## Compile- and runtime errors

```
final int public ❶ = 33;

final String s = null;
System.out.println(s.length())❷ ;
```

❶ Compile time error: public is a Java™ keyword not to be used as variable's name.

❷ Run time error: De-referencing `null` yields a `NullPointerException`.

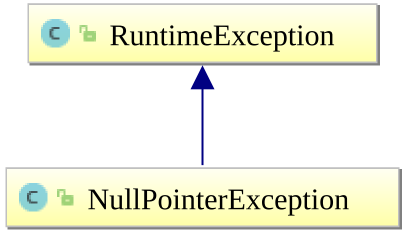
## NullPointerException (NPE for short)

```
final String s = null;
System.out.println(s.length());
```

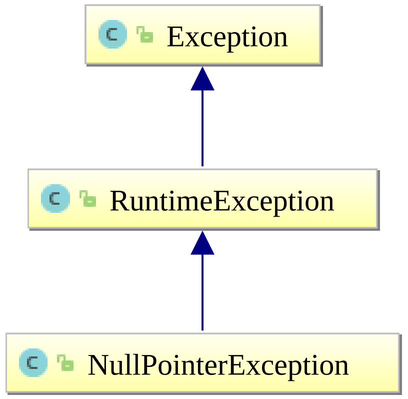
```
Exception in thread "main" java.lang.NullPointerException
	at exceptionhandling.Npe.main(Npe.java:7)
```

## `NullPointerException` is a class

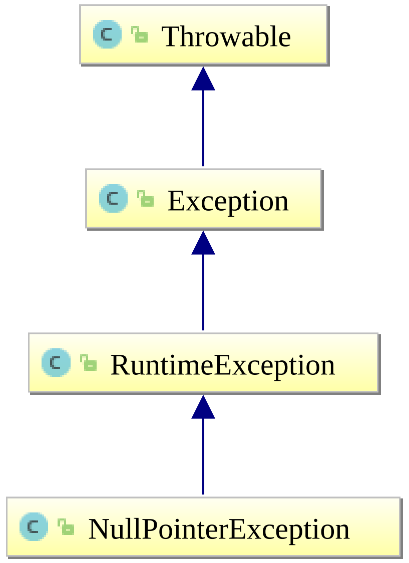
## `NullPointerException` is a class



## `NullPointerException` is a class



# `NullPointerException` is a class



## Throwing an exception

```
...
if (somethingBadHappens) {
  throw new NullPointerException();
}
...
```

**Note**

Without countermeasures your program will terminate

## Catching an exception by `try {...} catch {...}`

```java
final String s = null;
try {
  System.out.println(s.length()) ;
} catch (final NullPointerException e) {
  System.out.println("Dear user, something bad just happened");
}
System.out.println("Business as usual ...");
```

```
Dear user, something bad just happened
Business as usual ...
```

## Related exercises

Exercise 170: Mind your prey

## try {...} catch {...} syntax

```
try {
 [code that may throw an exception]
}[catch (ExceptionType-1 e) {
 [code that is executed when ExceptionType-1 is thrown]
}] [catch (ExceptionType-2 e) {
 [code that is executed when ExceptionType-2 is thrown]
}]
  ...
} [catch (ExceptionType-n e) {
  [code that is executed when ExceptionType-n is thrown]
}]
[finally {
  [code that runs regardless of whether an exception was thrown]]
}]
```

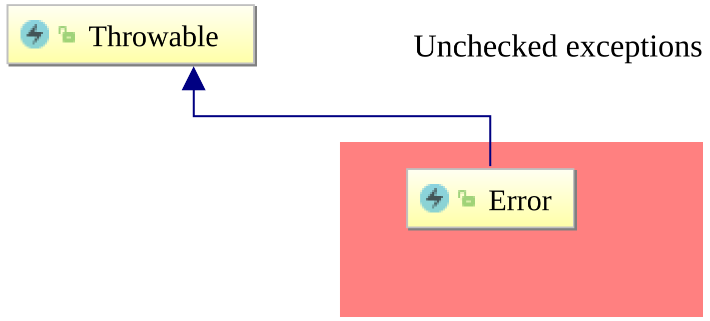
## Checked and unchecked exceptions

```
public static void main(String[] args) {
  final Path
    sourcePath = Paths.get("/tmp/test.txt"),
    destPath = Paths.get("/tmp/copy.java");

  // Compile time error:
  // Unhandled exception:
       java.io.IOException
  Files.copy(sourcePath, destPath);
...
```

