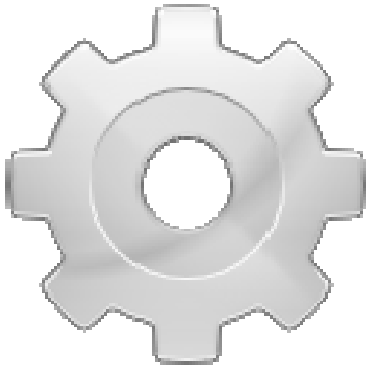
Programação Orientada a Objetos

Teste de Software e Tratamento de erros

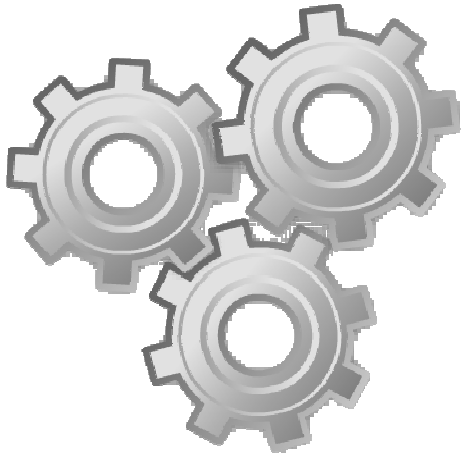
Testando Software

- Testar de forma eficiente é fundamental para construir software confiável e de boa qualidade
- Teste de Software tornou-se uma importante área de pesquisa dentro da Engenharia de Software e para a indústria de software em geral
- Há vários tipos de teste de software: unidade, integração, funcionais, desempenho, segurança e carga



Unit Test

Test a single class or a single method isolated.



Integration Test

Test a group of classes that colaborate to fulfill a functionality.

Functional Test

Test the software as a whole, verifying if the functionality is according to software requirements .

