
        }
    }
}
```

Example of finally block

Exception

1. Java Exception Handling
2. **Java Exception Class**
3. Java Exception Throw
4. Java Custom Exception

Exception Class

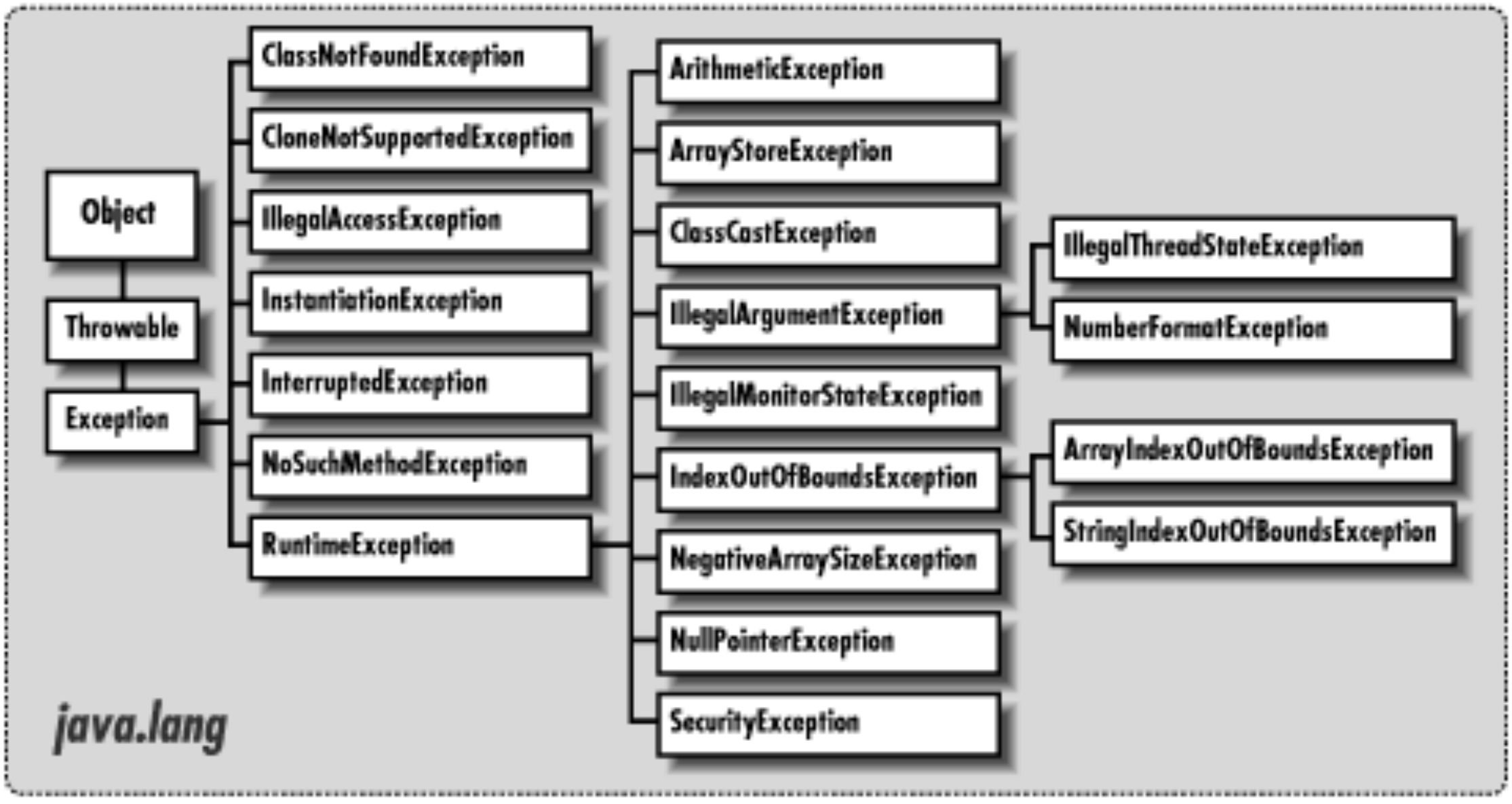
- The exception class hierarchy starts at the java.lang.Throwable class.

- When an exception is thrown, it must be an object of the Throwable class, or any of its subclasses.

- We can create our own exception classes by inheriting our classes from one of the exception classes.

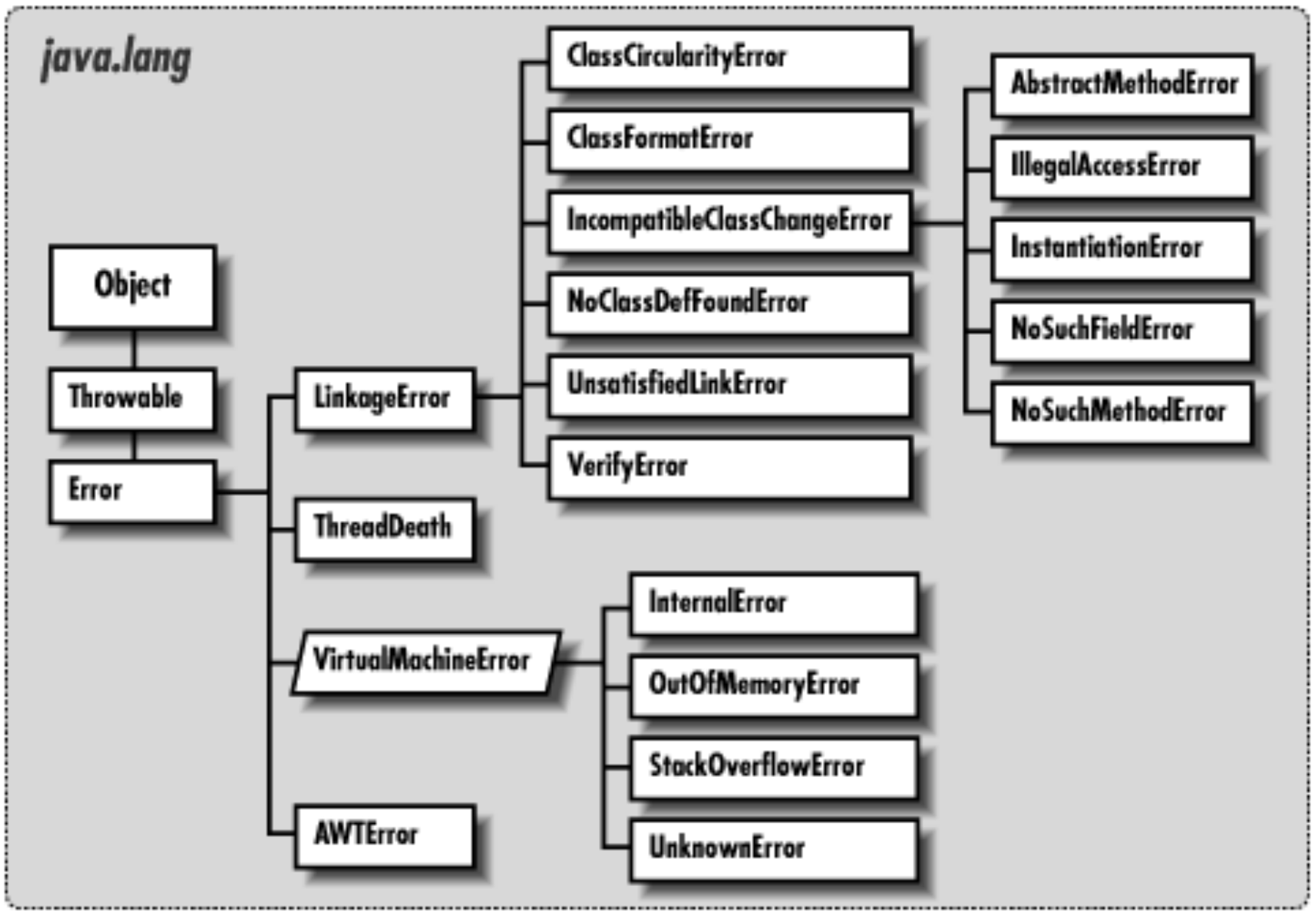
Exception Class Hierarchy

- The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: **Exception** and Error.



Exception Class Hierarchy

- The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and **Error**.



Types of Java Exceptions

- Checked Exception
  - The classes which directly inherit Exception class except RuntimeException are known as checked exceptions.
  - Checked exceptions are checked at compile-time.

- Unchecked Exception
  - The classes which inherit RuntimeException are known as unchecked exceptions
  - Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

- Error
  - The classes which inherit class Error are errors.
  - Errors are irrecoverable.

Some checked exception classes

- ClassNotFoundException
  - This exception is thrown to indicate that a class that is to be loaded cannot be found.
- CloneNotSupportedException
  - This exception is thrown when the clone() method has been called for an object that cannot be cloned.
- NoSuchMethodException
  - This exception is thrown when a specified method cannot be found.

Some unchecked exception classes

- ArithmeticException
  - This exception is thrown to indicate an exceptional arithmetic condition, such as integer division by zero.
- ArrayIndexOutOfBoundsException
  - This exception is thrown when an out-of-range index is detected by an array object. An out-of-range index occurs when the index is less than zero or greater than or equal to the size of the array.
- NullPointerException
  - This exception is thrown when there is an attempt to access an object through a null object reference. This can occur when there is an attempt to access an instance variable or call a method through a null object or when there is an attempt to subscript an array with a null object.

Some error classes

- OutOfMemoryError
  - This error is thrown when an attempt to allocate memory fails.
- StackOverflowError
  - This error is thrown when a stack overflow error occurs within the virtual machine.
- VirtualMachineError
  - The appropriate subclass of this error is thrown to indicate that the Java virtual machine has encountered an error.

Arranging Multiple catch Blocks

- Multiple catch blocks for a try block must be arranged from the most specific exception type to the most generic exception type.

