



# Sun Certified Java Programmer ( SCJP )

Noel Mamoghli





Noel Mamoghli



[es.linkedin.com/in/noelmd/](https://es.linkedin.com/in/noelmd/)

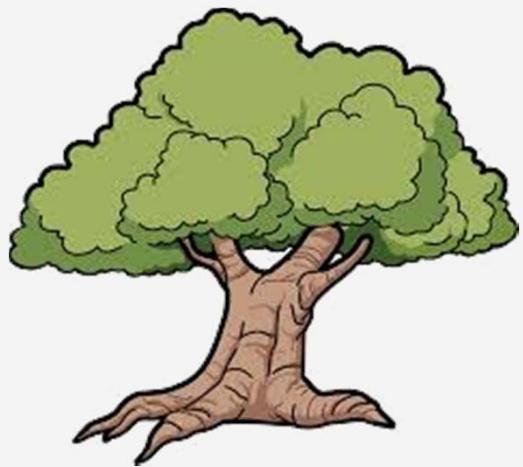


**noel.nmd@gmail.com**





- 
- #0 *Conceptos Generales e Infraestructura*
  - #1 Declaraciones, Inicialización y Ámbito
  - #2 Control de Flujo de Ejecución
  - #3 Contenidos APIs Principales
  - #4 Conurrencia
  - #5 Conceptos OO
  - #6 Colecciones y Genéricos
  - #7 Fundamentos



**James Gosling**



**THE NAME...**



**ACRONYM**

James Gosling,

Arthur

Van Hoff, y

Andy Bechtolsheim



**ACRONYM**

Just  
Another  
Vague  
**Acronym**



ACRONYM



Orientado a  
Objetos

Maquina  
Virtual



Multi-  
Dispositivo

Herencia  
Simple

# Java SE Version History (Green: Major; Blue: Minor)



# Java EE: Past & Present



Enterprise Java Platform

## J2EE 1.2

- Servlet
- JSP
- EJB
- JMS
- RMI/IOP

Robustness

## J2EE 1.3

- CMP
- Connector Architecture

Web Services

## J2EE 1.4

- Web Services Management
- Deployment
- Async. Connector

Ease of Development

## Java EE 5

- Ease of Development Annotations
- EJB 3.0
- Persistence
- New and Updated Web Services

Rightsizing

## Java EE 6

- Pruning
- Extensibility Profiles
- Ease of Development
- EJB Lite
- RESTful Services
- Dependency Ejection

## Web Profile

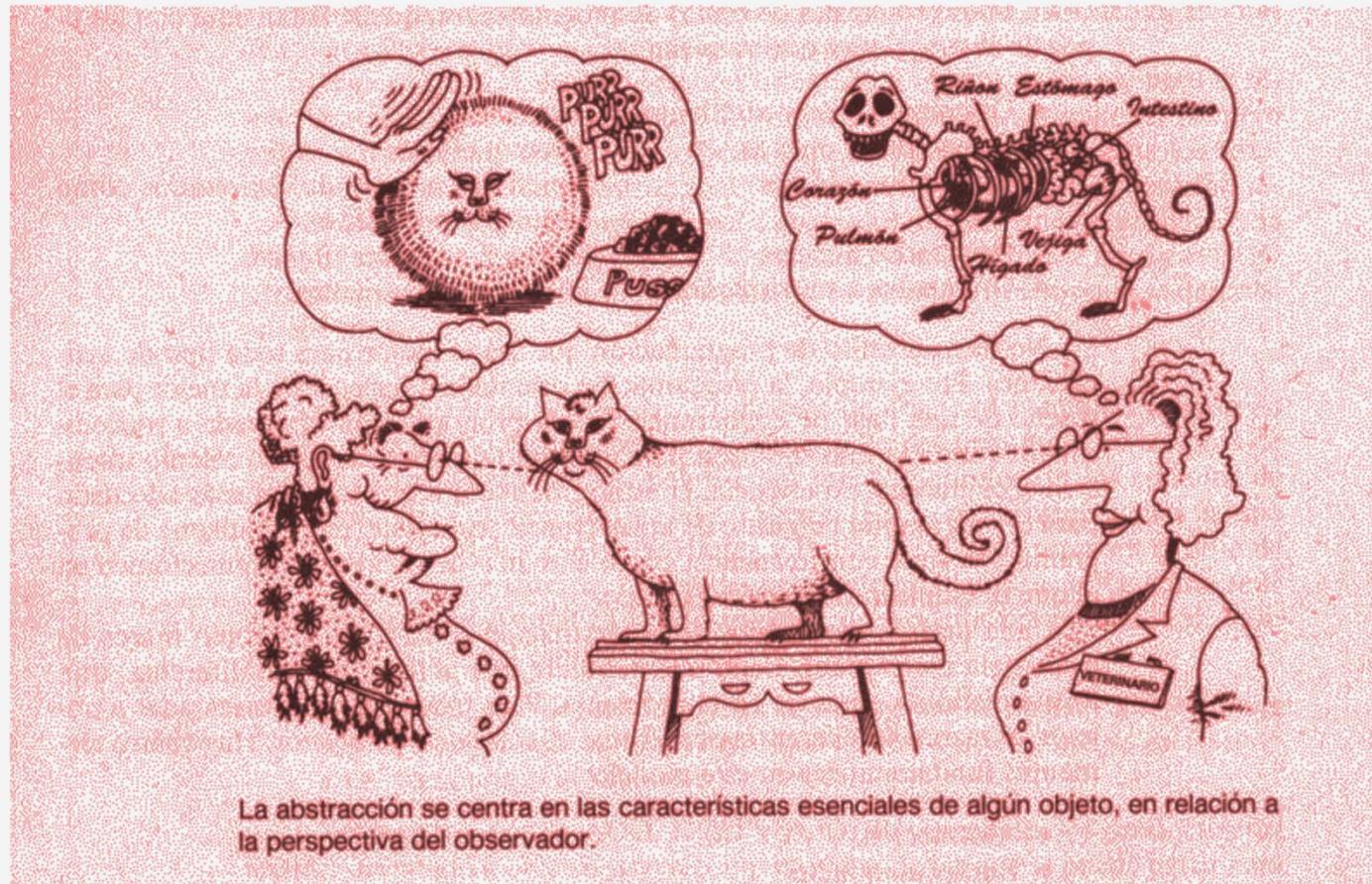
# Orientación a Objetos



# Orientación a Objetos



# Orientación a Objetos

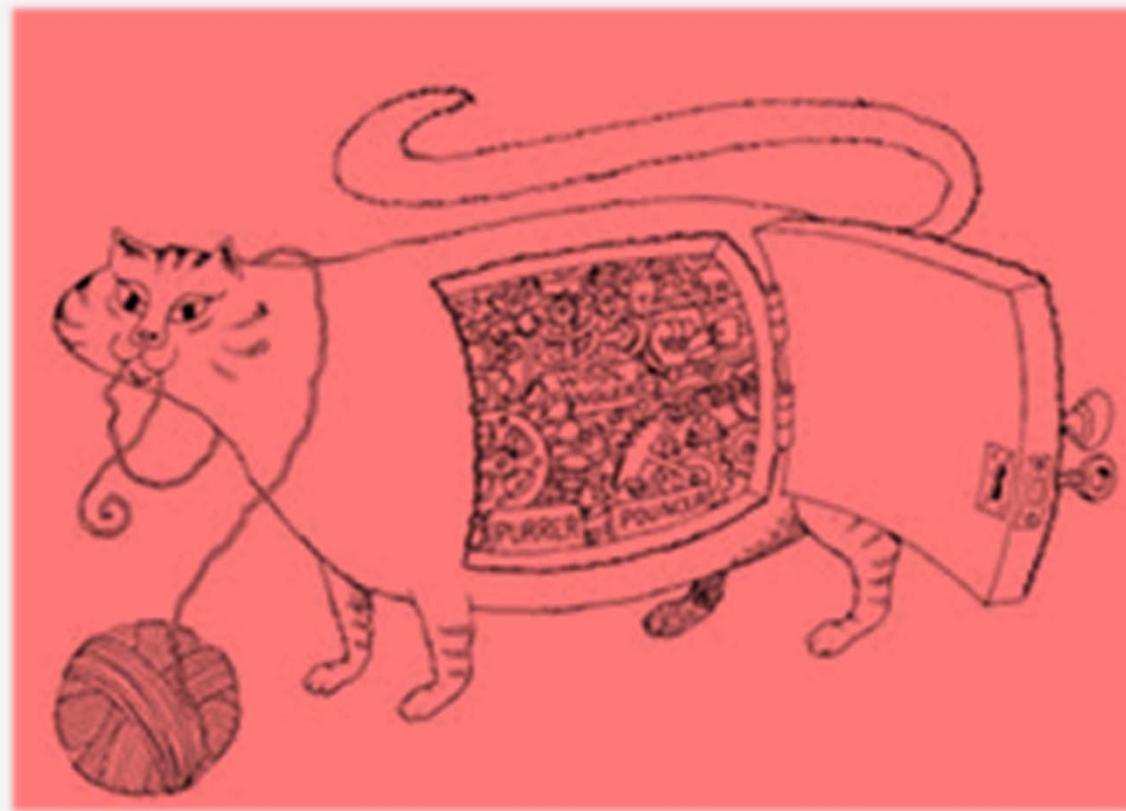


La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.