It is also possible to perform tests to verify non-functional characteristics

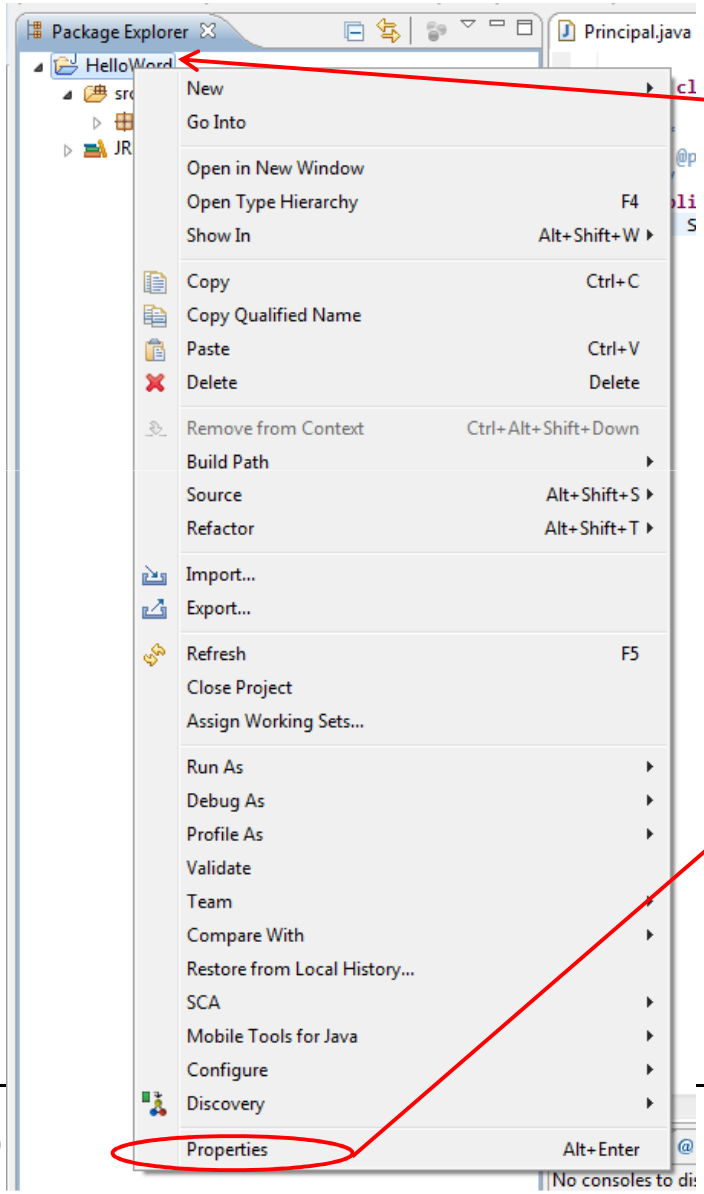
Performance Test

Security Test

Stress Test

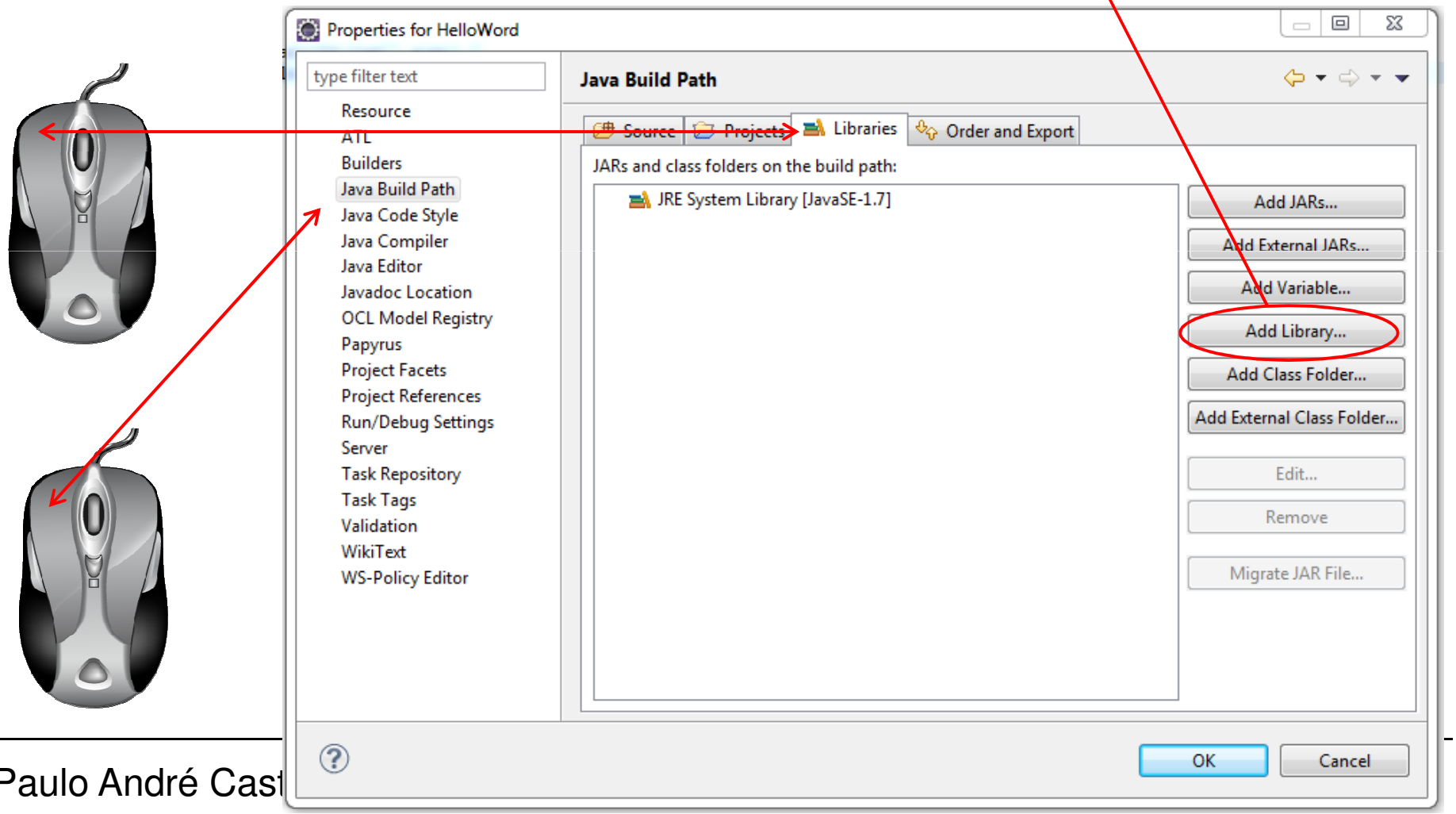
Testes de Unidade com o Framework JUnit

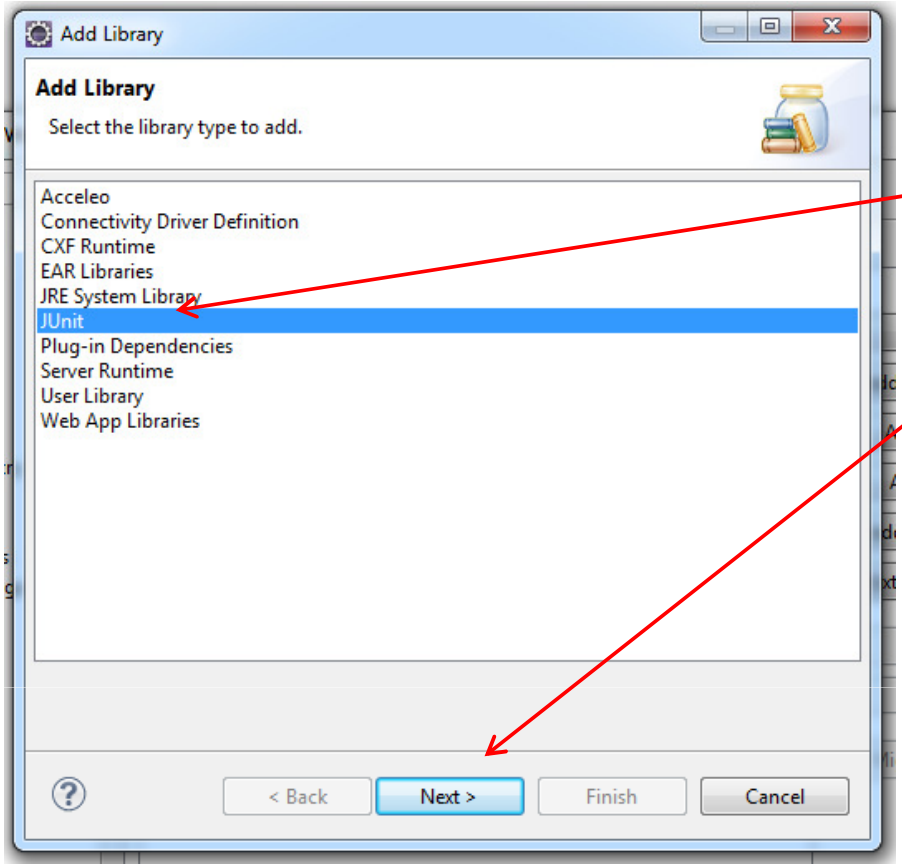
Step 1 *Add JUnit to classpath*

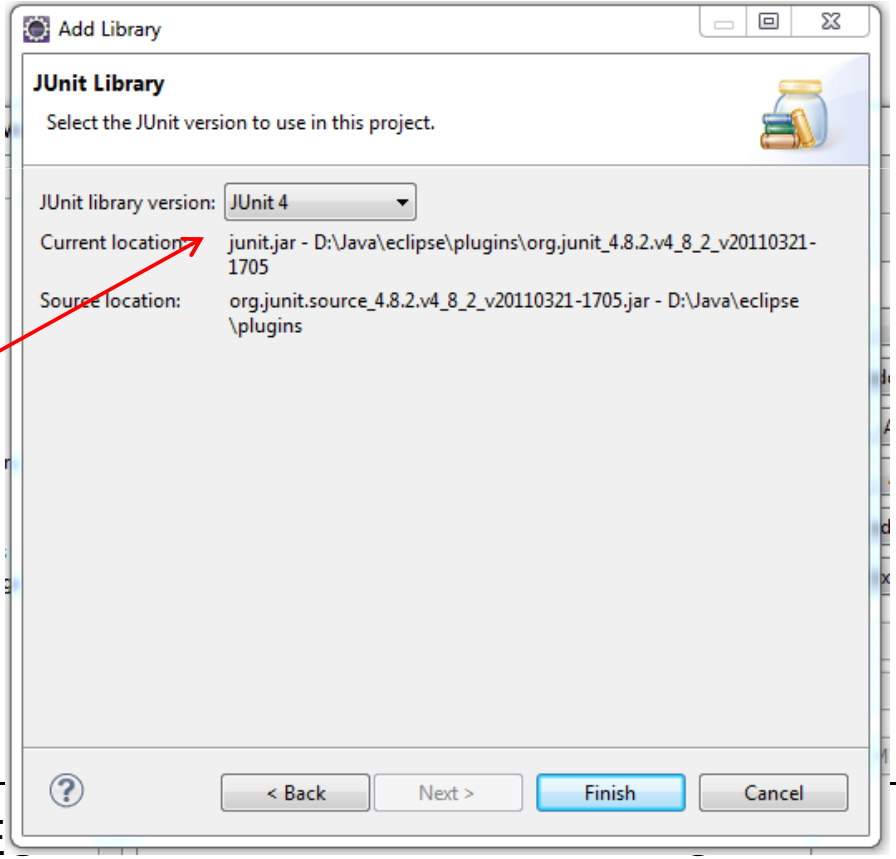


Access the project properties

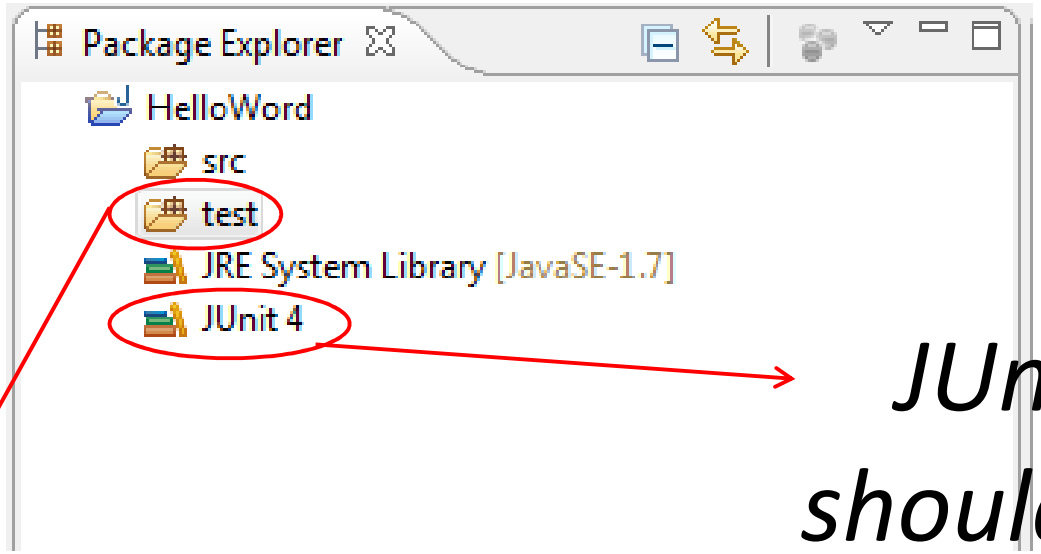
Click the button to add a library







Choose JUnit 4



JUnit library should appear on the project

A good practice is to create a new source folder to separate test code from application code

Step 2 *Create test class*

```
public class TestCalc
{
    ...
}
```

The class don't need anything special, it just need to have a test method!

