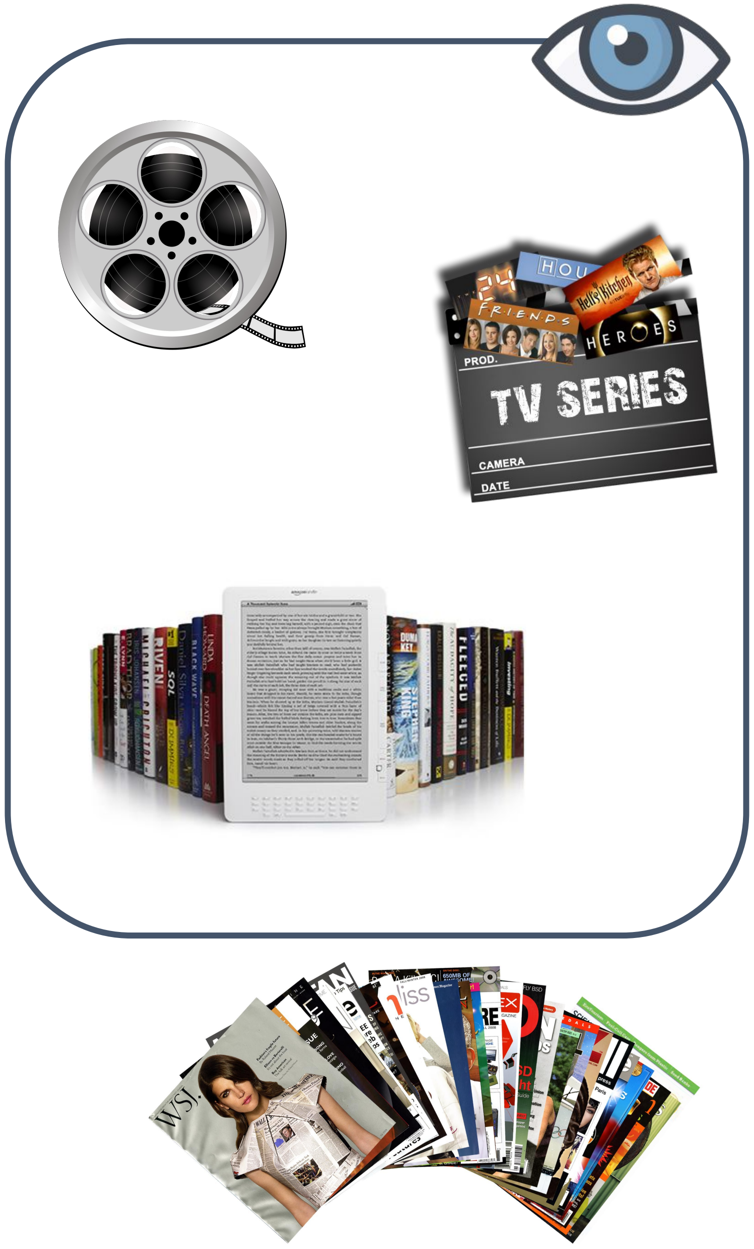
Curso Avanzado
JAVA SE

Presentación

AmazonViewer

AmazonViewer

AmazonViewer



Clases Avanzadas

Clases Avanzadas

CLASES ABSTRACTAS

Polimorfismo

Herencia Clases

Métodos sobreescritos

Muchas formas

Polimorfismo

Implementación **Interfaces**

Métodos sobreescritos

Muchas formas

¡Genial!

Interfaces

A veces no necesitamos
implementar todos los métodos

Herencia

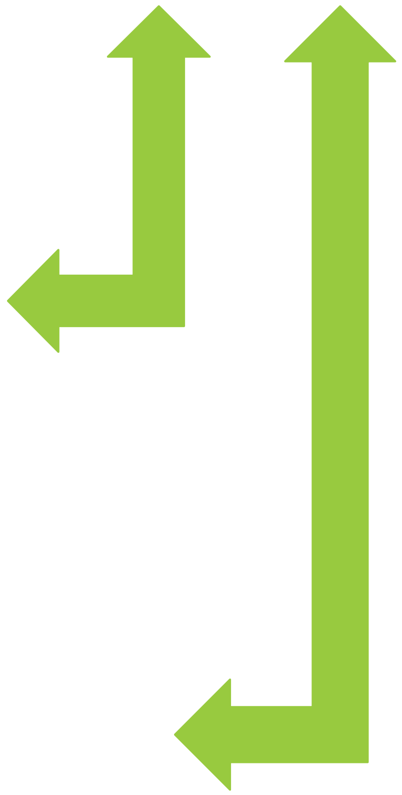
Las clases podrían <u>no</u> necesitar **heredar la implementación** de un método

Herencia

A veces no necesitamos crear **instancias de una clase padre**, porque es muy genérica

Clase Abstracta

- Intefaz
- Herencia



Clase Abstracta

No implementaremos todos los métodos

No crearemos instancias

```
public abstract class Figura {

    abstract void dibujate();

}
```

CLASES ABSTRACTAS

```java
public abstract class Figura {

    abstract void dibujate();

}
```

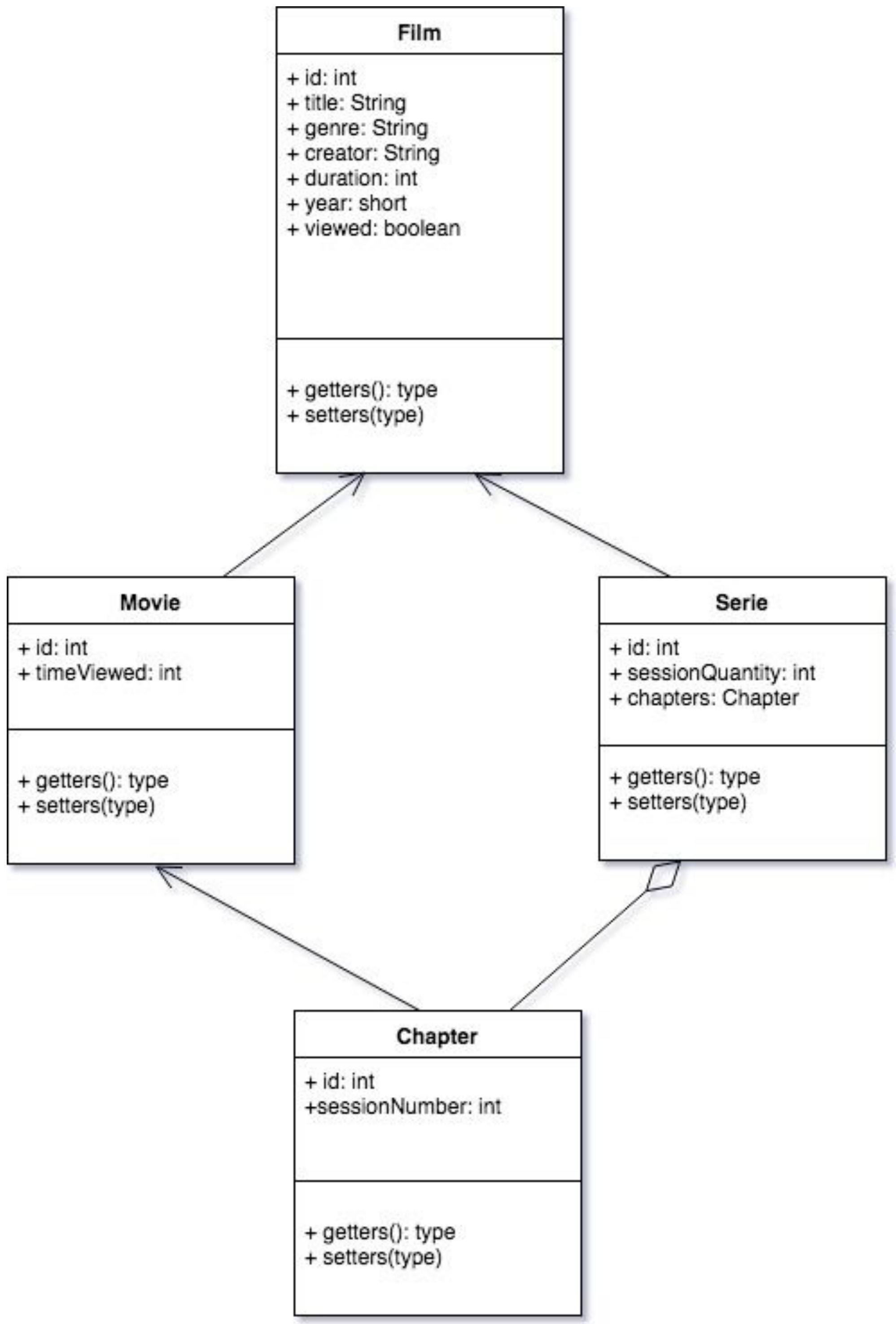
CLASES ABSTRACTAS

```
class Triangulo extends Figura {

    abstract void dibujate();

}
```

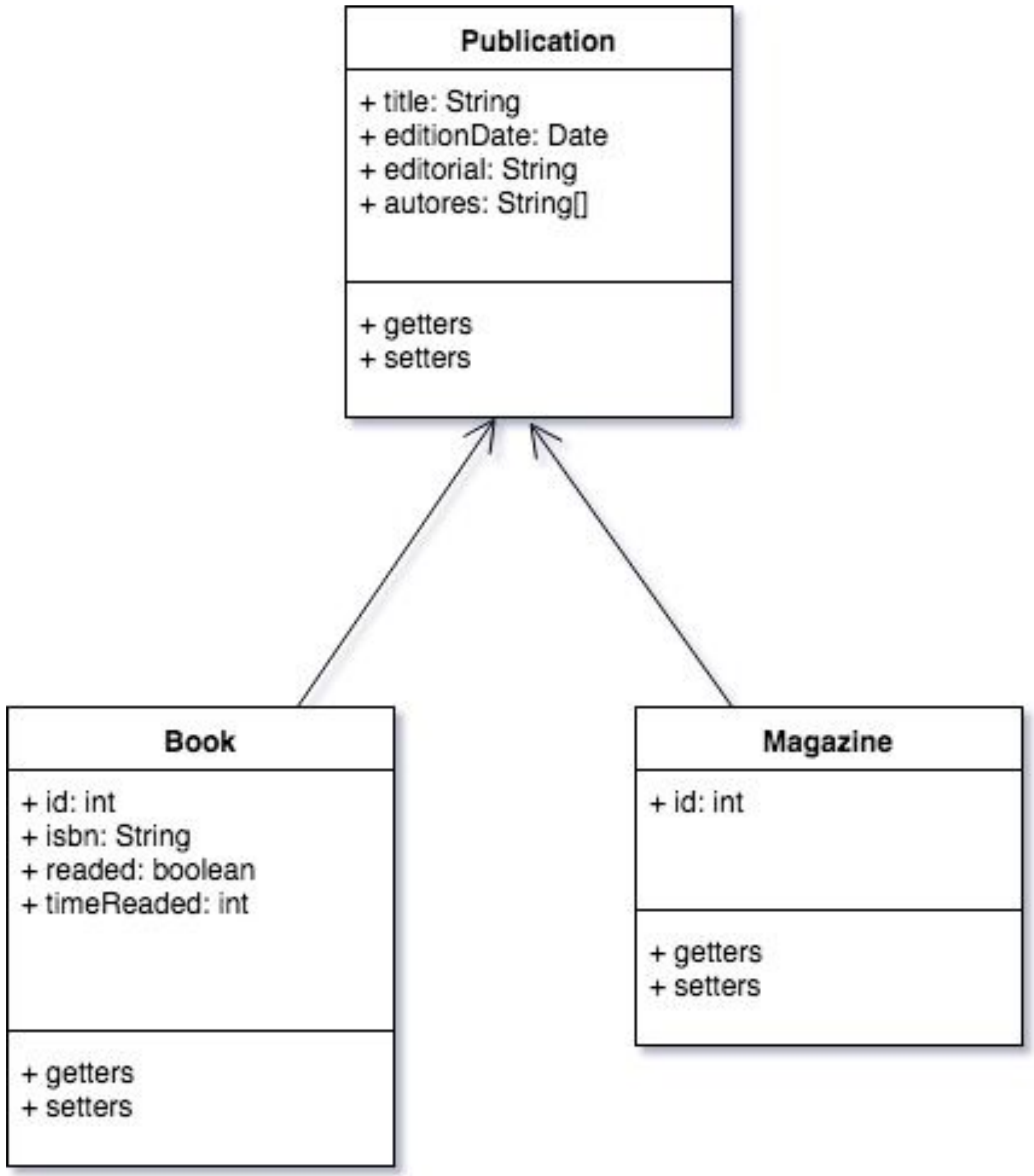
CLASES ABSTRACTAS

Nuestro proyecto

view()


Film
+ id: int
+ title: String
+ genre: String
+ creator: String
+ duration: int
+ year: short
+ viewed: boolean
+ getters(): type
+ setters(type)
Movie
+ id: int
+ timeViewed: int
+ getters(): type
+ setters(type)
Serie
+ id: int
+ sessionQuantity: int
+ chapters: Chapter
+ getters(): type
+ setters(type)
Chapter
+ id: int
+sessionNumber: int
+ getters(): type
+ setters(type)

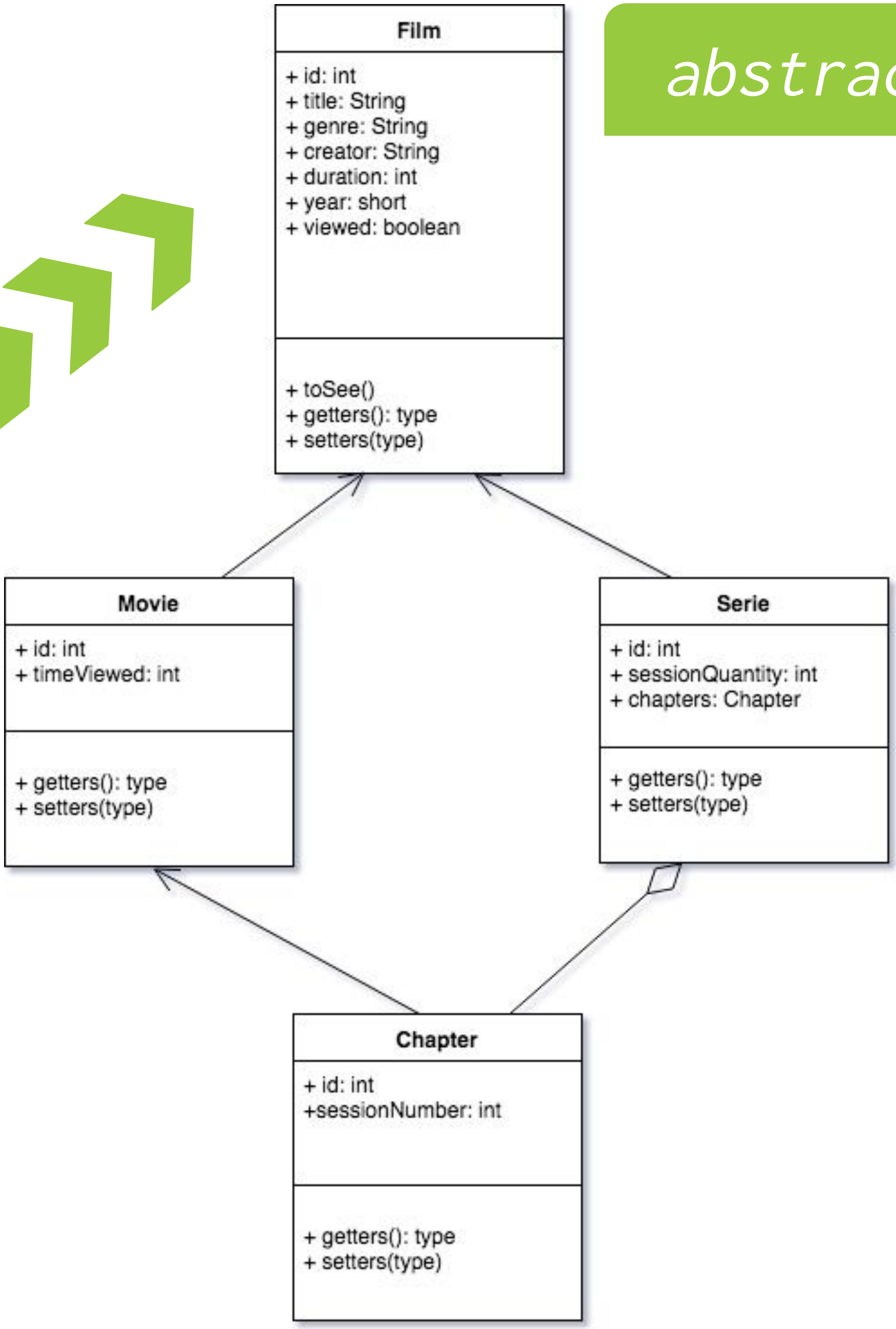
view()