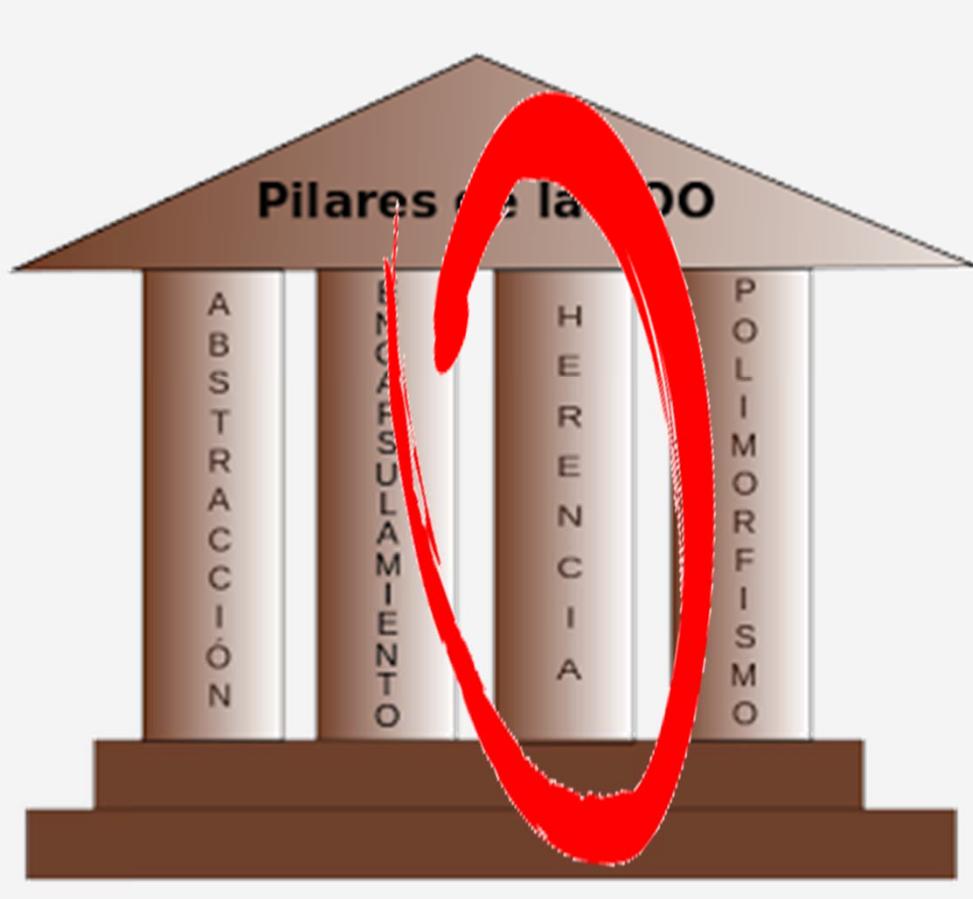
# Orientación a Objetos



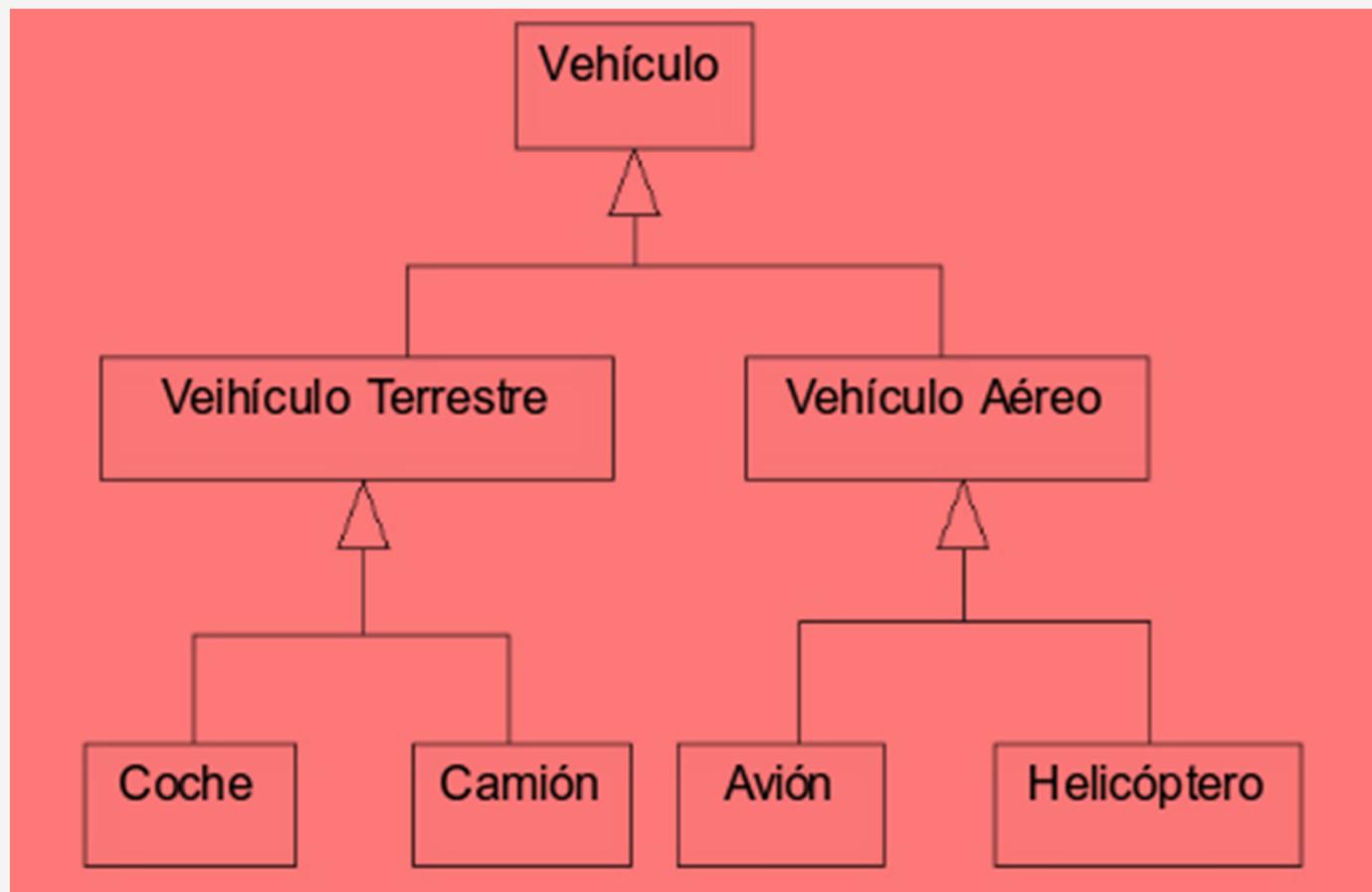
# Orientación a Objetos



# Orientación a Objetos



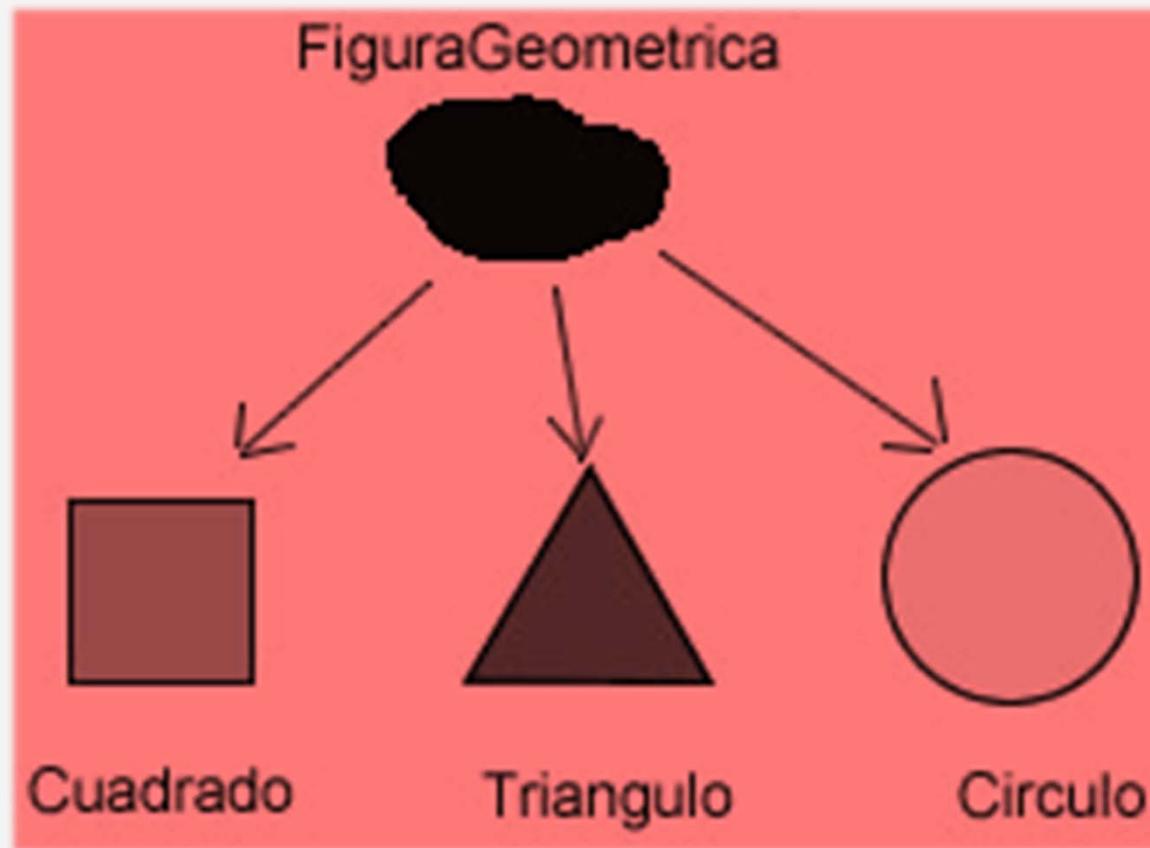
# Orientación a Objetos



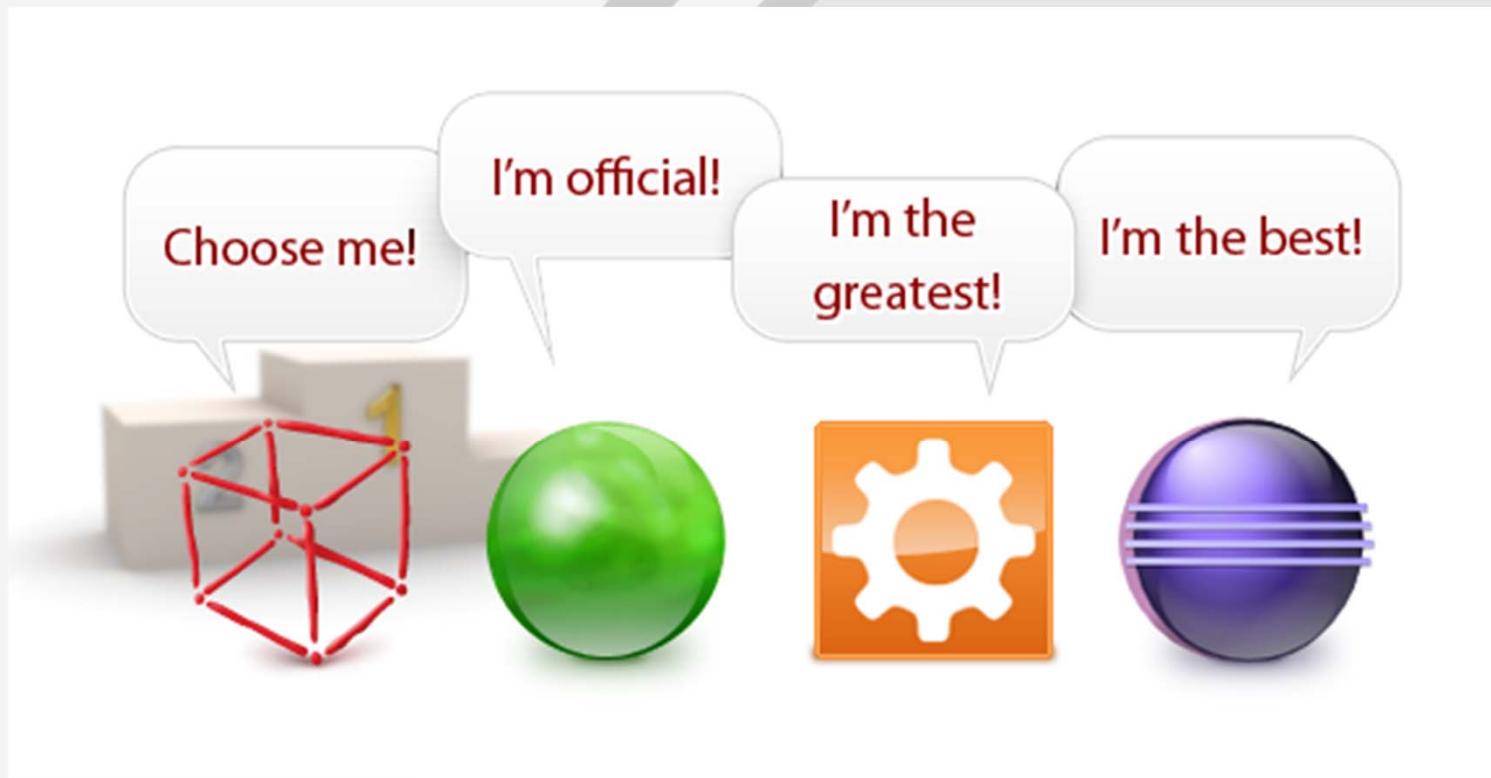
# Orientación a Objetos



# Orientación a Objetos



# JAVA IDEs



# JAVA IDEs

Resaltado de sintaxis

Generación de código

Resaltado de errores y warning

Control de Versiones

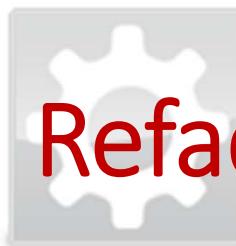
Navegación

Modo Debug

Refactorización

Completitud de código

Soporte a otras tecnologías



<https://netbeans.org/>



<https://eclipse.org/>



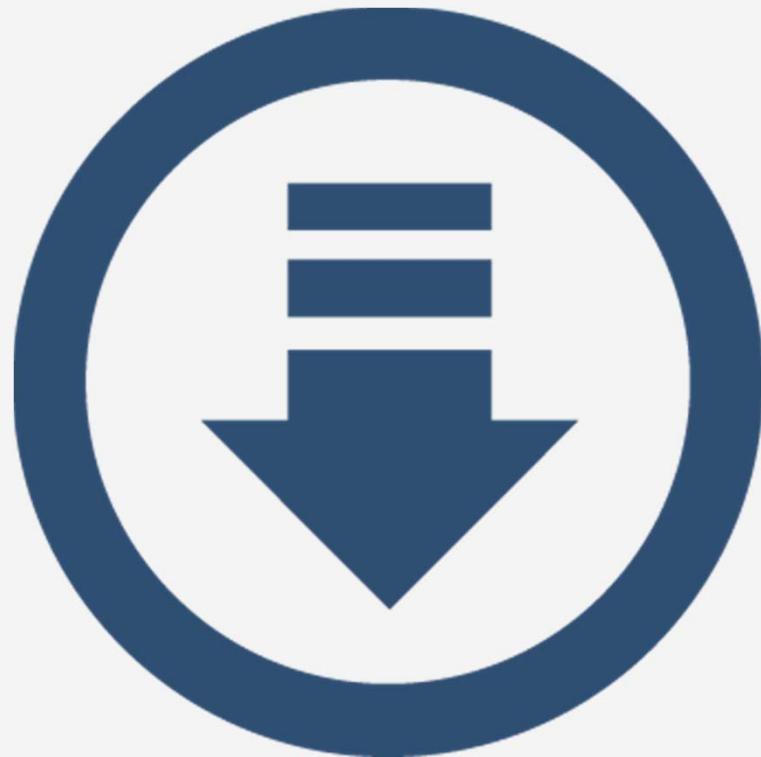
**hay otros...**

<http://www.jcreator.com/>



<https://www.jetbrains.com/idea/>

# Práctica: “JAVA IDEs”





#1

# Declaraciones, Inicialización y Ámbito

# Class, Import y Package



- Sólo una clase pública por fuente
- Comentarios no cuentan
- Si hay clase pública, el nombre del fuente debe coincidir
- La sentencia *package* debe ser siempre la primera del fuente.

# Class, Import y Package



```
package es.hpe.scjp.section1;  
  
import java.util.*;  
import es.noelmd.something.*;  
import this.is.anexample;  
  
class FirstClass { ...}  
class SecondClass { ...}  
public class PublicClass {...}
```

# Acceso a clases



```
package es.scjp.noelmd.c01;

public class TemaOne
{
    public static void main (String arfs [])
    {
        new ClassA();
    }
}

class ClassA
{
    private ClassB classB;
    private ClassC classC;

    public ClassA ()
    {
        System.out.println("Entrando en el constructor de ClassA");

        System.out.println("Creando instancia de ClassB");

        this.classB = new ClassB();

        System.out.println("Creando instancia de ClassC");

        this.classC = new ClassC();
    }
}

class ClassB
{
}

class ClassC extends ClassA
{
}
```

# Modificadores de Acceso



# Modificadores de Acceso

# default

Sin modificador específico.  
Cualquier clase del paquete puede  
acceder.



# Modificadores de Acceso

# public

Puede ser accedida desde  
cualquier clase que importe  
el paquete o la clase.



# Modificadores de Acceso

## final

La clase NO podrá ser extendida.



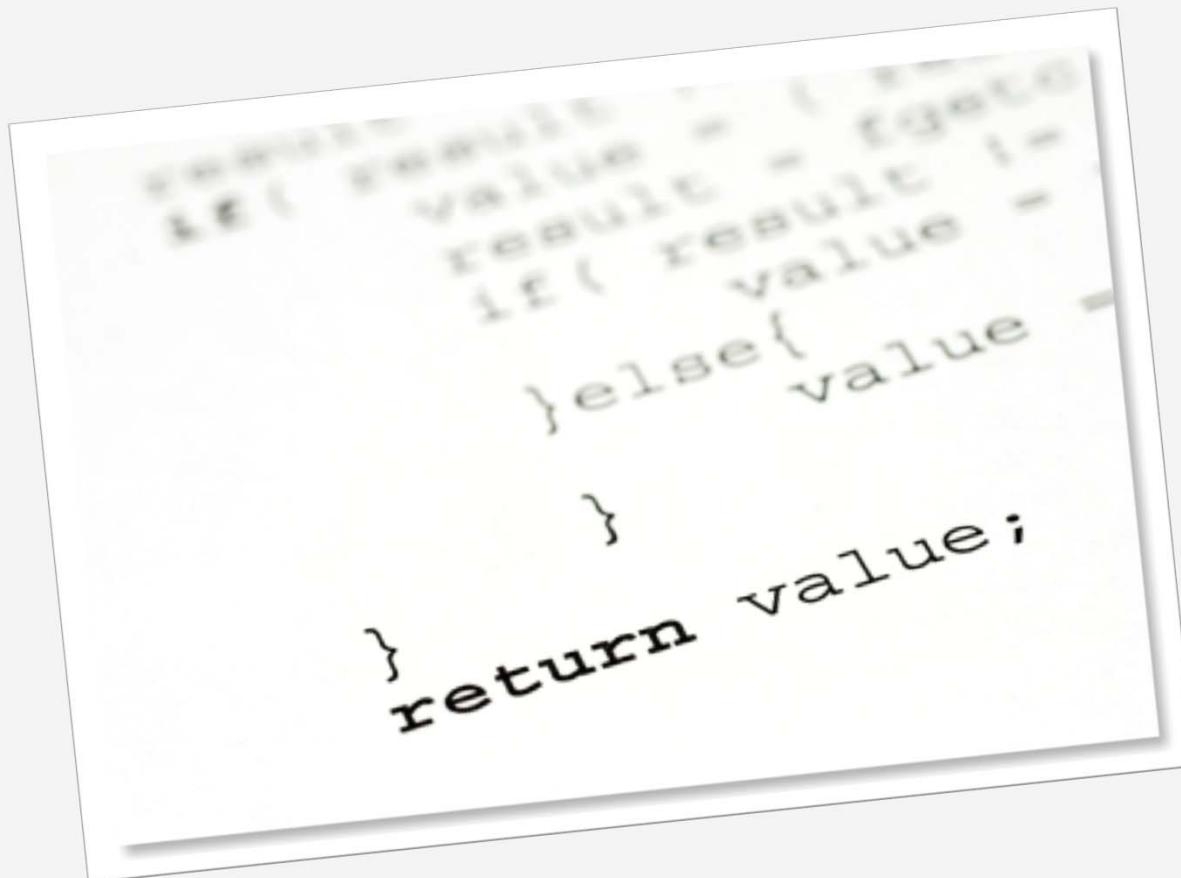
# Modificadores de Acceso

## abstract

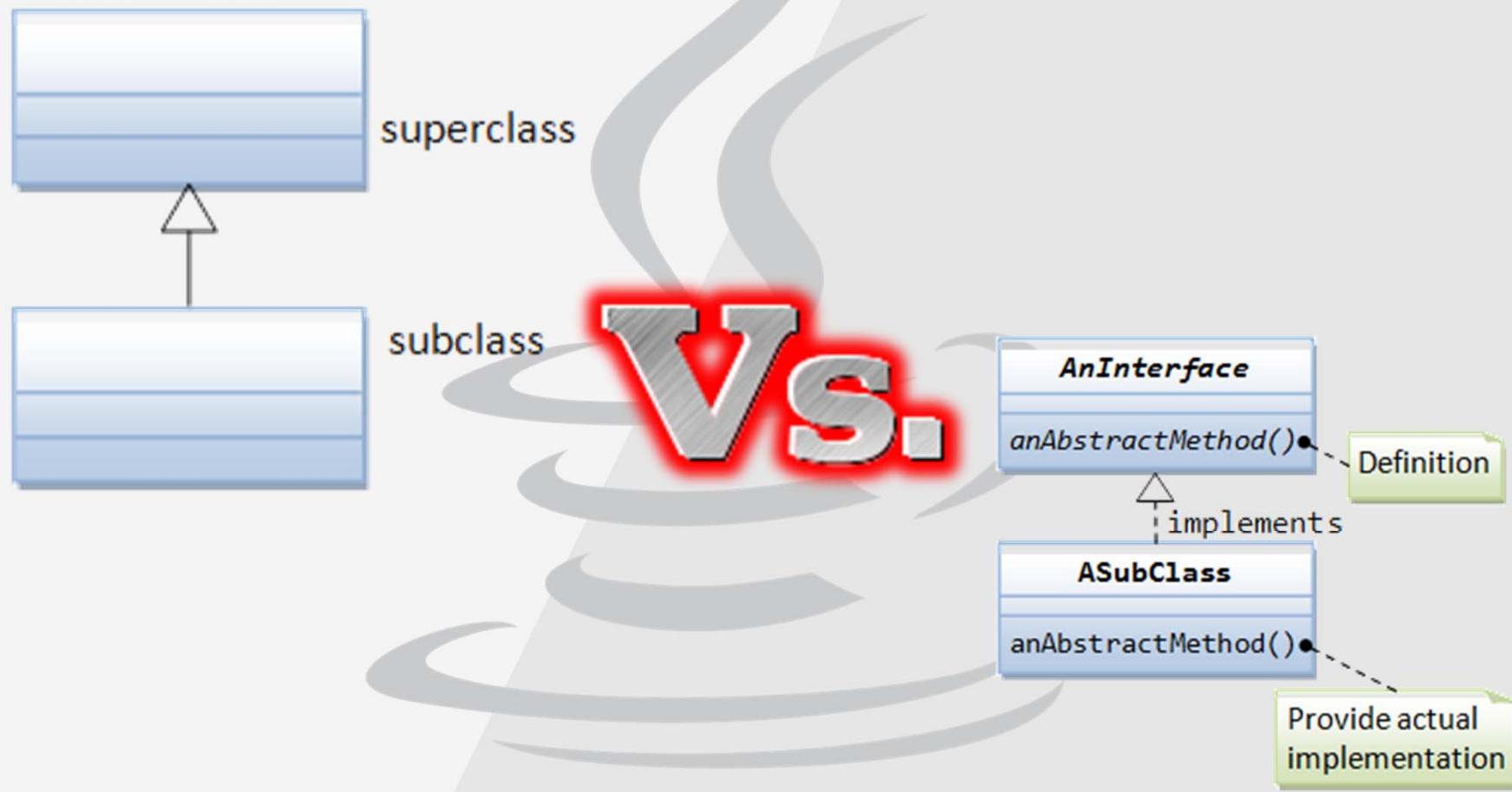
La clase NO podrá ser instanciada.