Step 3 *Create test method*

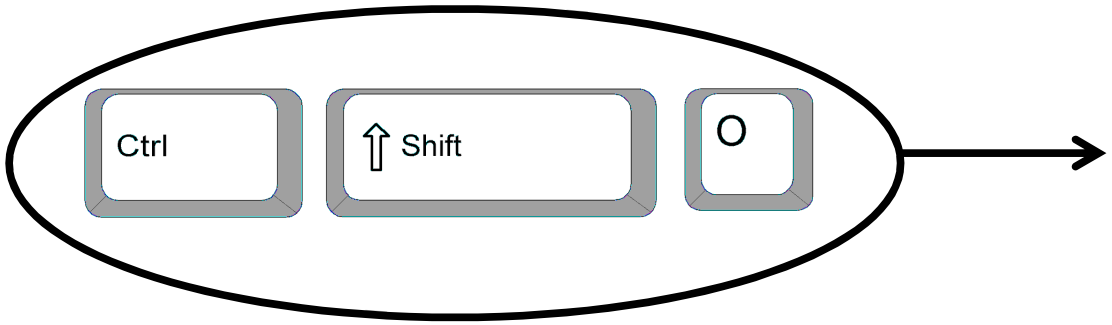
To be a test method it needs to have the @Test annotation

```
public class TestCalc {
    @Test
    public void testAdd(){
        ...
    }
}
```

The method needs to return void and don't have parameters

```
import
org.junit.Test;
```

You will need to import the @Test annotation in your class



Try that!

Step 4 *Exercise functionality*

```
public class TestCalc {
    @Test
    public void testAdd(){
        Calc c = new
Calc();
        int i = c.add(3,2);
    }
}
```

Step 5 *Assert expectations*

```
public class TestCalc {
    @Test
    public void testAdd(){
        Calc c = new Calc();
        int i = c.add(3,2);
        assertEquals(5, i);
    }
}
```

```
import static
org.junit.Assert.*;
```

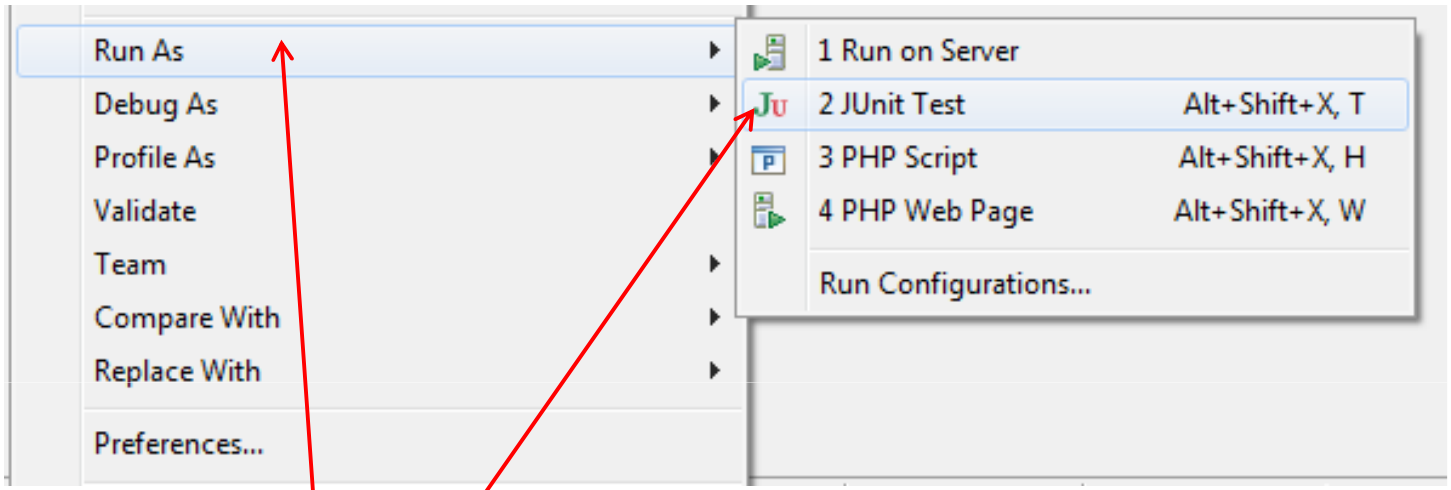
With this import statement, all functions from class Assert became available on the test class

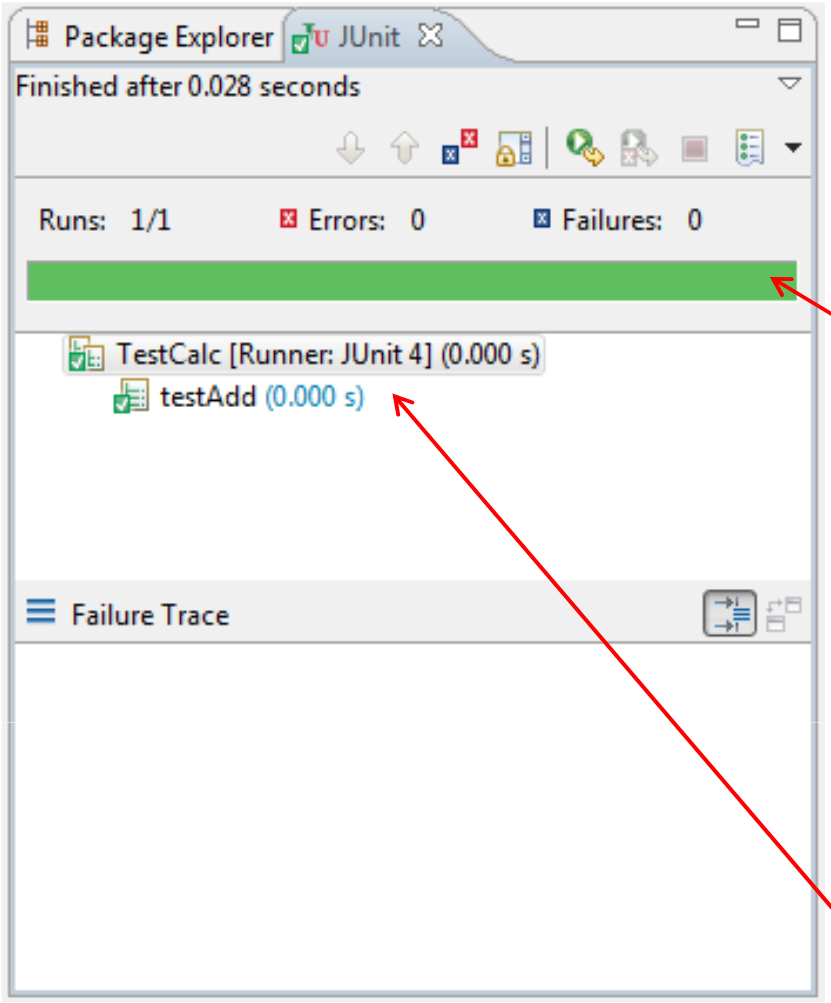
```
assertTrue()
assertFalse()
assertEquals()
assertNotEquals()
assertSame()
assertNotSame()
assertNull()
assertNotNull()
fail()
```

Stop printing in the console to test!

