

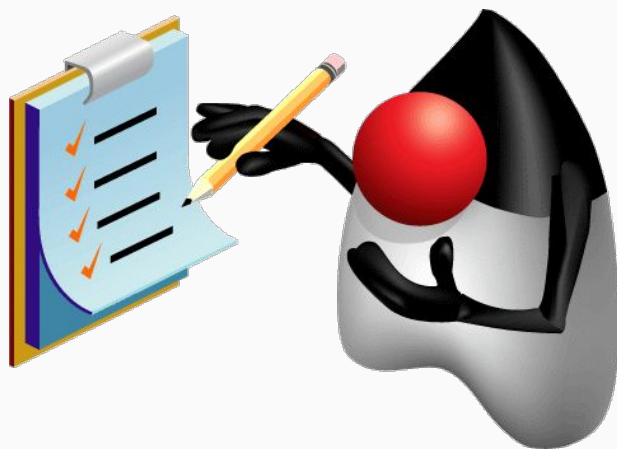


Fundamentos de P00

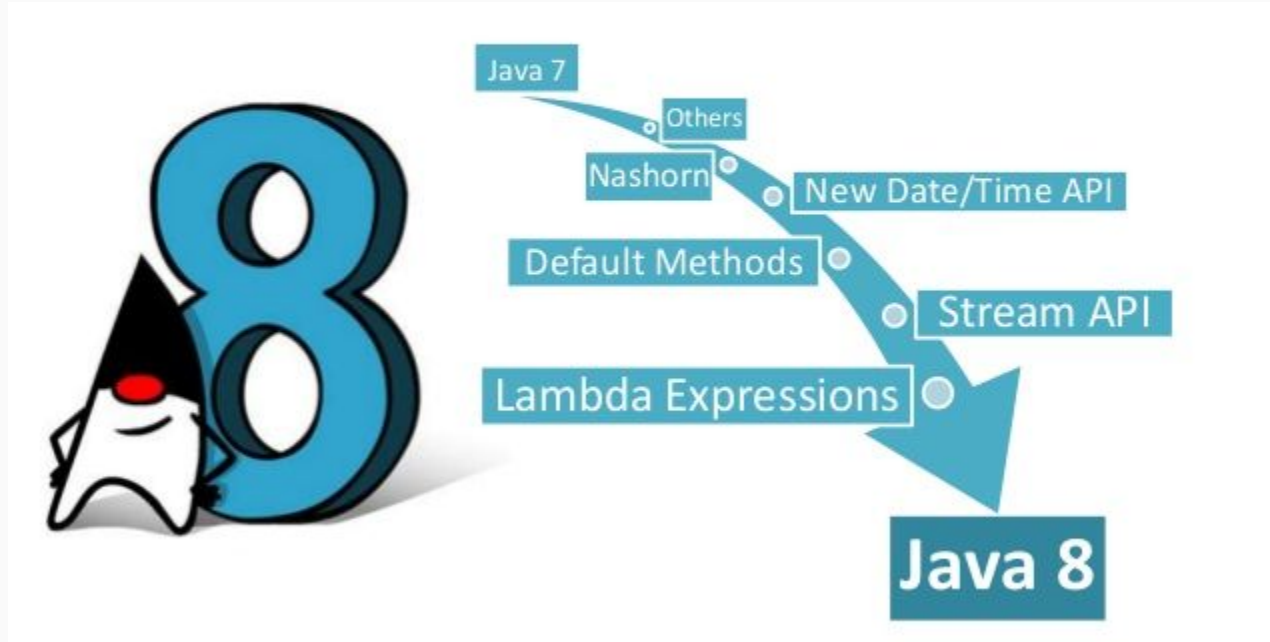
Dany Cenas
cvdany7@gmail.com

Objetivos

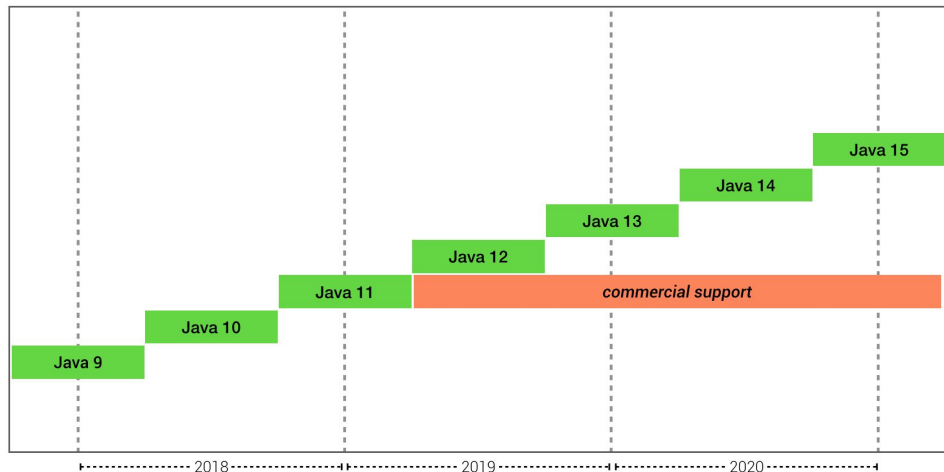
- Introducción a Java 8
- Evolución del JDK
- Configuración de Maven
- Programación Orientada a Objetos



Características importantes de Java 8

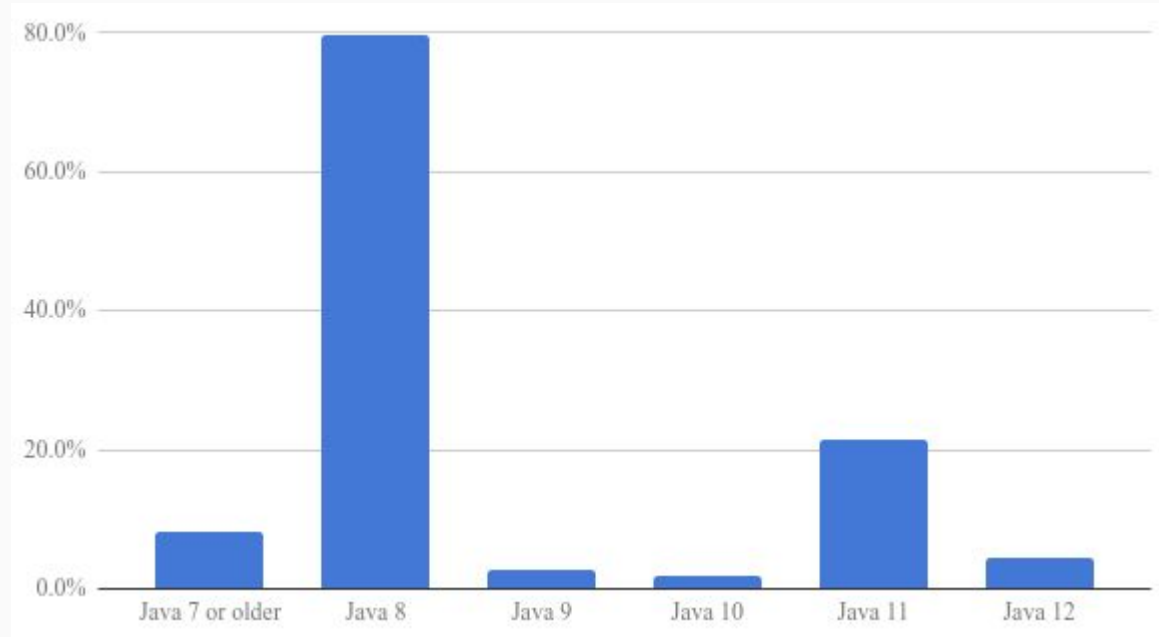


Evolución del JDK



Java SE 11 (LTS)	September 2018	At least August 2024 for Amazon Corretto September 2022 for AdoptOpenJDK	September 2026
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	TBA	TBA
Legend: Old version Older version, still maintained Latest version Latest preview version Future release			

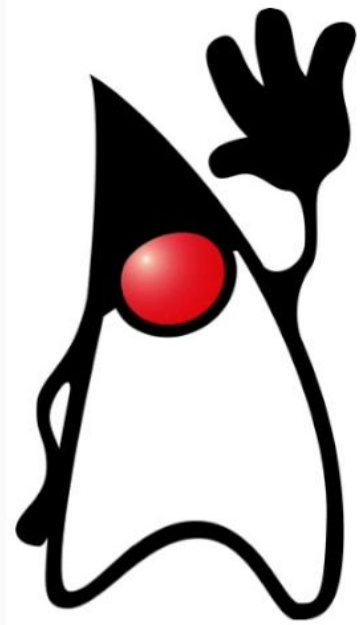
Adopción de Java

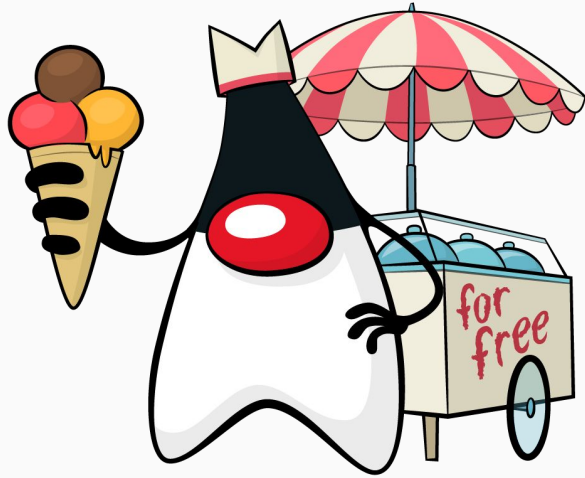


Actualizado al 2019. Fuente: <https://www.baeldung.com/java-in-2019>

Licenciamiento de uso Java 11

- Se paga licencia en entornos productivos. Esto no afecta a versiones anteriores del JDK, por lo que si usas Java 8, 9 o 10 sigue siendo gratuito.

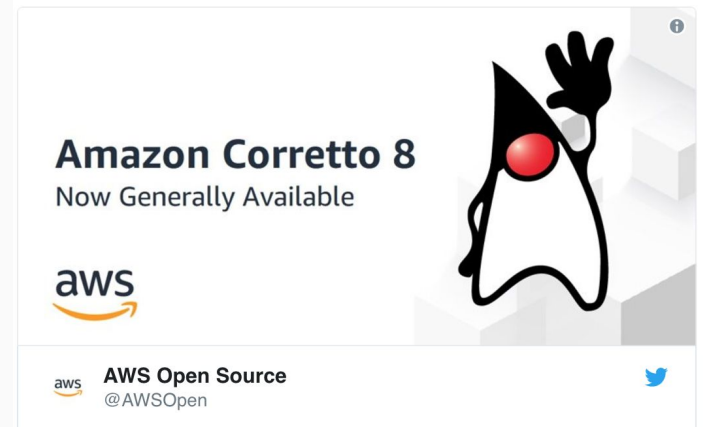




■ ■ ■ ■ AdoptOpenJDK

OpenJDK

OpenJ9



AZUL
SYSTEMS®

Requisitos

Para empezar a trabajar necesitamos instalar lo siguiente:



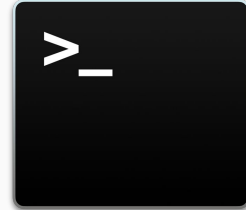
maven



Herramienta que realiza una función del control de versiones de código de forma distribuida.