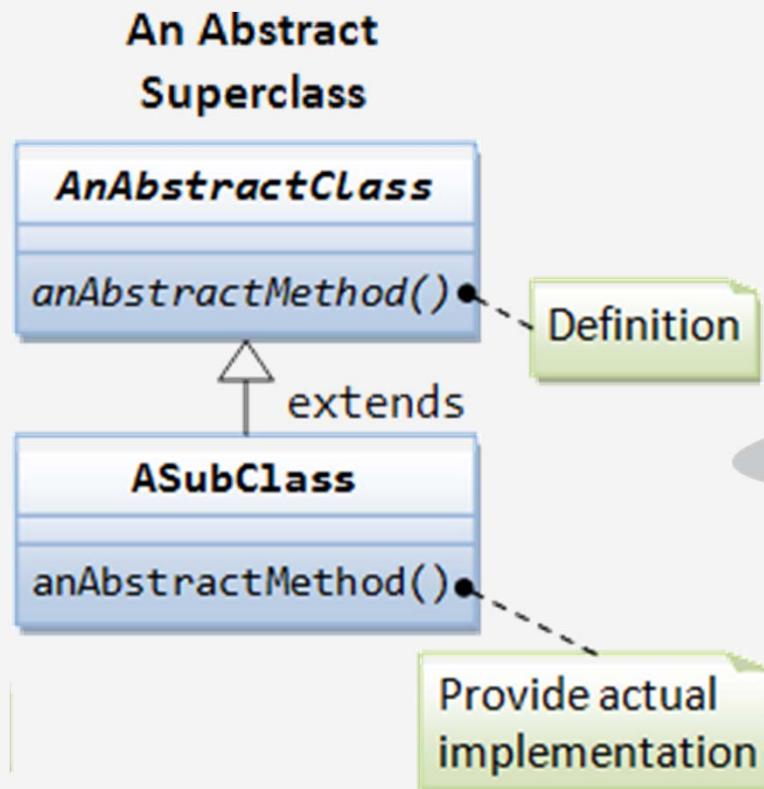
# Práctica: “Modificadores Acceso a Clase”



# Abstracción e Interfaces

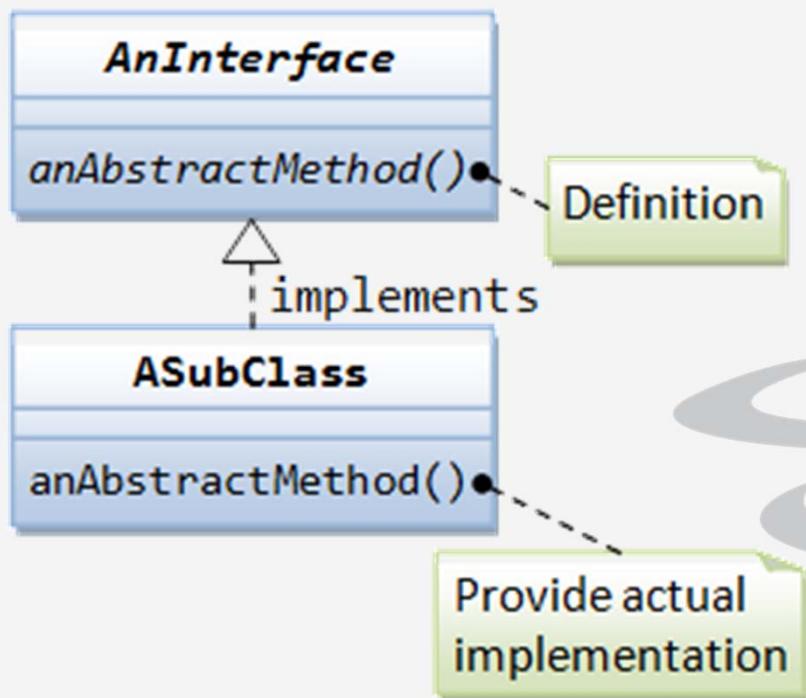


# Abstracción e Interfaces



- No es instanciable
- Métodos abstractos
- Métodos NO abstractos
- Subclases NO abstractas deben implementar métodos abstractos
- Subclases abstractas

# Abstracción e Interfaces



Un interfaz es un contrato.  
Especifica una serie de propiedades que debe cumplir quien acepta el contrato ( su implementación ).  
No dice nada en absoluto CÓMO debe cumplir el contrato.

Una interfaz define el QUÉ, pero no el CÓMO.

# Identificadores



# Identificadores

- Los identificadores de comenzar con una letra, el carácter \$ u otro carácter de unión como \_
- No pueden empezar con número.
- Después del primer carácter pueden incluir combinaciones de letras números y caracteres de unión.
- No hay límite en el número de caracteres.
- No se pueden utilizar palabras reservas del lenguaje
- **Son sensibles a mayúsculas – minúsculas (case-sensitive).**

# Modificadores de Acceso II



# Modificadores de Acceso II

public

Si la clase que lo contiene  
es visible cualquiera puede  
acceder.



# Modificadores de Acceso II

```
package book;
import cert.*;
class Goo {
    public static void main(String[] args) {
        Sludge o = new Sludge();
        o.testIt();
    }
}

package cert;
public class Sludge {
    public void testIt() {
        System.out.println("sludge");
    }
}
```

# Modificadores de Acceso II

## private

No puede ser accedido por ningún mecanismo.

No se puede realizar “sobre-escritura”, solo es homónimo.



# Modificadores de Acceso II

```
package cert;
public class Roo {
    private String doRooThings() {
        // imagine the fun code that goes here, but
        // no other class will know
        return "fun";
    }
}

package cert; //Cloo and Roo are in the same package
class Cloo extends Roo {
    //Still OK, superclass Roo is public
    public void testCloo() {
        System.out.println(doRooThings());
        //Compiler error!
    }
}
```

# Modificadores de Acceso II

## default

Sólo puede ser accedido  
por clases que se  
encuentran en el mismo  
paquete.



# Modificadores de Acceso II

## protected

Solo puede ser accedido por subclases de la clase que define este ámbito.



# Modificadores de Acceso II

Visibility	Public	Protected	Default	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, <i>through inheritance</i>	No	No
From any non-subclass class outside the package	Yes	No	No	No

# Modificadores de Acceso III



# Modificadores de Acceso III

final



# Modificadores de Acceso III

## final

Impide la sobreescritura de un método por las subclases de una superclase.



# Modificadores de Acceso III

```
package cert;
public class Suppa{
    public final String noMore() {
        // the code that goes here, but
        // no other subclass can override the method
        return "stop";
    }
}

package cert.fine.grained;
public class Child extends Suppa{
    @override
    public String noMore() {
        //Compiler error!
    }
}
```

# Modificadores de Acceso III

```
package cert;
public class Suppa{
    public final String noMore() {
        // the code that goes here, but
        // no other subclass can override the method
        return "stop";
    }
}

package cert.fin;
public class Ch {
    @override
    public String noMore() {
        //Com
    }
}
```

# Modificadores de Acceso III

## final

En un argumento obliga a que dicho valor se mantenga constante en dicho método.



# Modificadores de Acceso III

```
package cert;
public class myClass {
    public Record getRecord(int fileNumber,
                           final int recordNumber) {
        recordNumber++;
    }
}
```

# Modificadores de Acceso III

```
package cert;
public class myClass {
    public Record getRecord(int fileNumber,
                           final Record record) {
        record = new Record();
        record.setFileNumber(fileNumber);
        return record;
    }
}
```

# Modificadores de Acceso III

## abstract

Método que ha sido declarado pero no implementado.

Sólo propio de clases abstractas.



# Modificadores de Acceso III

```
package cert;  
public class IllegalClass{  
    public abstract void  
        doSomething();  
}
```

# Modificadores de Acceso III

```
package cert;
public class IllegalClass{
    public abstract void doSomething();
}
}
```

# Modificadores de Acceso III

## synchronized

Método que puede ser accedido por un único hilo concurrentemente.

Exclusivo para métodos  
(no para variables ni clases)



# Modificadores de Acceso III

```
package cert;

public class EmployeeDTO
{
    public synchronized Record
        updateUserInfo(UserInfo user) { }
}
```

# Modificadores de Acceso III

## native

Método implementado en el lenguaje de la plataforma de la arquitectura.

Exclusivo para métodos (no variables ni clases)



# Modificadores de Acceso III

## transient

Indica a la JVM que una variable de clase no será tenida en cuenta cuando la clase vaya a ser serializada.



# Modificadores de Acceso III

## volatile

Una variable volatile implica que los diferentes hilos que la acceden sincronizan el último valor de dicha variable.



# Modificadores de Acceso III

## strictfp

Fuerza a utilizar operaciones de coma flotante conforme a la especificación IEEE 754.

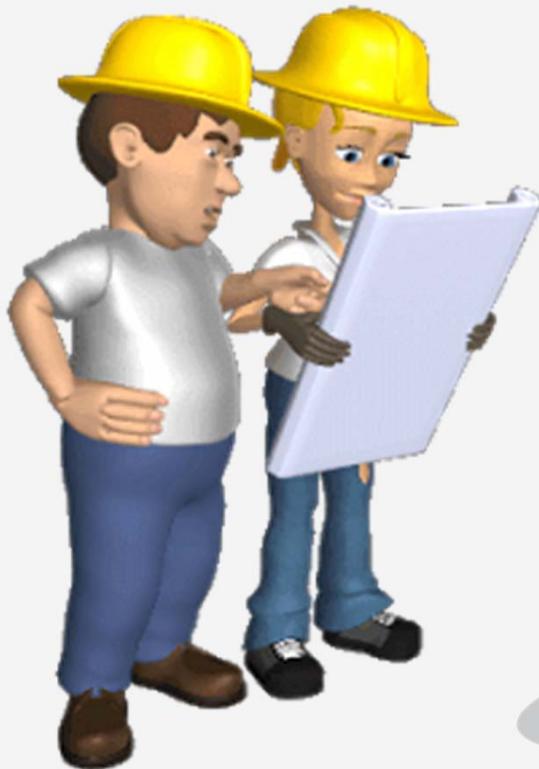
Exclusivo de clase y método.



# Constructores

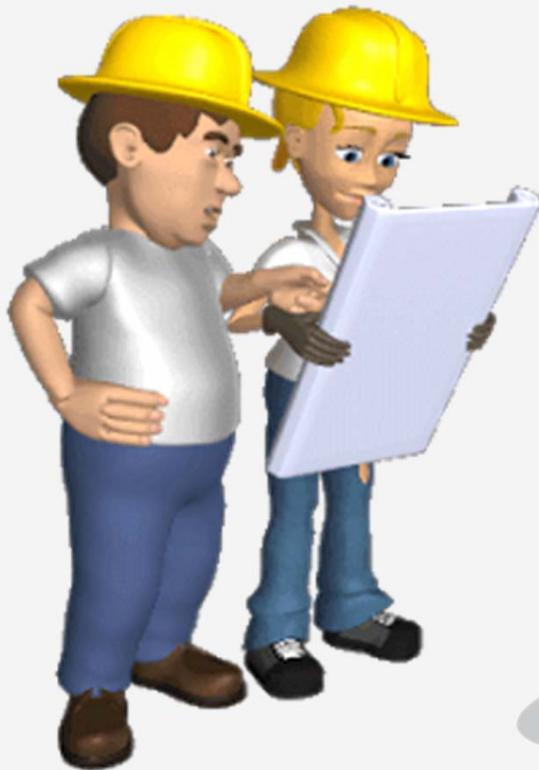


# Constructores



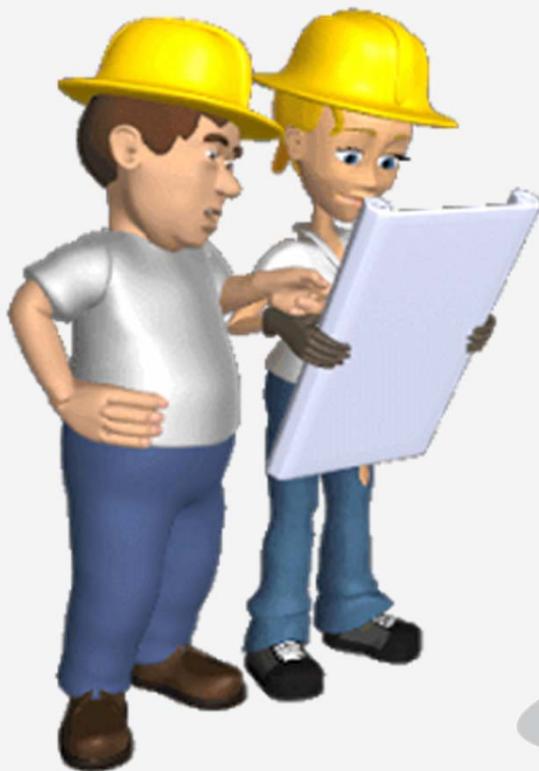
- Todas las clases deben tener al menos un constructor.
- Si no ponemos constructor se agrega el constructor por defecto.
- El constructor por defecto no tiene parámetros.
- Los constructores poseen el mismo nombre que la clase que los contiene.

# Constructores



- Todas las clases deben tener al menos un constructor.
- Si no ponemos constructor se agrega el constructor por defecto.
- El constructor por defecto no tiene parámetros.
- Los constructores poseen el mismo nombre que la clase que los contiene.
- **Los constructores no devuelven nada.**

# Constructores



- Pueden usar cualquier tipo de modificador.
- Pueden sobrecargarse.
- Las clases abstractas tienen constructores.
- Su primera sentencia implícita o explícita es siempre llamar a un constructor sobrecargado o al constructor super.
- Su ejecución es previa a cualquier llamada a método o acceso a variable de clase.
- **Sólo puede invocarse un constructor desde otro constructor o instanciando la clase.**

# Variables



# Variables

primitivo

char

boolean

long

int

float

byte

short

double



referencias

≈ punteros a  
objetos

# Variables

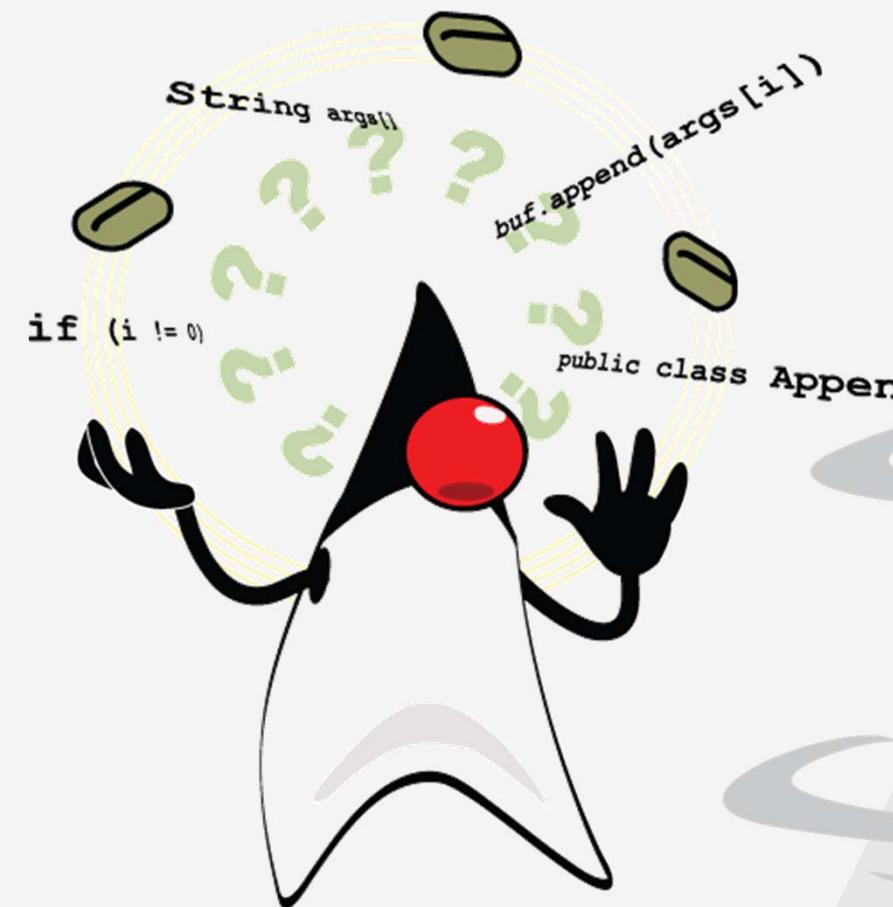
public, private, protected, default

final, transient

abstract, synchronized, strictfp, native

static → Variables de Clase

# Variables



- Las variables de clase son automáticamente inicializadas, las de método deben inicializarse específicamente.
- Una variable de clase es accesible desde cualquier punto de la clase, las de método son de acceso exclusivo en el método.
- Una variable de método puede llamarse igual que una de clase.
- El ámbito más cercano es el que prevalece, salvo cuando usamos el operador *this*.