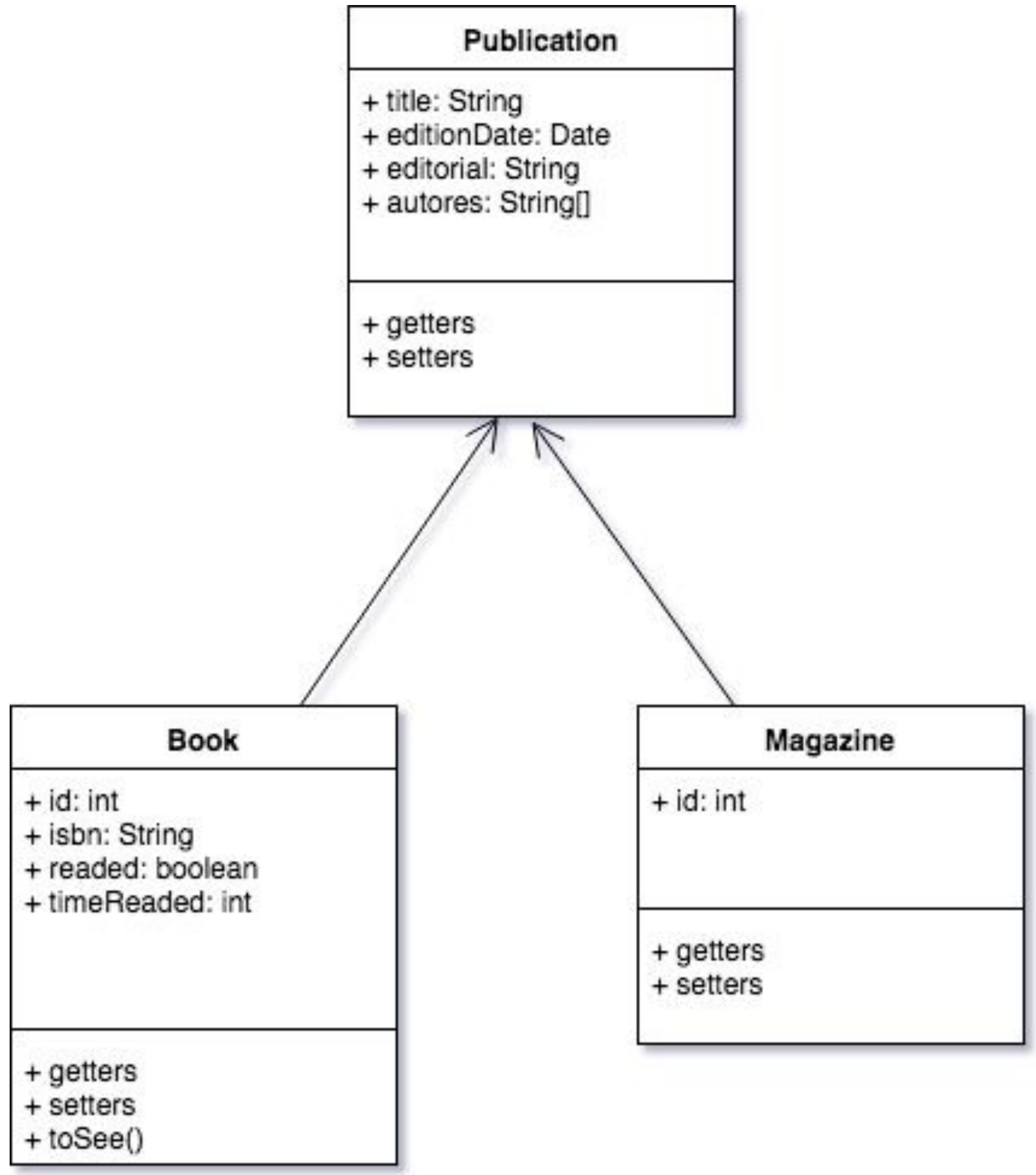
Publication
+ title: String
+ editionDate: Date
+ editorial: String
+ autores: String[]
+ getters
+ setters
Book
+ id: int
+ isbn: String
+ readed: boolean
+ timeReaded: int
+ getters
+ setters
Magazine
+ id: int
+ getters
+ setters

view()

abstract



view()



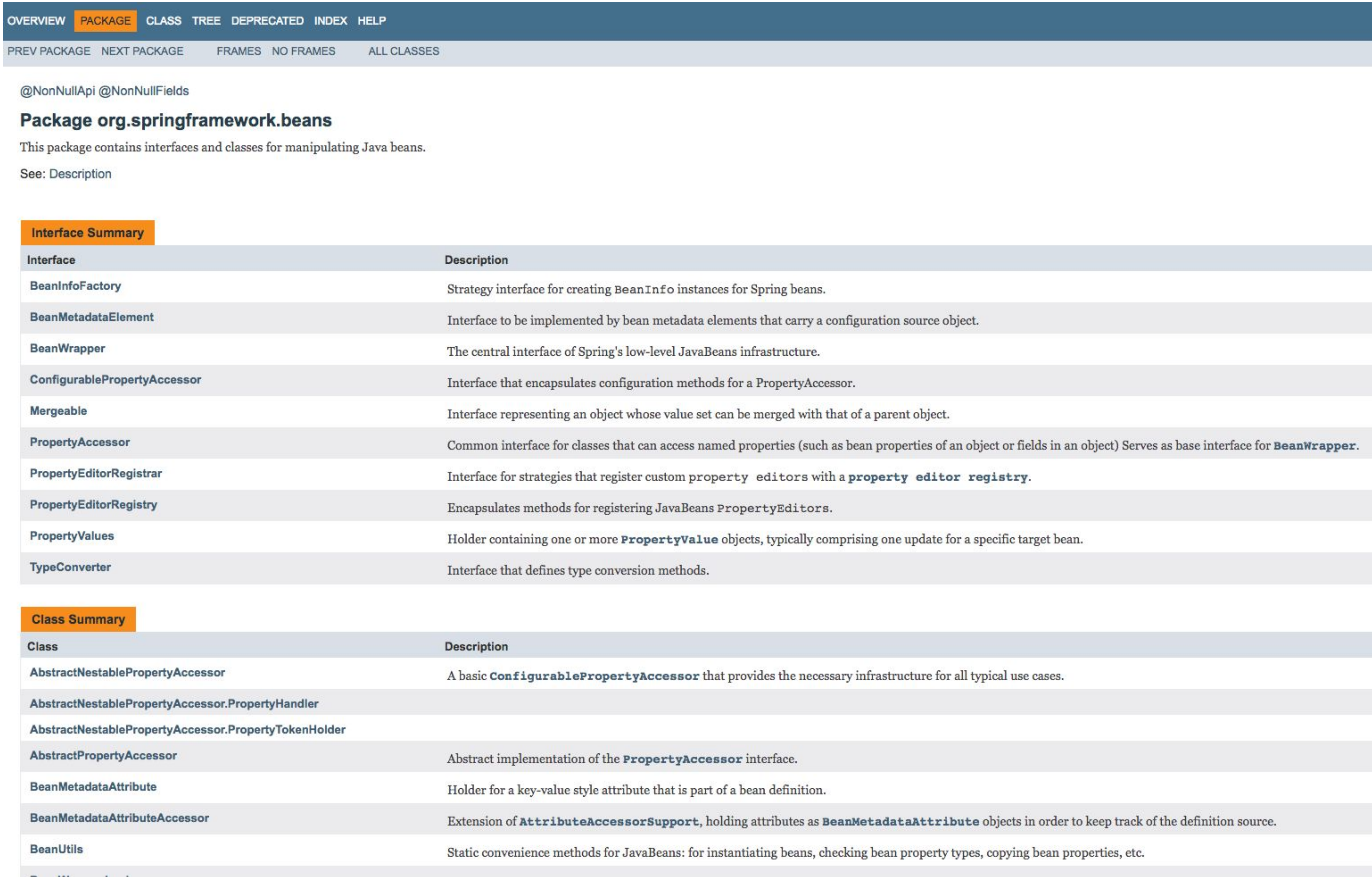
Publication
+ title: String
+ editionDate: Date
+ editorial: String
+ autores: String[]
+ getters
+ setters
Book
+ id: int
+ isbn: String
+ readed: boolean
+ timeReaded: int
+ getters
+ setters
+ toSee()
Magazine
+ id: int
+ getters
+ setters

JavaDoc

Javadoc

Generar **documentación** en **HTML** desde el código Java.

Spring



OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP
PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES ALL CLASSES

@NonNullApi @NonNullFields

Package org.springframework.beans

This package contains interfaces and classes for manipulating Java beans.

See: Description

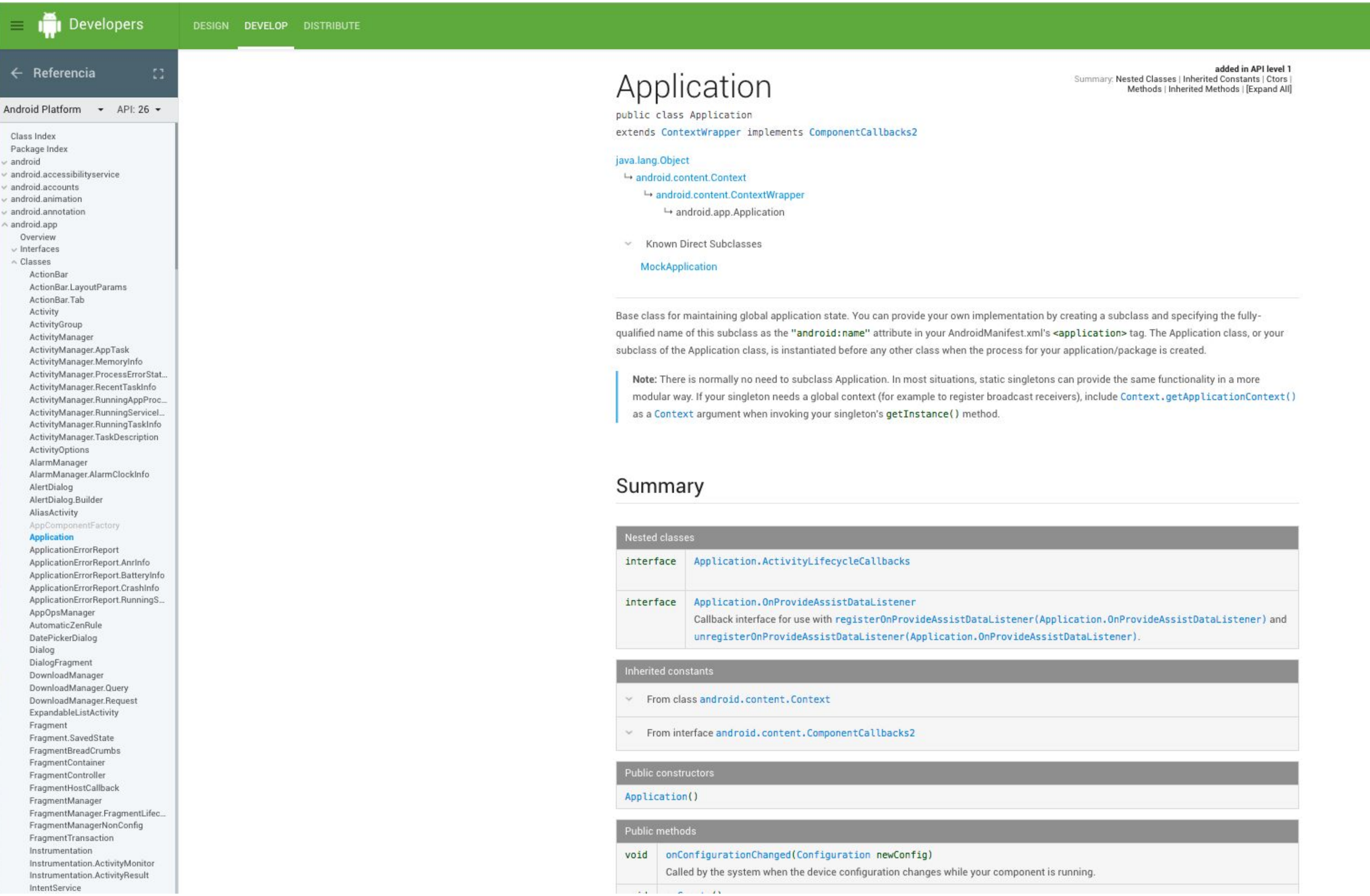
Interface Summary

Interface	Description
BeanInfoFactory	Strategy interface for creating `BeanInfo` instances for Spring beans.
BeanMetadataElement	Interface to be implemented by bean metadata elements that carry a configuration source object.
BeanWrapper	The central interface of Spring's low-level JavaBeans infrastructure.
ConfigurablePropertyAccessor	Interface that encapsulates configuration methods for a PropertyAccessor.
Mergeable	Interface representing an object whose value set can be merged with that of a parent object.
PropertyAccessor	Common interface for classes that can access named properties (such as bean properties of an object or fields in an object) Serves as base interface for `BeanWrapper`.
PropertyEditorRegistrar	Interface for strategies that register custom `property editors` with a `property editor registry`.
PropertyEditorRegistry	Encapsulates methods for registering JavaBeans `PropertyEditors`.
PropertyValues	Holder containing one or more `PropertyValue` objects, typically comprising one update for a specific target bean.
TypeConverter	Interface that defines type conversion methods.

Class Summary

Class	Description
AbstractNestablePropertyAccessor	A basic `ConfigurablePropertyAccessor` that provides the necessary infrastructure for all typical use cases.
AbstractNestablePropertyAccessor.PropertyHandler	
AbstractNestablePropertyAccessor.PropertyTokenHolder	
AbstractPropertyAccessor	Abstract implementation of the `PropertyAccessor` interface.
BeanMetadataAttribute	Holder for a key-value style attribute that is part of a bean definition.
BeanMetadataAttributeAccessor	Extension of `AttributeAccessorSupport`, holding attributes as `BeanMetadataAttribute` objects in order to keep track of the definition source.
BeanUtils	Static convenience methods for JavaBeans: for instantiating beans, checking bean property types, copying bean properties, etc.

Android



Developers DESIGN DEVELOP DISTRIBUTE

Referencia

Android Platform API: 26

Class Index
Package Index
android
android.accessibilityservice
android.accounts
android.animation
android.annotation
android.app
Overview
Interfaces
Classes
ActionBar
ActionBar.LayoutParams
ActionBar.Tab
Activity
ActivityGroup
ActivityManager
ActivityManager.AppTask
ActivityManager.MemoryInfo
ActivityManager.ProcessErrorStat...
ActivityManager.RecentTaskInfo
ActivityManager.RunningAppProc...
ActivityManager.RunningServiceI...
ActivityManager.RunningTaskInfo
ActivityManager.TaskDescription
ActivityOptions
AlarmManager
AlarmManager.AlarmClockInfo
AlertDialog
AlertDialog.Builder
AliasActivity
AppComponentFactory
Application
ApplicationErrorReport
ApplicationErrorReport.AnrInfo
ApplicationErrorReport.BatteryInfo
ApplicationErrorReport.CrashInfo
ApplicationErrorReport.RunningS...
AppOpsManager
AutomaticZenRule
DatePickerDialog
Dialog
DialogFragment
DownloadManager
DownloadManager.Query
DownloadManager.Request
ExpandableListActivity
Fragment
Fragment.SavedState
FragmentBreadCrumbs
FragmentContainer
FragmentController
FragmentHostCallback
FragmentManager
FragmentManager.FragmentLifec...
FragmentManagerNonConfig
FragmentTransaction
Instrumentation
Instrumentation.ActivityMonitor
Instrumentation.ActivityResult
IntentService

added in API level 1
Summary: Nested Classes | Inherited Constants | Ctors | Methods | Inherited Methods | [Expand All]

Application

`public class Application`
`extends ContextWrapper implements ComponentCallbacks2`

java.lang.Object
↳ android.content.Context
↳ android.content.ContextWrapper
↳ android.app.Application

Known Direct Subclasses

MockApplication

Base class for maintaining global application state. You can provide your own implementation by creating a subclass and specifying the fully-qualified name of this subclass as the `"android:name"` attribute in your AndroidManifest.xml's `<application>` tag. The Application class, or your subclass of the Application class, is instantiated before any other class when the process for your application/package is created.