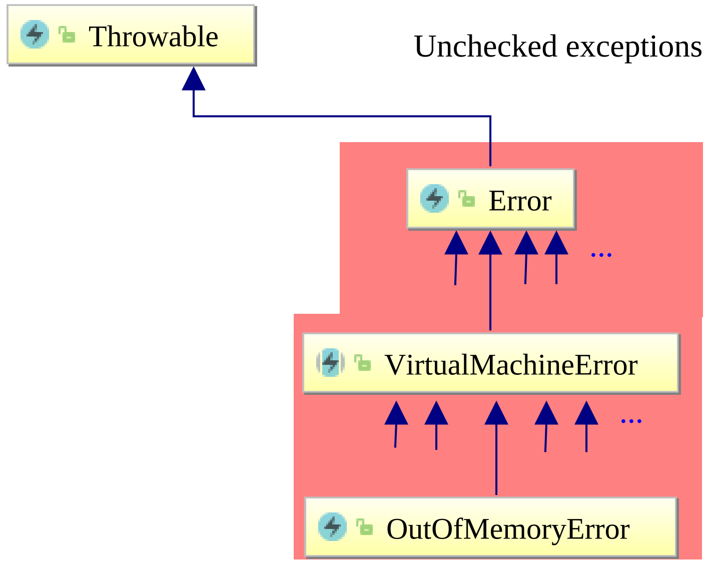
```
public static void
  main(String[] args) {

final String s = null;

// No problem
System.out.println(s.length());
```