# Arrays

## Arrays en Java

Array

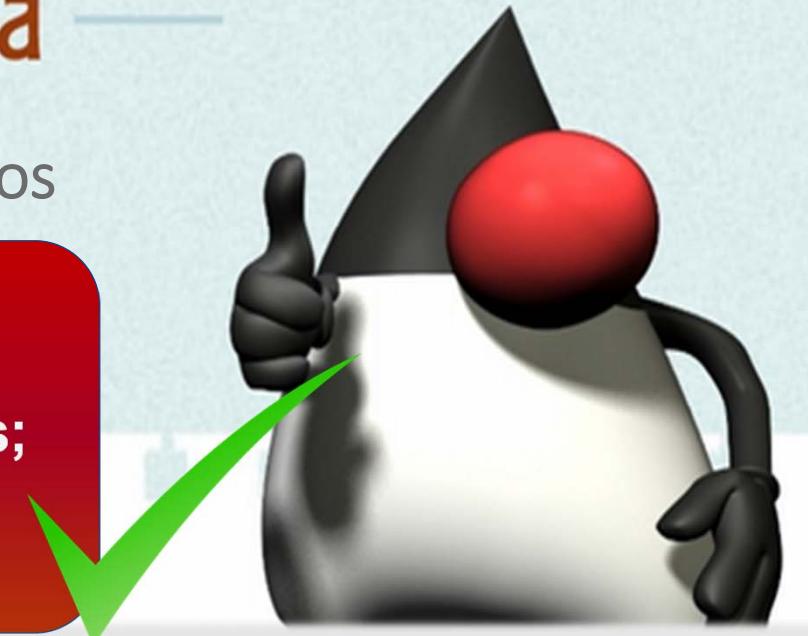


# Arrays: Declaración

## Arrays en Java

→ De tipos primitivos u objetos

```
int numeros [ ];  
float [ ] decimales;  
Coches [ ] array_coches;  
Animal pets[ ];  
String [ ] [ ] palabras;
```



# Arrays: Declaración

## Arrays en Java

→ De tipos primitivos u objetos

```
int nu[5];
```



# Arrays: Construcción

## Arrays en Java

- Se debe especificar el tamaño
- Se crean con el operador new

```
int[ ] testScores; // declaración  
testScores = new int[4]; // creación
```

```
// declaración + creación  
int[ ] testScores2 = new int[8];
```



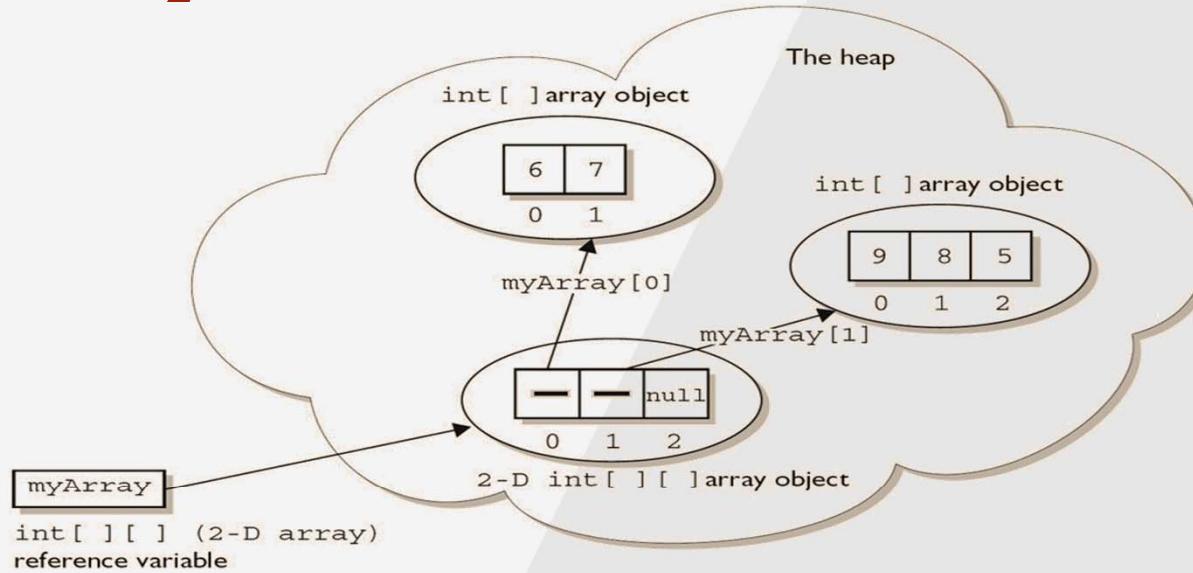
# Arrays: Multidimensión

## — Arrays en Java —

→ Son arrays que en cada posición, contienen otro array



# Arrays: Multidimensión



Picture demonstrates the result of the following code:

```
int[ ][ ] myArray = new int[3][ ];
myArray[0] = new int[2];
myArray[0][0] = 6;
myArray[0][1] = 7;
myArray[1] = new int[3];
myArray[1][0] = 9;
myArray[1][1] = 8;
myArray[1][2] = 5;
```

# Arrays: Inicialización

## Arrays en Java

- Inicializar = Asignar valores
- Por defecto se crean sin valores
- El acceso a los elementos es por índice utilizando el [ ]
- El primer elemento es siempre el 0 y el último la longitud -1



# Arrays: Inicialización

## — Arrays en Java —

- Inicializar = Asignar valores
- Por defecto se crean sin valores
- El acceso a los elementos es por índice utilizando el [ ]

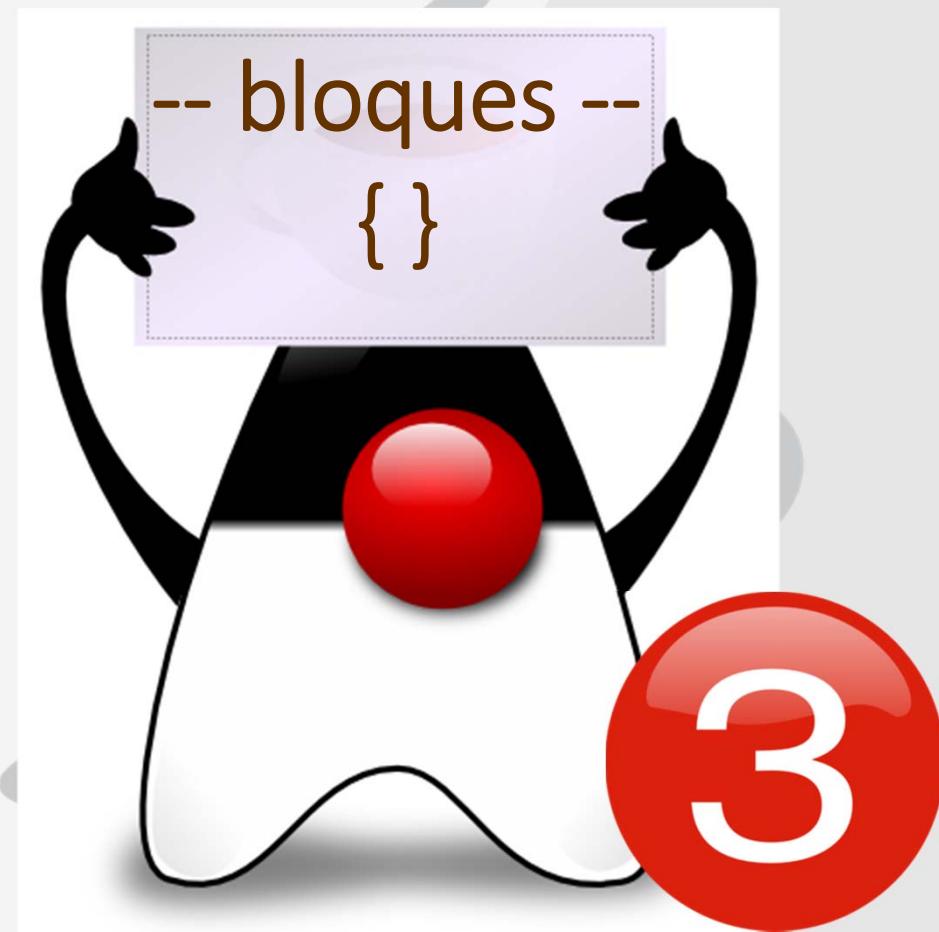
!! **ArrayIndexOutOfBoundsException** !!

El primer elemento es index 0  
0 y el ultimo la longitud -1

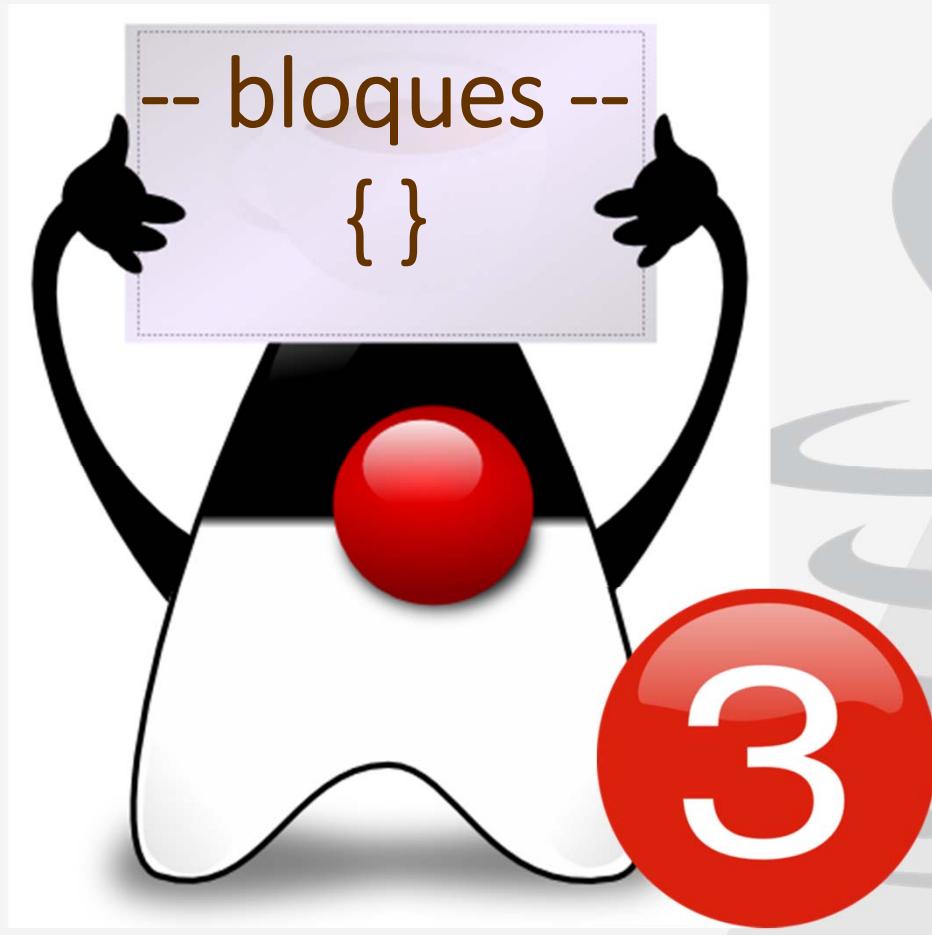
-- Error en tiempo de ejecución --



# Inicialización



# Inicialización



→ Constructores

→ Métodos

**→ Bloques**

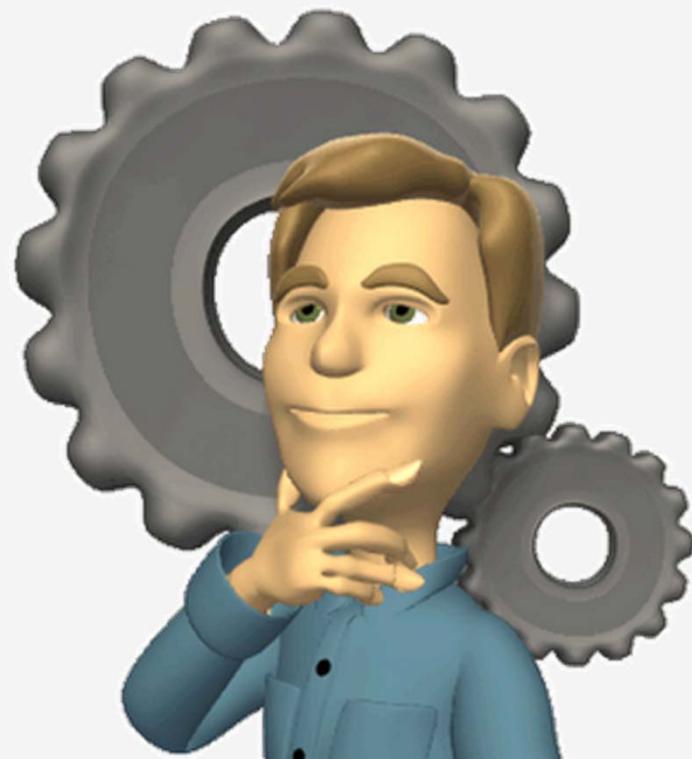
→ Bloques de instancia {}

-- Después llamada a super --

**→ Bloques estáticos o de clase**  
**static {}**

-- Al cargar la clase --

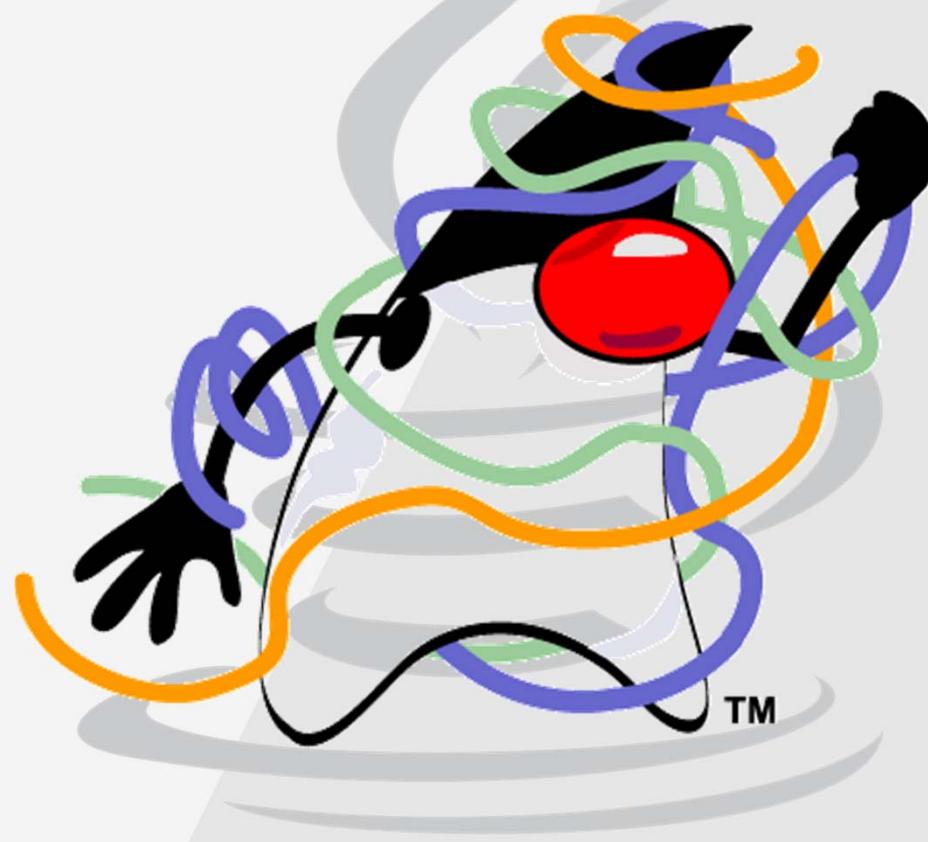
# Práctica: "Arrays"



# Práctica: "Arrays"

Código que cree un array  
Y muestre sus valores...