Note: There is normally no need to subclass Application. In most situations, static singletons can provide the same functionality in a more modular way. If your singleton needs a global context (for example to register broadcast receivers), include `Context.getApplicationContext()` as a `Context` argument when invoking your singleton's `getInstance()` method.

Summary

Nested classes	
`interface`	`Application.ActivityLifecycleCallbacks`
`interface`	`Application.OnProvideAssistDataListener` Callback interface for use with `registerOnProvideAssistDataListener(Application.OnProvideAssistDataListener)` and `unregisterOnProvideAssistDataListener(Application.OnProvideAssistDataListener)`.

Inherited constants
From class `android.content.Context`
From interface `android.content.ComponentCallbacks2`

Public constructors
`Application()`

Public methods	
`void`	`onConfigurationChanged(Configuration newConfig)` Called by the system when the device configuration changes while your component is running.

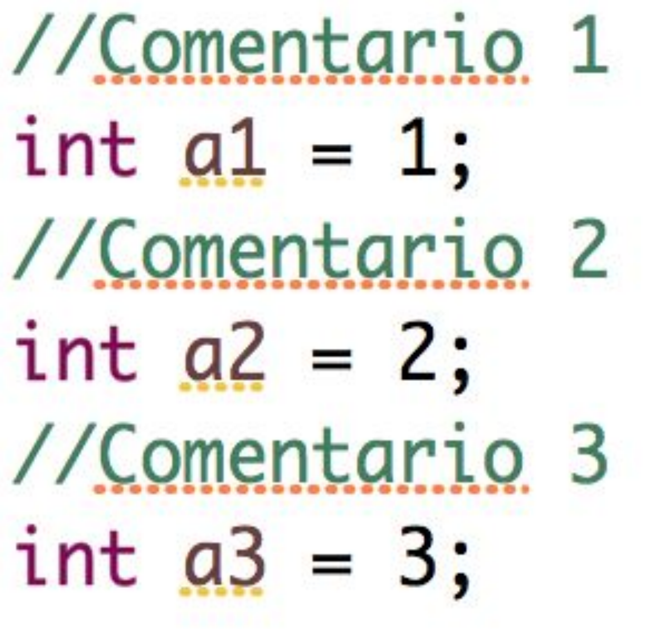
Comentarios

```
// Soy un comentario :)
```

3 formas de poner comentarios

// un comentario

Todo lo que esté en esa línea será ignorado por la computadora



```
//Comentario 1
int a1 = 1;
//Comentario 2
int a2 = 2;
//Comentario 3
int a3 = 3;
```

Comentario en una línea

3 formas de poner comentarios

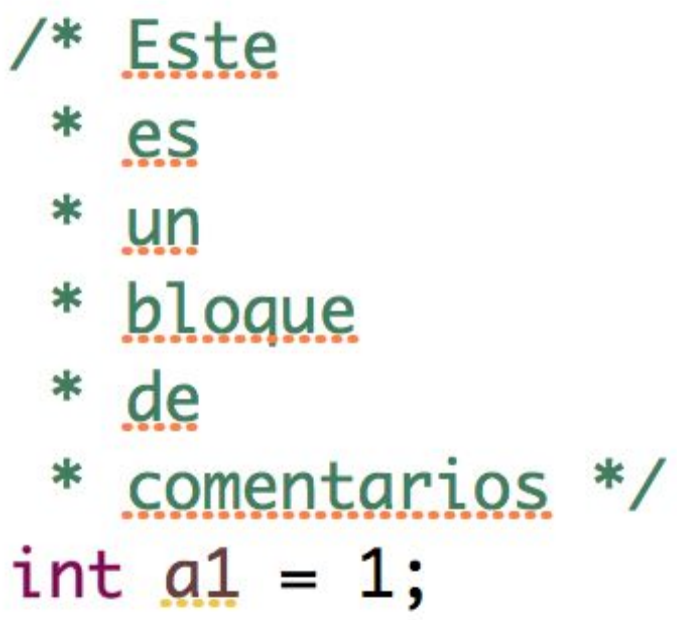
● // un comentario

```
Todo lo que esté en esa línea será
ignorado por la computadora
```

● **/* un bloque de comentarios */**

```
Todo lo que esté dentro será ignorado
```

●



```
/* Este
 * es
 * un
 * bloque
 * de
 * comentarios */
int a1 = 1;
```

Bloque de comentarios

3 formas de poner comentarios

● // un comentario

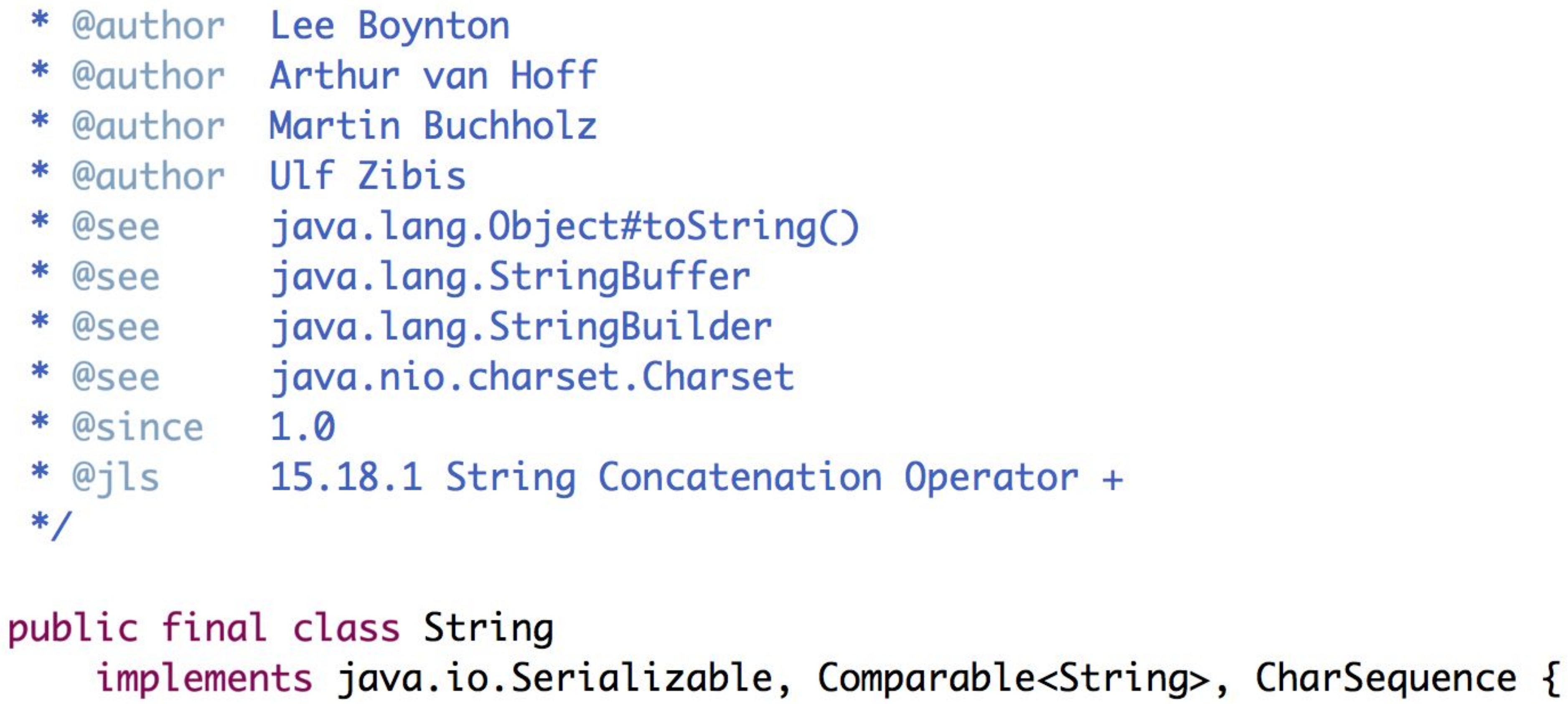
`Todo lo que esté en esa línea será ignorado por la computadora`

● /* un bloque de comentarios */

`Todo lo que esté dentro será ignorado`

● **/** documentacion */**

`Todo lo que esté dentro será un comentario de documentación llamado doc comment`



```
 * @author  Lee Boynton
 * @author  Arthur van Hoff
 * @author  Martin Buchholz
 * @author  Ulf Zibis
 * @see     java.lang.Object#toString()
 * @see     java.lang.StringBuffer
 * @see     java.lang.StringBuilder
 * @see     java.nio.charset.Charset
 * @since   1.0
 * @jls     15.18.1 String Concatenation Operator +
 */

public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {
```

JavaDoc comment

```
/**
* [descripción corta]
* <p>
* [descripción larga]
*
* [author, version, params,
returns, throws, see, other tags]
* [see also]
*/
```

Ejemplo

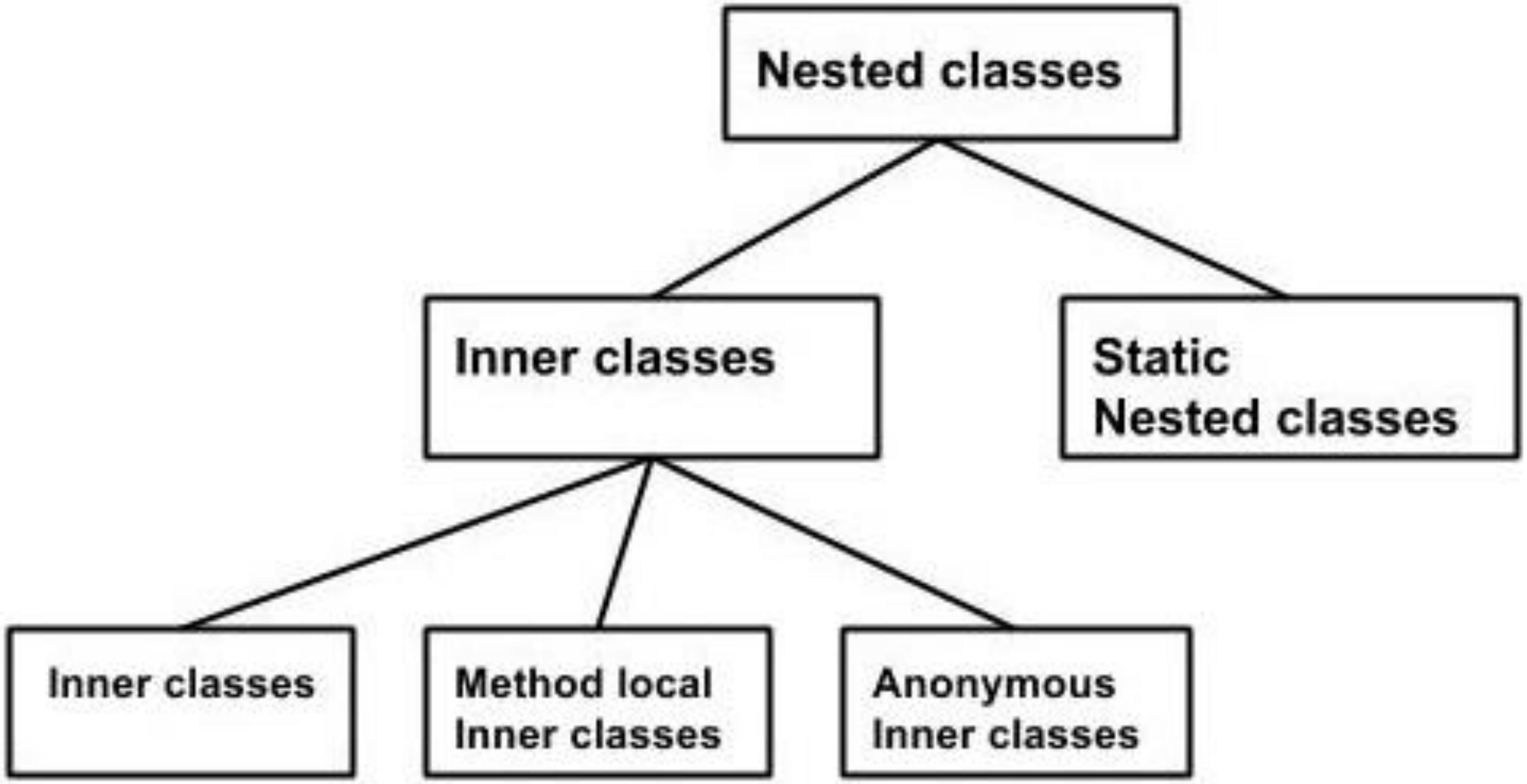
Platzi

Clases Avanzadas

CLASES ANIDADAS

```
class ClaseExterior {

    class ClaseAnidada {

    }

}
```

CLASES ANIDADAS


Nested classes
Inner classes
Static Nested classes
Inner classes
Method local Inner classes
Anonymous Inner classes

```
class ClaseExterior {
    static class ClaseStaticaAnidada {
    }
    class ClaseInterna {
    }
}
```