- Es muy potente
- Fue diseñada por Linus Torvalds
- No depende de un repositorio central
- Es software libre
- Con ella podemos mantener un historial completo de versiones.



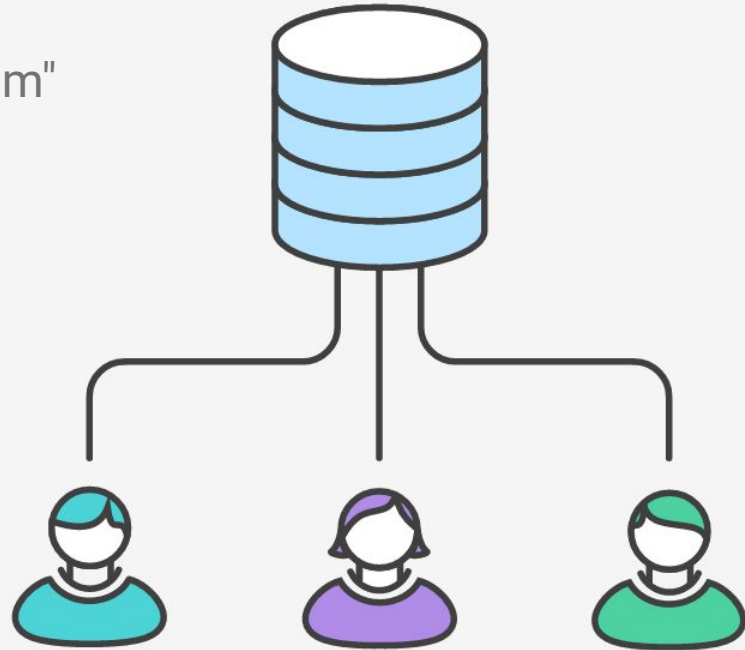


<https://www.youtube.com/watch?v=x3ybMcIOjp0>

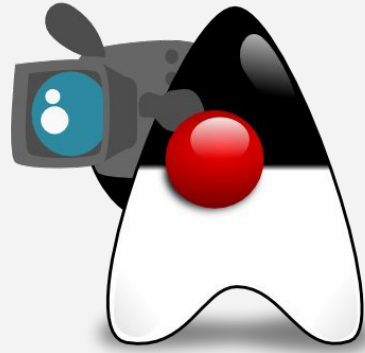
Configuración inicial

Como primer paso es necesario ejecutar lo siguiente:

```
git config --global user.name "Dany"  
git config --global user.email "cvdany7@gmail.com"
```



 **gitignore.io**



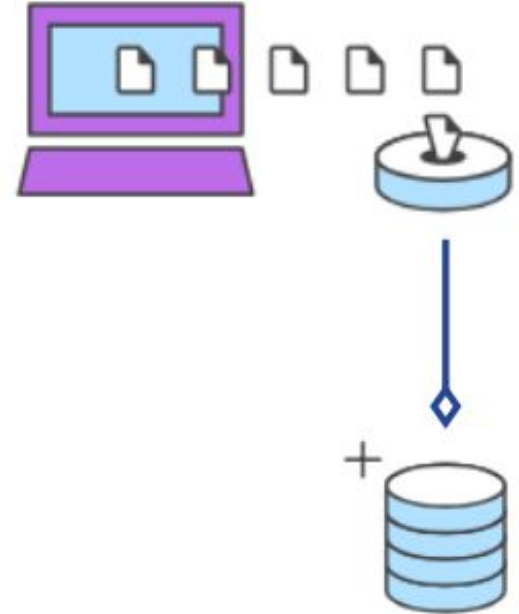
Crear un proyecto

`git init`

* Configuración inicial de un repositorio nuevo, crea automáticamente un subdirectorio `.git`.

`git status`

* Muestra el estado del working directory y el staging area.



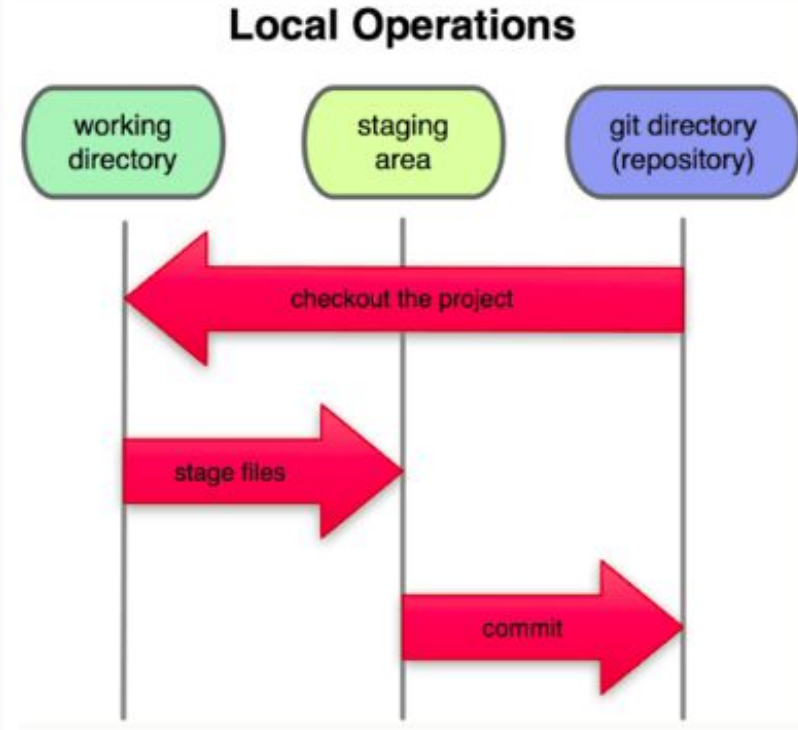
Secciones

`git add -A`

* Añade todos los archivos a staging area.

`git commit -m "Primer commit"`

* Confirma los cambios en git directory.



Cómo funciona



más comandos...

`git remote add origin <URL>`

* Añade la url del repositorio remoto.

`git remote set-url origin <URL>`

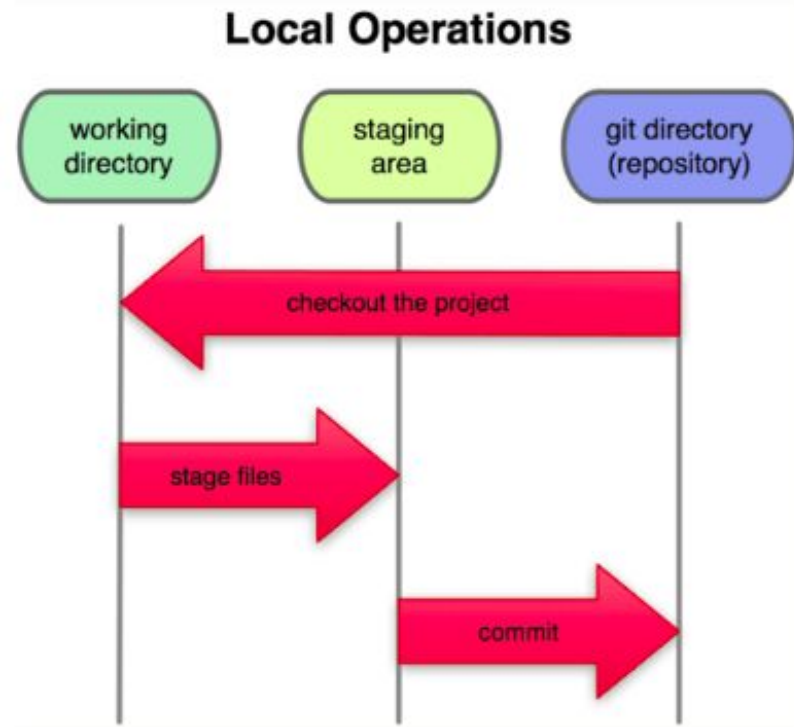
* Cambiar la url del repositorio remoto.

`git remote -v`

* Visualizar los repositorios remotos.

`git clone <URL>`

* Clona un repositorio remoto.



Otros repositorios remotos



Azure Repos

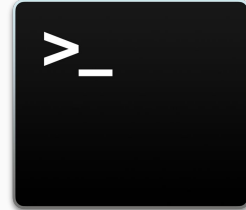


AWS CodeCommit

Es una herramienta opensource, que se utiliza para crear y administrar cualquier proyecto basado en Java.

- Hacer el proceso de construcción fácil.
- Proporcionar un sistema de construcción uniforme.
- Permitir la generación de diferentes perfiles para el despliegue.
- Permitir una migración transparente a nuevas funciones.





<https://www.youtube.com/watch?v=biBOXvSNaXg>

Los IDE's populares que apoyan el desarrollo con Maven son:



IntelliJ IDEA



Apache
NetBeans IDE

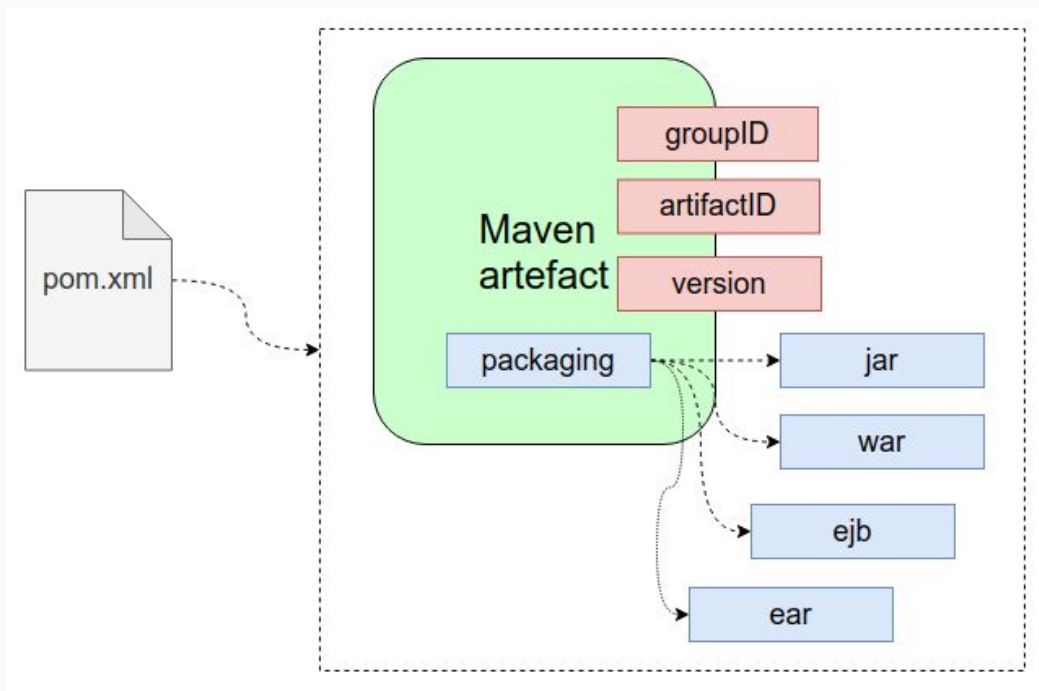


Desde línea de comandos

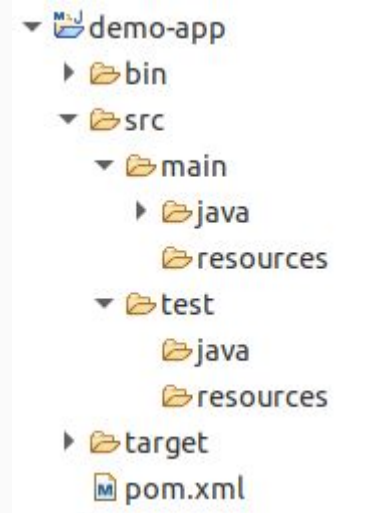
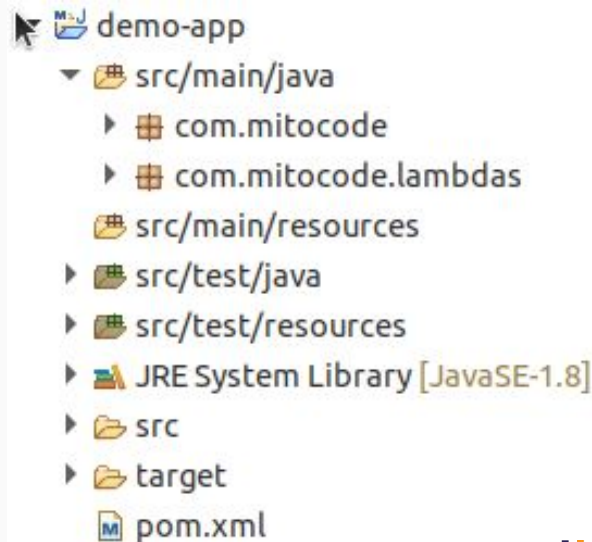
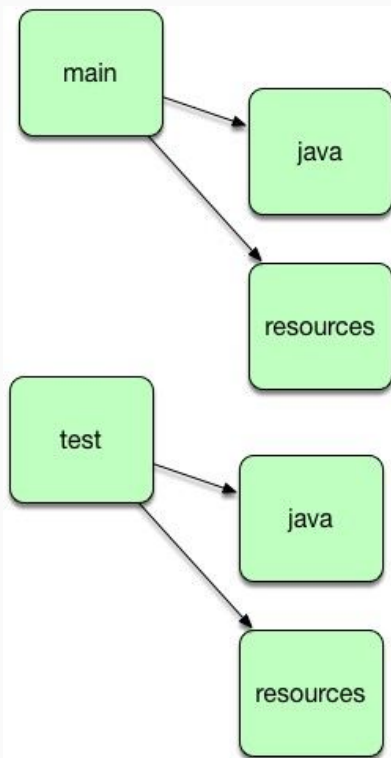


¿Qué es un artefacto?

Es un archivo, generalmente un JAR, que se implementa en un repositorio de Maven.



Estructura

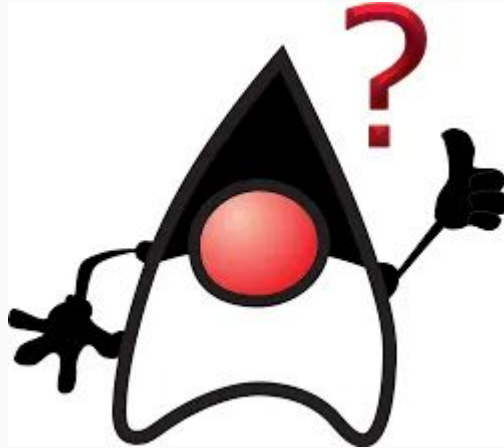


Programación Orientada a Objetos

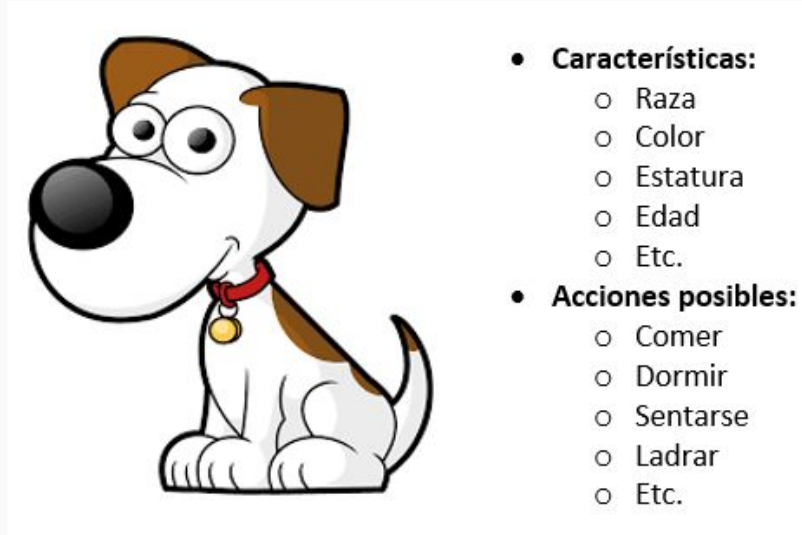


Introducción

Cualquier cosa que puedas pensar o ver en este instante es un objeto.
Por ejemplo: una pizarra, una laptop, un estudiante, un docente, etc.



Detalles de un objeto.



Todo perro es distinto, es decir, aunque compartan características, no todos son iguales.

Introducción

Aquí surge algo llamado clase. Todos los perros pertenecen a la clase Perro.



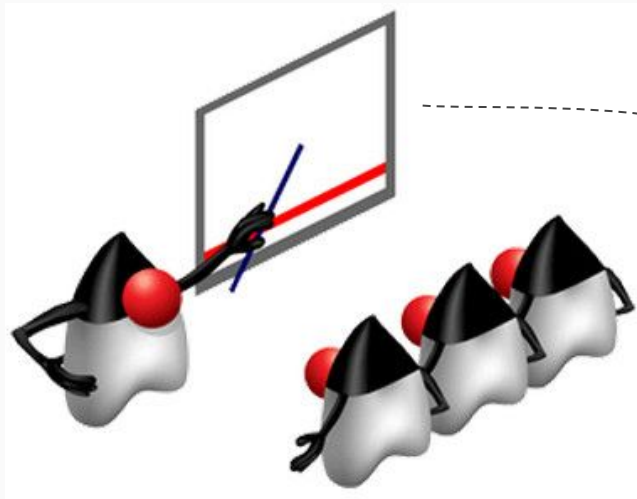
Perro	
	Nombre
	Raza
	Altura
	Comer()
	Dormir()
	LadRAR()

← **Propiedades**

← **Métodos**

Programación Orientada a Objetos

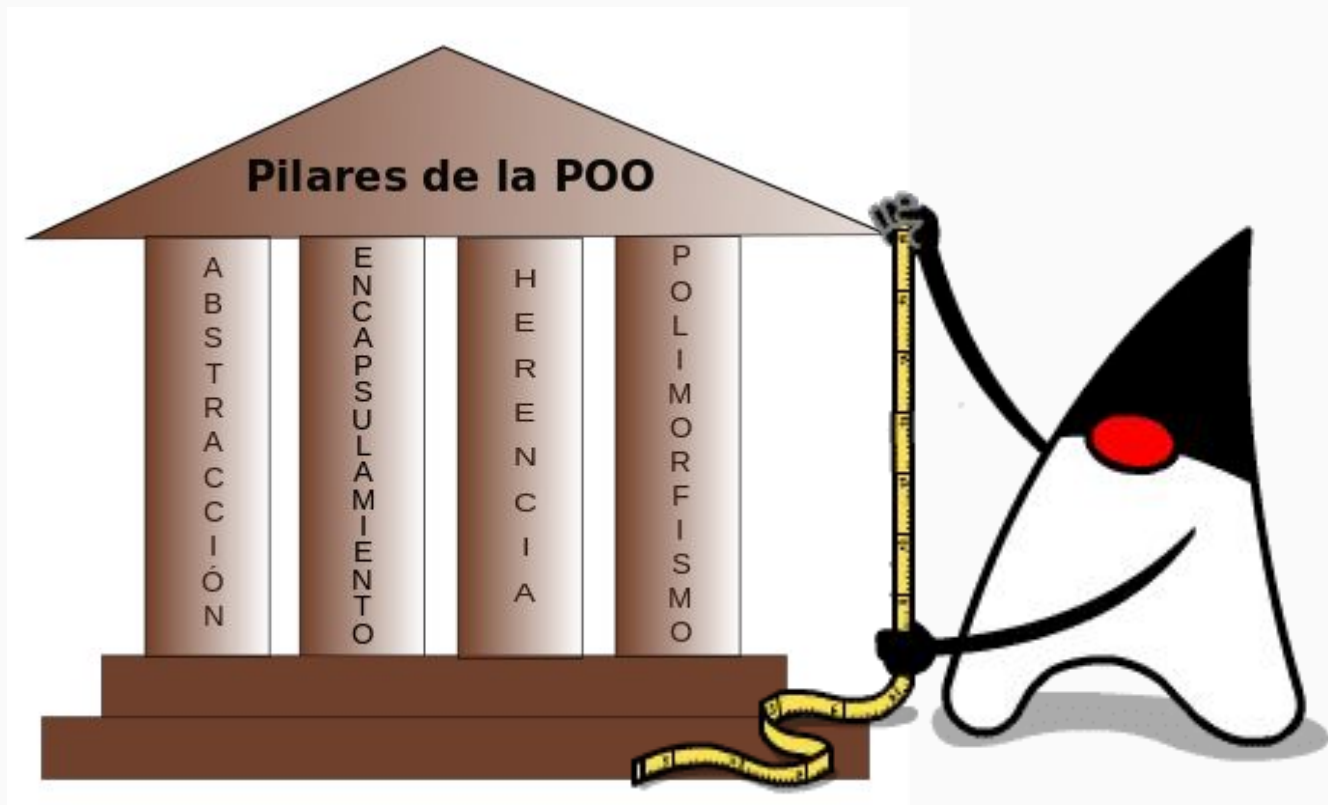
Es un paradigma de programación, la cual permite relacionar los objetos del mundo real y de alguna manera llevar estos conceptos a la programación.



```
1  
2 public class Perro {  
3  
4     private String nombre;  
5     private String raza;  
6     private double altura;  
7 }
```

Un paradigma de programación es un estilo de desarrollo de software.