Step 6 *Run the test!*

on the code or on the file

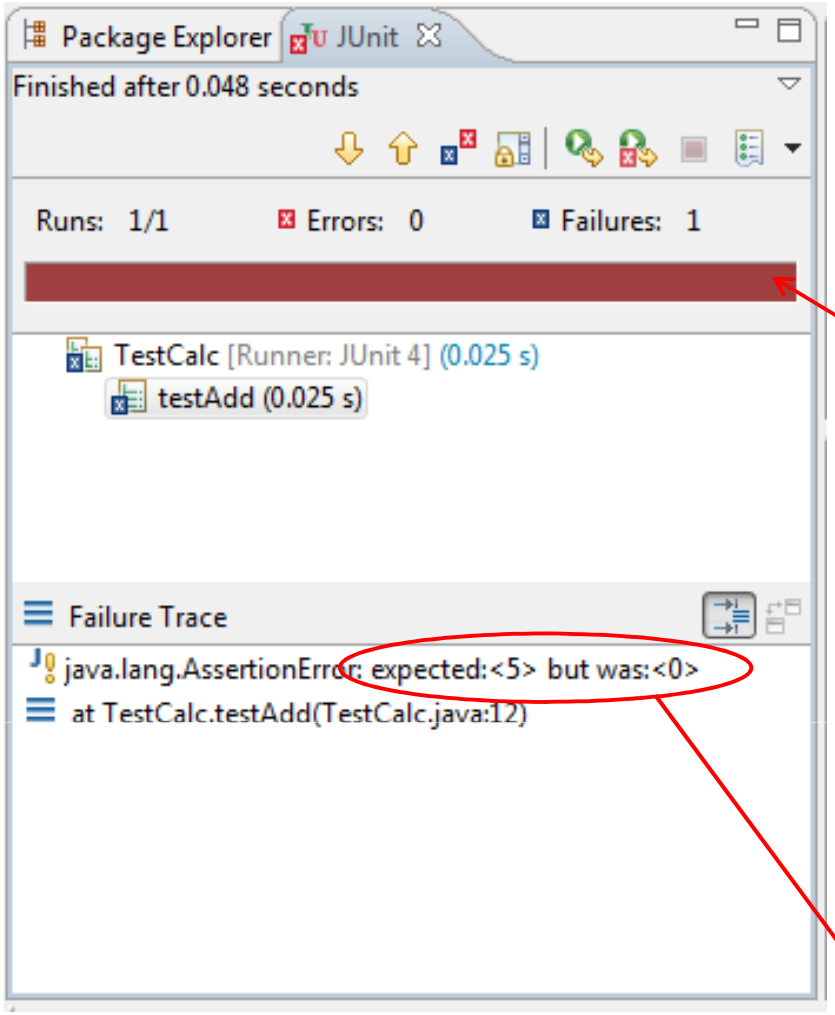




SUCCESS!

Green bar means that all tests had passed!

It lists all the classes and their respective test methods



FAILURE...

Red bar means that there are tests failing...

Selecting the test, it shows what went wrong in the execution

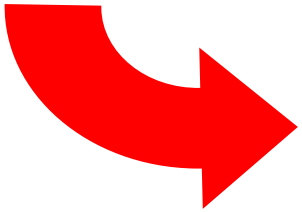
Sometimes there is something that you need to execute before or after each test

Initialization

```
@Test
public void
testAdd(){...}
```

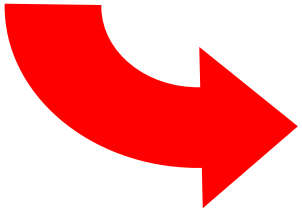

Clean up for next test

`@BeforeClass`



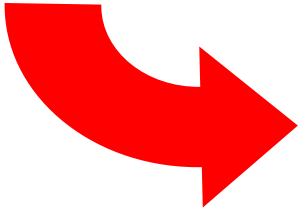
Execute before all tests

`@Before`



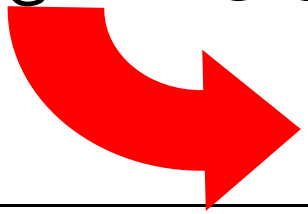
Execute before each test

`@After`



Execute after each test

`@AfterClass`



Execute after all tests

Exercício

1. Crie uma classe ArrayMath com métodos para calcular média, mínimo e somatório de um array de doubles.

2. Crie uma classe de testes com métodos para testar cada um dos métodos da classe ArrayMath. Cada método de testes deve usar pelo menos dois Asserts

3. Crie métodos usando @Before, @BeforeClass, @AfterClass and @After. Faça com que estes métodos de testes e inicialização escrevam algo no console e verifique a ordem de escrita dos métodos

```
01 package arrayMath;
02
03 import static org.junit.Assert.*;
04
05 import org.junit.After;
06 import org.junit.AfterClass;
07 import org.junit.Before;
08 import org.junit.BeforeClass;
09 import org.junit.Test;
10
11 public class ArrayMathTest {
12
13       @BeforeClass
14       public static void setUpBeforeClass() throws Exception {
15                System.out.println("@BeforeClass");
16       }
17
18       @AfterClass
19       public static void tearDownAfterClass() throws Exception {
20                System.out.println("@AfterClass");
21       }
22
23       @Before
24       public void setUp() throws Exception {
25                System.out.println("@Before");
26       }
27
28       @After
29       public void tearDown() throws Exception {
30                System.out.println("@After");
31       }
```

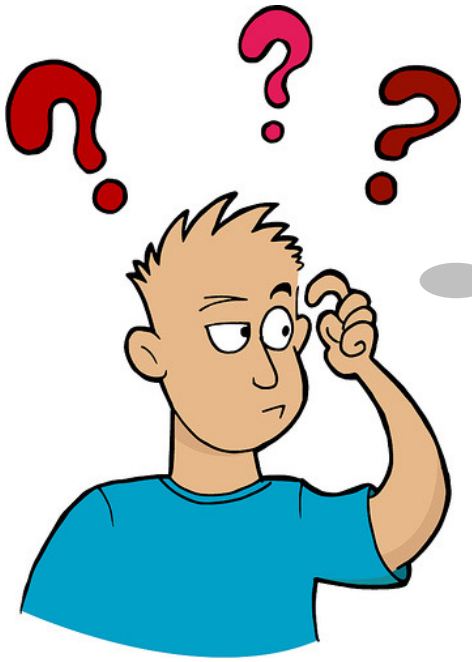
```
32      @Test
33      public final void testMinimo() {
34              ArrayMath a=new ArrayMath();
35              double array[]={1,2,3,4,5,6,7,8};
36              double m=a.minimo(array);
37              assertEquals(m,1.0,Double.MIN_VALUE);
38              //assertEquals(double,double) is deprecated.
39              //Use assertEquals(double a,double b,double epsilon)
40              // That means assertEquals return true if a-b<epsilon
41              System.out.println("Test Minimo");
42      }
43      @Test
44      public final void testMedia() {
45              ArrayMath a=new ArrayMath();
46              double array[]={1,2,3,4,5,6,7,8};
47              double m=a.media(array);
48              assertEquals(m,4.5,Double.MIN_VALUE);
49              System.out.println("Test Media");
50      }
51
52      @Test
53      public final void testSomatorio() {
54              ArrayMath a=new ArrayMath();
55              double array[]={1,2,3,4,5,6,7,8};
56              double s=a.somatorio(array);
57              assertEquals(s,36,Double.MIN_VALUE);
58              System.out.println("Test Somatorio");
59  } }
```

Tratamento de Erros em Java

- A melhor maneira de tratar erros Java é utilizar Exceções em substituição aos antigos códigos de retorno de Erro

```
public int portion(double value,
                   double percent){
    if(percernt > 100 || percent <
0)
        return -1;
    return value * percent / 100;
}
```

BAD IDEA



Why I'm receiving -1 from this method?

```
portion(5000,120)
;
```

It is not clear for the clients how errors are notified and what kind of error can happen.