# Enums



# Enums



- Son variables con valores restringidos a una lista
- Se declaran en una clase propia o como miembro de clase

# Enums

→ Clase Propia

```
enum CoffeeSize { BIG, HUGE, OVERWHELMING }

class Coffee {
    CoffeeSize size;
}

public class CoffeeTest1 {
    public static void main(String[] args) {
        Coffee drink = new Coffee();
        drink.size = CoffeeSize.BIG;
    }
}
```



# Enums

→ En otra Clase



```
class Coffee2 {  
    enum CoffeeSize {BIG, HUGE,OVERWHELMING }  
    CoffeeSize size;  
}  
  
public class CoffeeTest2 {  
    public static void main(String[] args) {  
        Coffee2 drink = new Coffee2();  
        drink.size = Coffee2.CoffeeSize.BIG;  
    }  
}
```

# Enums



```
public class CoffeeTest1 {  
    public static void main(String[] args) {  
        enum CoffeeSize { BIG, HUGE,  
OVERWELMING }  
        Coffee drink = new Coffee();  
        drink.size = CoffeeSize.BIG;  
    }  
}
```

# Enums

→ Puede tener constructores

```
enum CoffeeSize {  
    BIG(8), HUGE(10), OVERWHELMING(16);
```

```
CoffeeSize(int ounces)  
{  
    this.ounces = ounces;  
}  
  
public int getOunces()  
{  
    return ounces;  
}
```



# Enums



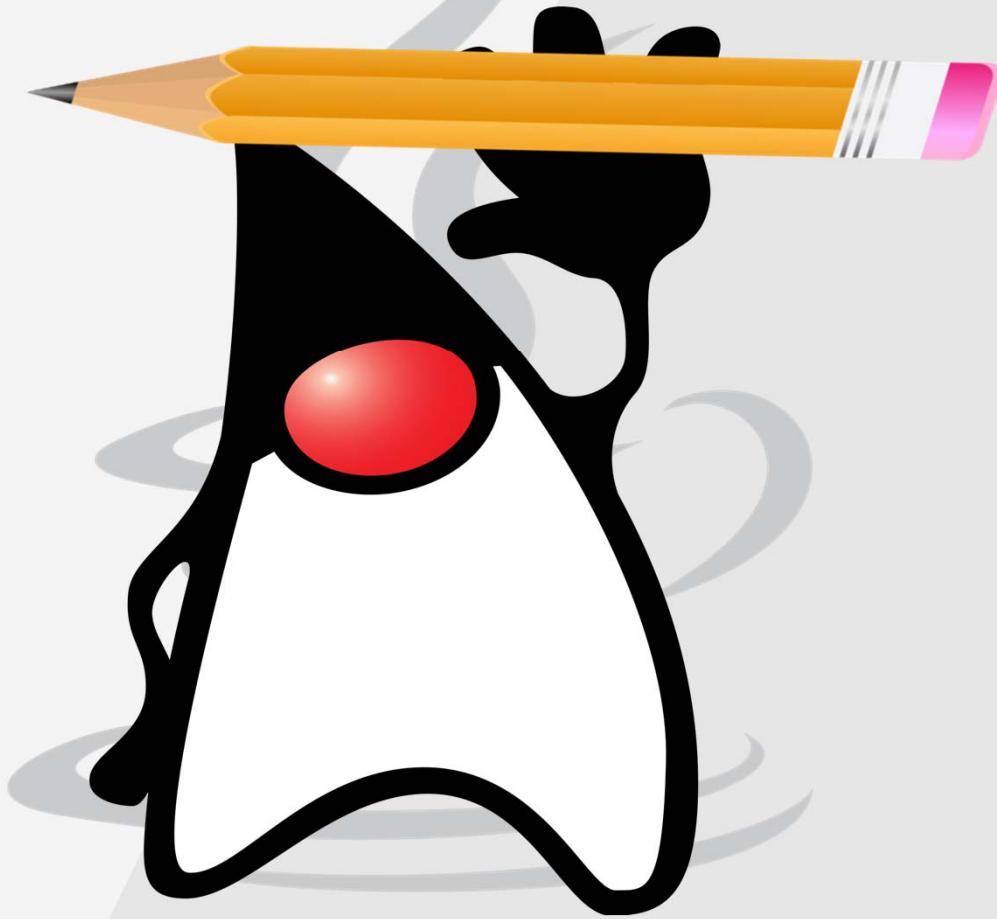
```
class Coffee {  
    CoffeeSize size;  
    public static void main(String[] args)  
    {  
        Coffee drink1 = new Coffee();  
        drink1.size = CoffeeSize.BIG;  
        Coffee drink2 = new Coffee();  
        drink2.size = CoffeeSize.OVERWHELMING;  
        System.out.println(drink1.size.getOunces());  
  
        for(CoffeeSize cs: CoffeeSize.values())  
            System.out.println(cs + " " +  
                cs.getOunces());  
    }  
}
```

# Enums



```
class Coffee {  
    CoffeeSize size;  
    public static void main(String[] args)  
    {  
        Coffee drink1 = new Coffee();  
        drink1.size = CoffeeSize.BIG;  
        C  
        size = CoffeeSize.MEDIUM; HELMING;  
        System.out.println(drink1.size.getOunces());  
  
        for(CoffeeSize cs: CoffeeSize.values())  
            System.out.println(cs + " " +  
                cs.getOunces());  
    }  
}
```

# Sobreescritura



# Sobreescritura

```
public class Animal
{
    public void comer()
    {
        System.out.println("Animal
                           Genérico comiendo");
    }
}
```

```
public class Caballo extends Animal
{
    public void comer()
    {
        System.out.println("Animal
                           Caballo comiendo");
    }
}
```

# Sobreescritura

- 
- El argumento debe coincidir con el original.
  - El tipo de retorno debe ser igual o subclase .
  - El nivel de acceso no puede ser más restrictivo.
  - No se pueden lanzar otras excepciones declaradas pero si no declaradas ( run –time ).
  - Métodos estáticos o finales NO pueden ser sobreescritos.
  - Sólo métodos heredados pueden sobreescribirse.

# Sobreescritura

→ Invocación al método original.

```
public class Animal {  
    public void comer() {}  
    public void correr()  
    }  
}  
  
class Caballo extends Animal {  
    public void correr() {  
        super.correr();  
    }  
}
```

# Sobreescritura

→ Invocación polimórfica.

```
class Animal {  
    public void comer() throws Exception {  
        // throws an Exception  
    }  
}  
class Perro extends Animal {  
    public void comer() {  
        // no Exceptions  
    }  
    public static void main(String [] args) {  
        Animal a = new Perro();  
        Perro p = new Perro();  
        p.comer();  
        a.comer();  
    }  
}
```

# Sobreescritura

→ Invocación polimórfica.

```
class Animal {  
    public void comer() throws Exception {  
        // throws an Exception  
    }  
}  
class Perro extends Animal {  
    public void comer() {  
        no Exceptions  
    }  
}  
public static void main(String [] args) {  
    Animal a = new Perro();  
    Perro p = new Perro();  
    p.comer();  
    a.comer();  
}
```

# Sobrecarga



# Sobrecarga



```
class Animal {  
    public void comer()  
    {  
        // código comer generico  
    }  
}  
  
class Perro extends Animal {  
    public boolean comer (Integer unidades)  
    {  
        // código comer unidades  
    }  
    public void comer (Alimento [ ] alimentos)  
        throws IndigestionException  
    {  
        // código comer objetos Alimento  
    }  
}
```

# Sobrecarga



- Implica un cambio en la lista de argumentos de la firma del método.
- Puede cambiar el tipo de retorno.
- Puede cambiar el modificador de acceso.
- Puede cambiar las excepciones.
- Se puede dar a nivel de clase y de subclase.

# Test de Conocimientos





#2

# Control de Flujo

# Control de Flujo

condiciones

bucles

excepciones

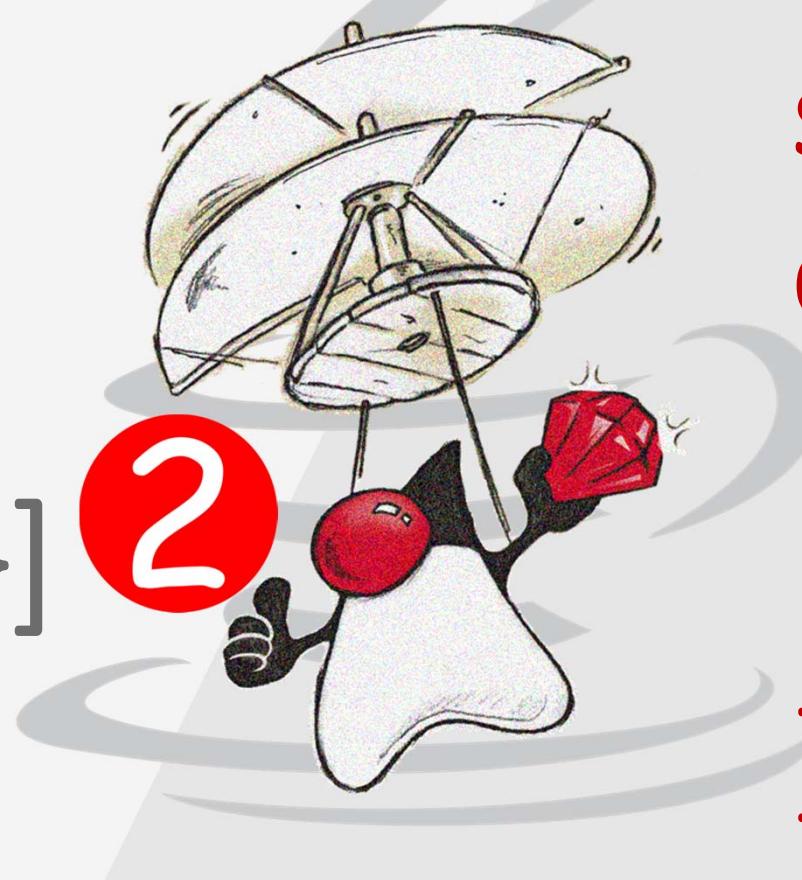
aserciones



# Condiciones

```
If {...}  
[else {...}]  
[elseif {...}]
```

```
switch {  
case ... :  
[break;]  
[default ... :]  
}
```



# Control de Flujo

If {...} Expression booleana  
[else {...}]  
[elseif {...}]

# Control de Flujo

```
If ( contador == 100 ){
    System.out.println("100");
}
elseif ( contador > 100){
    System.out.println(">100");
}
else{
    System.out.println("<100");
}
```

# Control de Flujo

```
If ( 4 < 1)  
{  
    System.out.println("Works or not");  
}
```



# Control de Flujo

```
If ( 4 < 1)
{
    System.out.println("Works or not");
}
```



# Control de Flujo

```
switch (expression) {  
    short, char, int, Byte, Short,  
    Character, Integer, or an enum type  
    case constant1: code block  
        [break;]  
    case constant2: code block  
        [break;]  
    [default: code block]  
}
```

# Control de Flujo

```
switch (expression) {  
    case constant1: code block  
        [break;]  
    case constant2: code block  
        [break;]  
    [default: code block]  
}
```

Indeterminado  
número de bloques  
case

# Control de Flujo

```
switch (expression) {  
    case constant1: code block  
        [break;] ← Detiene la ejecución del switch  
    case constant2: code block  
        [break;]  
    [default: code block]  
}
```

# Control de Flujo

```
switch (expression) {  
    case constant1: code block  
        [break;]  
    case constant2: code block  
        [break;]  
    [default: code block] ← No se cumple ninguna  
}                                            condición
```

# Práctica: “Control de Flujo”



# Práctica: “Control de Flujo”

```
switch (x) {  
    case 2: System.out.println("2");  
    default:  
        System.out.println("default");  
    case 3: System.out.println("3");  
    case 4: System.out.println("4");  
}
```

# Práctica: “Control de Flujo”

```
switch (x) {  
    case 2: System.out.println("2");  
    default:  
        System.out.println("default");  
    case 3: System.out.println("3");  
    case 4: System.out.println("4");  
}
```

?  
? 5,2 y 3

# Práctica: "Control de Flujo"

X = 5	X=2	X = 3
default	2	3
3	default	4
4	3	
	4	

# Bucles

while (condicion)

{

...

}

do{

...

} while (condicion)

for (inicio; fin; incremento){

for (TipoDato x : Coleccion ){

}



# While

Evaluación booleana

```
while (expresion) {  
    //do something;  
}
```

↑  
Condición de salida

# While

```
while(false) {  
    System.out.println("Compila...?");  
}
```

¿?

# While

```
while(false) {  
    System.out.println("Compilation error");  
}
```

# Do... While

```
do  
{  
    //do something;  
} while (expresion) ;
```



Evaluación booleana

# Do... While

```
do          Condición de salida  
{  
    //do something;  
} while (expresion) ;
```

↑  
Evaluación booleana

# Do... While

```
do  
{  
    System.out.println("Compila...?");  
} while(false);
```



# Do... While

```
do
{
    System.out.println("Compila...?");
} while(false);
```



# For

Evaluación booleana



```
for ( /*Initialization*/ ; /*Condition*/ ; /* Iteration */ )  
{  
    /* loop body */  
}
```

# For

```
for (int i = 0; i<10; i++)  
{  
    System.out.println("i vale " + i);  
}
```

# For

```
for (int x = 10, y = 3; y > 3; y++) { }
```

¿?

# For

```
for (int x = 10, y = 3; y > 3; y++) { }
```



# For

```
for( ; ; )  
{  
    System.out.println("Never Land");  
}
```



# For

```
for( ; ; )  
{  
    System.out.println("Never Land");  
}
```



# For Mejorado

Compatible con objeto de la colección

```
↓  
for (datatype iterator : collection)  
{  
    /*loop body*/ ↑  
}  
          Array o java.lang.Iterable
```

# For Mejorado

```
int [] array_enteros = new int [5];  
  
for (int entero : array_enteros){  
    System.out.println("Valor: " + entero);  
}  
  
Object [] array_objetos = new Object [9];  
  
for (Object instancia : array_objetos){  
    System.out.println("Valor: " + instancia);  
}
```

# Práctica: “For Mejorado”



# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};
```

```
for(long y : la );
```



# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};
```

```
for(long y : la );
```



# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};
```

```
for(x2 : la );
```



# Práctica: "For Mejorado"

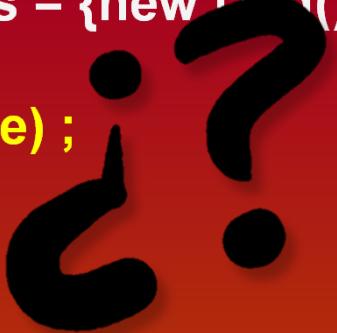
```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};
```

for(x2 ↙ Sentencia inválida

# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};
```

```
for(int x2 : twoDee) ;
```



# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};
```

for(int x2 : twoDee); ← Tipos incompliables



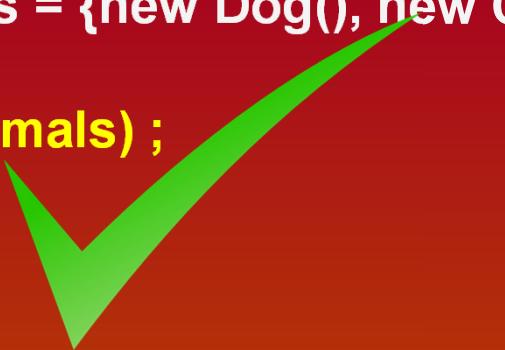
# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};  
  
for(Animal a : animals) ;
```



# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};  
  
for(Animal a : animals) ;
```



# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};  
  
for(int[ ] n : twoDee);
```



# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};  
  
for(int[ ] n : twoDee);
```



# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};  
  
for(Dog d : animals) ;
```



# Práctica: "For Mejorado"

```
long x2 = 0L;  
long [ ] la = {7L, 8L, 9L};  
int [ ] [ ] twoDee = {{1,2,3}, {4,5,6}, {7,8,9}};  
String [ ] sNums = {"one", "two", "three"};  
Animal [ ] animals = {new Dog(), new Cat()};
```

for(Dog d : animals) ← Tipos incompliables



# Break / Continue

Unlabeled:

```
boolean problem = true;  
while (true) {  
    if (problem) {  
        System.out.println("There was a problem");  
        break;  
    }  
}
```

←Fuerza salida bucle

Labeled:

```
char row = 'A';  
outer: while(row <= 'D') {  
    System.out.print(row++);  
    inner: for(int i = 1; i <= 5; i++) {  
        if(i%2 == 0)  
            continue; //Goes to next iteration inside for loop  
        if(i%3 == 0) {  
            System.out.println();  
            break outer; //Stops the whole while loop  
        }  
        System.out.print(i);  
    }  
}
```

# Break / Continue

**Unlabeled:**

```
boolean problem = true;
while (true) {
    if (problem) {
        System.out.println("There was a problem");
        break;
    }
}
```

**Labeled:**

```
char row = 'A';
outer: while(row <= 'D') {
    System.out.print(row++);
    inner: for(int i = 1; i <= 5; i++) {
        if(i%2 == 0)
            continue; // ← Salta a la siguiente iteracion
        if(i%3 == 0) {
            System.out.println();
            break outer; //Stops the whole while loop
        }
        System.out.print(i);
    }
}
```

 Salta a la siguiente iteracion

# Break / Continue

Unlabeled:

```
boolean problem = true;
while (true) {
    if (problem) {
        System.out.println("There was a problem");
        break;
    }
}
```

Labeled:

```
char row = 'A';
outer: while(row <= 'D') {
    System.out.print(row++);
    inner: for(int i = 1; i <= 5; i++) {
        if(i%2 == 0)
            continue; //Goes to next iteration inside for loop
        if(i%3 == 0) {
            System.out.println();
            break outer; //Breaks the outer while loop
        }
        System.out.print(i);
    }
}
```