Pilares de POO



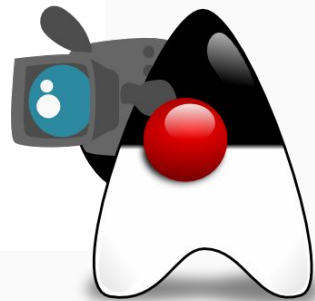
P00 en Java



Archivo pom.xml

sesion01/pom.xml

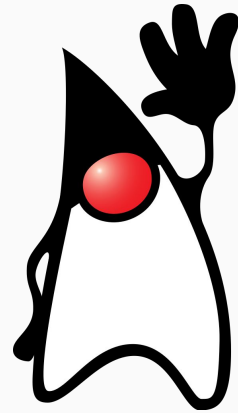
```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>com.mitocode</groupId>
6   <artifactId>app01</artifactId>
7   <version>0.0.1-SNAPSHOT</version>
8
9   <properties>
10     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
11     <maven.compiler.source>1.8</maven.compiler.source>
12     <maven.compiler.target>1.8</maven.compiler.target>
13   </properties>
14 </project>
```



Método main

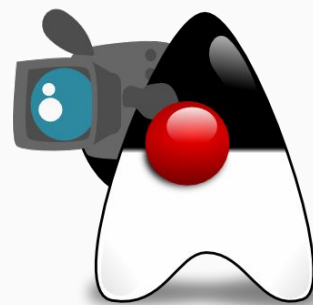
HelloWorld.java

```
1 import java.util.Scanner;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6
7         // Imprimir en consola
8         System.out.println("Bienvenidos al curso");
9
10        // Obtener valores de la consola
11        Scanner sc = new Scanner(System.in);
12        System.out.print("¿Cuál es tú nombre? ");
13        String nombre = sc.nextLine();
14
15        System.out.println("Bienvenido " + nombre);
16
17        sc.close();
18
19    }
20 }
```



Estructura de Paquetes

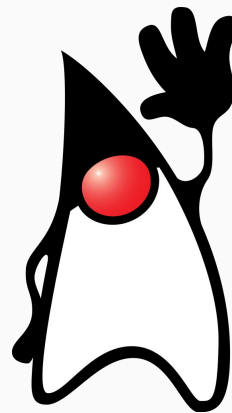
- src/main/java
 - > com.coveros.speaker.configuration
 - > com.coveros.speaker.configuration.core
 - > com.coveros.speaker.exception
 - > com.coveros.speaker.model
 - > com.coveros.speaker.repository
 - > com.coveros.speaker.service
 - > com.coveros.speaker.util
 - > com.coveros.speaker.web.controller.api
 - > com.coveros.speaker.web.controller.view
- src/main/resources
 - > db.migration
 - > app.properties
 - > log4j.properties
- src/test/java
 - > com.coveros.speaker.configuration
 - > com.coveros.speaker.model
 - > com.coveros.speaker.service
 - > com.coveros.speaker.test.util
 - > com.coveros.speaker.web.controller.api
- src/seleniumTest/java
 - > com.coveros.selenium
 - > com.coveros.selenium.workflows
- src/main/generated
 - > com.coveros.speaker.model



Clase

Representa al conjunto de objetos que comparten una estructura y un comportamiento comunes.

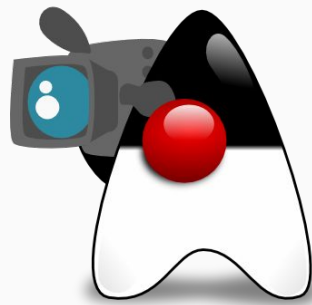
```
Perro.java ✖  
1  
2 public class Perro {  
3  
4     // Definición de atributos  
5     String nombre;  
6     String raza;  
7     double altura;  
8
```



Objeto

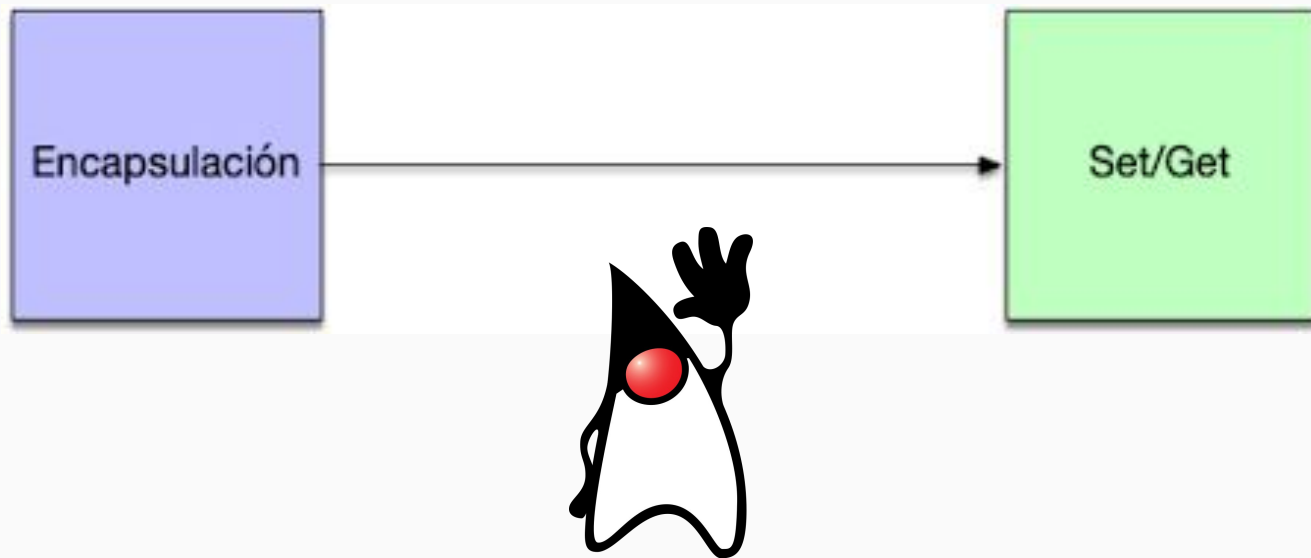
Es la referencia e instancia de una clase.

```
50 public static void main(String[] args) {  
6  
7     // Instancia de la clase Perro  
8     Perro p = new Perro();  
9     p.nombre = "Toby";  
10    p.raza = "Dogo";  
11    p.altura = 60;  
12  
13    // Imprimimos los valores  
14    System.out.println("Nombre : " + p.nombre);  
15    System.out.println("Raza : " + p.raza);  
16    System.out.println("Altura: " + p.altura);  
17 }  
18 }
```



Encapsulamiento

Consiste en la ocultación del estado o de los datos miembro de un objeto, de forma que sólo es posible modificar los mismos mediante los métodos definidos para dicho objeto.



Encapsulamiento

Perro.java

```
1
2 public class Perro {
3
4     // Atributos privados
5     private String nombre;
6     private String raza;
7     private double altura;
8
9     public String getNombre() {
10         return nombre;
11     }
12
13     public void setNombre(String nombre) {
14         this.nombre = nombre;
15     }
16
17     public String getRaza() {
18         return raza;
19     }
20
21     public void setRaza(String raza) {
22         this.raza = raza;
23     }
```

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

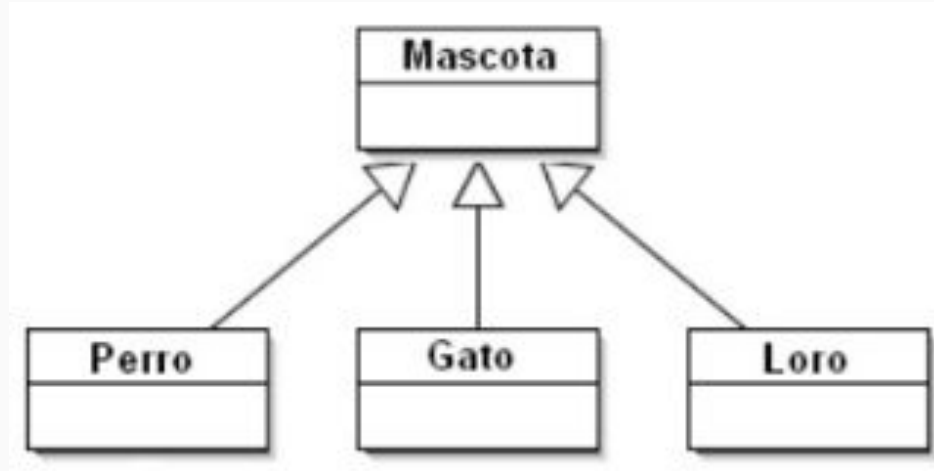
    // Asignamos valores por el método set
    Perro p = new Perro();
    p.setNombre("Toby");
    p.setRaza("Dogo");
    p.setAltura(60);