CLASES ANIDADAS

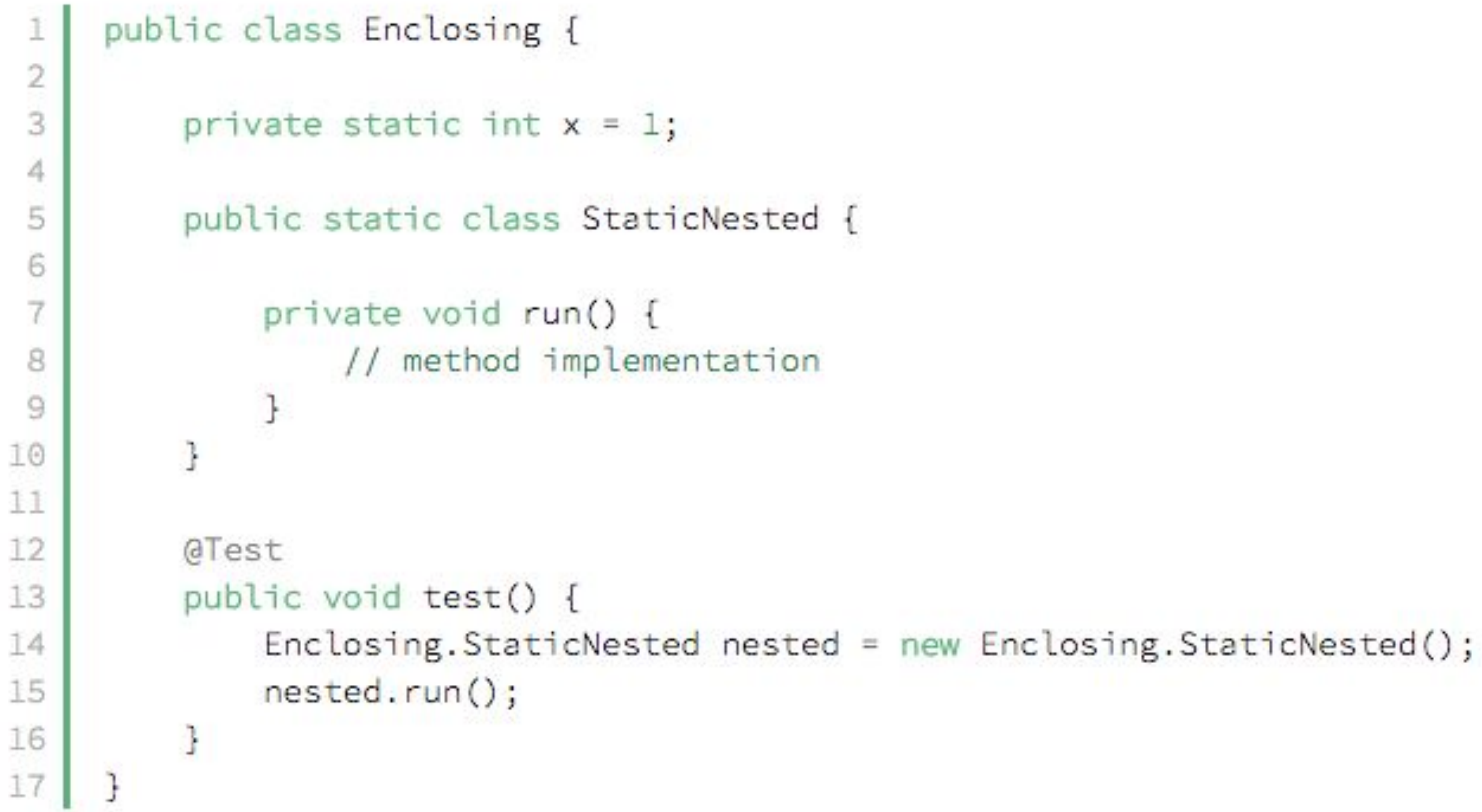
Clases anidadas

`static`

- Estáticas
- No Estáticas

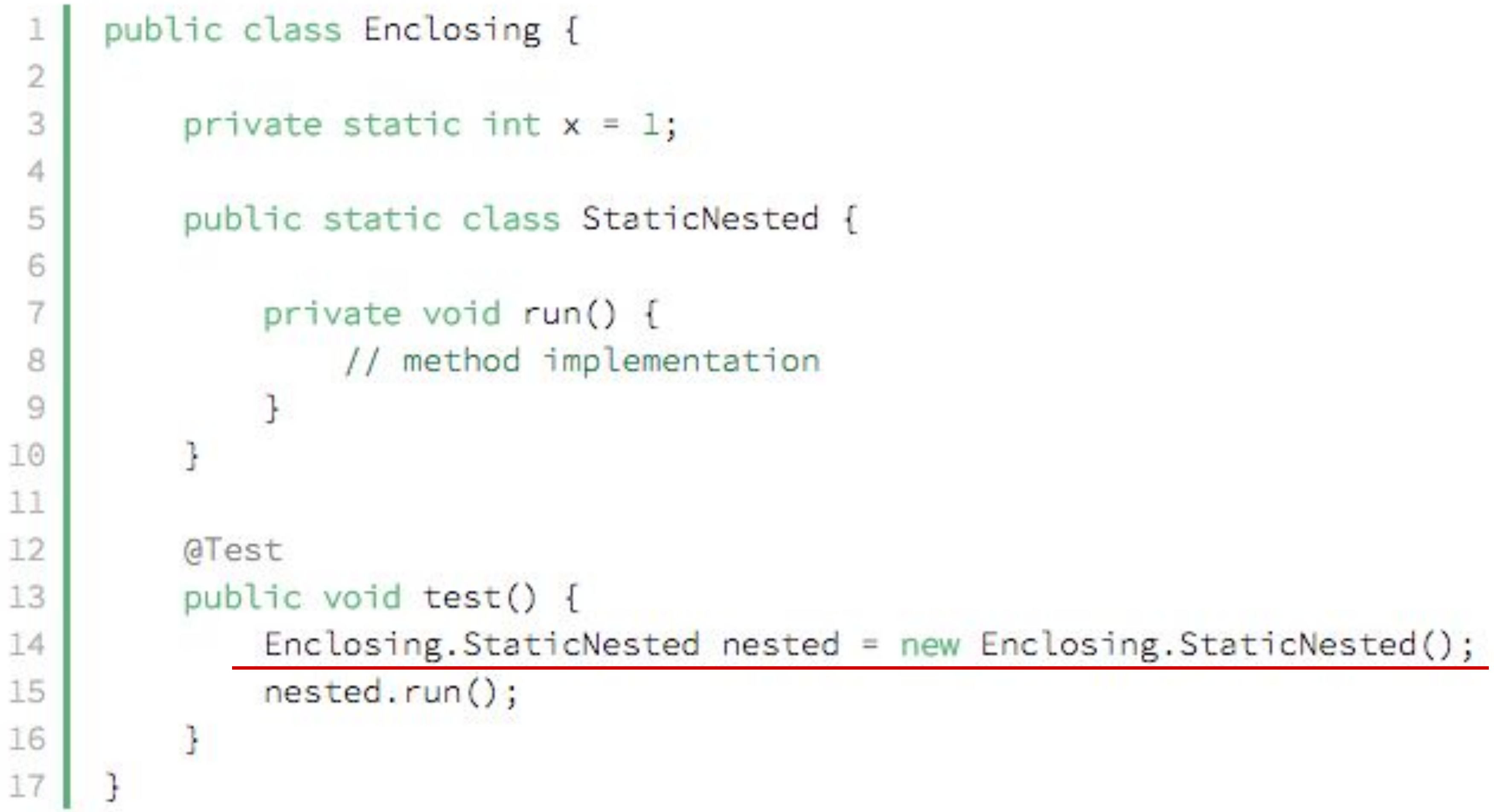
Clases estáticas

No se necesitan crear instancias para llamarlas



```java
public class Enclosing {

    private static int x = 1;

    public static class StaticNested {

        private void run() {
            // method implementation
        }
    }

    @Test
    public void test() {
        Enclosing.StaticNested nested = new Enclosing.StaticNested();
        nested.run();
    }
}
```

Clases Estáticas

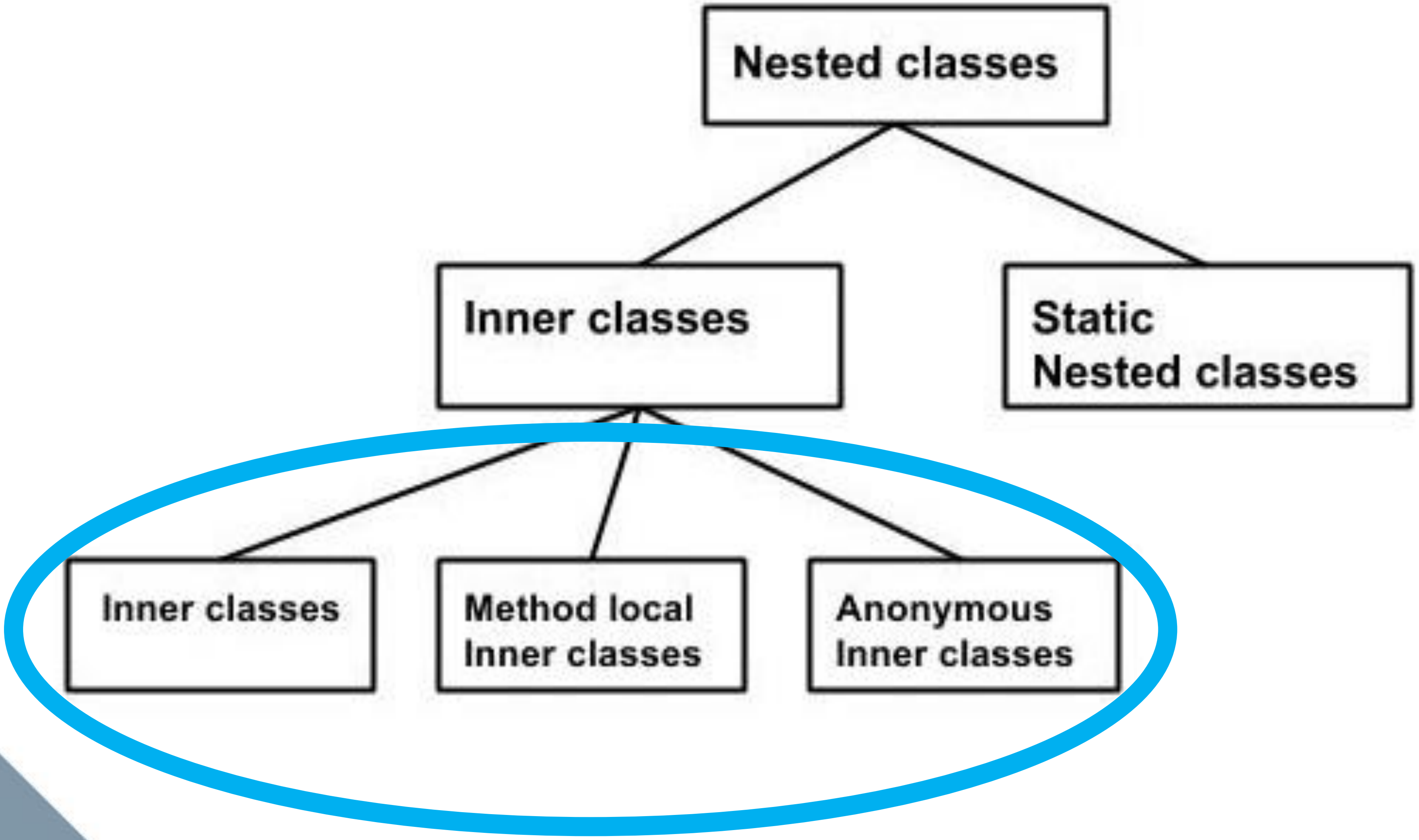


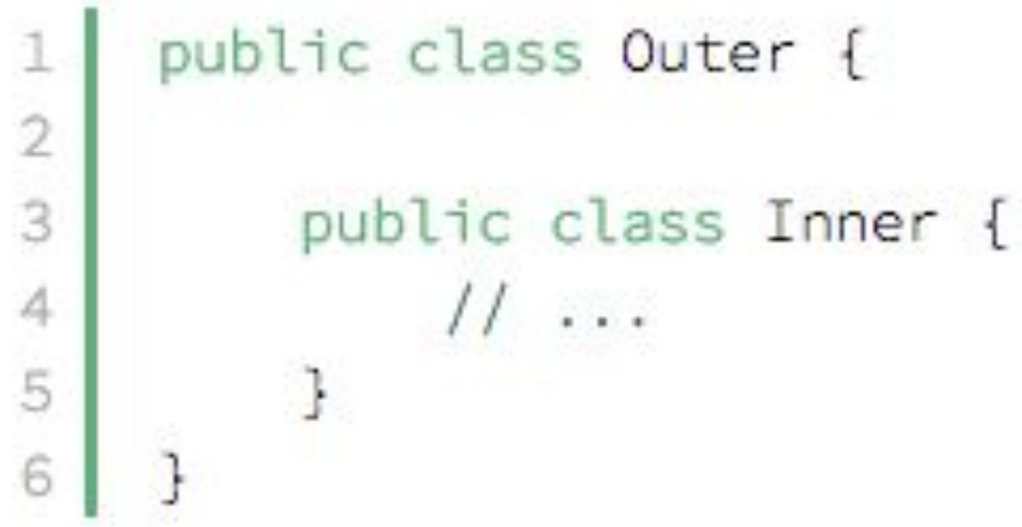
```java
public class Enclosing {

    private static int x = 1;

    public static class StaticNested {

        private void run() {
            // method implementation
        }
    }

    @Test
    public void test() {
        Enclosing.StaticNested nested = new Enclosing.StaticNested();
        nested.run();
    }
}
```

Clases Estáticas

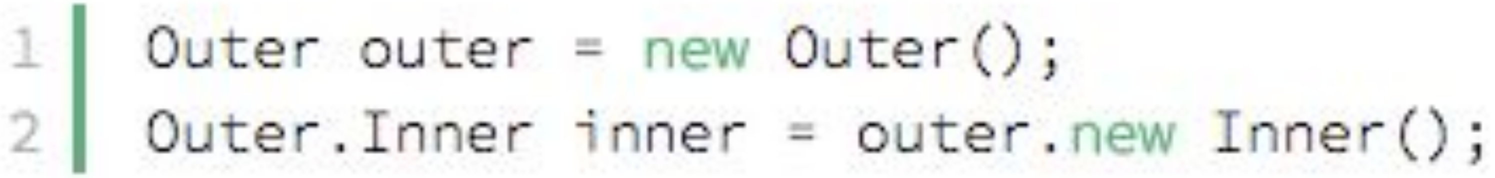
Clases estáticas

Solo se pueden llamar a los métodos estáticos


Nested classes
Inner classes
Static Nested classes
Inner classes
Method local Inner classes
Anonymous Inner classes

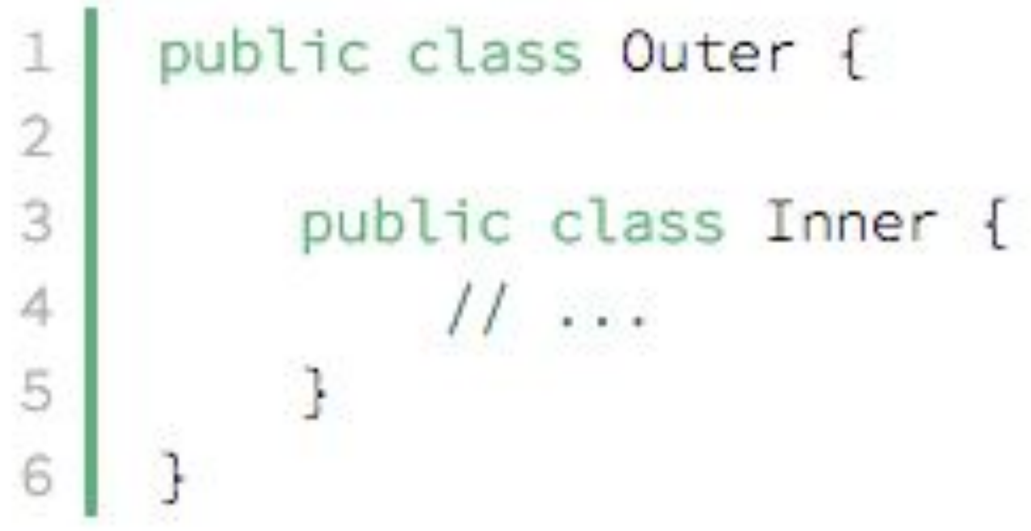


```java
public class Outer {

    public class Inner {
        // ...
    }
}
```

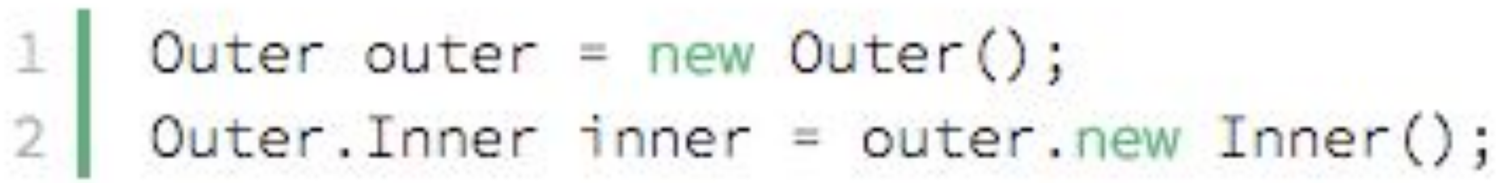


```java
Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
```

Clases Anidadas - Inner

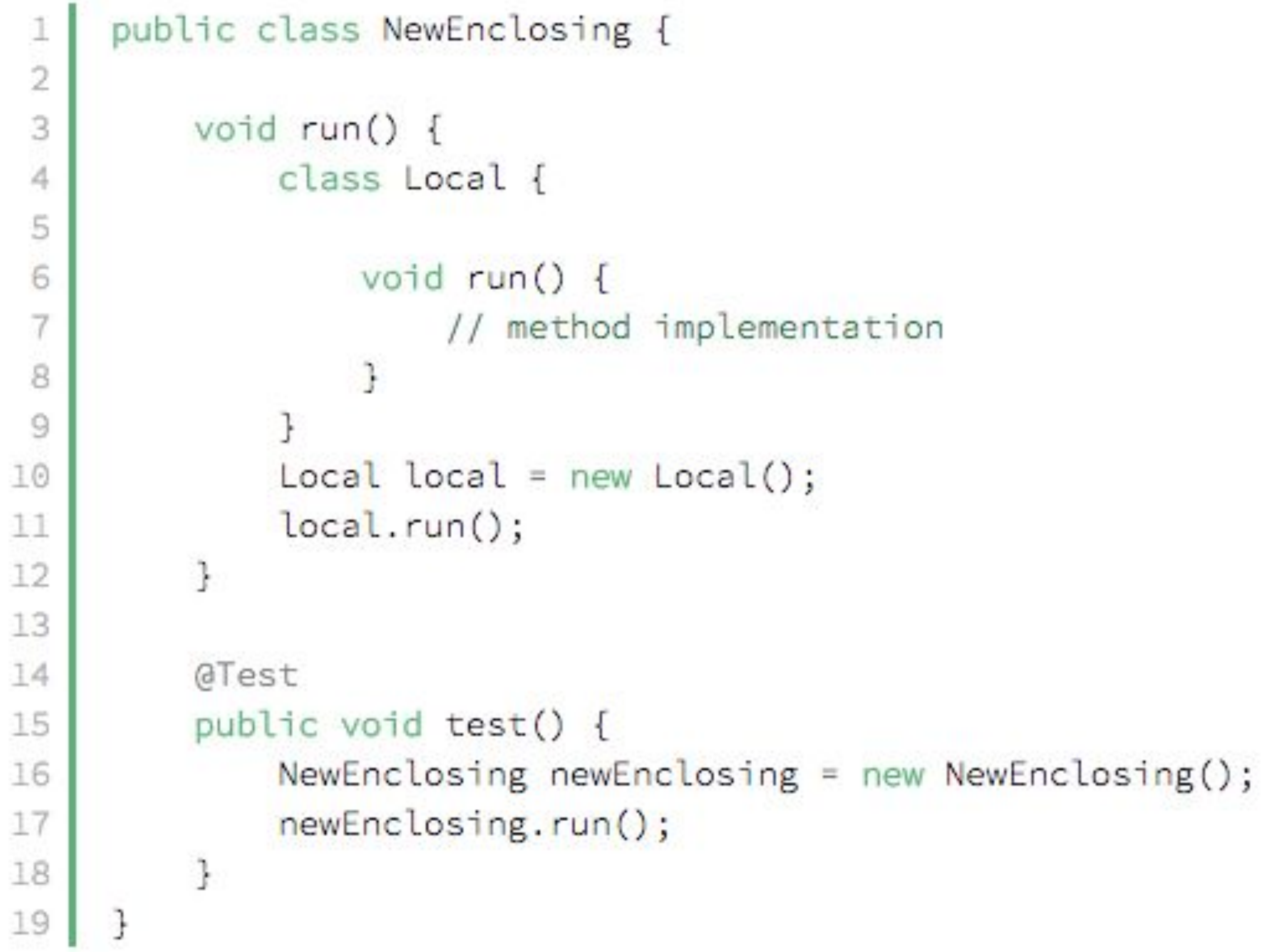


```java
public class Outer {

    public class Inner {
        // ...
    }
}
```



