



an NTT DATA Company

The background of the slide is a photograph of a city skyline, likely Tokyo, with a person sitting on a rooftop in the foreground looking out over the city. The image has a blue and green color grade. Overlaid on the image are several white, glowing, circular lines that spiral outwards from the center, creating a sense of motion or data flow. In the top left corner, there is a green-to-white gradient bar.

BECA Java **Verano 2017**

Enrique Mingorance Cano

enrique.Mingorance.cano@everis.com





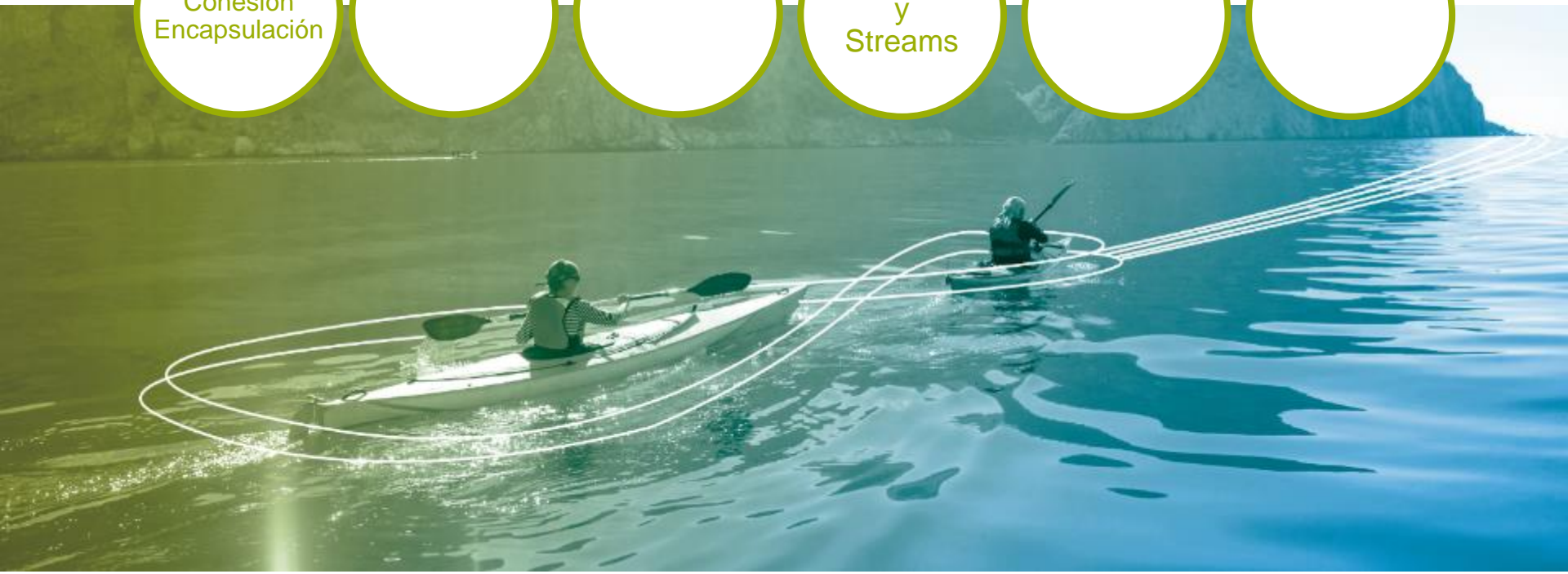
Java 8

Acoplamiento
Cohesión
Encapsulación

Métodos

Constructores

Lambdas
y
Streams



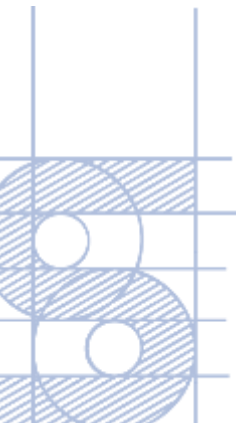
01

Acoplamiento
Cohesión
Encapsulación

Acoplamiento Cohesión Encapsulación I

Alta Encapsulación, Alta Cohesión y Bajo Acoplamiento

- Es el objetivo de POO
- Permite
 - Una mejor comprensión del software
 - Una inversión menor en tareas de refactorización
 - Facilitar las tareas del desarrollo de tests unitarios