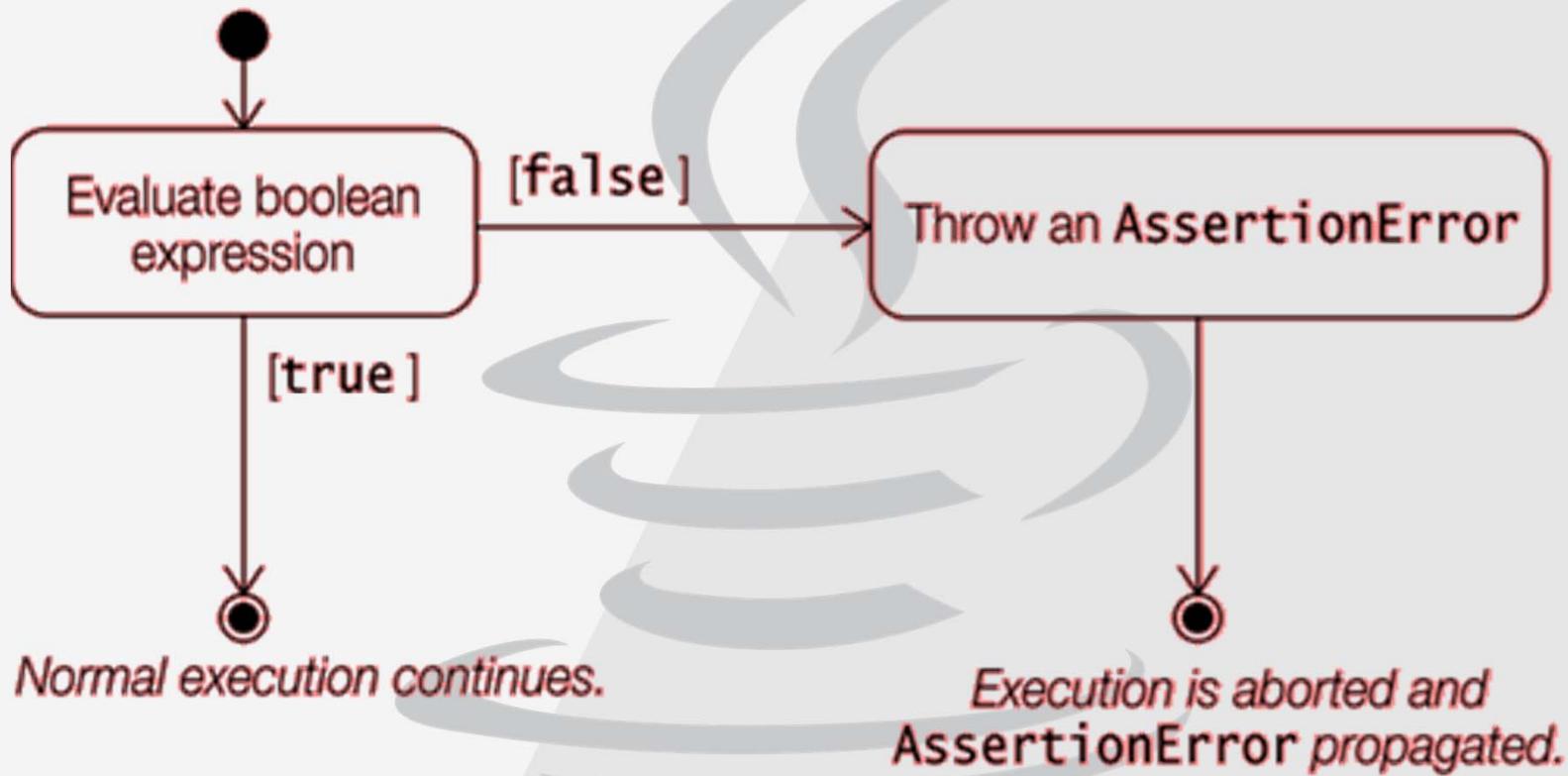
← Código etiquetado para salto

← Ejecución de salto

# Aserciones



# Aserciones



# Aserciones

```
assert boolean_expression;
```

```
assert boolean_expression : error_message;
```

# Aserciones

```
java -ea -da:com.testingassertions.Foo Main
```

```
java -ea -da:com.testingassertions... Main
```

# Aserciones

*Apropiado*

**vs.**

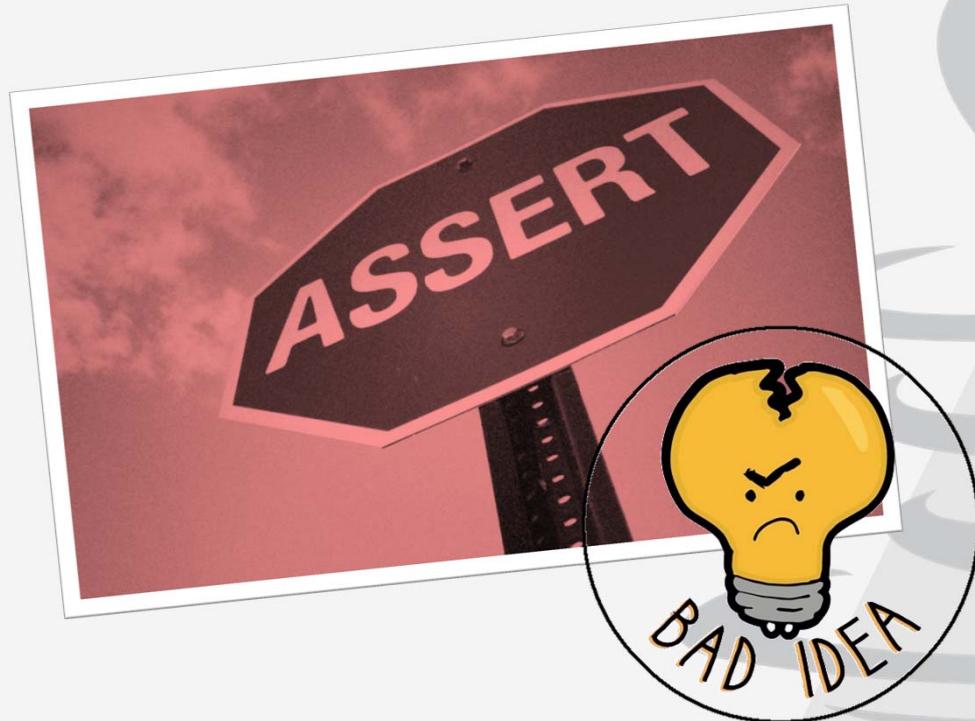
*Legal*

# Aserciones



- Validación de argumentos en métodos privados.
- Validar / probar casos incluso en método público que rarísima vez pueden darse.

# Aserciones



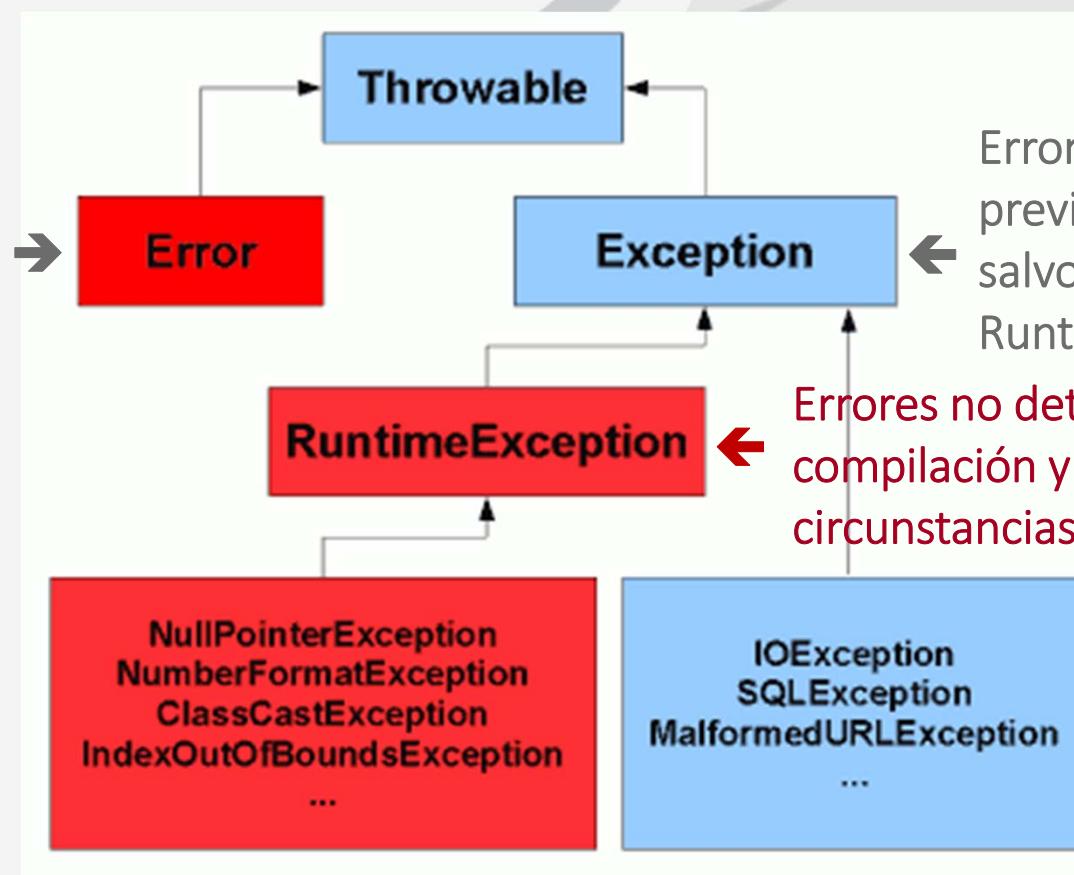
- Validación de argumentos en métodos públicos.
- Validación de argumentos por linea de comandos.
- Usar aserciones con efectos colaterales: *(assert(modifyThings()))*

# Excepciones



# Excepciones

Fallos no recuperables,  
situaciones especiales, no  
derivadas de errores en la  
programación



Errores de programación  
previsibles, deben ser siempre  
salvo que estén en la jerarquía de  
RuntimeException

Errores no detectables en tiempo de  
compilación y que se dan sólo por  
circunstancias de la ejecución del código.

# Excepciones

```
try {  
    /* Código protegido, debe ir en este bloque todo código susceptible  
       de lanzar algún tipo de excepción */  
}  
[catch(FirstException fe) {  
    /* Código específico para manejar la excepción indicada */  
}  
catch(SecondException se) {  
    /* Código específico para manejar otra excepción indicada */  
}  
catch(NException ne) {  
    /* Código específico para manejar la excepción n indicada */  
}  
]  
[finally {  
    /* Código que se ejecuta siempre se haya o no se haya producido una  
       excepción. Siempre se ejecuta este bloque tras el try o el catch  
       según sea correspondiente */  
}]
```

# Excepciones



- El orden en que las excepciones se declaran en el bloque catch es el patrón que se sigue para encontrar el correspondiente manejador.
- Si no se encuentra un manejador específico se busca uno para el supertipo.
- Si tampoco se encuentra un manejador para el supertipo se propaga.

# Excepciones



- Sólo se ejecuta un único bloque catch o ninguno.
- Los manejadores más específicos deben declararse siempre antes.

```
try {  
    // do risky IO things  
} catch (IOException e) {  
    // handle general IOExceptions  
} catch (FileNotFoundException ex) {  
    // handle specific FileNotFoundException  
}
```

# Excepciones



- Sólo se ejecuta un único bloque catch o ninguno.
- Los manejadores más específicos deben declararse siempre antes.

```
try {  
    // do risky IO things  
} catch (IOException ex) {  
    // handle general exception  
} catch (FileNotFoundException ex) {  
    // handle specific exception  
}
```



# Excepciones



- Un bloque try obliga a que siempre se declare al menos o un bloque catch o finally. ( Pueden ir los dos )
- Si un método ejecuta código susceptible de lanzar una excepción y no lo trata explícitamente con un bloque catch, debe obligatoriamente declarar que propaga la excepción mediante la cláusula throws.

# Excepciones



```
public class TestTryCathFinally {  
  
    public void metodoLanzaException () throws TCFException{  
        System.out.println("Lanzando excepcion desde:  
            metodoLanzaException()");  
    }  
  
    public void testTCF() {  
        try{  
            this.metodoLanzaException();  
        }  
        finally {  
            System.out.println("No hace falta Catch");  
        }  
    }  
  
    class TCFException extends Exception {  
    }  
}
```



# Excepciones



```
public class TestTryCathFinally {  
  
    public void metodoLanzaException () throws TCFException{  
        System.out.println("Lanzando excepcion desde:  
            metodoLanzaException()");  
    }  
  
    public void testTCF() {  
        try{  
            this.metodoLanzaException();  
        } finally {  
            System.out.println("Finalmente ejecutando el finally");  
        }  
    }  
  
    class TCFException extends Exception {  
    }  
}
```

# Excepciones



```
public class TestTryCathFinally {  
  
    public void metodoLanzaException () throws TCFException{  
        System.out.println("Lanzando excepcion desde:  
        metodoLanzaException()");  
    }  
  
    public void testTCF() throws TCFException  
    {  
        try{  
            this.metodoLanzaException();  
        }  
        finally {  
            System.out.println("No hace falta Catch");  
        }  
    }  
  
    class TCFException extends Exception {  
    }  
}
```



# Excepciones

## ArrayListOutOfBoundsException

- Thrown when attempting to access an array with an invalid index value (either negative or beyond the length of the array). [By the JVM](#)

## ClassCastException

- Thrown when attempting to cast a reference variable to a type that fails the IS-A test. [By the JVM](#)

## IllegalArgumentException

- Thrown when a method receives an argument formatted differently than the method expects. [Programmatically](#)

## IllegalStateException

- Thrown when the state of the environment doesn't match the operation being attempted, e.g., using a Scanner that's been closed. [Programmatically](#)

## NullPointerException

- Thrown when attempting to access an object with a reference variable whose current value is null. [By the JVM](#)

## NumberFormatException

- Thrown when a method that converts a String to a number receives a String that it cannot convert. [Programmatically](#)

## AssertionError

- Thrown when a statement's boolean test returns false. [Programmatically](#)

## ExceptionInInitializerError

- Signals that an unexpected exception has occurred in a static initializer. An `ExceptionInInitializerError` is thrown to indicate that an exception occurred during evaluation of a static initializer or the initializer for a static variable. [By the JVM](#)

## StackOverflowError

- Typically thrown when a method recurses too deeply. (Each invocation is added to the stack.) [By the JVM](#)

## NoClassDefFoundError

- Thrown when the JVM can't find a class it needs, because of a command-line error, a classpath issue, or a missing .class file. [By the JVM](#)



#3

# API's Principales

# API's Principales

Wrappers

I/O

Formato y Parseo

String/StringBuffer/  
StringBuilder

Expresiones  
Regulares

# Wrappers

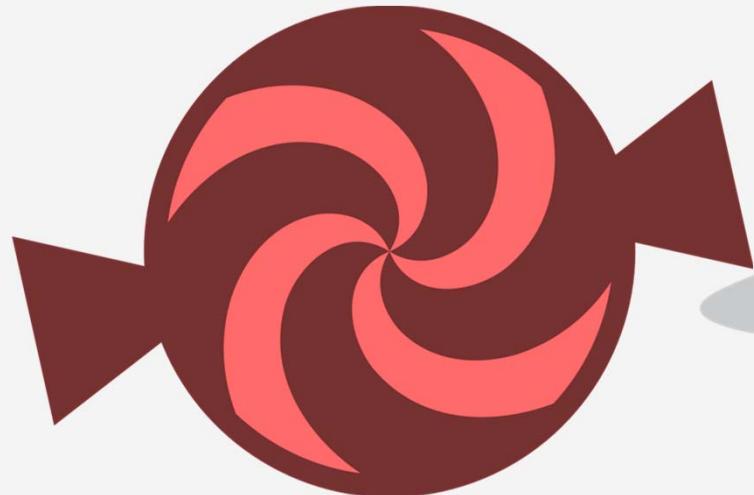


# Wrappers

→ Mecanismo para envolver valores primitivos en objetos, para que estos puedan trabajar como ellos ( Colecciones, devolver / pasar valores en métodos ).



# Wrappers



- Mecanismo para envolver valores primitivos en objetos, para que estos puedan trabajar como ellos ( Colecciones, devolver / pasar valores en métodos ).
- Conjunto de utilidades para la conversión a y desde objetos String y diferentes bases ( binario, octal, decimal ).

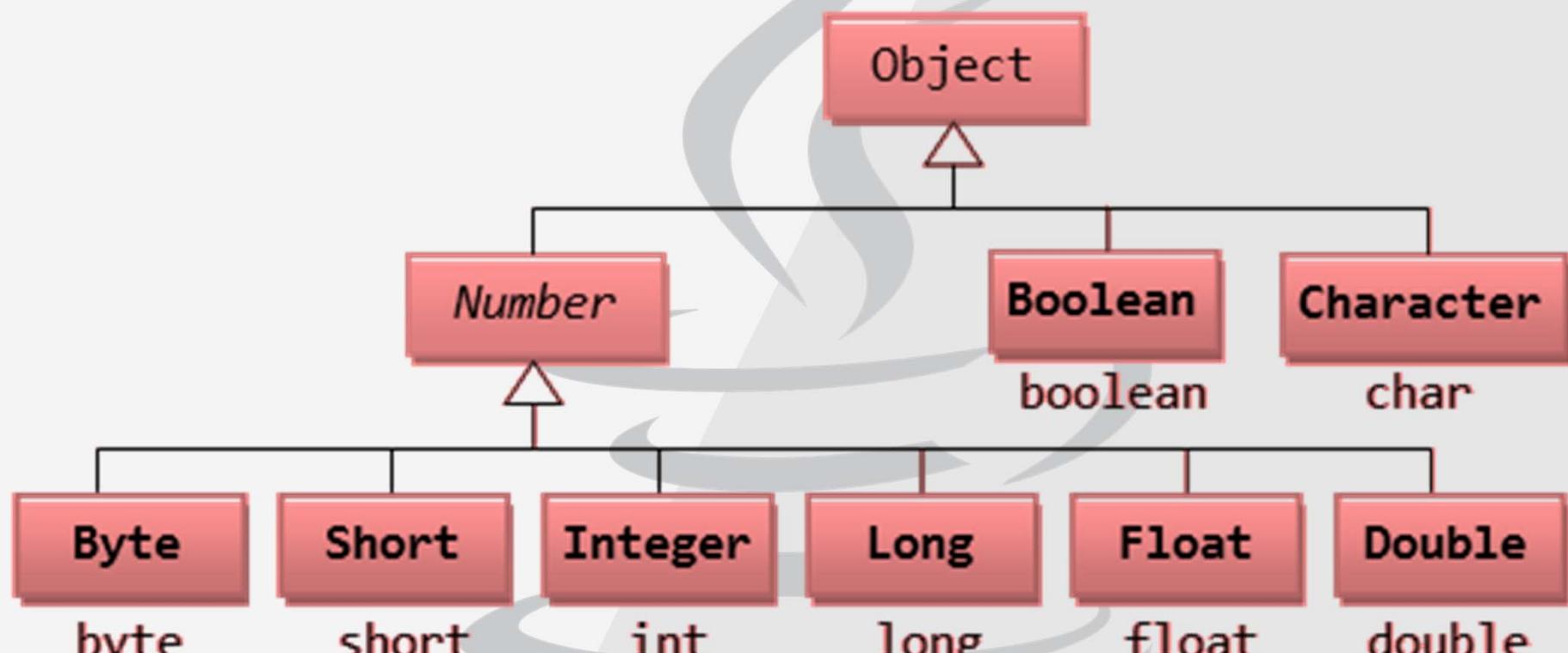
# Práctica: “Transformaciones Numéricas”



# Práctica: “Transformaciones Numéricas”

```
1. public class Convers {
2.
3.     public static void main(String args [])
4.     {
5.         Integer number = 60;
6.
7.         System.out.println("Decimal: " + number +
8.                         " Hex: " + Integer.toHexString(number) +
9.                         " Octal: " + Integer.toOctalString(number) +
10.                          " Binary: " + Integer.toBinaryString(number));
11.
12.         System.out.println("Base 4: " + number.toString(6,4));
13.         System.out.println("Base 32: " + number.toString(6,4));
14.     }
15. }
```

# Wrappers



# Wrappers

Tipo Primitivo	Clase Wrapper	Argumentos del Constructor
boolean	Boolean	boolean/String
byte	Byte	byte/String
char	Character	char
double	Double	double/String
float	Float	float,double,String
int	Integer	int/String
long	Long	long/String
short	Short	short/String

# Wrappers

**xxxValue()**

→ Convierte de Wrapper a tipo primitivo. No tiene argumentos

**parseXxx /  
valueOf()**

→ Un método parseXxx por cada tipo de Wrapper. Convierte a tipo primitivo/Wrapper un objeto String, lanzan NumberFormatException. Pueden trabajar con diferentes bases para tipos enteros.

parseXxx ( devuelve primitivos)  
valueOf ( devuelve Wrappers )

# Wrappers

## **toString()**

→ Heredado de Object, es la representación en String que implementan todos los Wrappers del método.