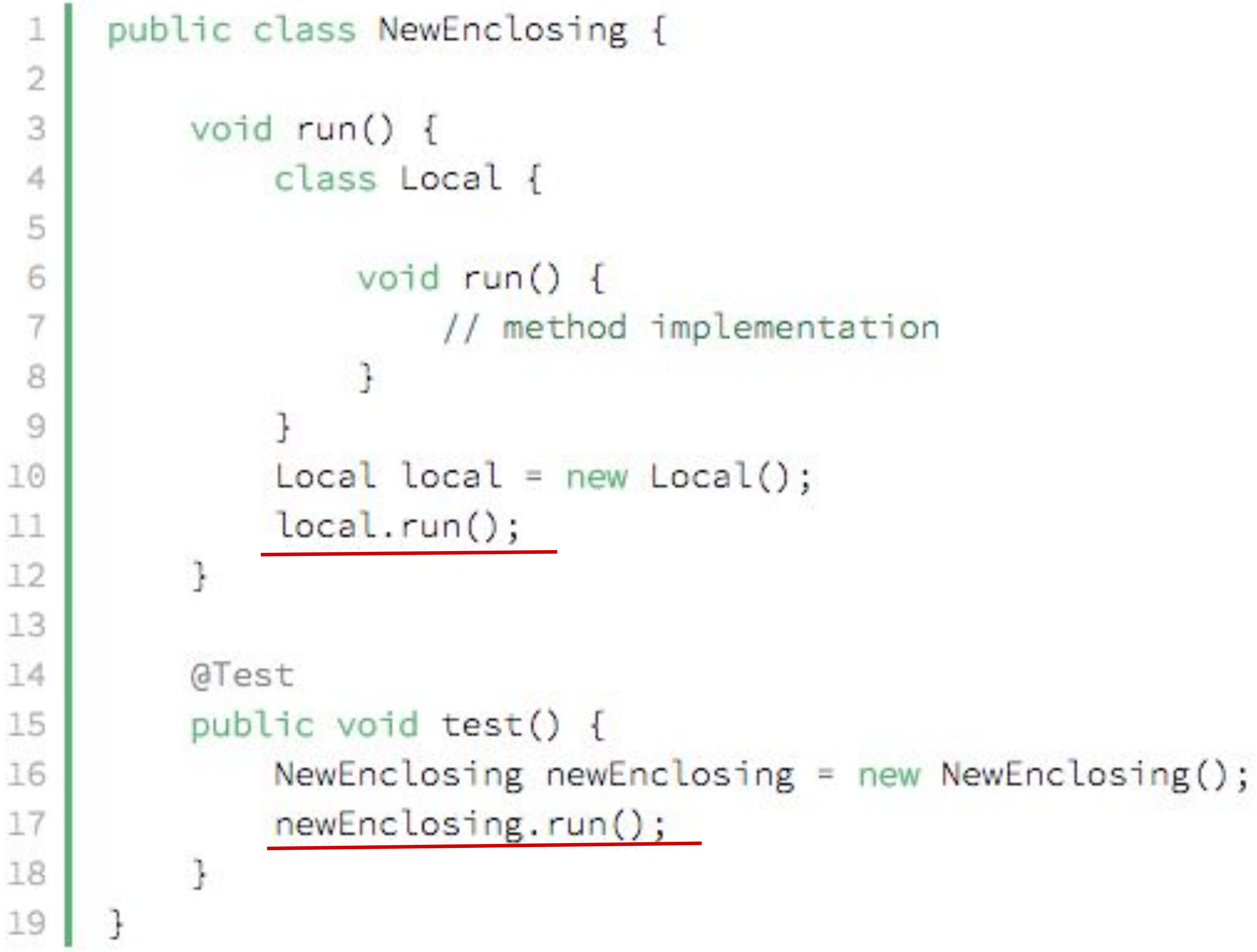
```java
Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
```

Clases Anidadas - Inner



```java
public class NewEnclosing {

    void run() {
        class Local {

            void run() {
                // method implementation
            }
        }
        Local local = new Local();
        local.run();
    }

    @Test
    public void test() {
        NewEnclosing newEnclosing = new NewEnclosing();
        newEnclosing.run();
    }
}
```

Clases Locales a Método



```java
public class NewEnclosing {

    void run() {
        class Local {

            void run() {
                // method implementation
            }
        }
        Local local = new Local();
        local.run();
    }

    @Test
    public void test() {
        NewEnclosing newEnclosing = new NewEnclosing();
        newEnclosing.run();
    }
}
```

Clases Locales a Método

```java
abstract class SimpleAbstractClass {
    abstract void run();
}
```

```java
public class AnonymousInnerTest {

    @Test
    public void whenRunAnonymousClass_thenCorrect() {
        SimpleAbstractClass simpleAbstractClass = new SimpleAbstractClass() {
            void run() {
                // method implementation
            }
        };
        simpleAbstractClass.run();
    }
}
```

Clases Anónimas

Clases anidadas

Clases Helper
Agrupadas por lógica
Encapsulación

Clases
Estáticas vs Anidadas

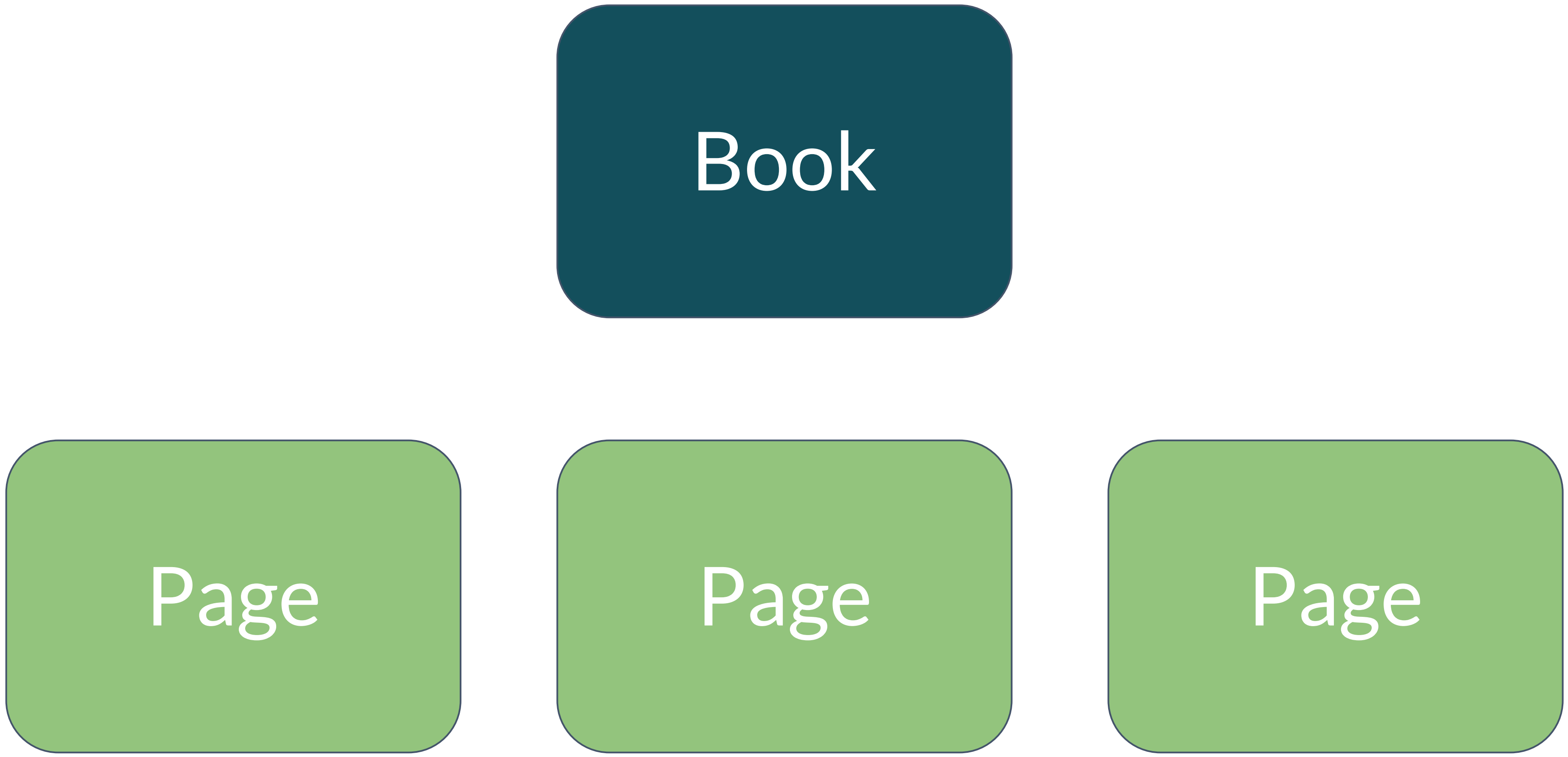
Estáticas solo podemos llamar a métodos y elementos de su misma naturaleza

Clases **Estáticas vs Anidadas**

Anidadas pueden llamar a cualquier tipo de elemento o método

Ejemplo

Book


Book
Page
Page
Page

Platzi

Interfaces Avanzadas

Interfaces avanzadas

Métodos Abstractos
Campos constantes

Interfaces avanzadas

Tipo de referencia
Polimorfismo similar Clases
Abstractas

Java 8 y 9

Java 8

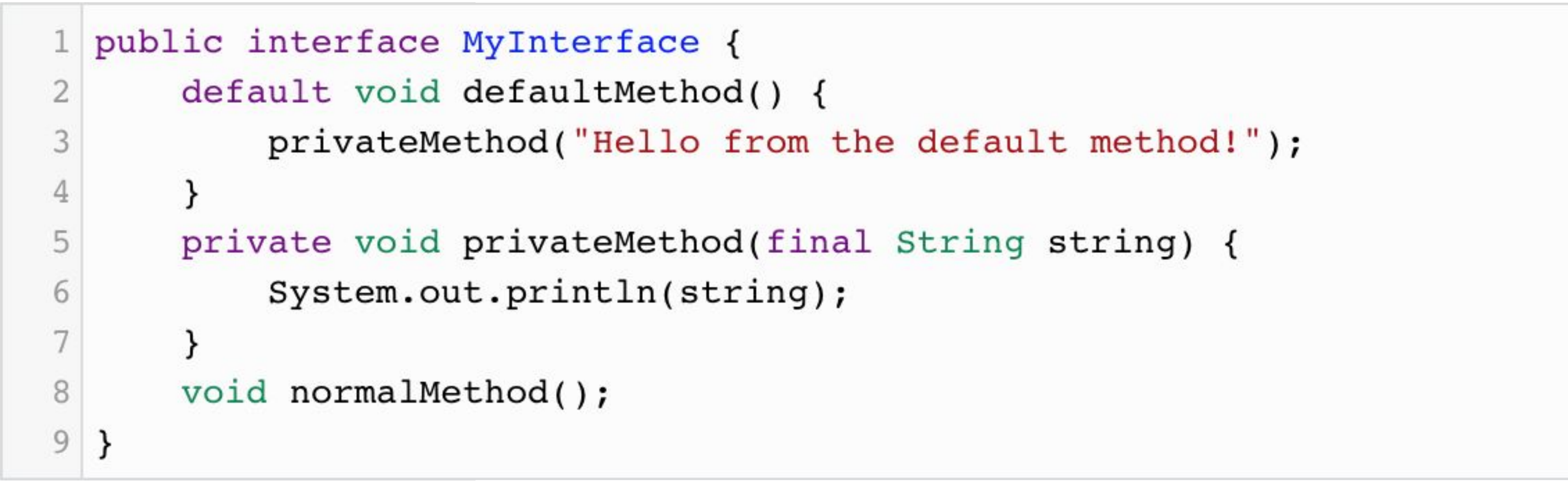
`default`

Java 9

`private`

Interfaces avanzadas

Ahora podemos tener
implementación en métodos



```java
public interface MyInterface {
    default void defaultMethod() {
        privateMethod("Hello from the default method!");
    }
    private void privateMethod(final String string) {
        System.out.println(string);
    }
    void normalMethod();
}
```

`default y private` Methods

DAO

Data Access Object

DAO - Data Access Object

Patrón de diseño

Métodos CRUD

(Create, Read, Update y Delete).

Ejemplo

Platzi

Interfaces Avanzadas

Interfaces Funcionales

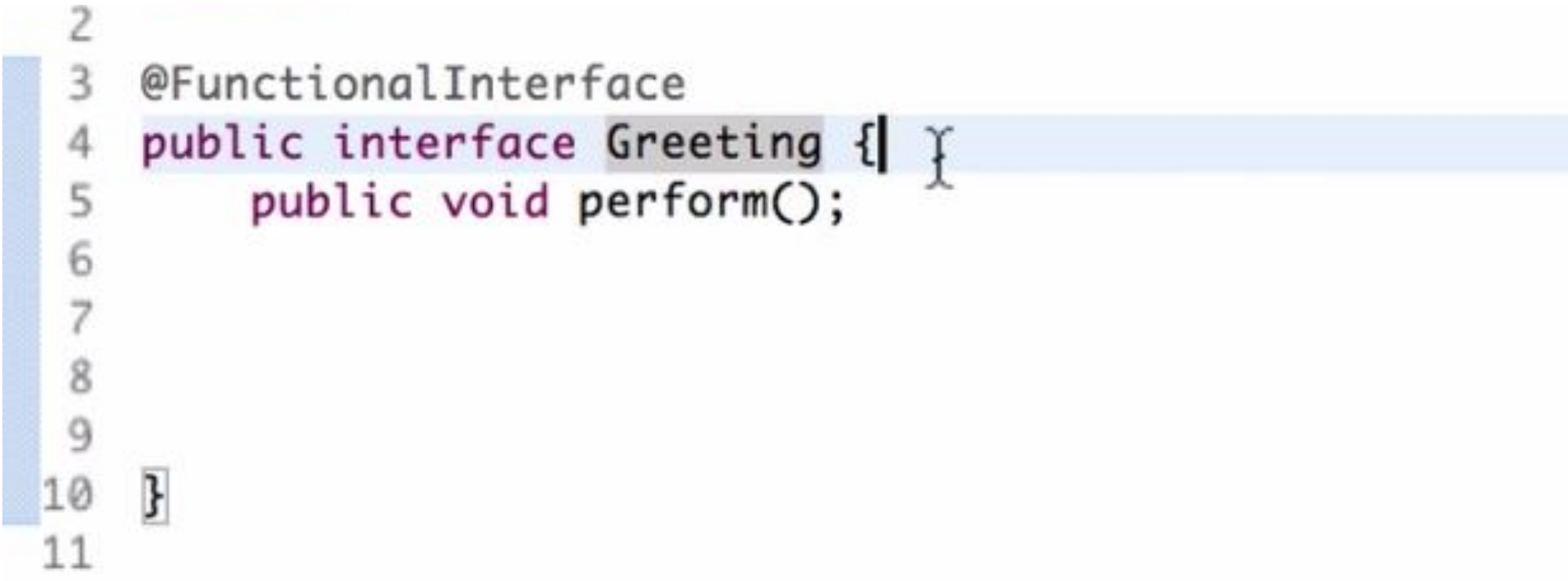
Interfaces funcionales

Tienen un solo método abstracto

SAM(Single Abstract Method)