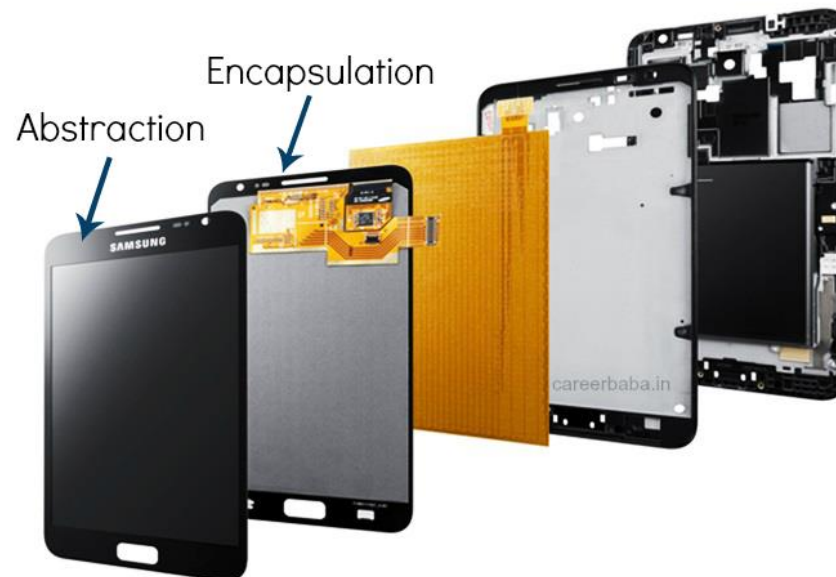


Acoplamiento Cohesión Encapsulación II

ALTA Encapsulación

- Es la ocultación de la información
- Consiste en exponer una API que nos de acceso sólo a las partes que queramos exponer al resto de clases
- Todos los atributos de una clase deben ser privados y exponer getters y setters para la lectura/escritura respetando la visibilidad de los métodos


```
public class Wheel implements VehiclePiece {  
    private int inches;  
  
    public int getInches() {  
        return inches;  
    }  
  
    public void setInches(int inches) {  
        this.inches = inches;  
    }  
}
```


























Acoplamiento Cohesión Encapsulación III

ALTA Cohesión

- Representa el grado en el que cada pieza de software forma una unidad lógica aislada y atómica



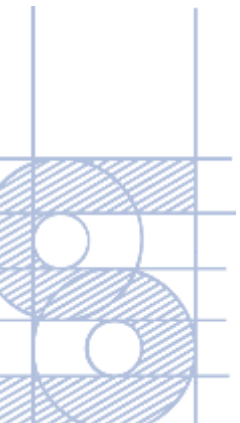
 BookinManager	 VehicleManager	 ClientManager
  getBookin(String) Booking	  getVehicle(String) Vehicle	  getClient(String) Client
  saveBooking(Booking) Booking	  saveVehicle(Vehicle) Vehicle	  saveClient(Client) Client
  deleteBooking(Booking) Booking	  deleteVehicle(Vehicle) Vehicle	  deleteClient(Client) Client
  checkVehicleStatus(Vehicle) Boolean		

Acoplamiento Cohesión Encapsulación III

BAJO Acoplamiento

- Es el grado en el que dos o más componentes son dependientes entre sí.
- La sugerencia es que todos los módulos deben ser lo más independientes posibles.
- Afecta directamente al grado de riesgo de errores en el resto de componentes cuando un componente es modificado.