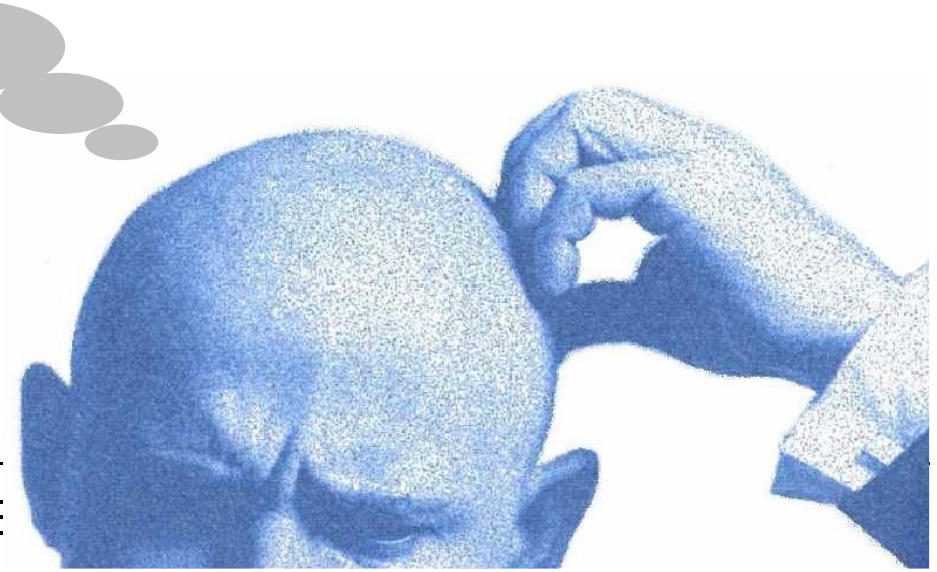
It can return -1 without error

```
portion(-100,1);
```

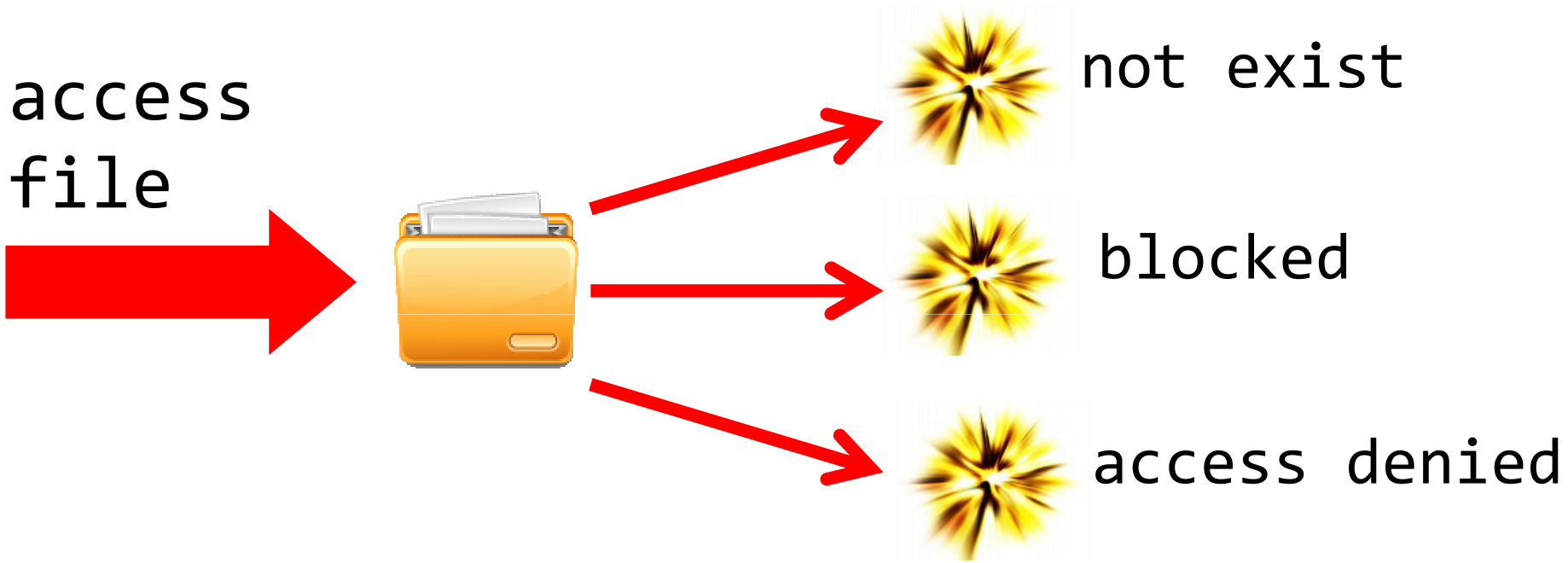
Sometimes all values are a valid return in a correct method invocation

```
String someMethod(){...}
Person otherMethod(){...}
```

What can I do if the return is another class?



The application may perform different actions for different kinds of errors.



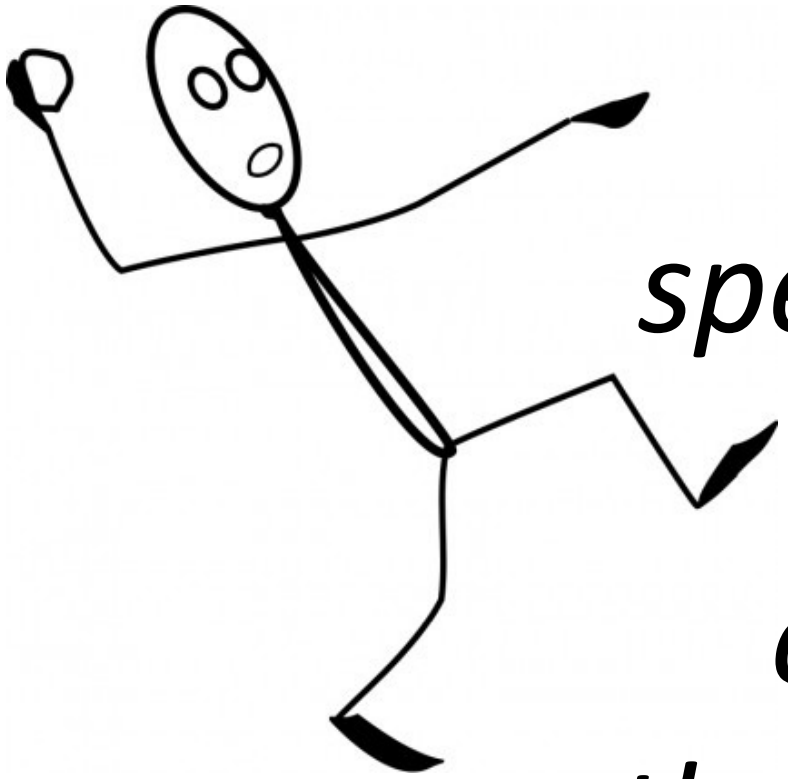
How can you communicate what happened?