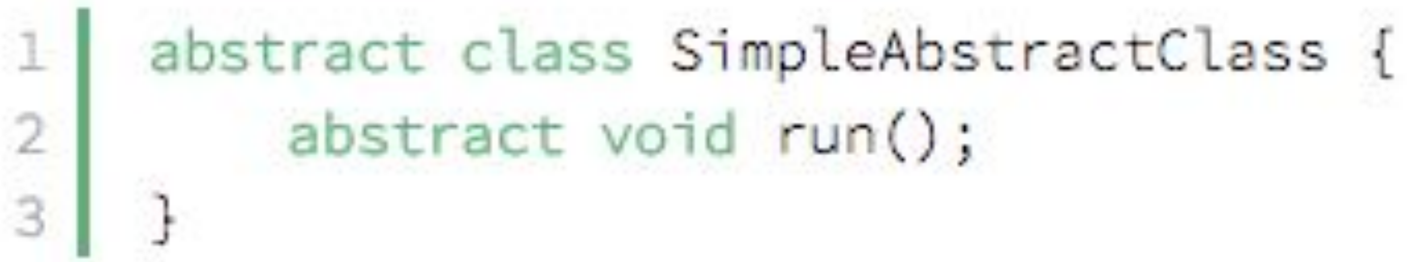
```
@FunctionalInterface
```

BUENA PRÁCTICA

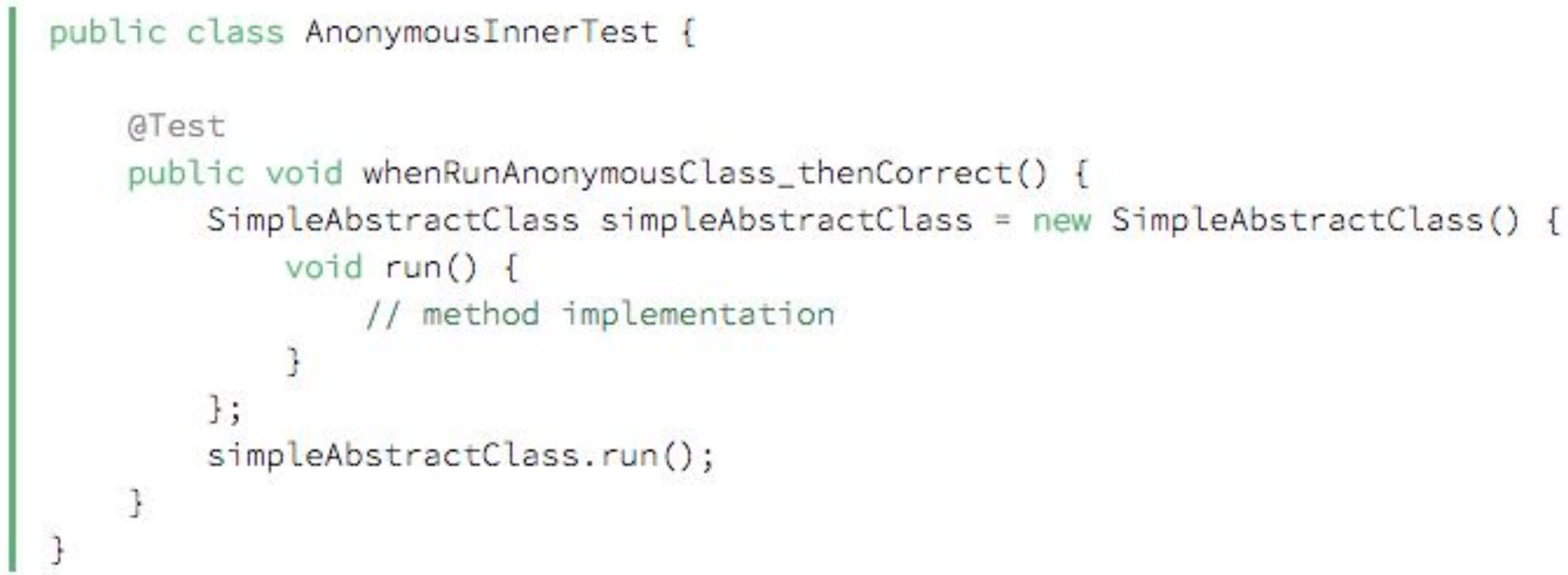


```java
@FunctionalInterface
public interface Greeting {
    public void perform();




}
```

@FunctionalInterface
SAM (Single Abstract Method)



```java
abstract class SimpleAbstractClass {
    abstract void run();
}
```



```java
public class AnonymousInnerTest {

    @Test
    public void whenRunAnonymousClass_thenCorrect() {
        SimpleAbstractClass simpleAbstractClass = new SimpleAbstractClass() {
            void run() {
                // method implementation
            }
        };
        simpleAbstractClass.run();
    }
}
```

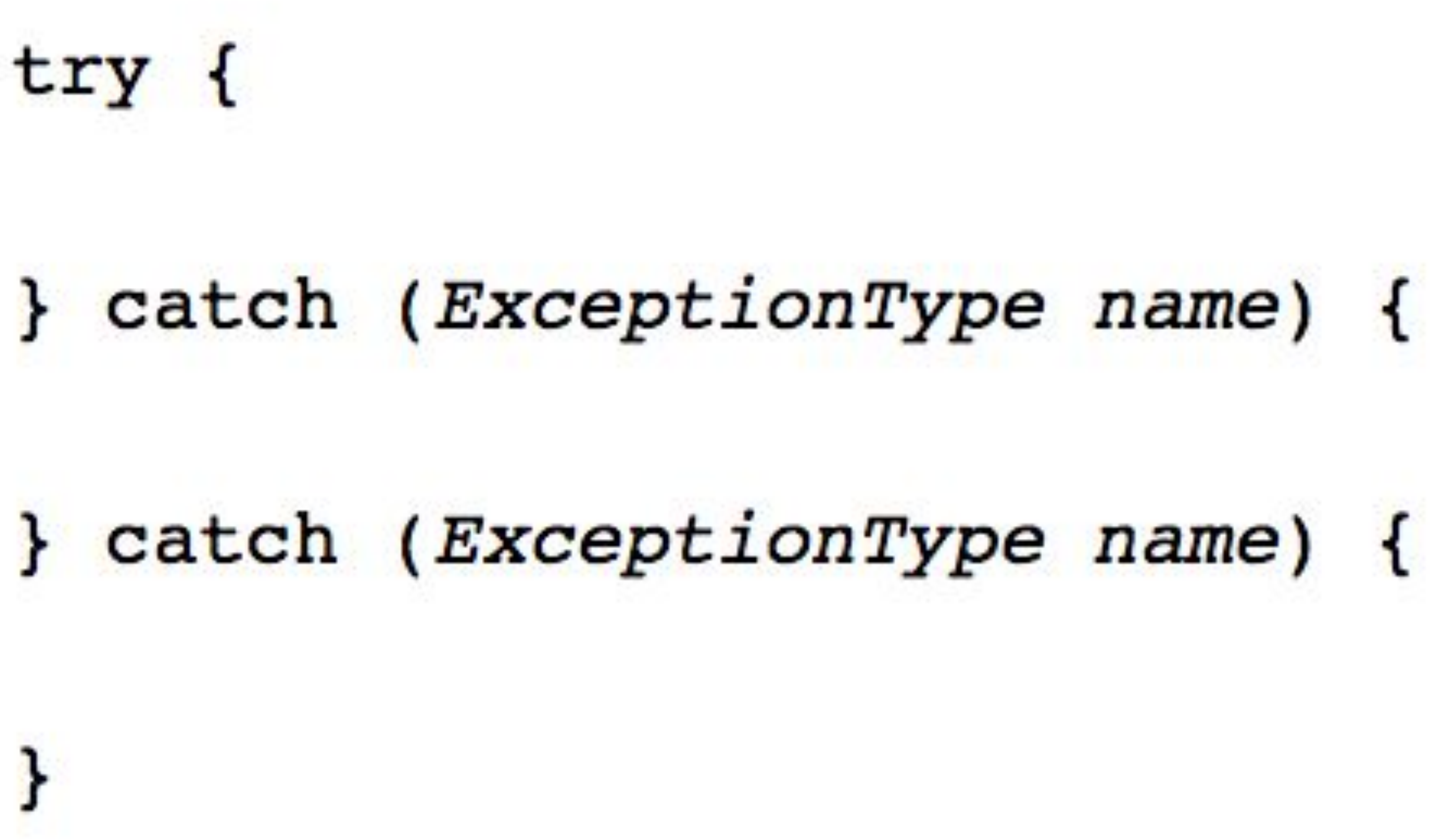
Clases Anónimas

Excepciones

Try-catch-finally

Excepciones

Manejar Excepciones significa que añadirás un bloque de código para manejar un error

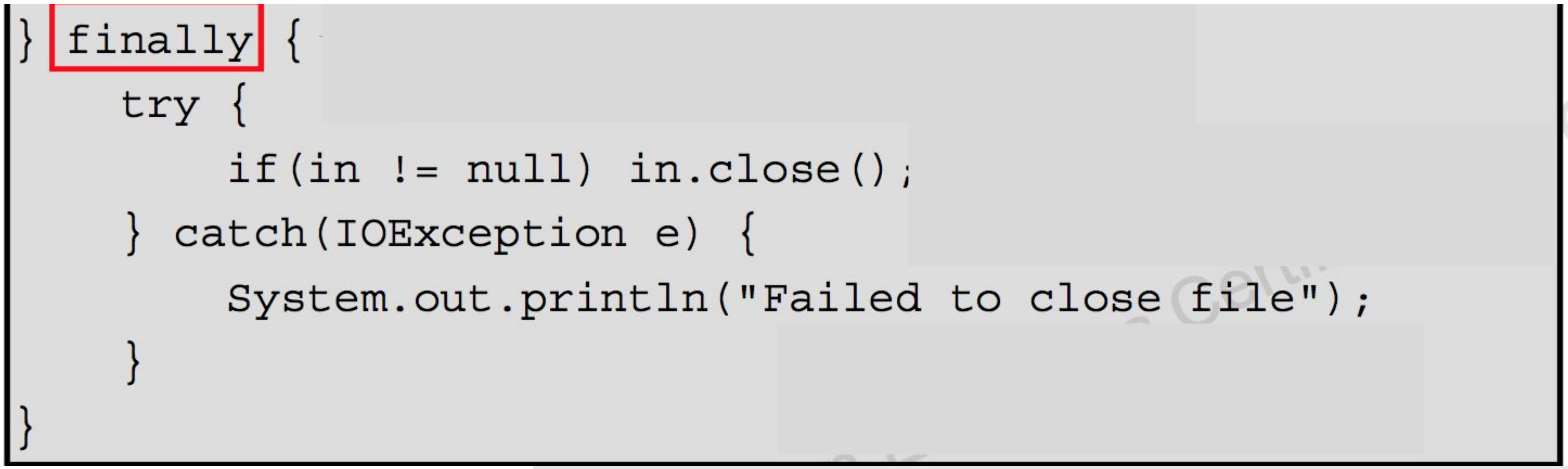


```
try {

} catch (ExceptionType name) {

} catch (ExceptionType name) {

}
```

try - catch

```
finally {
    if (out != null) {
        System.out.println("Closing PrintWriter");
        out.close();
    } else {
        System.out.println("PrintWriter not open");
    }
}
```

finally



```
} finally {
    try {
        if(in != null) in.close();
    } catch(IOException e) {
        System.out.println("Failed to close file");
    }
}
```

Cerrar recursos

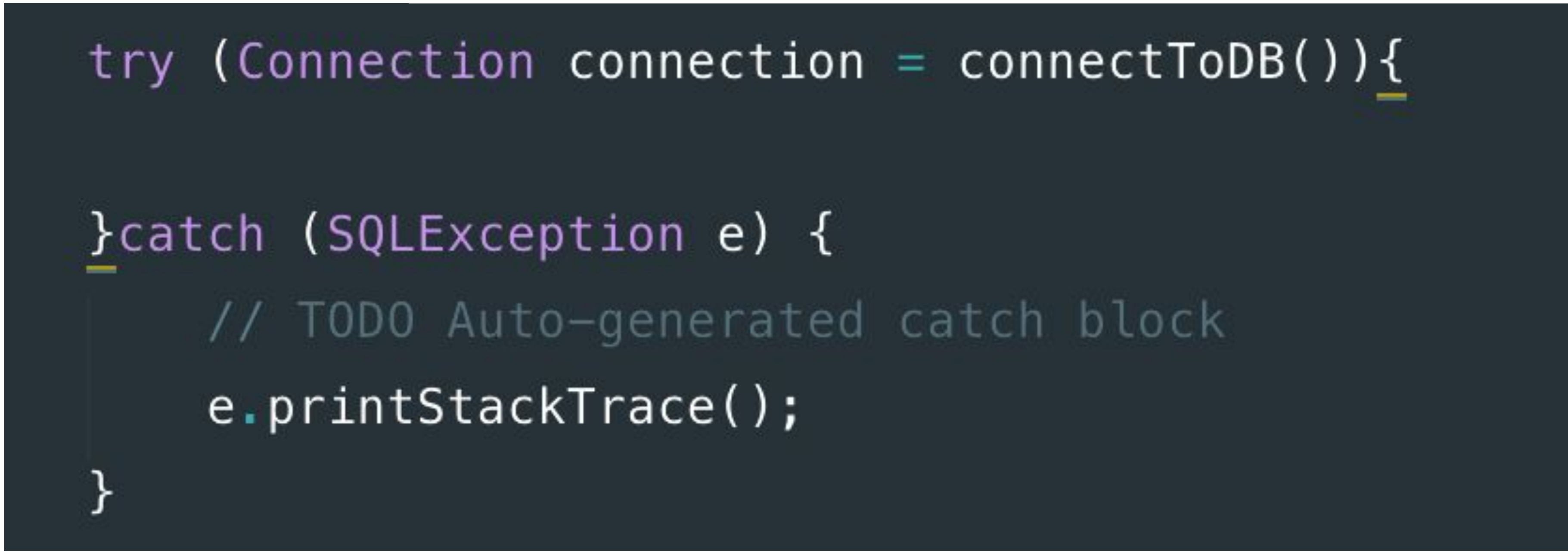
Try-with-resources

```
BufferedReader reader = newBufferedReader(new InputStreamReader(System.in));

try(BufferedReader r1 = reader) {
  //sentencias

} catch(Exception e) {
  //sentencias

}
```

Aquí se ve la variable `r1` que francamente estaría de más, ahora podemos ponerlo así:

```
BufferedReader reader = newBufferedReader(new InputStreamReader(System.in));

try(reader) {
  //sentencias

} catch(Exception e) {
  //sentencias

}
```