## Checked and unchecked exceptions

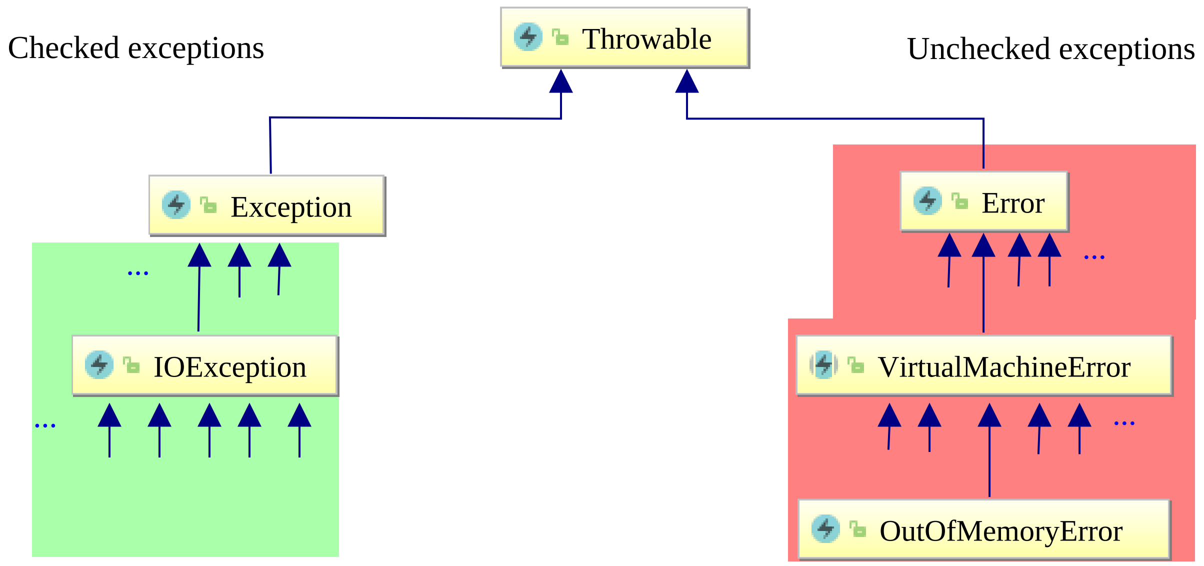
## Checked and unchecked exceptions



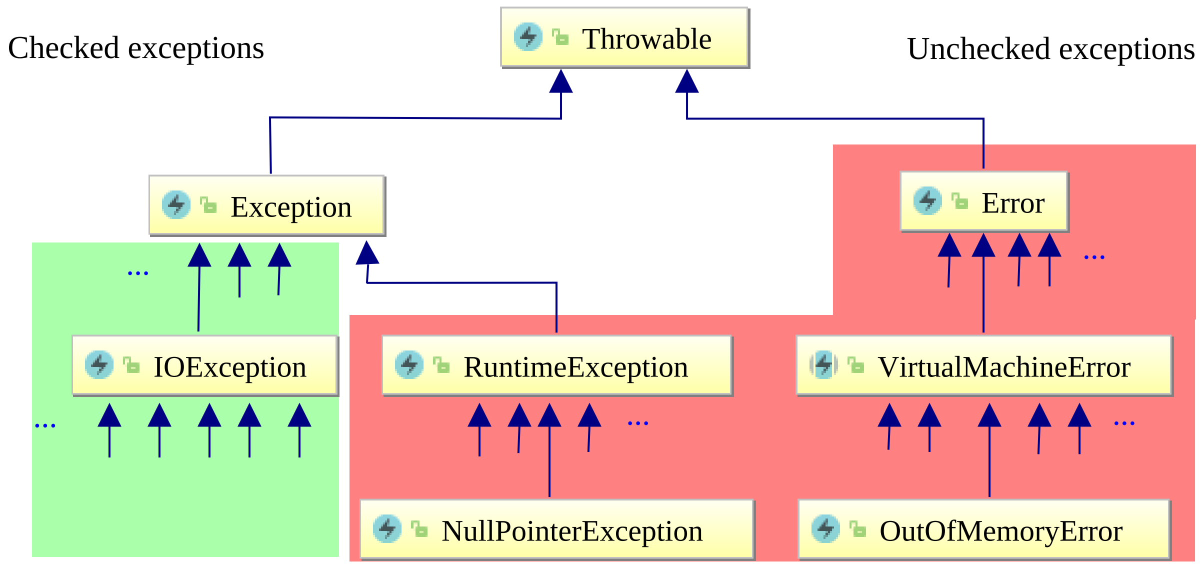
## Checked and unchecked exceptions



# Checked and unchecked exceptions



# Checked and unchecked exceptions



## Expected exceptions in Junit

```
@Test(expected = FileAlreadyExistsException.class)
public void copyFile() throws IOException {
  final Path
    source = Paths.get("/tmp/source.txt"),
    dest   = Paths.get("/tmp/dest.txt");

  Files.copy(source, dest);  // May work.
  Files.copy(source, dest);  // Failure: FileAlreadyExistsException
}
```

## Related exercises

Exercise 171: Expected exception test failure

## Just `finally`, no `catch`

```java
Scanner scanner = null;
try {
  scanner = new Scanner(System.in);
   ... // Something may fail
} finally {
  if (null != scanner) {
    scanner.close(); // Clean up, save resources!
  }
}
```

## try-with-resources (Java™ 7)

```
try (final Scanner❶  scanner❷ = new Scanner(System.in)) {
   ... // Something may fail
}❸ // implicitly calling scanner.close()
```

❶ Class must implement interface `AutoCloseable`.

❷ Variable `scanner`'s scope limited to block.

❸ `close()` method will be called automatically before leaving block scope.

## Scanner implementing `AutoCloseable`

```
public class Scanner
  implements AutoCloseable ❶, ... {

  ...

  public void close() {...} ❷

}
```

```
Interface AutoCloseable {
  public void close(); // Signature, no
                       // implementation
}
```

❶ Promise to implement all methods being declared in `AutoCloseable`.

❷ Actually implementing a `close()` method.