```
try  {
// Do  something, which  might  throw  Exception
}
catch(ArithmeticException e1)  {
// Handle  ArithmeticException first
}
catch(RuntimeException e2)  {
// Handle  RuntimeException after  ArithmeticException
}
```

Exception

1. Java Exception Handling
2. Java Exception Class
3. **Java Exception Throw**
4. Java Custom Exception

Throwing an Exception

- We can throw an exception in our code using a throw statement. The syntax for a throw statement is

```
throw <A throwable object reference>;
```

- throw is a keyword, which is followed by a reference to a throwable object.

- A throwable object is an instance of a class, which is a subclass of the Throwable class, or the Throwable class itself.

Throwing an Exception Example

```
// Create an object of IOException
IOException e1  = new IOException("File not  found");
// Throw the IOException
throw  e1;
```

- If we throw a checked exception, we must handle it with a try-catch block, or using a throws clause in the method or constructor declaration.

- These rules do not apply if you throw an unchecked exception.

Throws clause

- If a piece of code may throw a checked exception, we have two options: handle it with a try-catch block, or specify in your method declaration with throws clause.

- Syntax for a throws clause is

```
methodName(<params>) throws<List of Exceptions>{
    \\method body
}
```

Throws clause example

```
public class Main {
  public static void readChar() throws IOException {
    int input = System.in.read();
    System.out.println(input);
  }
  public static void main(String[] args) {
    try {
      readChar();
    } catch (IOException e) {
      System.out.println("Error occurred.");
    }
  }
}
```

Another example

```
public class Main {
  public static void readChar() throws IOException {
    int input = System.in.read();
    System.out.println(input);
  }
  public static void main(String[] args) throws IOException {
    readChar();
  }
}
```

Exception

1. Java Exception Handling
2. Java Exception Class
3. Java Exception Throw
4. **Java Custom Exception**

Create our own exception

- An exception class is like any other classes in Java, it can be inherited.

- If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

```
public class MyException  extends  Exception  {
    \\custom codes
}
```

```
class InvalidAgeException extends Exception {
    InvalidAgeException(String s) {
        super(s);
    }
}

class TestCustomException {
    static void validate(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("not valid");
        }
        System.out.println("welcome to vote");
    }

    public static void main(String args[]) {
        try {
            validate(13);
        } catch (Exception m) {
            System.out.println("Exception occured: " + m);
        }
        //rest of the code...
    }
}
```

Custom exception example

Methods of exceptions

- Throwable class is the superclass of all exception classes in Java. All of the methods shown in this table are available in all exception classes.
  - String getMessage() — returns the detailed message of the exception.
  - StackTraceElement[] getStackTrace() — returns an array of stack trace elements.
  - void printStackTrace() — prints the stack trace on the standard error stream.
  - String toString() — returns a short description of the exception object.

```
public class TestExceptionMethods {
   public static void main(String args[]) {
        try {
            int[] array = new int[10];
            System.out.println(array[10]);
        } catch (Exception e) {
            System.out.println(e);
            System.out.println(e.getMessage());
            System.out.println(e.getStackTrace());
            e.printStackTrace();
        }
    }
}
```

```
java.lang.ArrayIndexOutOfBoundsException: 10
10
[Ljava.lang.StackTraceElement;@7852e922
java.lang.ArrayIndexOutOfBoundsException: 10
        at TestExceptionMethods.main(TestExceptionMethods.java:7)
```

Custom Exceptions can override methods

```
class MyException
        extends Exception {
    String msg;
    MyException(String msg) {
        this.msg = msg;
    }
    public String getMessage() {
        return msg;
    }
}
```

```
public class TestExceptionMethods {
    public static
            void main(String args[]) {
        try {
            throw new MyException
                    ("Something wrong....");
        } catch (Exception e) {
            System.out.println
                    (e.getMessage());
        }
    }
}
```

Homework

Write a class that keeps a running total of all characters passed to it (one at a time) and throws an exception if it is passed a non-alphabetic character.

Why do Java developers wear glasses? ...
Because they don't C#.