Cerrar recursos



```
try (Connection connection = connectToDB()){

}catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Cerrar recursos

JDBC

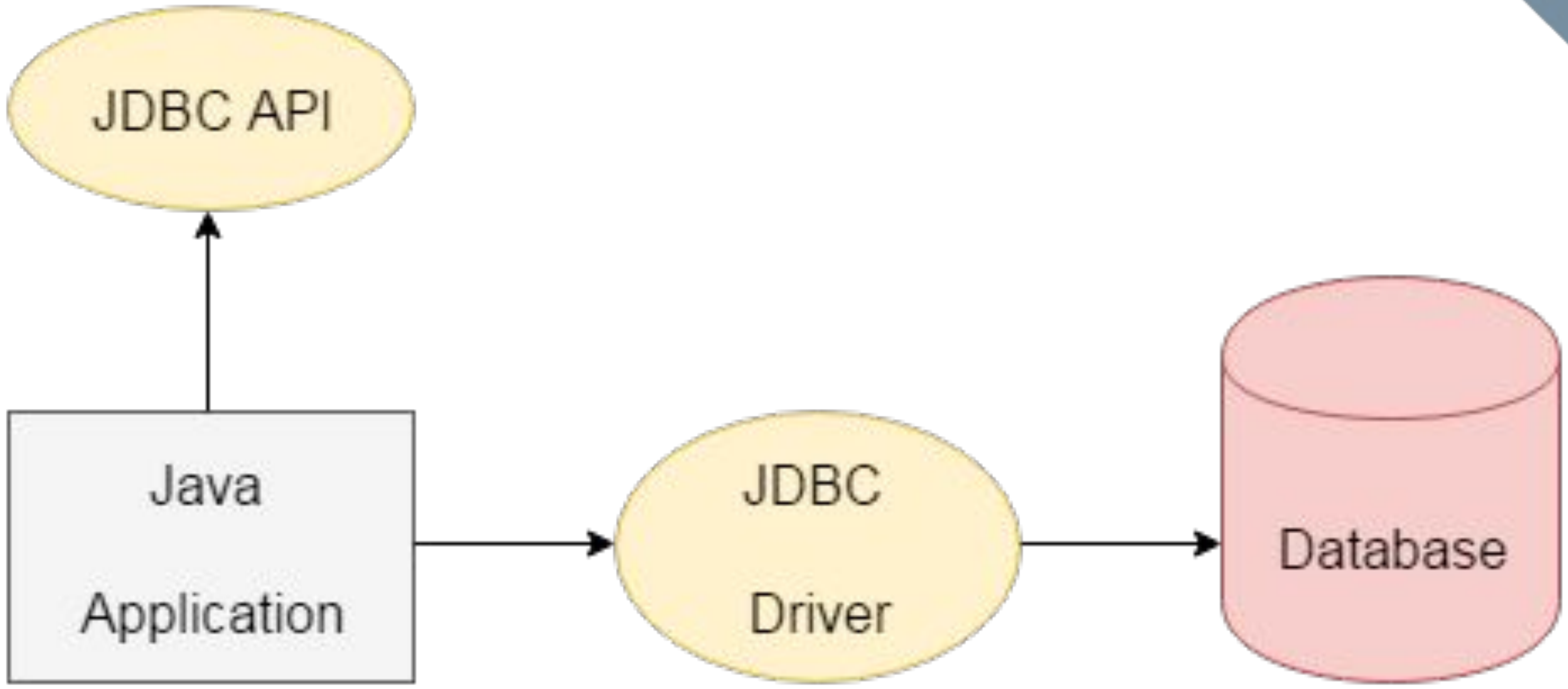
JDBC

Java Data Base Connectivity

JDBC

Es un API compuesta por
varias clases
Operaciones a base de datos

JDBC



Componentes

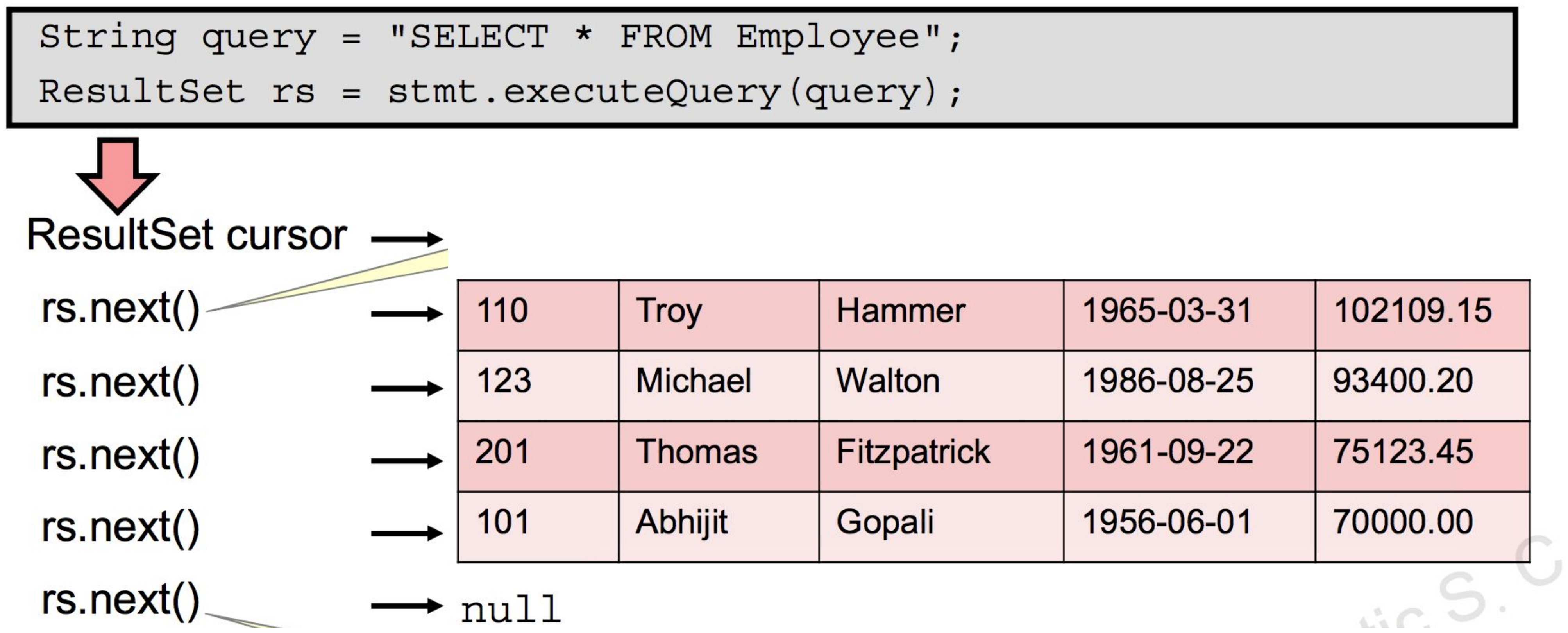
DriverManager

Connection

Statement

PreparedStatement

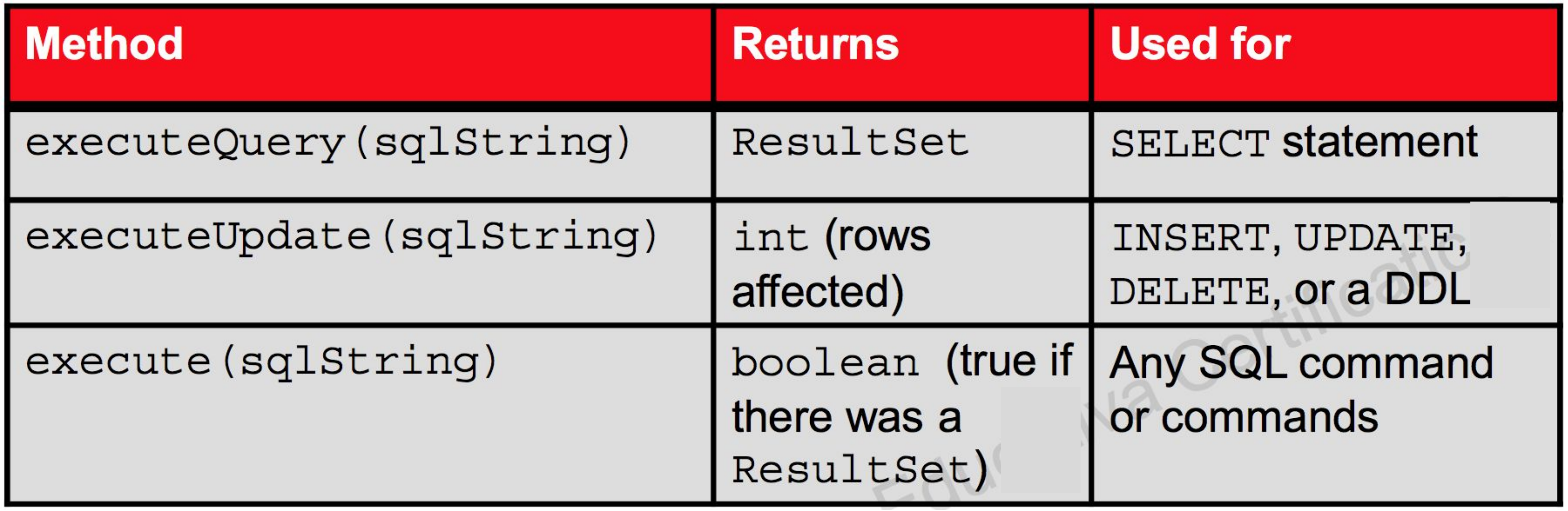
ResultSet



```
String query = "SELECT * FROM Employee";
ResultSet rs = stmt.executeQuery(query);
```

110	Troy	Hammer	1965-03-31	102109.15
123	Michael	Walton	1986-08-25	93400.20
201	Thomas	Fitzpatrick	1961-09-22	75123.45
101	Abhijit	Gopali	1956-06-01	70000.00

ResultSet



Method	Returns	Used for
`executeQuery(sqlString)`	`ResultSet`	`SELECT` statement
`executeUpdate(sqlString)`	`int` (rows affected)	`INSERT`, `UPDATE`, `DELETE`, or a DDL
`execute(sqlString)`	`boolean` (true if there was a `ResultSet`)	Any SQL command or commands

CRUD

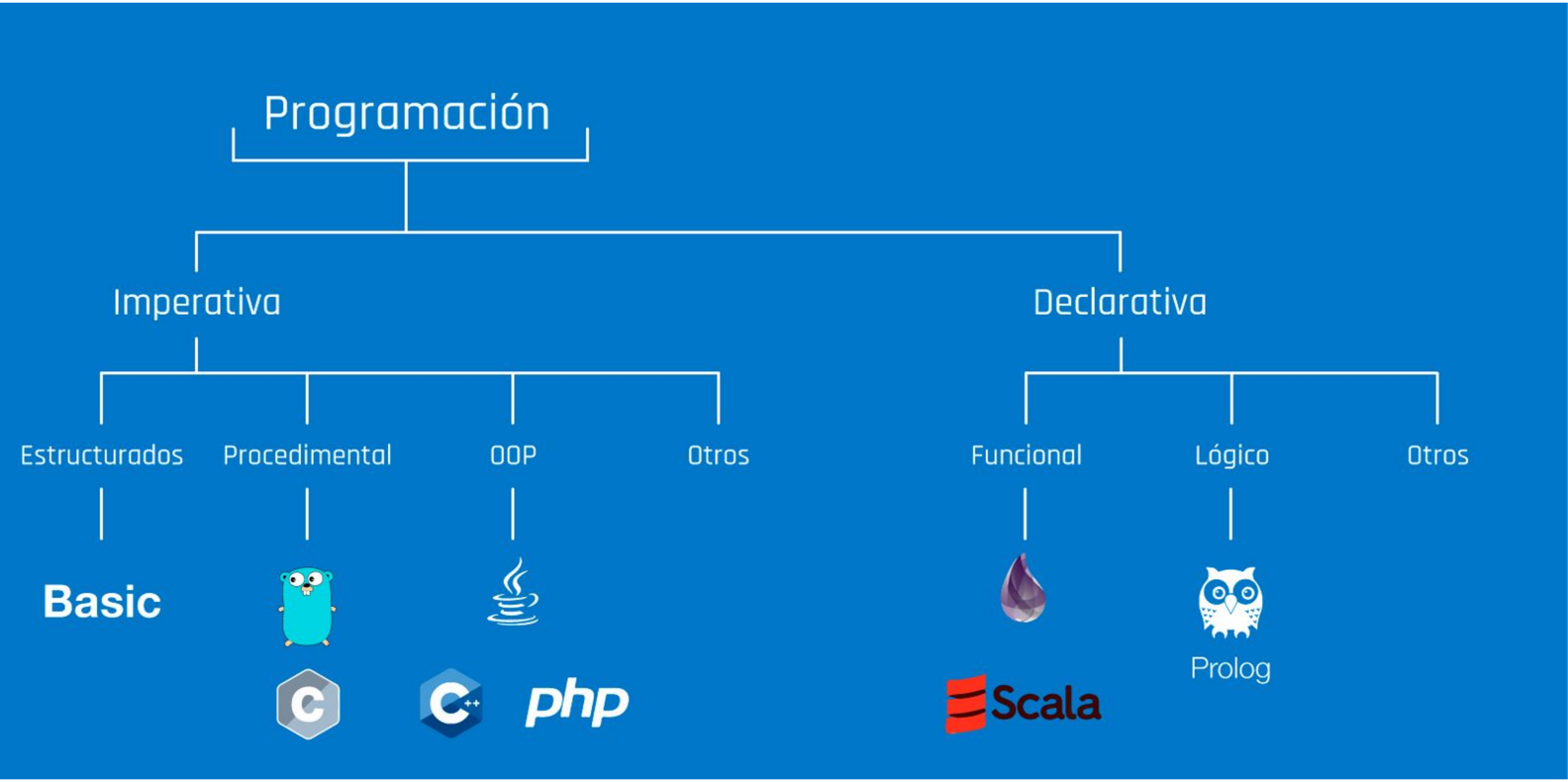
Programación Funcional

Programación Funcional

Paradigma de programación

Programación Funcional

Paradigma declarativo
vs.
Paradigma imperativo



https://medium.com/@Loopa/paradigmas-de-programaci%C3%B3n-programaci%C3%B3n-imperativa-y-programaci%C3%B3n-declarativa-4c4a4182fd87

Qué

Programación Funcional

Cómo

Programación imperativa

Qué

Programación Funcional

Cómo

Programación imperativa

ERLANG Haskell

elm

Java

Lenguajes Funcionales

Lenguajes con características Funcionales

Programación Funcional

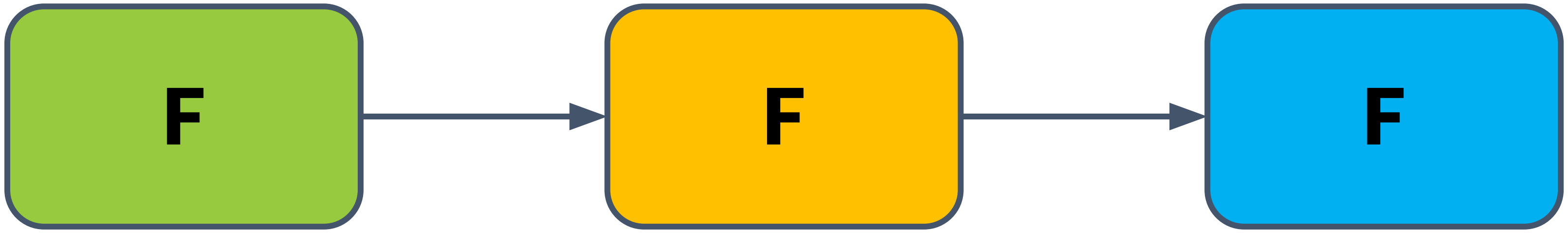
Funciones

Programación Funcional

Funciones
Entrada y Salida

Programación Funcional

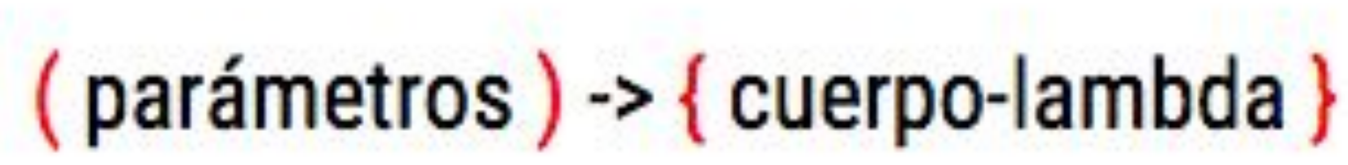
Programación Funcional



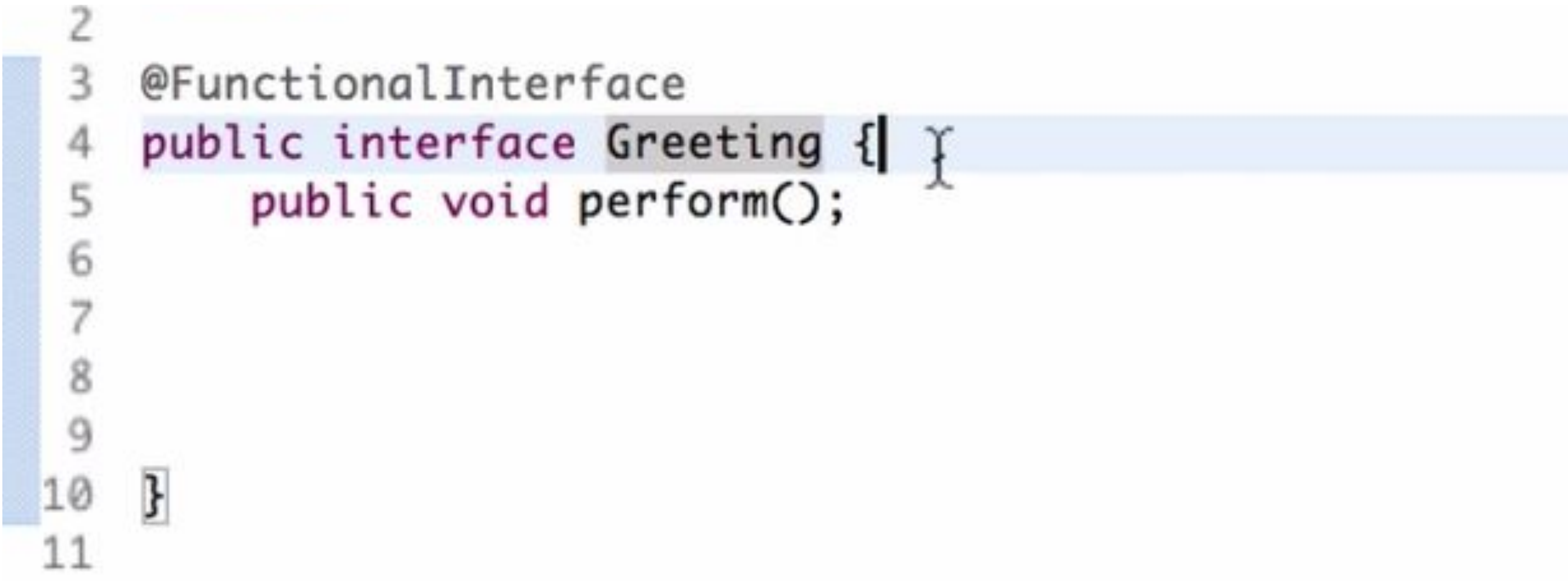
Funciones de orden superior

Lambdas

Lambdas

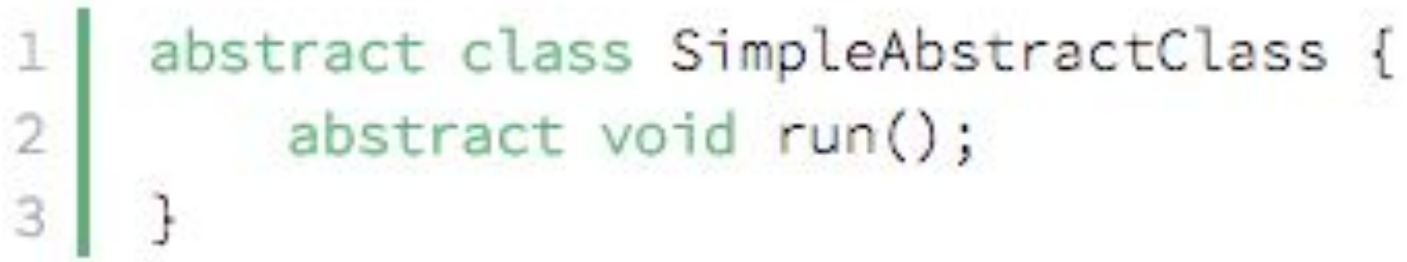


(parámetros) -> { cuerpo-lambda }

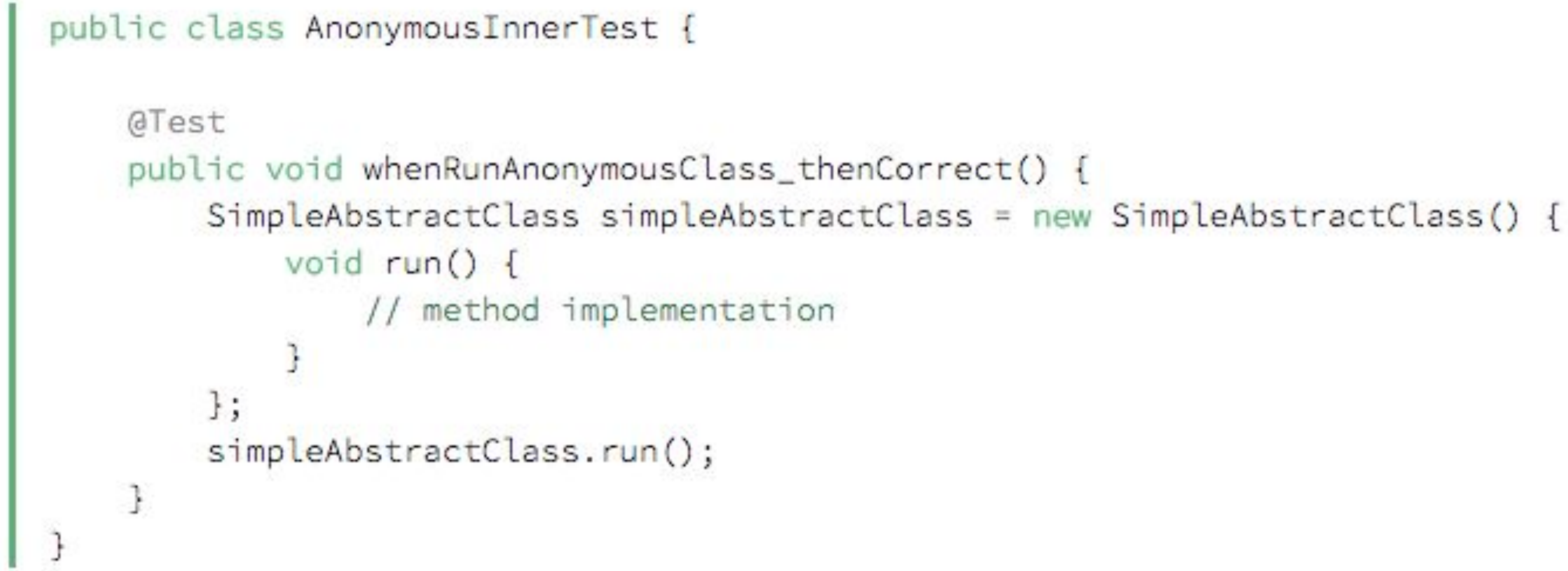