Don't even think about putting errors in a global varible!

Bad for concurrency!

You can forget to clean up and other components will find previous erros!

Exceptions to the Job

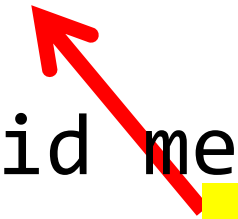


Exceptions are a special type of class that represent errors and can be thrown to the caller.

```
public class MyException
                    extends
Exception{
    ...
}
```

To be an exception, the class should extend the class Exception.

A method should declare explicitly which exceptions it can throw.



```
public void method(int param)
                    throws MyException{
    …
    if(error){
        throw new
MyException(“problem“);
    }
    …
}
```

A new exception instance should be created to be thrown.


```
public void caller(){
    …
    try{
instance.method();
    }catch(MyException e){
        //handle error
    }
    …
}
```

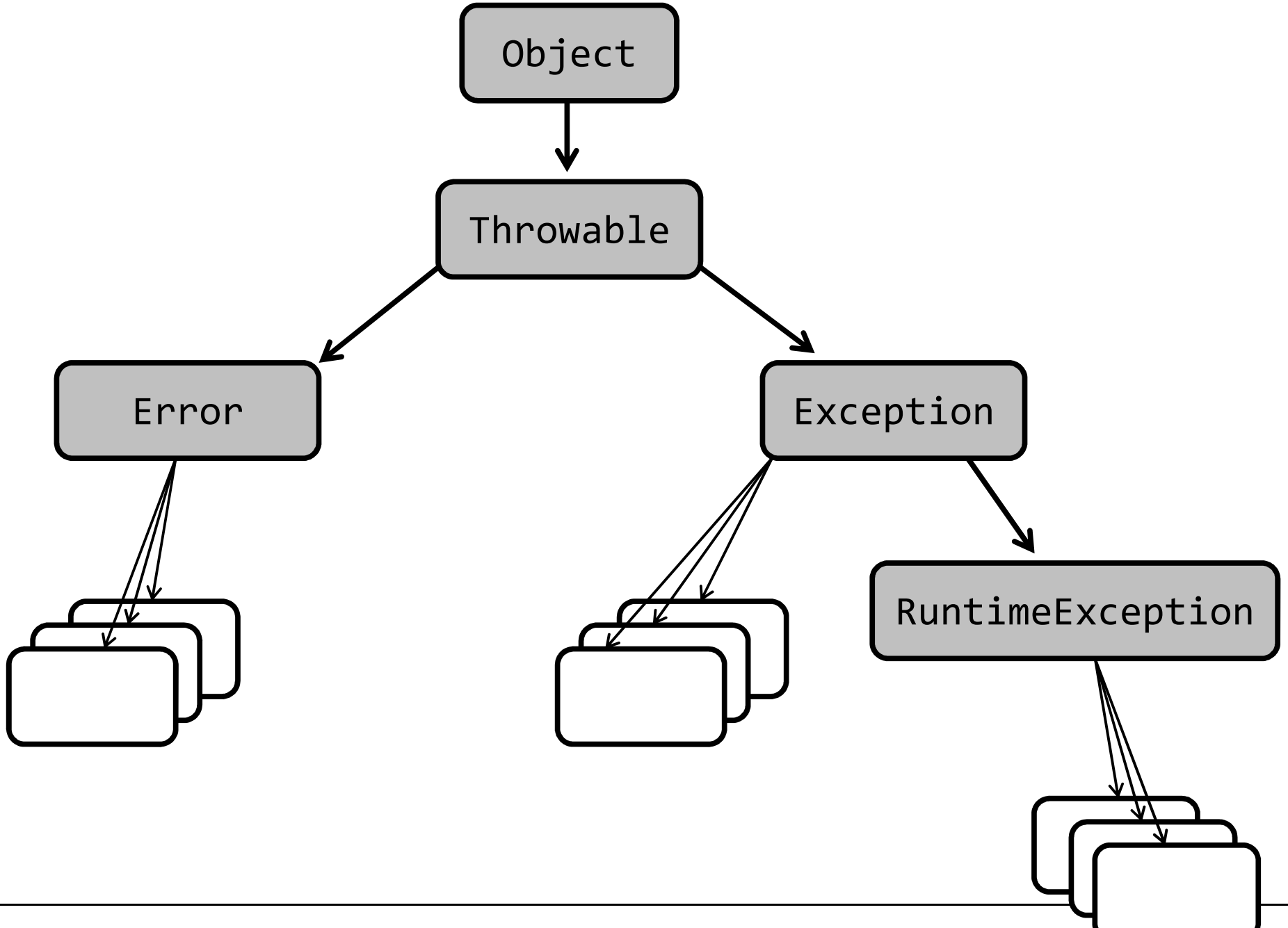
A caller can catch the exception using the try/catch block.

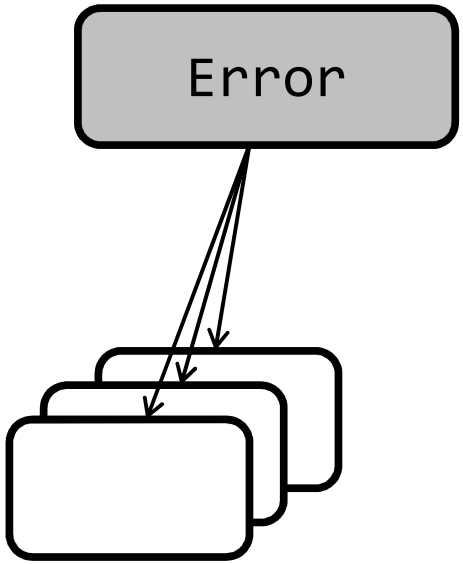
```
public void caller()
                throws MyException
{
    …
    instance.method();
    …
}
```

The caller can declare that he throws the exception and let his caller handle it.

You can handle the problem or send it to another one...

... but you can't ignore it!