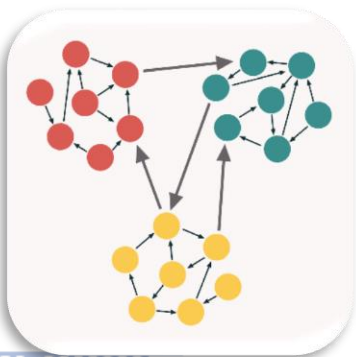
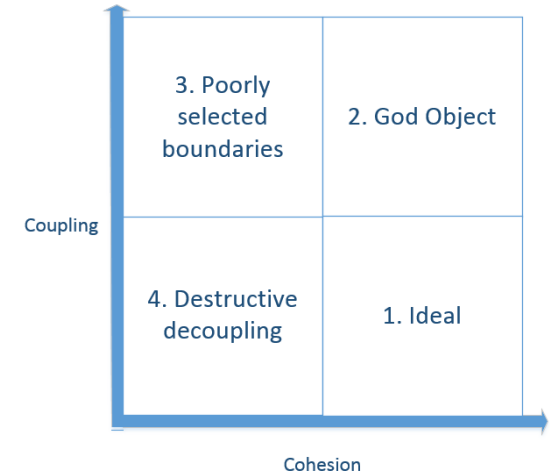
<https://stackoverflow.com/a/227334>



Acoplamiento Cohesión Encapsulación

Alta Encapsulación, Alta Cohesión y Bajo Acoplamiento

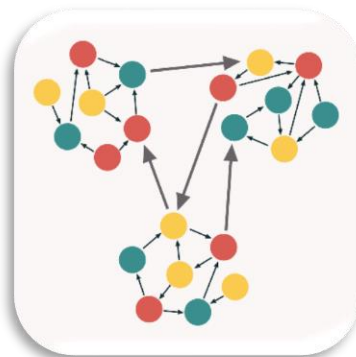
- <http://enterprisecraftsmanship.com/2015/09/02/cohesion-coupling-difference/>



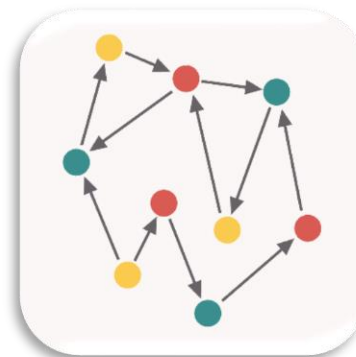
1. Ideal



2. God Object



3. Poorly ...



4. Hell

Acoplamiento Cohesión Encapsulación

Modificadores de acceso

- Junto con los paquetes y los métodos nos permiten mantener el balance adecuado en los tres parámetros
- Aplicable a clases, interfaces, atributos de clase o interfaz, métodos y constructores

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				

+ : accessible
blank : not accessible

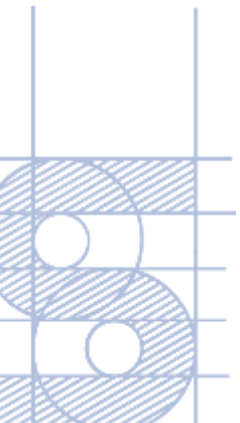
<https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>

<https://stackoverflow.com/a/215505>

Acoplamiento Cohesión Encapsulación

Modificadores de acceso y herencia

- Los métodos declarados con un modificador de acceso X en una superclase deben ser igual o menos restrictivos en todas las subclases.
- Los métodos declarados como privados no son heredados, por tanto no hay regla al respecto.
- Los métodos declarados en una interfaz son implícitamente públicos por defecto y no pueden tener un modificador distinto a este.
- Los atributos de una interfaz son implícitamente public static y final