```java
@FunctionalInterface
public interface Greeting {
    public void perform();




}
```

@FunctionalInterface
SAM (Single Abstract Method)



```java
abstract class SimpleAbstractClass {
    abstract void run();
}
```



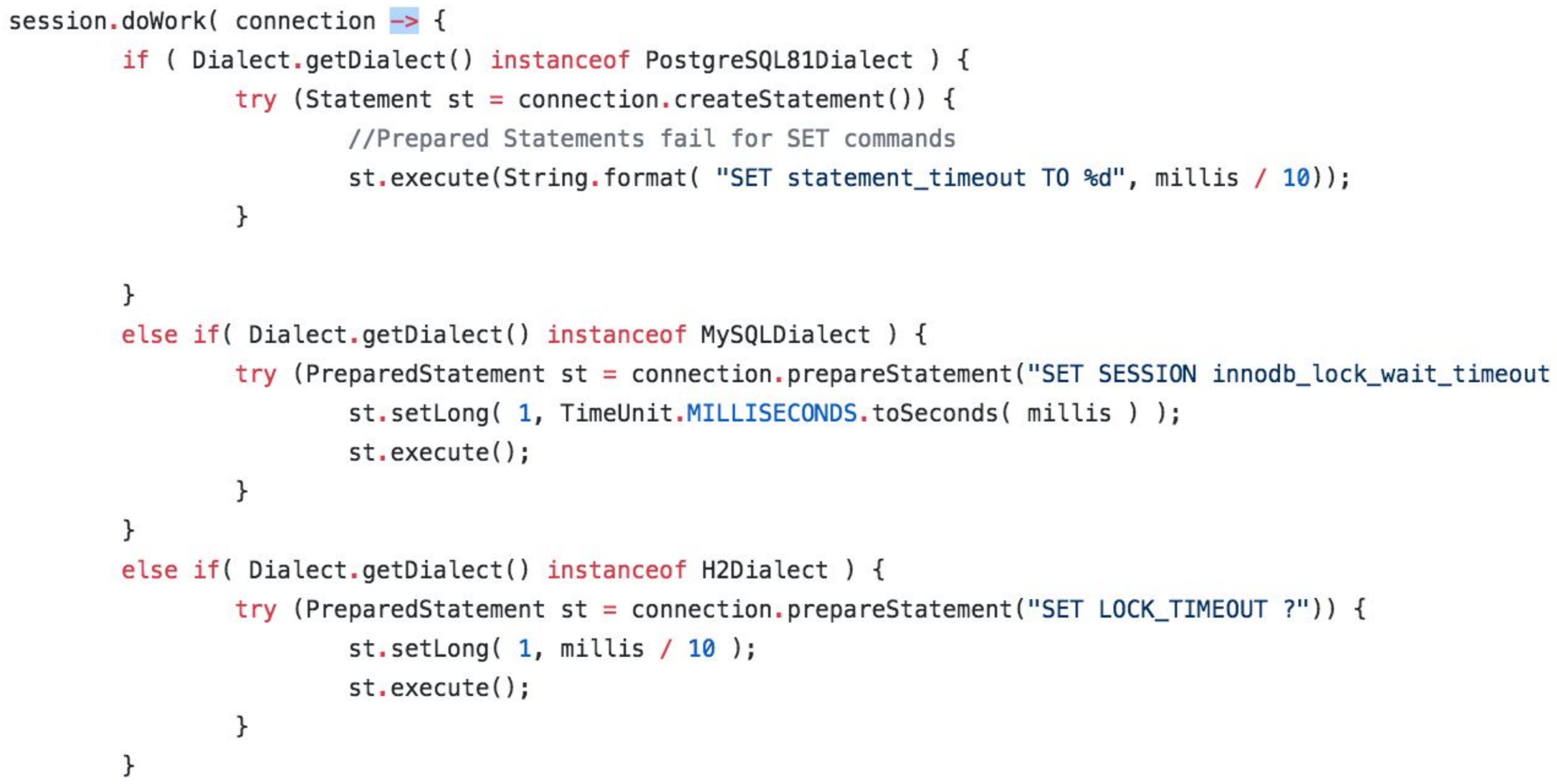
```java
public class AnonymousInnerTest {

    @Test
    public void whenRunAnonymousClass_thenCorrect() {
        SimpleAbstractClass simpleAbstractClass = new SimpleAbstractClass() {
            void run() {
                // method implementation
            }
        };
        simpleAbstractClass.run();
    }
}
```

Clases Anónimas

```
mButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // do something here
    }
});
```

```
mButton.setOnClickListener((View v) -> {
    // do something here
});
```

Android



```
session.doWork( connection -> {
        if ( Dialect.getDialect() instanceof PostgreSQL81Dialect ) {
                try (Statement st = connection.createStatement()) {
                        //Prepared Statements fail for SET commands
                        st.execute(String.format( "SET statement_timeout TO %d", millis / 10));
                }

        }
        else if( Dialect.getDialect() instanceof MySQLDialect ) {
                try (PreparedStatement st = connection.prepareStatement("SET SESSION innodb_lock_wait_timeout
                        st.setLong( 1, TimeUnit.MILLISECONDS.toSeconds( millis ) );
                        st.execute();
                }
        }
        else if( Dialect.getDialect() instanceof H2Dialect ) {
                try (PreparedStatement st = connection.prepareStatement("SET LOCK_TIMEOUT ?")) {
                        st.setLong( 1, millis / 10 );
                        st.execute();
                }
        }
```

Java Hibernate

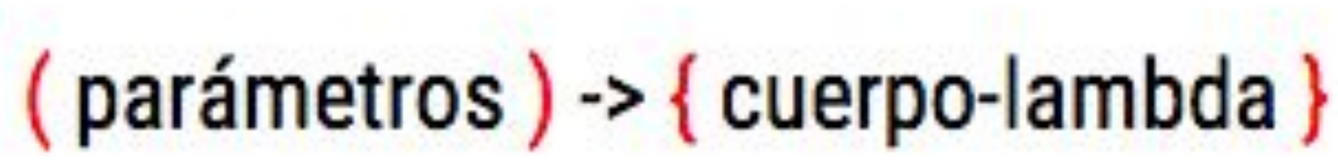
```java
mButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // do something here
    }
});
```

```java
mButton.setOnClickListener((View v) -> {
    // do something here
});
```

Android

Lambdas
Como variables

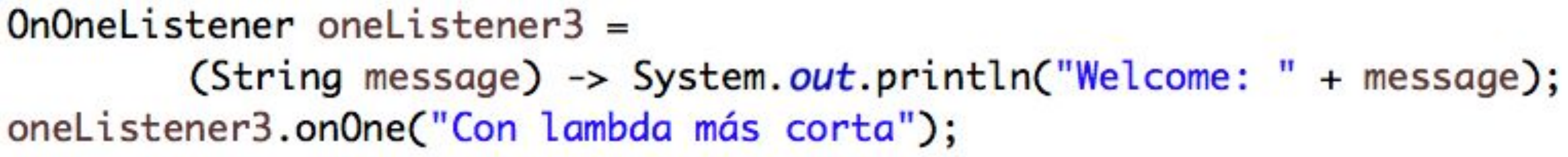
Lambdas



```
( parámetros ) -> { cuerpo-lambda }
```

Lambdas

Listener listener = (parámetros) -> { cuerpo-lambda }



```java
OnOneListener oneListener3 =
        (String message) -> System.out.println("Welcome: " + message);
oneListener3.onOne("Con lambda más corta");
```

Lambdas como variables

```
mButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // do something here
    }
});
```

```
mButton.setOnClickListener((View v) -> {
    // do something here
});
```

Android

Programación Funcional y Lambdas

No Iteración

No Iteración ✗

Sí Recursividad ✓

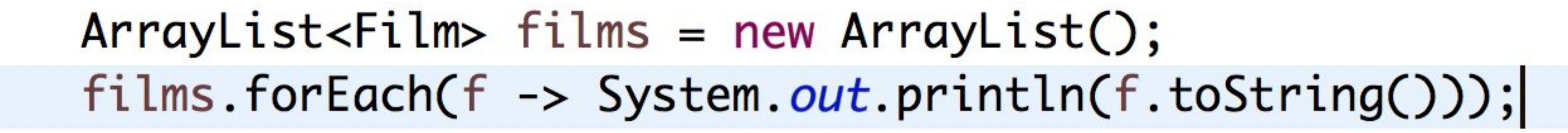
No Iteración ✗

Sí Recursividad ✓

Expresar Problemas

Recursividad

objects.**forEach()**



```java
ArrayList<Film> films = new ArrayList();
films.forEach(f -> System.out.println(f.toString()));
```

Recursividad

```
forEach(System.out::println)
```

RETO

Stream y Filter

Stream

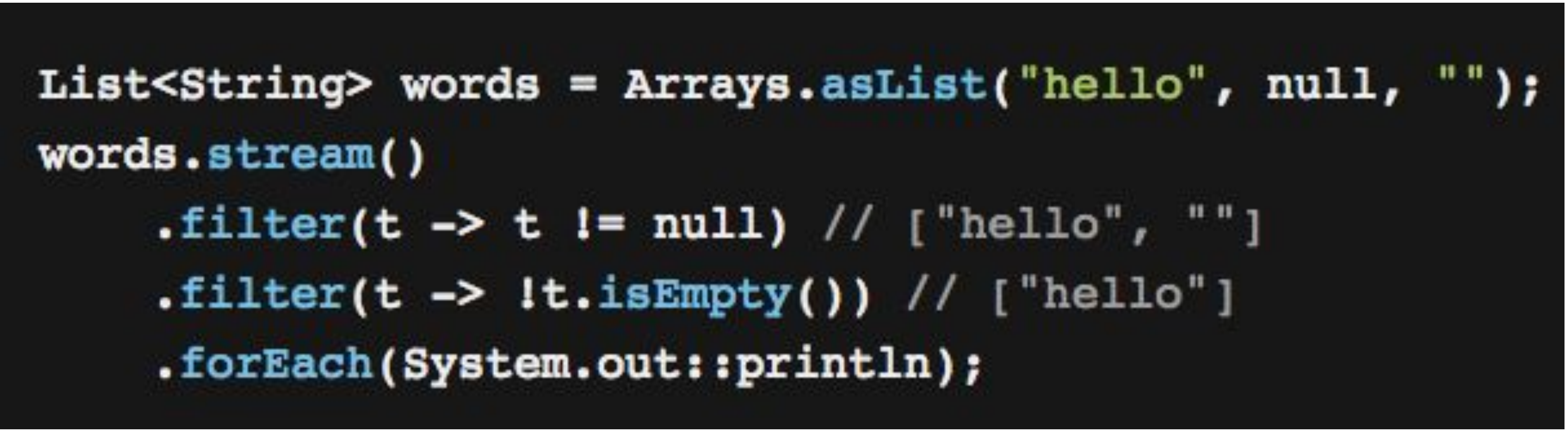
Un método que añadido a
todas las colecciones

Streams

objects.**stream()**

Filter

objects.stream()**.filter()**



```
List<String> words = Arrays.asList("hello", null, "");
words.stream()
    .filter(t -> t != null) // ["hello", ""]
    .filter(t -> !t.isEmpty()) // ["hello"]
    .forEach(System.out::println);
```

Filter

No asignaciones

No asignaciones ✗

Sí Inmutabilidad ✓

```
contentReport += m.toString() + "\n";
```