    // Obtenemos valores por método get e imprimimos
    System.out.println("Nombre : " + p.getNombre());
    System.out.println("Raza : " + p.getRaza());
    System.out.println("Altura: " + p.getAltura());
}
```



Herencia

Permite heredar las características (atributos y métodos) de otra clase.

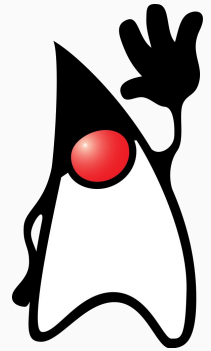


Ejercicio 1

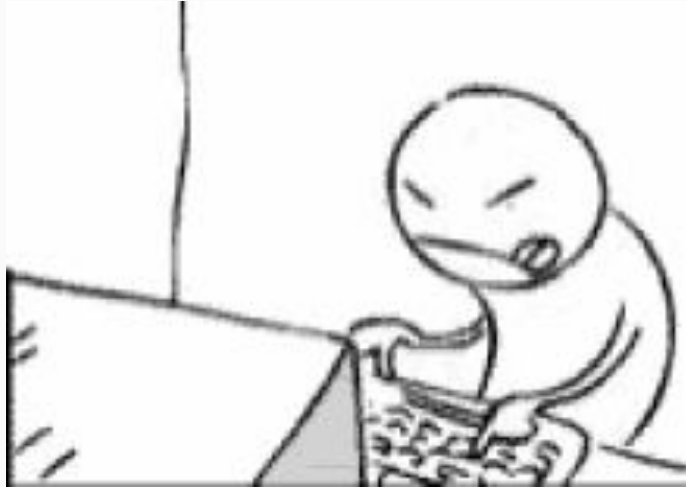
En un banco, para registrar una cuenta bancaria, es necesario informar el número de la cuenta, titular, y el valor de depósito inicial que el titular depositó al abrir la cuenta. Este valor de depósito sin embargo, es opcional, es decir: si el titular no tiene dinero que depositar en el momento de abrir su cuenta el depósito inicial no se hará y el saldo inicial de la cuenta será, naturalmente, cero.

Importante: el saldo de la cuenta no puede ser modificado libremente. Es necesario un mecanismo para proteger ella. El saldo sólo aumenta por medio de depósitos, y sólo disminuye por medio de retiros. Para cada saque el banco cobra una tasa de \$ 5.00. Nota: la cuenta puede quedar con saldo negativo si el saldo no es suficiente para realizar el retiro.

Usted debe hacer un programa que realice el registro de una cuenta, dando opción para que sea o no informado del valor de depósito inicial.



A programar..

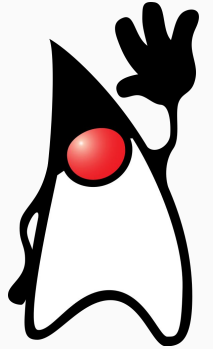


<https://github.com/danycenas/sesion01>

Uso de static - Ejercicio 2

Haga un programa que lea el tipo de cambio del dólar, luego ingrese el monto de dólares a comprar. Muestre el monto en soles a pagar, considerando que se tendrá que pagar un 6% de comisión.

Crear una clase ConvertidorMoneda.

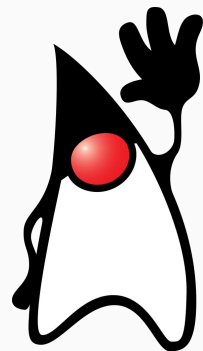


Herencia - Ejercicio 3

Un banco tiene tanto cuentas comunes como para las empresas. Una cuenta de empresa contiene todas las características de una cuenta común, adicionalmente un límite de préstamo y una operación de préstamo.

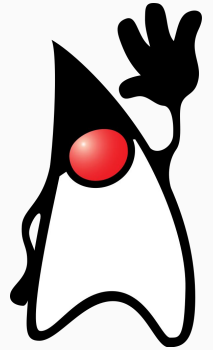
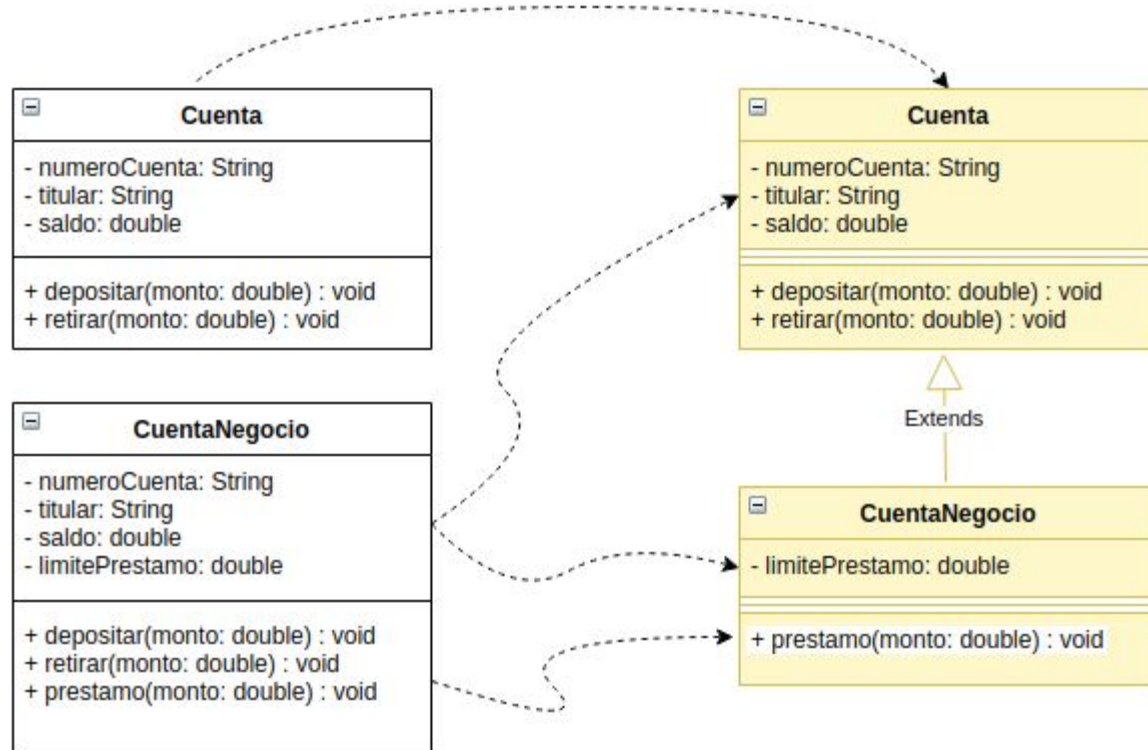
Cuenta
- numeroCuenta: String - titular: String - saldo: double
+ depositar(monto: double) : void + retirar(monto: double) : void

CuentaNegocio
- numeroCuenta: String - titular: String - saldo: double
+ depositar(monto: double) : void + retirar(monto: double) : void + prestamo(monto: double) : void

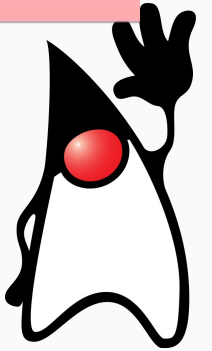
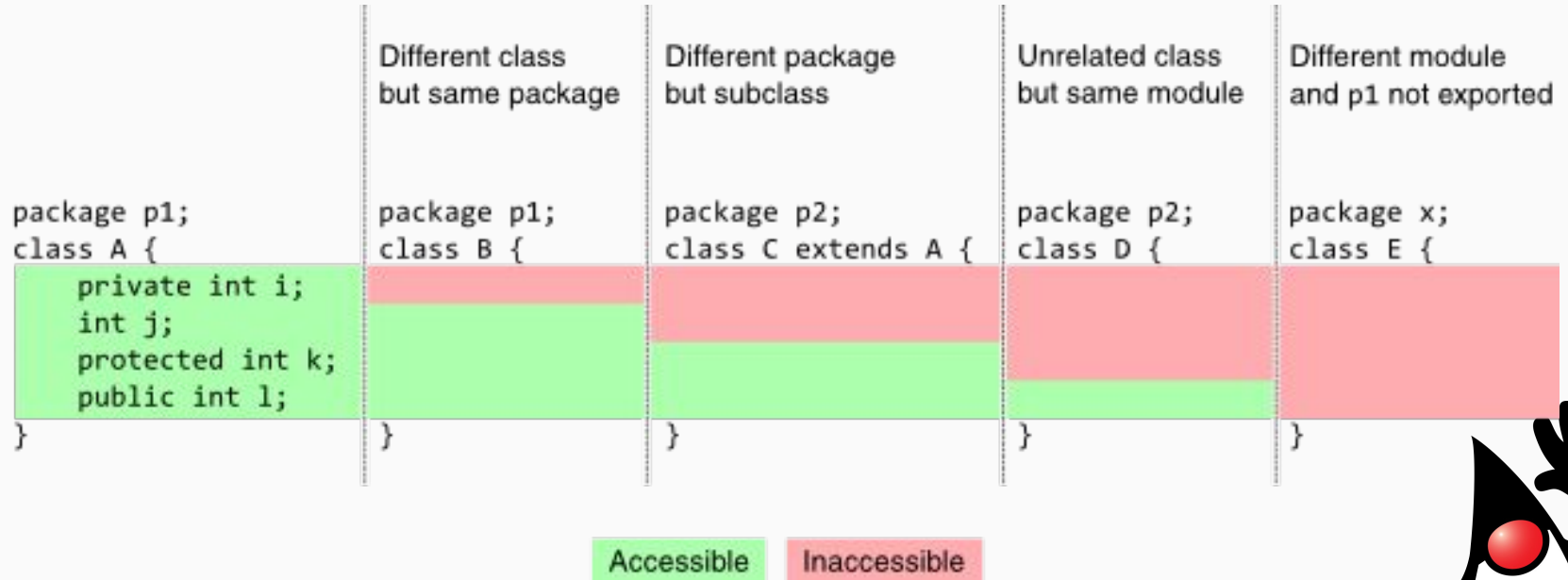


Herencia - Ejercicio 3

Quedan de la siguiente manera:

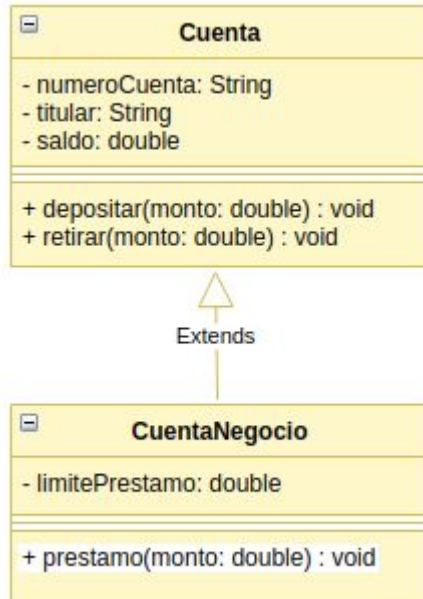


Modificadores

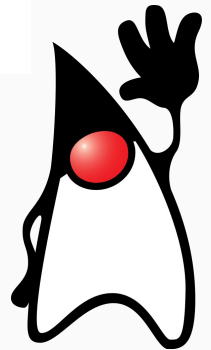


Modificadores - Ejercicio 4

Supongamos que, para realizar un préstamo, es descontado una tasa por valor de 10.0
Esto produce un error:



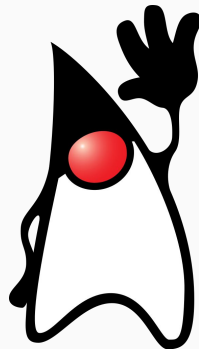
```
public void prestamo(double monto) {  
    if (monto <= limitePrestamo) {  
        saldo += monto - 10.0;  
    }  
}
```



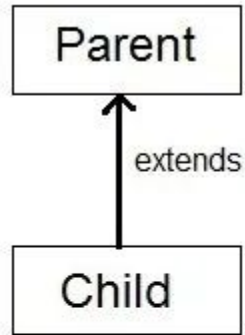
Sobrescritura

Es la implementación de un método de una superclase en la subclase

- Es muy recomendable utilizar la anotación @Override en un método sobrescrito.
- Facilita la lectura y la comprensión código.
- Avisamos al compilador (buena práctica)



Upcasting / Dowcasting