02

Métodos

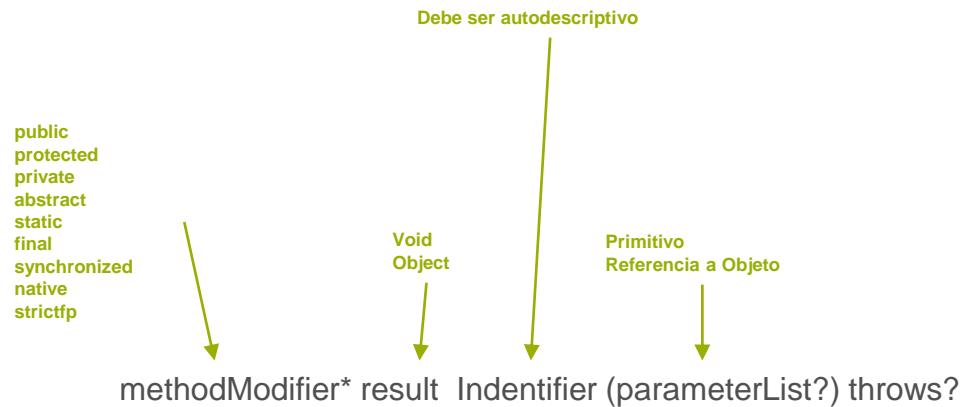
Definición de método

- Son las funciones definidas dentro de una clase
- Los métodos no-static necesitan de una instancia para ser ejecutados
- Para referirnos a un método dentro de una clase se utiliza el punto “.”



Definición de método

- Tienen la siguiente estructura



Varargs

- Es la posibilidad de definir el último parámetro del método como un array opcional

```
public class Calculator {  
    public Integer sum(Integer a, Integer b) {  
        return a+b;  
    }  
}
```

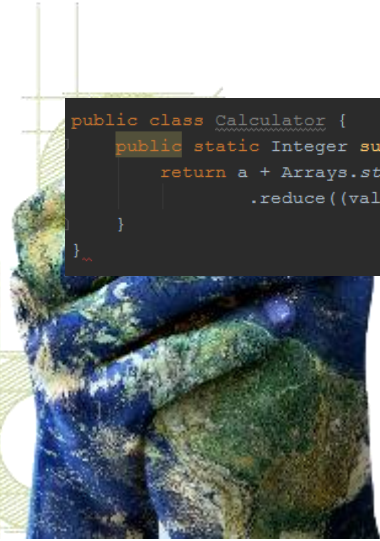


```
public void printSum() {  
    System.out.println(Calculator.sum( a: 1, b: 2));  
}
```

```
public class Calculator {  
    public static Integer sum(Integer a, Integer... b) {  
        return a + Arrays.stream(b)  
            .reduce((valA, valB) -> valA += valB).get();  
    }  
}
```

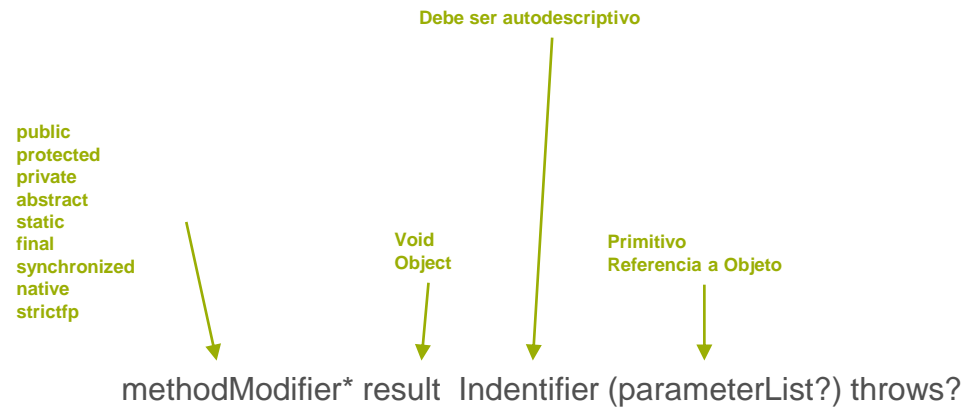


```
public void printSum() {  
    System.out.println(Calculator.sum( a: 1, ...b: 2, 3, 4, 5));  
}
```



Definición de método

- Tienen la siguiente estructura



Definición de sobrecarga

- Es la propiedad que una clase tiene de definir diferentes funcionalidades con el mismo nombre.
- Los métodos deben tener diferente firma
- No es posible declarar dos métodos con la misma firma en la misma clase
- No es posible definir dos métodos con la misma firma salvo el tipo de resultado devuelto

Definición de sobreescritura

- Es la propiedad que una clase hija tiene de redefinir la funcionalidad de una clase padre
- El método debe tener la misma firma