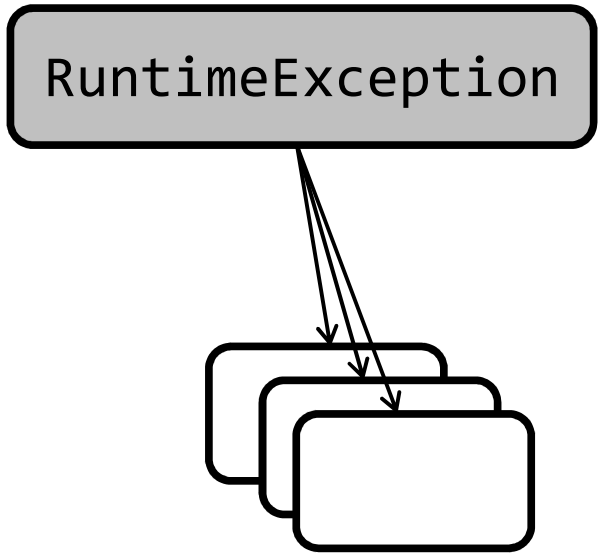

Object
Throwable
Error
Exception
RuntimeException



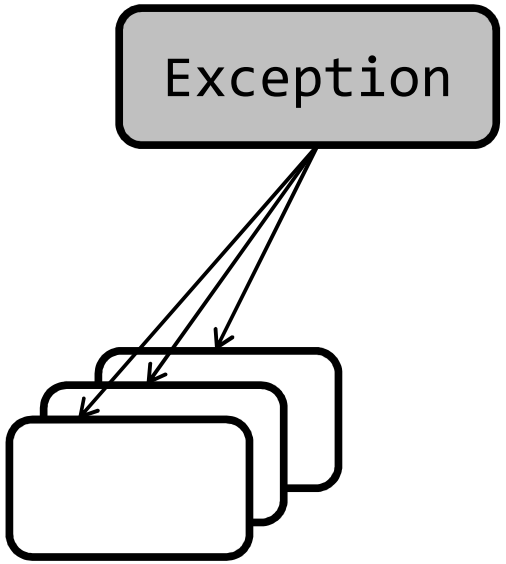
Indicates serious problems and abnormal conditions that a usually you should not try to catch.

```
CoderMalfunctionError

FactoryConfigurationError

VirtualMachineError
```

Indicates normal problems that are not manadatory to be handled.




```
ArithmeticException

ClassCastException

NullPointerException

IndexOutOfBoundsException
```

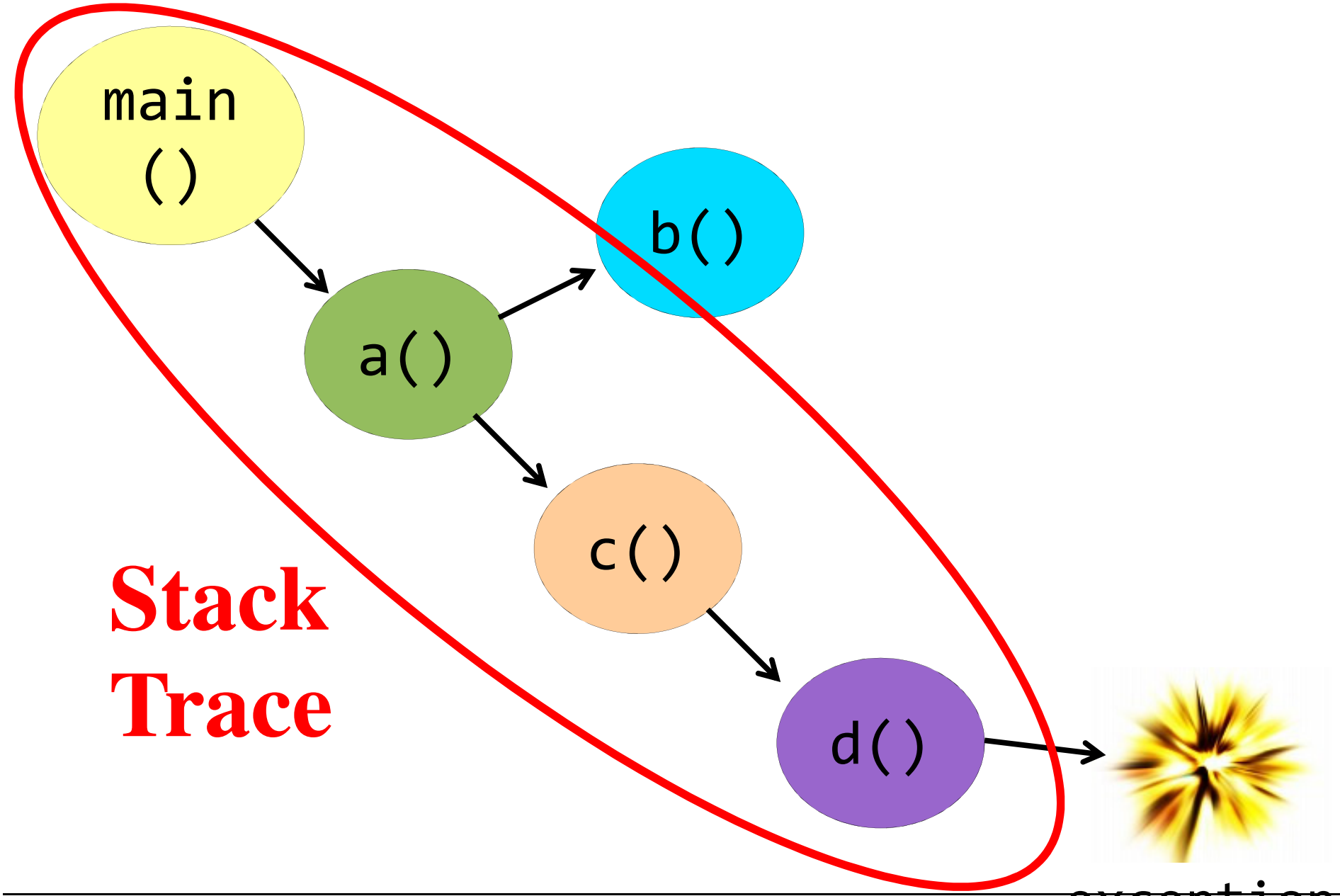


Indicates problems that can happen on class execution that need to be handled.

```
SQLException

IOException

PrintException

DataFormatException
```

Stack Trace

When an exception is created, it records all the call stack from that point.



Stack Trace

To know how to read a stack trace is important to find errors!

```
org.firebirdsql.jdbc.FBSQLException: Invalid column index.
        at org.firebirdsql.jdbc.FBResultSet.getField(FBResultSet.java:617)
        at org.firebirdsql.jdbc.FBResultSet.getField(FBResultSet.java:592)
        at org.firebirdsql.jdbc.FBResultSet.getFloat(FBResultSet.java:466)
        at com.myapplication.AccessServlet.doGet(AcessoServlet.java:50)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:689)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:252)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
        at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:213)
        at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:178)
        at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:126)
        at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:105)
        at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:107)
        at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:148)
        at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:869)
        at org.apache.tomcat.util.net.PoolTcpEndpoint.processSocket(PoolTcpEndpoint.java:527)
        at org.apache.tomcat.util.net.LeaderFollowerWorkerThread.runIt(LeaderFollowerWorkerThread.java:
        at org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run(ThreadPool.java:684)
        at java.lang.Thread.run(Thread.java:595)
```

The exception class can show what kind of error happen.

org.firebirdsql.jdbc.FBSQLException: Invalid column index.

```
at org.firebirdsql.jdbc.FBResultSet.getField(FBResultSet.java:617)
at org.firebirdsql.jdbc.FBResultSet.getField(FBResultSet.java:592)
at org.firebirdsql.jdbc.FBResultSet.getFloat(FBResultSet.java:466)
at com.myapplication.AccessServlet.doGet(AcessoServlet.java:50)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:689)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:252)
at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:213)
at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:178)
at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:126)
at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:105)
at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:107)
at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:148)
at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:869)
at org.apache.tomcat.util.net.PoolTcpEndpoint.processSocket(PoolTcpEndpoint.java:527)
at org.apache.tomcat.util.net.LeaderFollowerWorkerThread.runIt(LeaderFollowerWorkerThread.java:
at org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run(ThreadPool.java:684)
at java.lang.Thread.run(Thread.java:595)
```

The message usually present details of the error.