## **toXxxString**

→ Método para Integer y Long que permite operar con diferentes sistemas numéricicos de base.

Desarrollados específicamente ToBinaryString(), ToOctalString() y toHexString().

# AUTOBOXING



# AUTOBOXING

Java 1.4

```
import java.util.*;  
  
class Main {  
    public static void main() {  
        // Does not work, 5  
        someMethod(5);  
    }  
  
    public static void someMethod(Object a) {  
        System.out.println(a.toString());  
    }  
}
```

Java 1.4: Solution

```
import java.util.*;  
  
class Main {  
    public static void main(String [] args) {  
        Integer temp = new Integer(5);  
        someMethod(temp);  
    }  
  
    public static void someMethod(Object a) {  
        System.out.println(a.toString());  
    }  
}
```

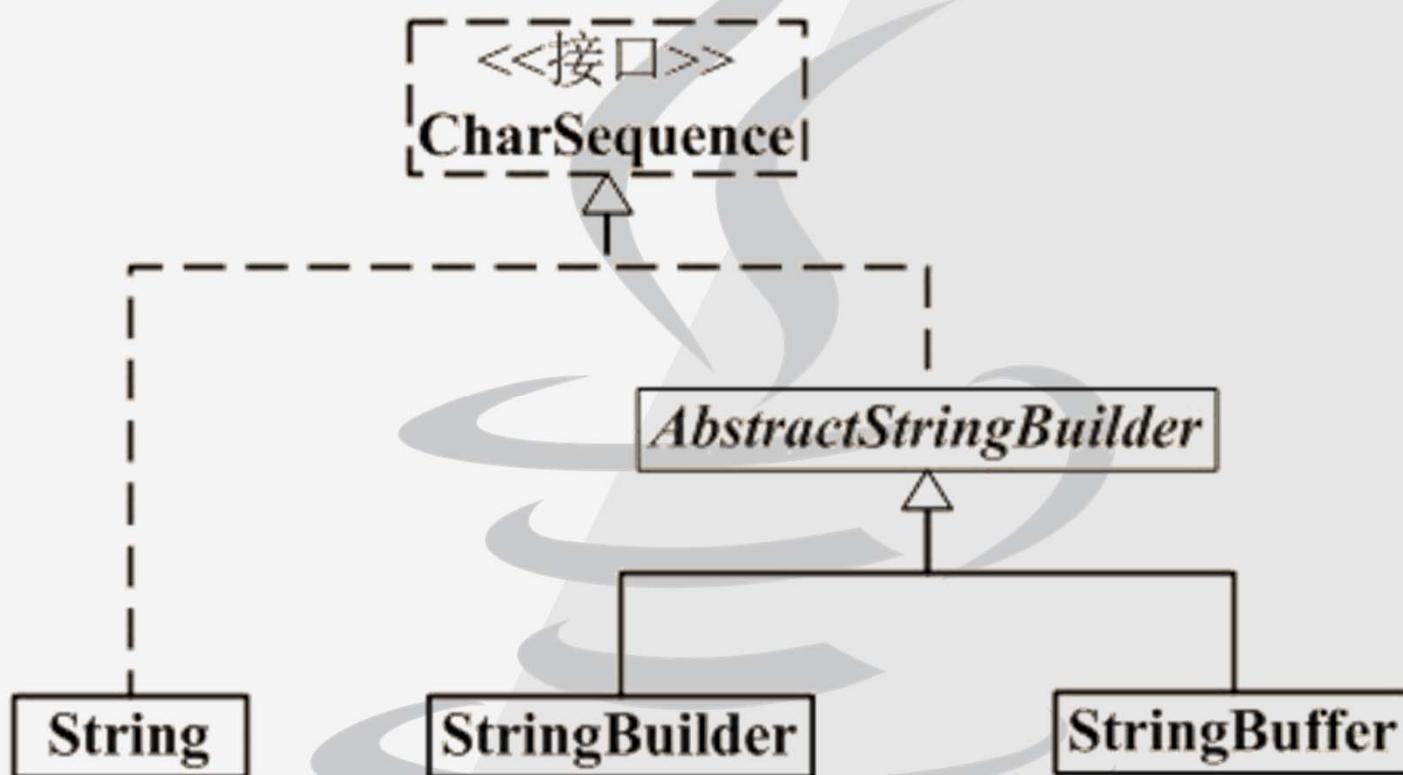
Java 1.5

```
public static void main(String [] args) {  
    // Works!  
    someMethod(5);  
}  
  
public static void someMethod(Object a) {  
    System.out.println(a.toString());  
}
```

# String / StringBuilder/ StringBuffer



# String / StringBuilder/ StringBuffer



# String / StringBuilder/ StringBuffer



- Son objetos inmutables. No se pueden alterar.
- En java todos los literales son objetos String.
- Su instanciaión es implícita con el uso del operador de asignación '='. \*
- Se almacenan en un espacio reservado de memoria compartida denominado string pool.
- El uso de String con el operador + lleva un coste debido a la instanciaión de múltiples objetos.

# String / StringBuilder/ StringBuffer



- ➔ **charAt()** Devuelve el carácter en la posición correspondiente del índice.
- ➔ **concat()** Concatena una cadena con otra ( Como el operador +)
- ➔ **equalsIgnoreCase()** Determina si dos cadenas son iguales ignorando el caso ( case-insensitive)
- ➔ **length()** Devuelve el número de caracteres de una cadena

# String / StringBuilder/ StringBuffer



- ➔ replace() Reemplaza las ocurrencias de un caracter por otro nuevo.
- ➔ substring() Devuelve una parte de la cadena.
- ➔ toLowerCase() Devuelve una cadena en minúsculas.
- ➔ toString() Obtiene el valor de la cadena.

# String / StringBuilder/ StringBuffer



- `toUpperCase()` Devuelve una cadena en mayúsculas.
- `trim()` Elimina los espacios en blanco de una cadena

# **String / StringBuilder/ StringBuffer**

## **StringBuilder**



- Son objetos similares.
- StringBuilder es No Sincronizado, mientras que StringBuffer sí es Sincronizado.
- StringBuilder es más eficiente cuando no hay que sumir concurrencia.

# **String / StringBuilder/ StringBuffer**

## **StringBuilder**



- public synchronized StringBuffer append(String s)
- public StringBuilder delete(int start, int end)
- public StringBuilder insert(int offset, String s)
- public synchronized StringBuffer reverse()

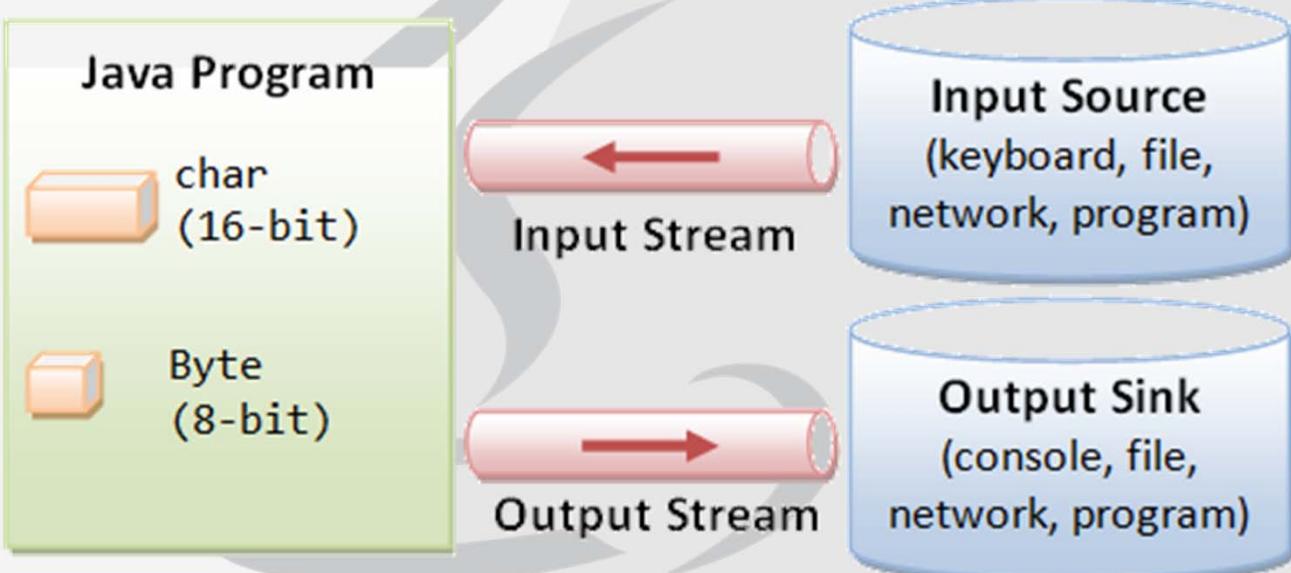
# Input/Output



# Input/Output

“Character” Streams  
(Reader/Writer)

“Byte” Streams  
(InputStream/  
OutputStream)



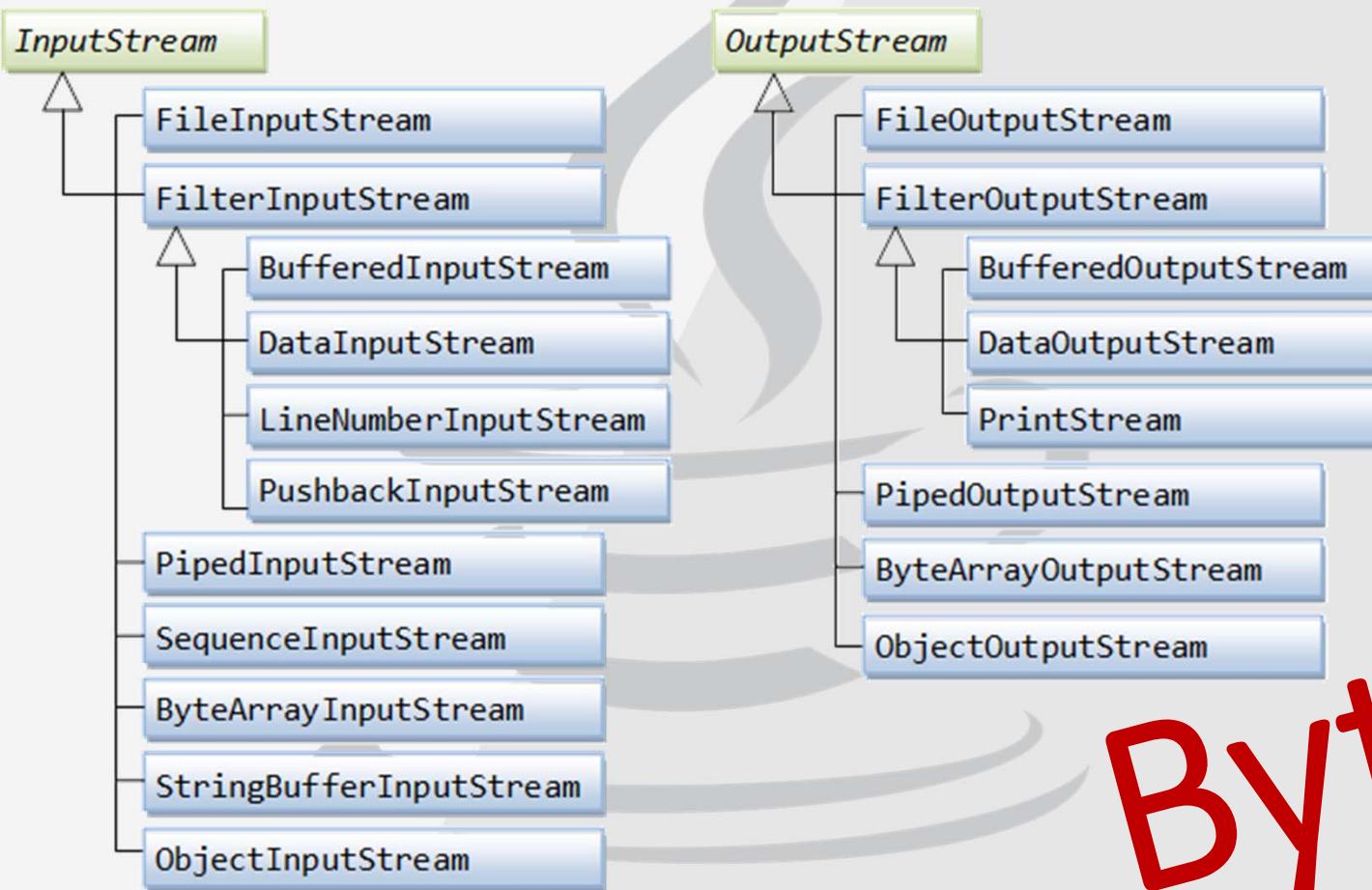
Internal Data Formats:

- Text (char): UCS-2
- int, float, double, etc.