Buenas prácticas

- Verbos con la primera palabra en minúscula y las restantes con la primera palabra en mayúsculas
- Breve conciso e intuitivo
- Un getter de Boolean debe usar is o has en lugar de get evitando el uso de la palabra Not

03

Constructores

Definición de constructor

- Es un método especial, que es invocado automáticamente por la JVM cuando el objeto es creado.
- Es el método ideal para las inicializaciones necesarias en la clase.
- Su nombre es el mismo que el de la clase.
- No tiene valor de retorno.
- Si no se define un constructor el compilador creará uno por defecto sin parámetros
- Puede existir más de un constructor, gracias a la sobrecarga de métodos
- Lo primero que hace el constructor cuando la instancia es creada es llamar al método `super()` independientemente de si se indica explícitamente o no.
- Un constructor puede invocar a otros constructores con el uso de `this` y `super`

Definición de constructor

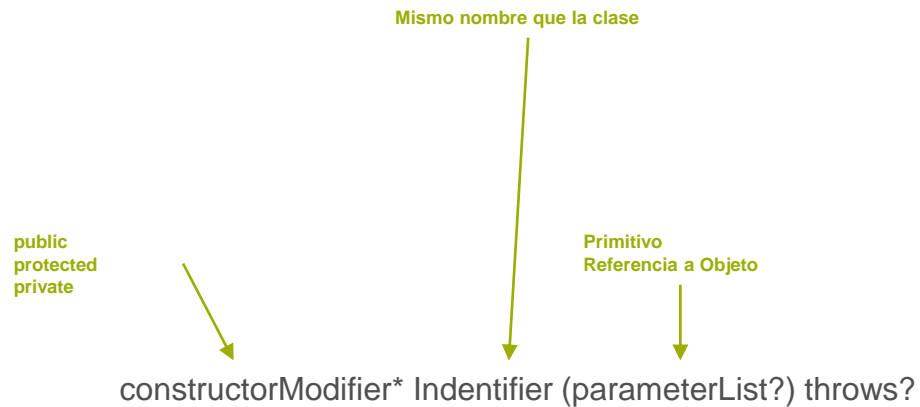
- Tienen la siguiente estructura

public
protected
private

Mismo nombre que la clase

Primitivo
Referencia a Objeto

constructorModifier* Indentifier (parameterList?) throws?



04

Lambdas y Streams



Definición de Lambda

- El cálculo lambda Es una teoría matemática (Alonzo Church, 1934) que fue introducida en el mundo de la programación en los 60 (LISP, diseñado en 1958), posteriormente ML, Miranda y Haskell
- La mayoría de los lenguajes de programación soporta lambdas, C++, C#, JavaScript, Python...
- Disponible desde Java 8 casi 60 años después o_O
- Eleva tu habilidades de programación
 - Eleva la inmutabilidad
 - Facilita la paralelización de ejecuciones
 - Código más claro y conciso

```
for (Integer value : a) {  
    System.out.println(value);  
}
```



```
Arrays.asList(a).forEach(System.out::println);
```

Method reference



Method Reference

- Esta funcionalidad está relacionada con las expresiones lambdas
- Posibilita referenciar constructores o métodos sin necesidad de ejecutarlos
- <https://blog.idrsolutions.com/2015/02/java-8-method-references-explained-5-minutes/>

Funcionalidades Lambda

- map
- filter
- reduce

https://es.slideshare.net/scottleber/java8-8-lambda-expressions?qid=d8fe604e-e610-4f09-b9f9-823079737fdb&v=&b=&from_search=5

Lambdas, stream y Stream API

- Las lambdas son usadas allá donde los tipos de parámetros aceptados sea una interfaz funcional
- `java.util.Stream` ofrece muchas funcionalidades funcionales
- Los streams son mónadas
 - Monada: Es simplemente un monoide en la categoría de endofunctor!!!! O_O
 - Monada (para gente normal): Son contextos, representan contextos con los que trabajar, flujos de valores sin estructura
- Juegan un papel fundamental trayendo la programación funcional al mundo de java
 - En la programación funcional las instrucciones cíclicas (`for`, `do..while`, ...) no existen