org.firebirdsql.jdbc.FBSQLException: Invalid column index.

```
at org.firebirdsql.jdbc.FBResultSet.getField(FBResultSet.java:617)
at org.firebirdsql.jdbc.FBResultSet.getField(FBResultSet.java:592)
at org.firebirdsql.jdbc.FBResultSet.getFloat(FBResultSet.java:466)
at com.myapplication.AccessServlet.doGet(AcessoServlet.java:50)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:689)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:252)
at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:213)
at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:178)
at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:126)
at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:105)
at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:107)
at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:148)
at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:869)
at org.apache.tomcat.util.net.PoolTcpEndpoint.processSocket(PoolTcpEndpoint.java:527)
at org.apache.tomcat.util.net.LeaderFollowerWorkerThread.runIt(LeaderFollowerWorkerThread.java:8
at org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run(ThreadPool.java:684)
at java.lang.Thread.run(Thread.java:595)
```

When the error doesn't happen on your classes, look for the first one on the stack trace.

```
org.firebirdsql.jdbc.FBSQLException: Invalid column index.
        at org.firebirdsql.jdbc.FBResultSet.getField(FBResultSet.java:617)
        at org.firebirdsql.jdbc.FBResultSet.getField(FBResultSet.java:592)
        at org.firebirdsql.jdbc.FBResultSet.getFloat(FBResultSet.java:466)
        at
com.myapplication.AccessServlet.doGet(AcessoServlet.java:50)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:689)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
        at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:252)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
        at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:213)
        at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:178)
        at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:126)
        at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:105)
        at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:107)
        at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:148)
        at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:869)
        at org.apache.tomcat.util.net.PoolTcpEndpoint.processSocket(PoolTcpEndpoint.java:527)
        at
```

Handling Exceptions

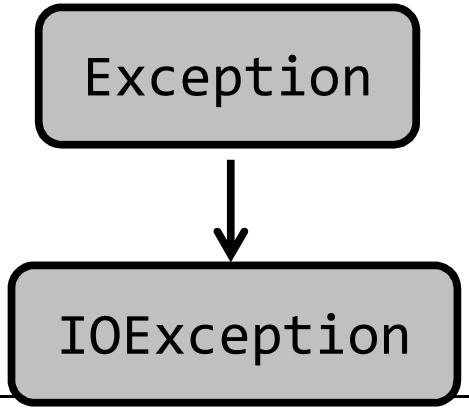
```
beforeTry();
try{
    beforeError();

instance.method();
    afterError();
}catch(MyException e){
    handleError(e);
}
afterTry();
```

When a exception happens, the execution jumps to the catch block and continues just after it.

You can catch more than one exception type, but ***only one*** *will be executed.*

```
try{
    instance.method();
}catch(IOException e){
    handleIO(e);
}catch(Exception e){
    handleOther(e);
}
```

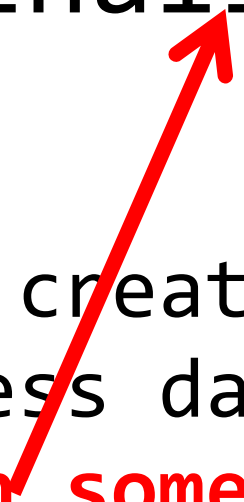
Subclass comes first!



```
try{
    con =
createConnection();
    //access database
}catch(SQLException e){
    handleError(e);
}finally{
    con.close()
}
```

You can use the block finally to ensure that something will be executed with or without exception!

Even with a return statement `finally` is executed!



```
try{
     con = createConnection();
     //access database
     return something;
}catch(SQLException e){
     handleError(e);
}finally{
     con.close()
}
```

Watch where you put your try/catch blocks!

```
try{
    a();
}catch(Exception e){
    handleError(e);
}
try{
    b();
}catch(Exception e){
    handleError(e);
}
```

```
try{
    a();
    b();
}catch(Exception e){
    handleError(e);
}
```

If error happen on a(), b() will not be executed!

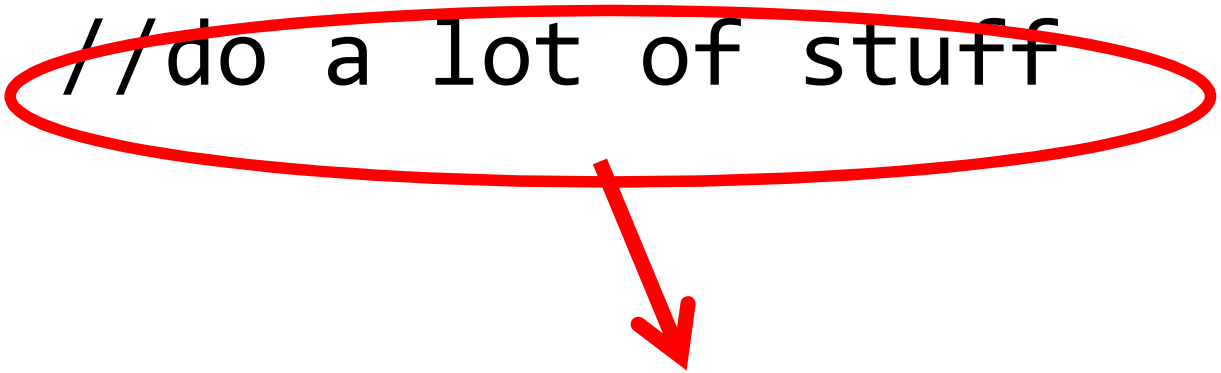
Testing Exceptions

Paulo André Castro

When you need to verify if a class throw an exception in a given situation!

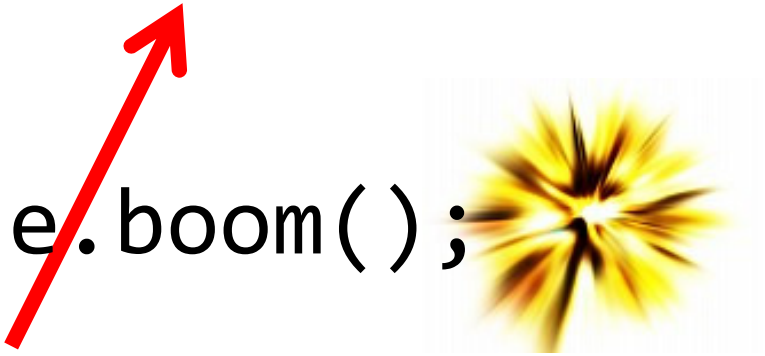
```
@Test(expected=BoomException.class)
public void testException(){
    instance.boom();
}
```

```
@Test(expected=MyException.class)
public void testException()
throws Exception {
    //do a lot of stuff
}
```



But the test will be correct if the exception is thrown anywhere in the test method!

Execution should never reach this line!



```
try{
    instance.boom();
    fail();
}catch(BoomException e){
    //verify exception if needed
}
```

If the exception is expected to be thrown in a specific point, you should use try/fail.

```
@Test
public void testException()
                              throws
BoomException{
    //This scenario should
    //never throw exception
}
```

If your test don't care about the exception, just declare that the method throw it...

Exercício Testando Exceptions

1. Crie uma nova versão da classe ArrayMath. Seus métodos devem lançar exceções caso recebam um array nulo como parâmetro.
2. A exceção lançada deve ser uma ArrayMathException e conter uma mensagem sobre o erro.
3. Crie uma nova versão da classe ArrayMathTest. Corrija os métodos de testes anteriores para que estes funcionem corretamente.
4. Crie novos métodos de teste para verificar se é lançada a exceção esperada ao ser passado null como parâmetro.

É possível executar todos os testes de um project, package, ou souce folder com um click