```
Parent p = new Parent( );
```

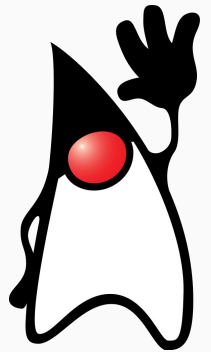
```
Child c = new Child( );
```

```
Parent p = new Child( );
```

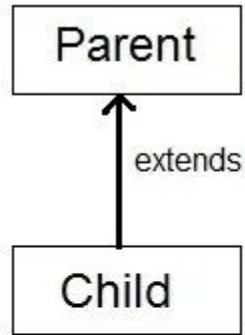
Upcasting

```
Child c = new Parent( );
```

incompatible type



Upcasting / Dowcasting



```
Parent p = new Parent( );
```

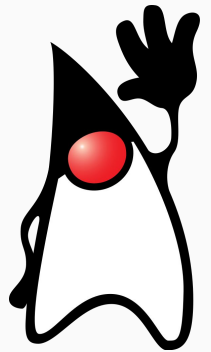
```
Child c = new Child( );
```

```
Parent p = new Child( );
```

Upcasting

```
Child c = new Parent( );
```

incompatible type



Polimorfismo

Upcasting

Casting de la subclase para la superclase

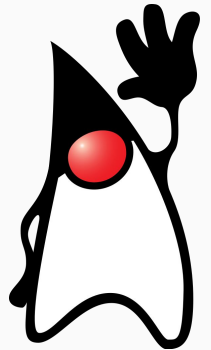
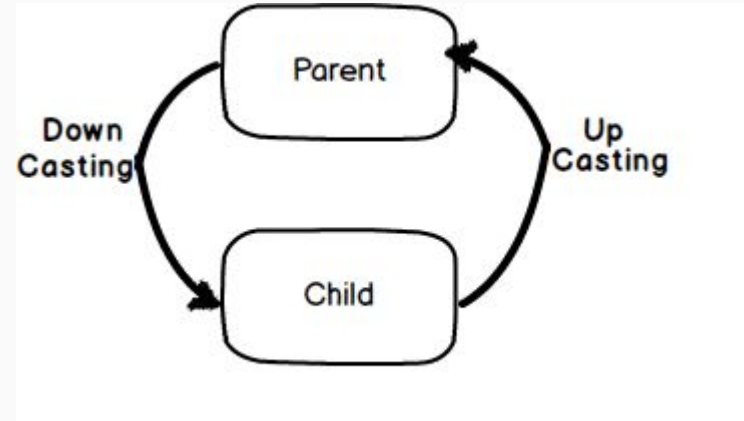
Uso común: polimorfismo

Downcasting

Casting de la superclase para subclase

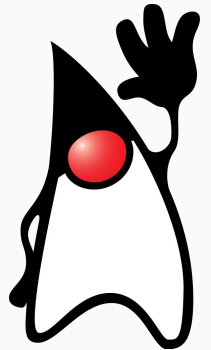
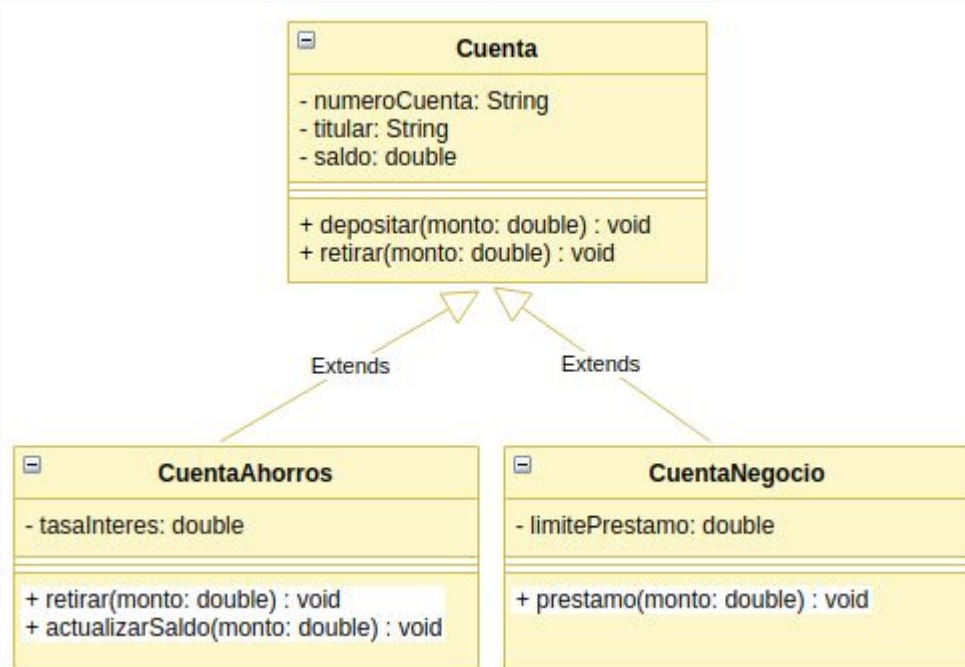
Palabra instanceof

Uso común: métodos que reciben parámetros genéricos (por ejemplo, Equals)



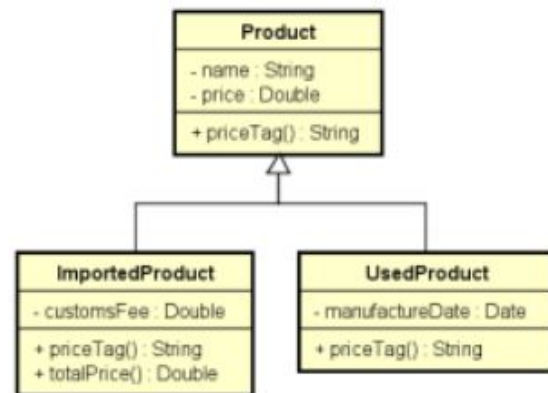
Polimorfismo - Ejercicio 5

Supongamos que la operación de retiro tiene un costo de 5.0. Sin embargo, para las cuentas de ahorros no debe cobrarse. ¿Cómo resolver esto?



Ejercicio

Hacer un programa para leer los datos de N productos (N proporcionado por el usuario). Al final, mostrar la etiqueta de precio de cada producto en la el mismo orden en que se hayan introducido. Todo producto tiene nombre y precio. productos importados tienen una tasa de aduana, y los productos usados tienen fecha de fabricación. Estos datos específicos deben añadidos en la etiqueta de precio conforme (página siguiente). Para productos importados, la tasa y la aduana debe añadida al precio final del producto.

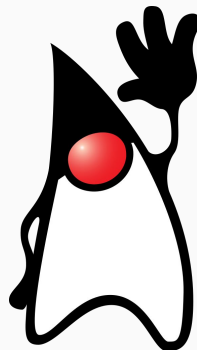


Enums

Es un tipo de datos especial que permite que una variable sea un conjunto de constantes predefinidas.

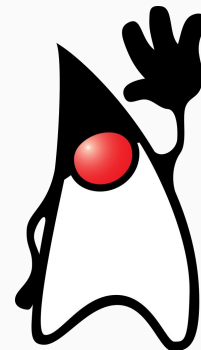
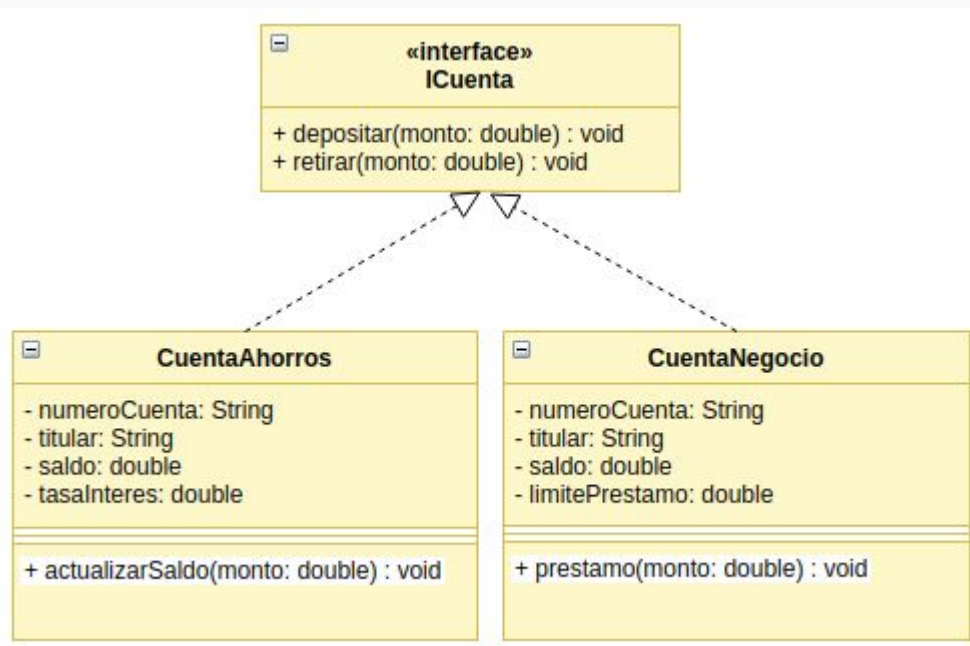
Cómo son constantes, los nombres de los campos de un Enum deben estar en mayúsculas.