## No `close()` method in e.g. `class` String

```
try (final String s = new String()) { // Error: Required type: AutoCloseable; Provided: String
   ...
}
```

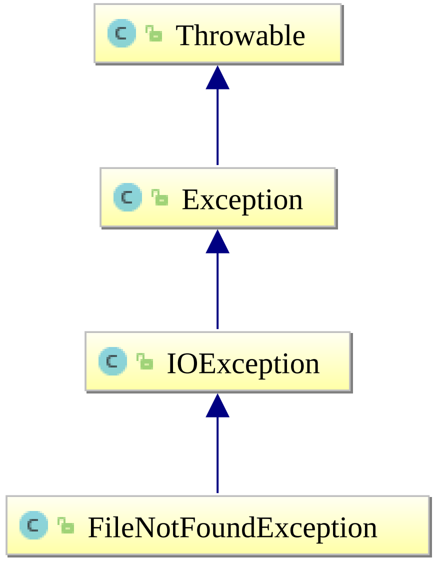
## Method `printStackTrace()`

```
 1 package exceptionhandling;
   public class StackTrace {
     public static void main(
          String[] args){
 5     a();
     }
     static void a() { b();}
     static void b() { c();}
     static void c() {
10     String s = null;
       s.length();
     }
   }
```

```
Exception in thread "main"
   java.lang.NullPointerException
 at ex.Trace.c(Trace.java:10)
 at ex.Trace.b(Trace.java:7)
 at ex.Trace.a(Trace.java:6)
 at ex.Trace.main(Trace.java:4)
```

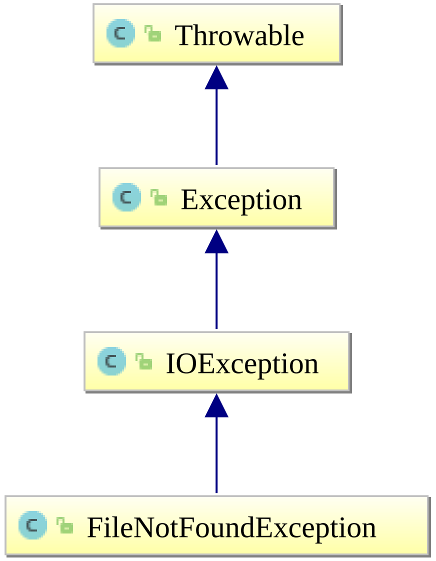
## Ascending inheritance ordering

```
try {
  FileInputStream f = new FileInputStream(
     new File("test.txt"));
} catch(final FileNotFoundException e) {
  System.err.println( "File not found");
} catch (final IOException e) {
  System.err.println( "IO error");
} catch(final Exception e) {
  System.err.println("General error");
}
```



# Descending inheritance ordering

```
try {
  FileInputStream f = new FileInputStream(
     new File("test.txt"));
} catch(Exception e) {
      System.err.println("General error");
} catch (IOException e) {
      System.err.println( "IO error");
} catch(FileNotFoundException e) {
      System.err.println("File not found");
}
```



## Implementing `convert`

```
/* Translate {"one", "two", "three"} to {"first", "second", "third"}
 * @param input The input String to be translated.
 * @return See above explanation. */
static public String convert(final String input) {
 switch (input) {
   case "one": return "first";
   case "two": return "second";
   case "three": return "third";
  default: return "no idea for " + input;
  }
}
```

## Problem: “Silent” errors

- Return false result, application continues.
- Solution: Throw an exception. Steps:

  1. Find a suitable exception base class.
  2. Derive a corresponding exception class
  3. Throw the exception accordingly.
  4. Test correct behaviour.

## Step 1: Find exception base class

- Problem happens on wrong argument to `convert(...)`.
- Use `IllegalArgumentException`.

## Step 2: Derive `CardinalException`

```java
public class CardinalException
  extends IllegalArgumentException {

  public CardinalException(final String msg) {
    super(msg);
  }
}
```

## Step 3: Throwing `CardinalException`

```java
/**
 * Translate {"one", "two", "three"} to {"first", "second", "third"}
 * @param input The input String to be translated.
 * @return See above explanation.
 * @throws CardinalException If input not from list.
 */
static public String convert(final String input)
  throws CardinalException {

  switch (input) {
    case "one": return "first";
    case "two": return "second";
    case "three": return "third";
  }
  throw new CardinalException(
            "Sorry, no translation for '" + input + "' on offer");
}
```

## Step 4: Unit test throwing `CardinalException`

```
@Test public void testRegular() {
  Assert.assertEquals("second", Cardinal.convert("two"));
}

@Test(expected = CardinalException.class)
public void testException() {
  Cardinal.convert("four"); // No assert...() required
}
```