```
public enum Day {  
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,  
    THURSDAY, FRIDAY, SATURDAY  
}
```



Interface

Una interface es una clase completamente abstracta, una clase sin implementación.

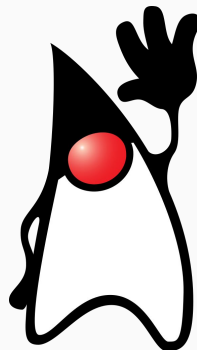


Clase Abstracta vs Interfaces

Diferencias importantes:

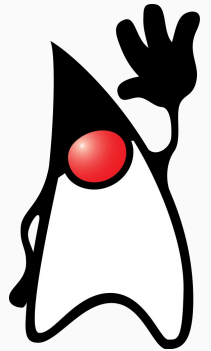
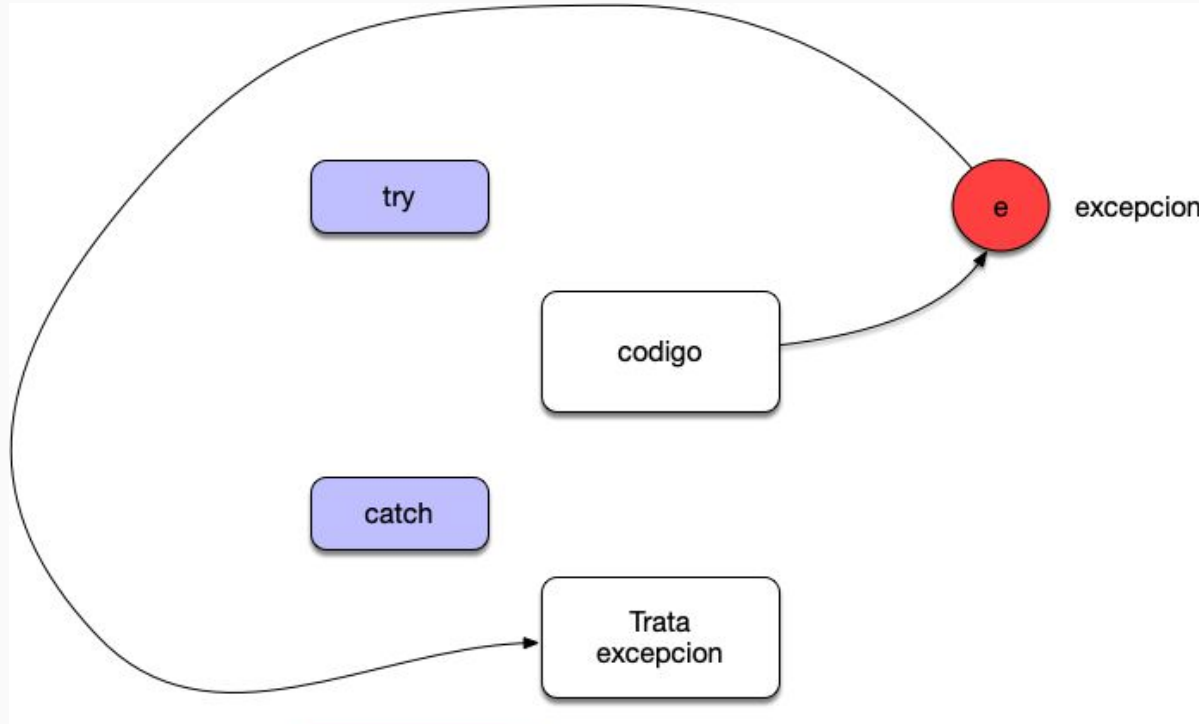
Una clase abstracta puede estar ***parcialmente implementada***. Una interfaz está limitada a métodos ***públicos y abstractos***.

Una clase abstracta puede heredar de ***una única clase***. La interfaz proporciona una forma de ***herencia múltiple***.

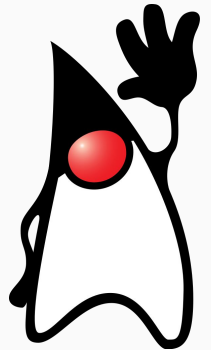
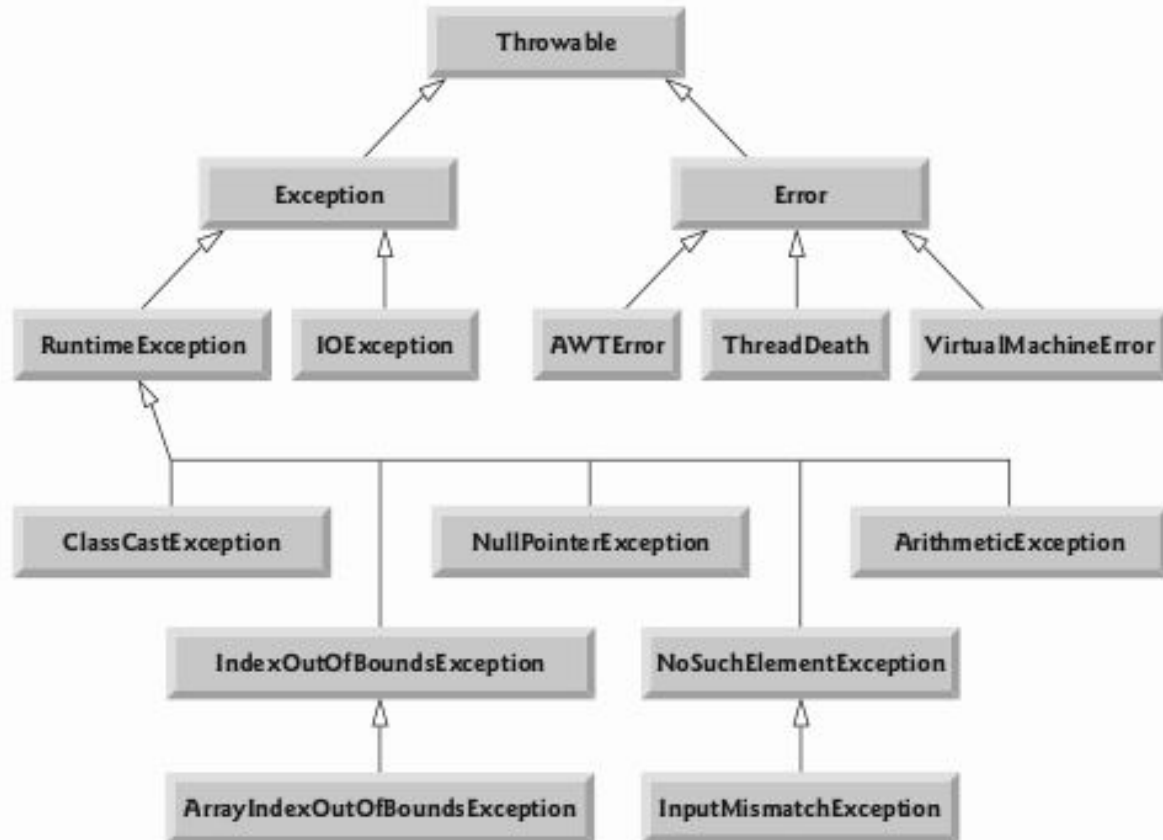


Excepcion

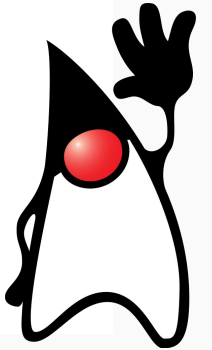
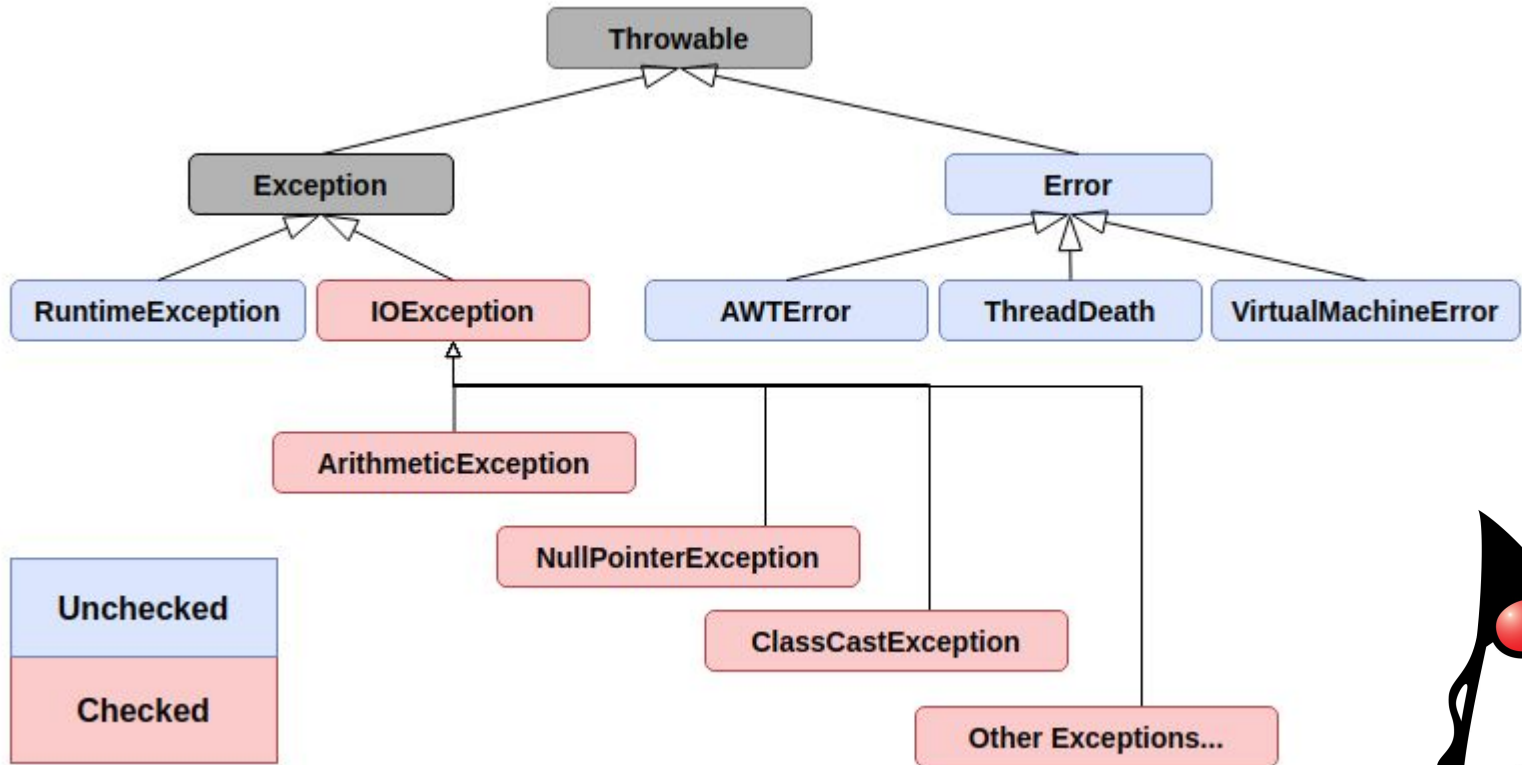
Es la indicación de un problema que ocurre durante la ejecución de un programa.



Jerarquía de excepciones



Jerarquía de excepciones



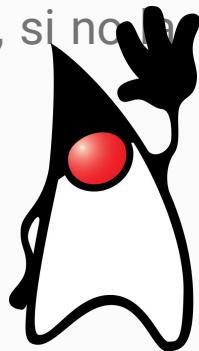
Jerarquía de excepciones

Unchecked

- Son aquellas que dejan al programador tomar la decisión de la conveniencia de atraparla o no.

Checked

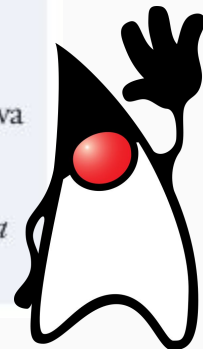
- Deben ser capturadas en forma obligatoria por nuestro programa, si no lo capturamos no se compila nuestra aplicación.



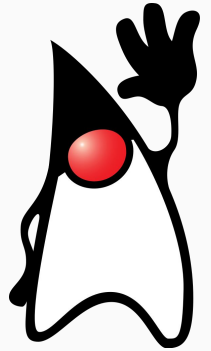
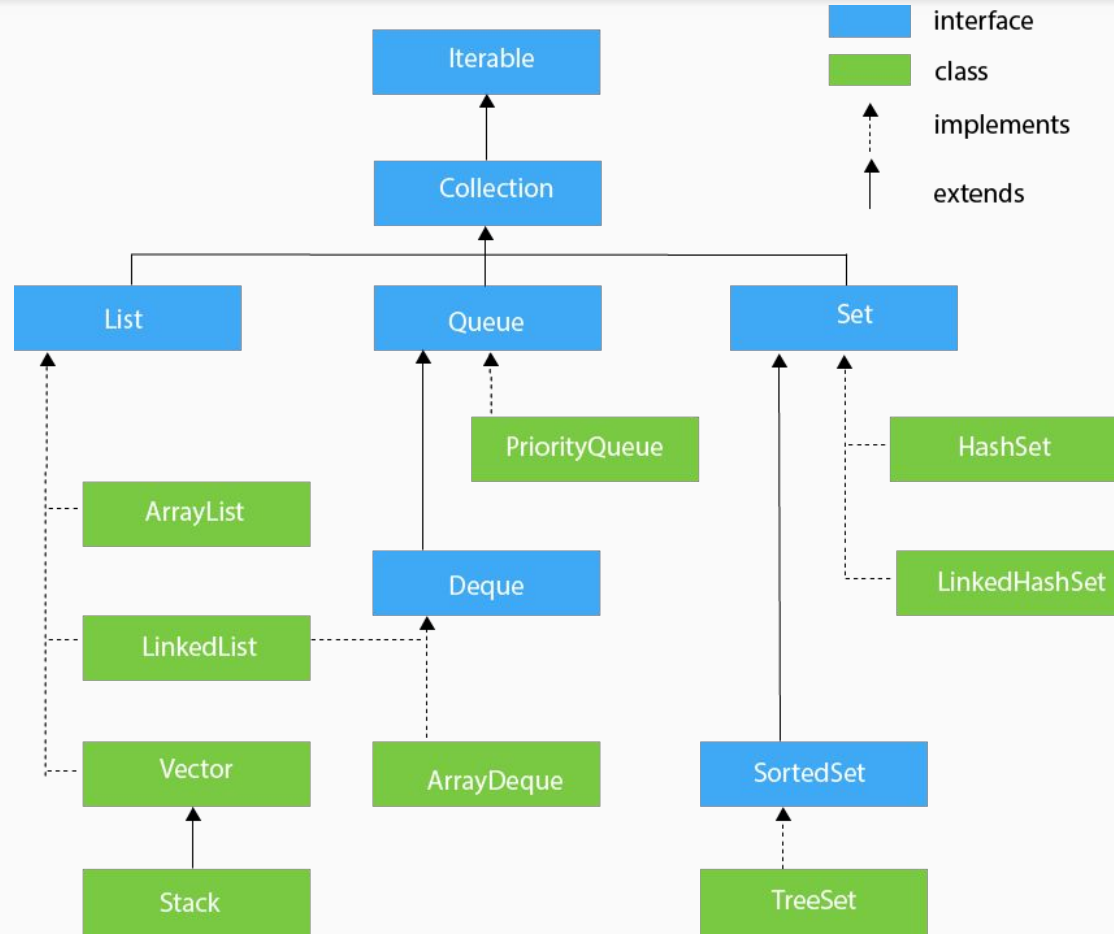
Collections

Java dispone de un conjunto de clases para agrupar objetos; se conocen como colecciones.

Interfaz	Descripción
Collection	La interfaz raíz en la jerarquía de colecciones a partir de la cual se derivan las interfaces <code>Set</code> , <code>Queue</code> y <code>List</code> .
Set	Una colección que <i>no</i> contiene duplicados.
List	Una colección ordenada que <i>puede</i> contener elementos duplicados.
Map	Una colección que asocia claves con valores y <i>no puede</i> contener claves duplicadas. <code>Map</code> no se deriva de <code>Collection</code> .
Queue	Por lo general es una colección del tipo <i>primero en entrar, primero en salir</i> , que modela a una <i>línea de espera</i> ; pueden especificarse otros órdenes.

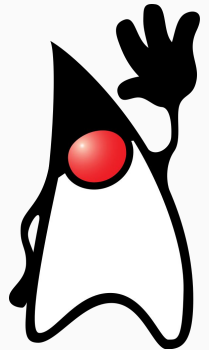
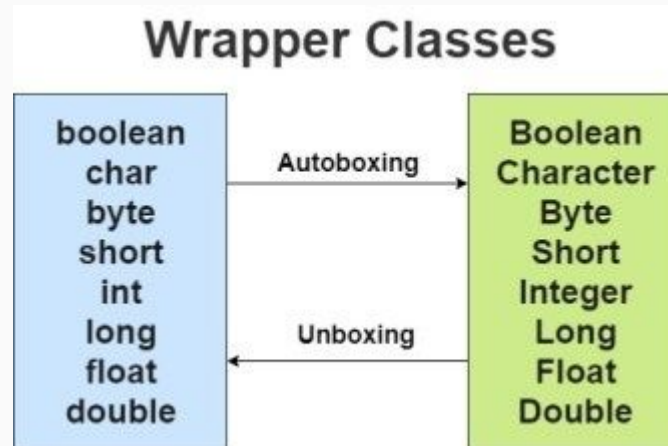


Collections



Wrapper Classes

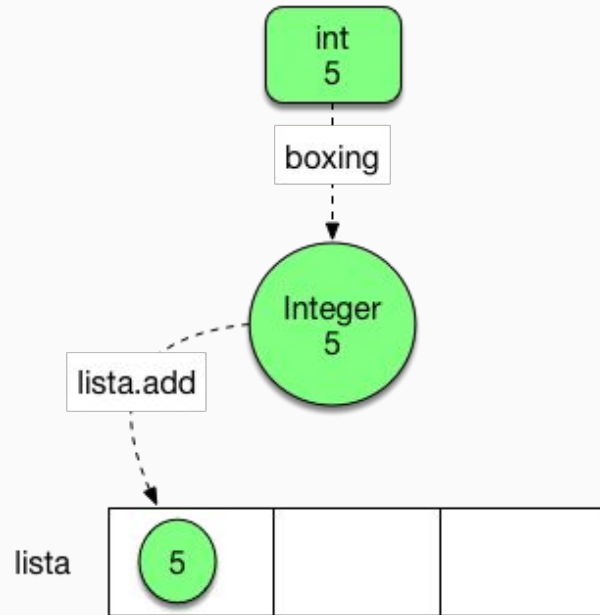
Cada tipo primitivo (listado en el apéndice D) tiene su correspondiente clase de envoltura de tipo (en el paquete java.lang). Estas clases se llaman Boolean , Byte , Character , Double , Float , Integer , Long y Short ; nos permiten manipular valores de tipos primitivos como objetos.



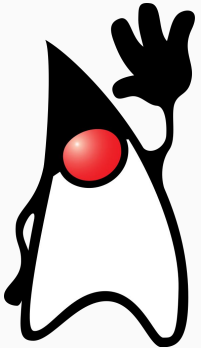
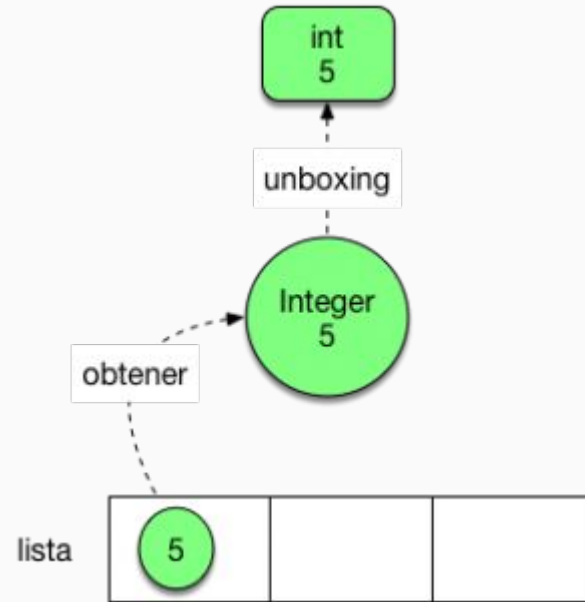
Unboxing vs Boxing

```
Integer[] lista = new Integer[10];
```

```
lista[0] = 5; /* boxing */
```



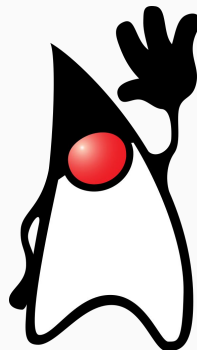
```
int valor = lista[0]; /* unboxing */
```



Interfaz Collection

Contiene operaciones como agregar, borrar y comparar objetos (o elementos) en una colección.

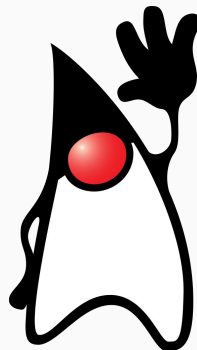
Además proporciona un método que devuelve un objeto Iterator, el cual permite recorrer toda la colección y eliminar elementos de la misma durante la iteración.



Interface List

Un objeto List (conocido como secuencia) es un objeto Collection ordenado que puede contener elementos duplicados.

La interfaz List es implementada por varias clases, incluyendo a ArrayList , LinkedList y Vector .

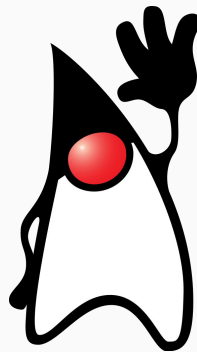



Interface List

En Java 7 se introdujo la inferencia de tipos con la notación <> (conocida como la notación diamante) en instrucciones que declaran y crean variables y objetos de tipo genérico.

```
List<String> lista = new ArrayList<String>();
```

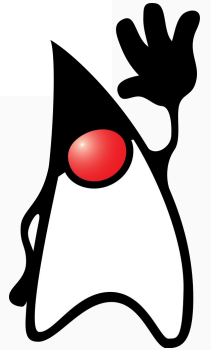
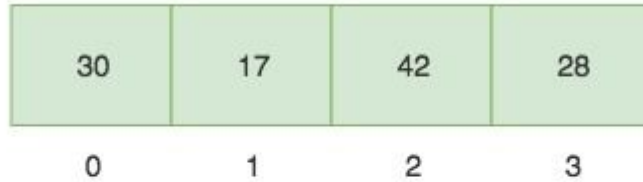
```
List<String> lista = new ArrayList<>();
```



ArrayList

Proporciona una matriz de tamaño variable e implementa la interfaz List.

Java ArrayList
Representation



LinkedList

Se basa en la implementación de listas doblemente enlazadas.

Java LinkedList
Representation