External Data Formats:

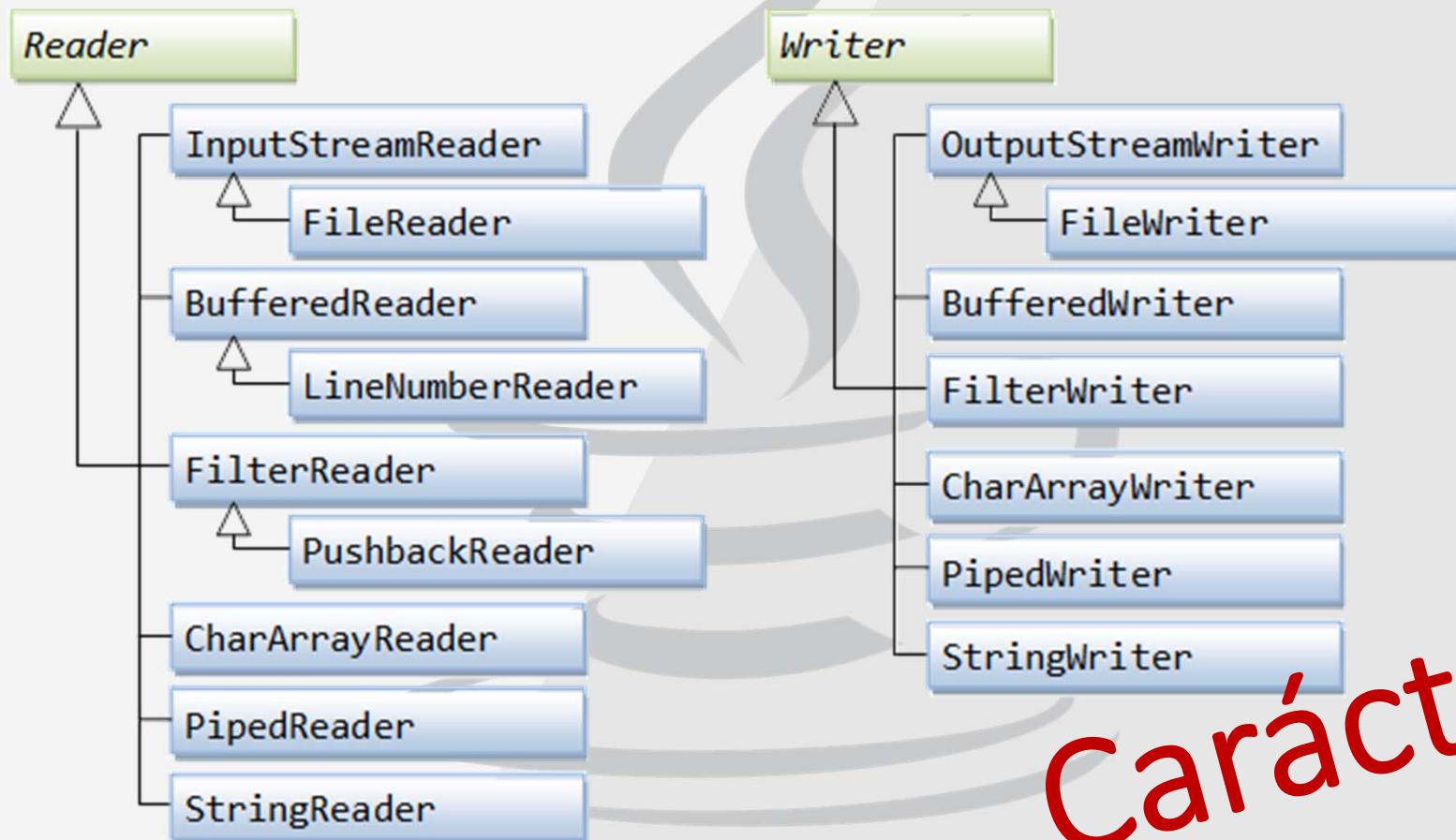
- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

# Input/Output



Bytes

# Input/Output



Carácteres



# Input/Output

## Class File

java.lang.Object  
java.io.File

### All Implemented Interfaces:

Serializable, Comparable<File>

<http://docs.oracle.com/javase/7/docs/api/java/io/File.html>



# Input/Output

- exists, determina si un fichero existe
- canRead, determina si hay permisos de lectura
- canWrite, determina si hay permisos de escritura
- canExecute, determina si hay permisos de ejecución
- createNewFile, crea un nuevo fichero,
- mkdir, crea un nuevo directorio
- delete, borra un fichero o directorio
- **list, listFiles, lista los contenidos de un directorio**

# Práctica: “Listar ficheros de un directorio”



# Input/Output

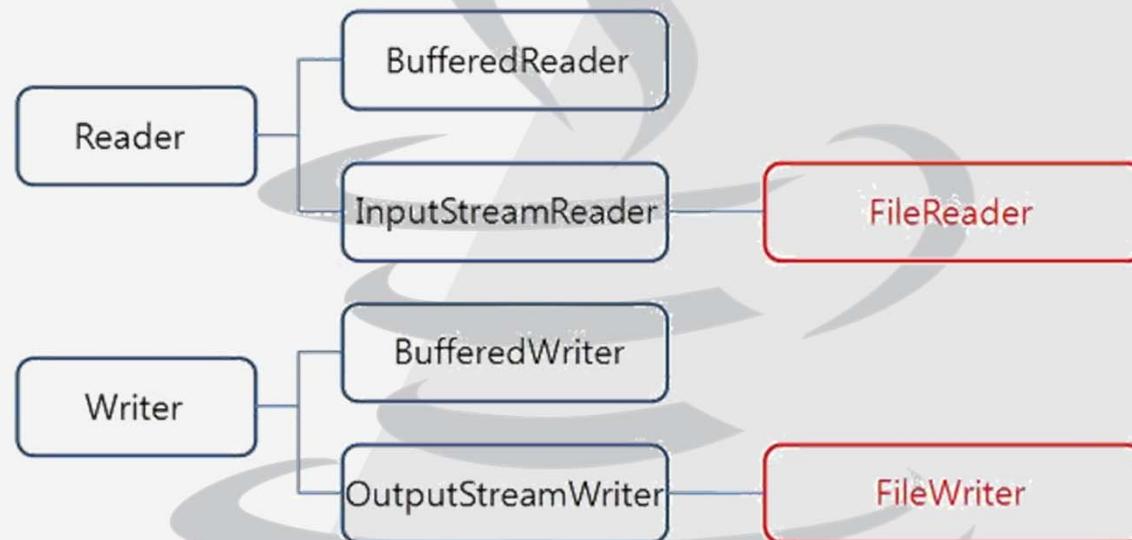
```
1. import java.io.File;  
2. public class ListarFichero {  
3.     public static void main(String[] args)  
4.     {  
5.         new ListarFichero().imprimirListado();  
6.     }  
7.  
8.     public void imprimirListado()  
9.     {  
10.        File f = new File(".");  
11.  
12.        if (f.canExecute() && f.canRead())  
13.        {  
14.            String [] ficheros = f.list();  
15.  
16.            for (String fichero : ficheros)  
17.            {  
18.                System.out.println("Fichero: " + fichero);  
19.            }  
20.  
21.        }  
22.    }  
23.}
```



FileReader  
FileWriter

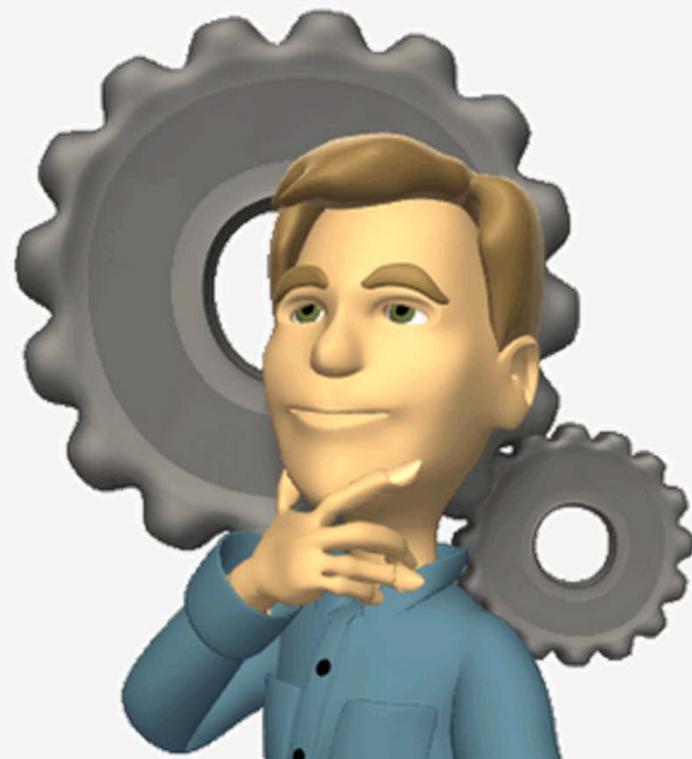
# Input/Output

<http://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html>



<http://docs.oracle.com/javase/7/docs/api/java/io/FileWriter.html>

# Práctica: “Escribir / Leer un fichero”



# Input/Output

```
1. import java.io.File;
2. import java.io.FileWriter;
3. import java.io.FileReader;
4. import java.io.IOException;

5. class WriteRead {
6.     public static void main(String [] args) {
7.         //Definicion de un tamaño de buffer de 50
8.         char[] in = new char[50];
9.         int size = 0;
10.        try {
11.            //Creacion del fichero
12.            File file = new File("MyStory.txt");
13.            //Instanciamos un FileWriter
14.            FileWriter fw = new FileWriter(file);
15.            //Escribimos
16.            fw.write("Erase una vez...\nUna historia Javesca\n");
17.            //Vaciamos el buffer
18.            fw.flush();
19.            //Cerramos fichero
20.            fw.close();
21.
22.            //Instanciamos un FileReader
23.            FileReader fr = new FileReader(file);
24.
25.            //Leemos
26.            size = fr.read(in);
27.            System.out.println("Numero Caracteres leidos: " + size);
28.
29.            for(char c : in)
30.                System.out.print(c);
31.            //Cerramos fichero
32.            fr.close();
33.        }
34.        catch(IOException e) {
35.            e.printStackTrace();
36.        }
37.    }
38. }
```

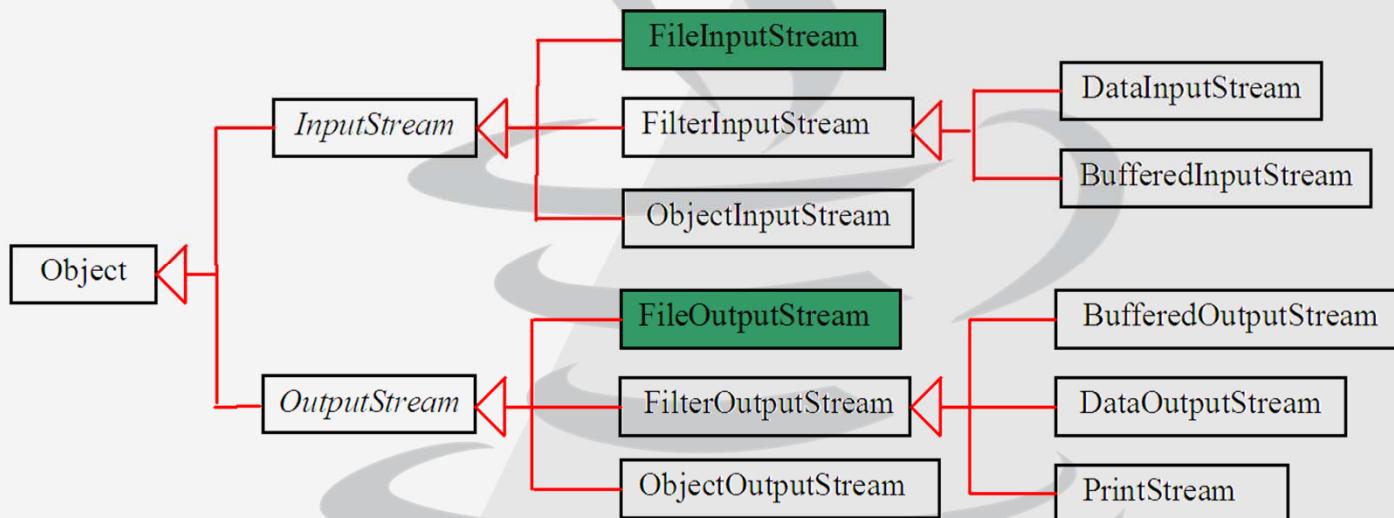


# Input/Output

FileInputStream

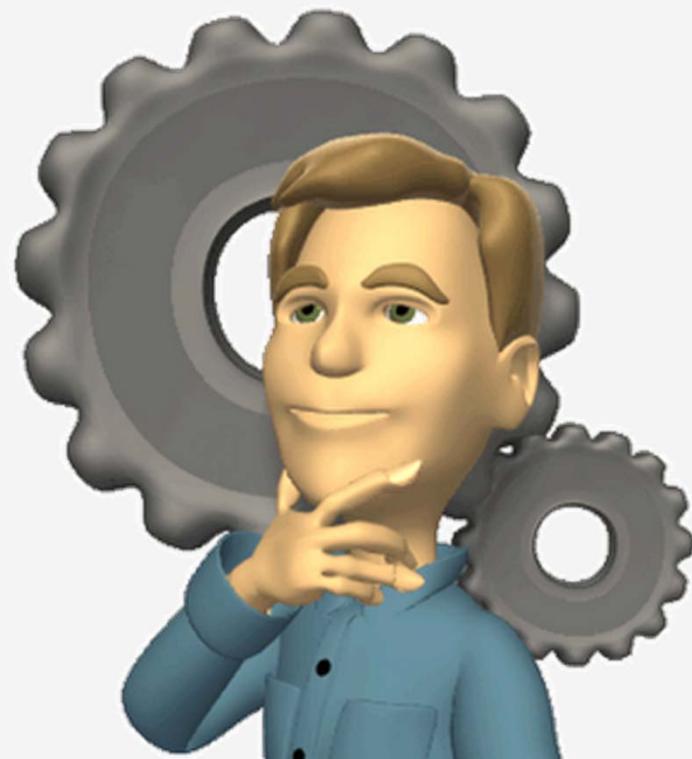
FileOutputStream

<http://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html>



<http://docs.oracle.com/javase/7/docs/api/java/io/FileOutputStream.html>

# Práctica: “Copiar un archivo”



# Input/Output

```
1. import java.io.FileInputStream;
2. import java.io.File;
3. import java.io.FileOutputStream;
4. import java.io.IOException;

5. public class CopiarArchivo {
6.
7.     public static void main(String[] args) {
8.         File src = new File(args[0]);
9.         File dest = new File(args[1]);
10.
11.        try {
12.            new CopiarArchivo().copiar(src, dest);
13.        }
14.        catch (IOException ioex){
15.            ioex.printStackTrace();
16.        }
17.    }
18.
19.    public void copiar(File src, File dest) throws IOException
20.    {
21.        FileInputStream in = new FileInputStream(src);
22.        FileOutputStream out = new FileOutputStream(dest);
23.        int bytesLeidos;
24.
25.        try {
26.            while((bytesLeidos = in.read()) != -1) {
27.                out.write(bytesLeidos);
28.            }
29.        }
30.        finally{
31.            in.close();
32.            out.close();
33.        }
34.    }
35.}
```

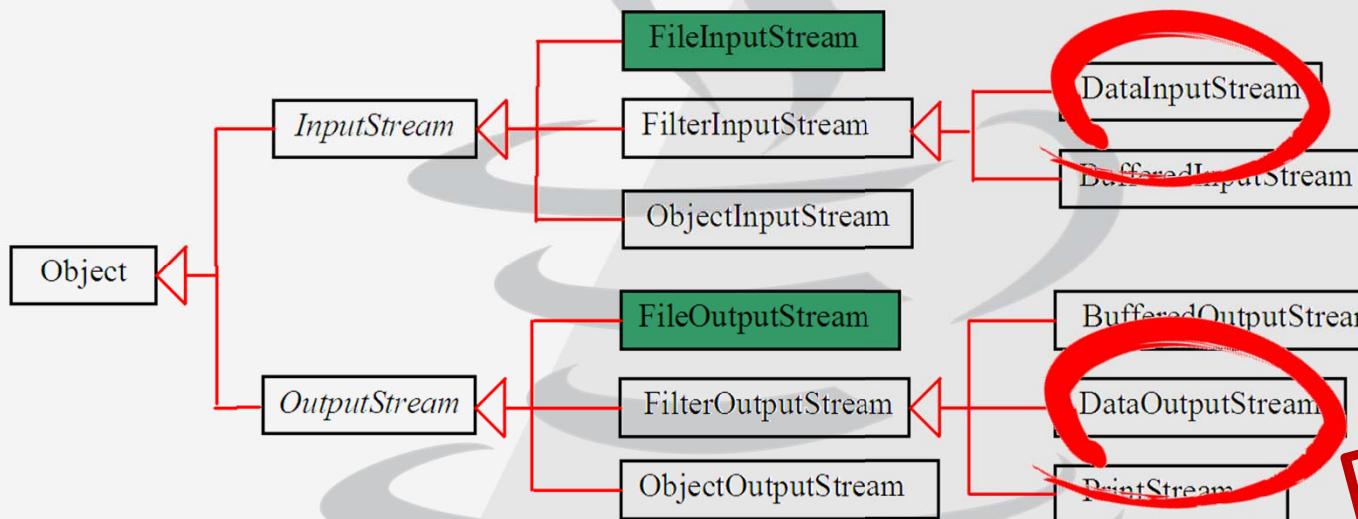


# Input/Output

DataInputStream

DataOutputStream

<http://docs.oracle.com/javase/7/docs/api/java/io/DataInputStream.html>



Tipos  
Primitivos

<http://docs.oracle.com/javase/7/docs/api/java/io/DataOutputStream.html>

# Práctica: “Agenda”



# Input/Output

```
class Contact{  
  
    private String name;  
    private Integer age;  
    private Long cellPhone;  
  
    public Contact (String name, Integer age, Long cellPhone){  
        this.name = name;  
        this.age = age;  
        this.cellPhone = cellPhone;  
    }  
  
    public String getName() {  
        return (this.name);  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public Integer getAge() {  
        return (this.age);  
    }  
  
    public void setAge(Integer age) {  
        this.age = age;  
    }  
  
    public Long getCellPhone() {  
        return (this.cellPhone);  
    }  
  
    public void setCellPhone(Long cellPhone) {  
        this.cellPhone = cellPhone;  
    }  
  
    public String toString(){  
        return "Name: " + this.name + " Age: " + this.age + " Phone: " + this.cellPhone;  
    }  
}
```

# Input/Output

```
package agenda.com;

import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.DataOutputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.DataInputStream;

public class Agenda {

    private static final String AGENDA_NAME = "agenda.dat";

    private static final File agendaFile = new File(agenda.AGENDA_NAME);

    public static void main(String[] args) {
        String name = args[0];
        Integer age = Integer.parseInt(args[1]);
        Long cellPhone = Long.parseLong(args[2]);

        Contact contact = new Contact(name,age,cellPhone);

        Agenda agenda = new Agenda();

        try{
            agenda.addContact(contact, Agenda.agendaFile);
        }catch (IOException ioex){
            ioex.printStackTrace();
        }

        try{
            agenda.retrieveContactByName(name);
        }catch (IOException ioex){
            ioex.printStackTrace();
        }
    }
}
```

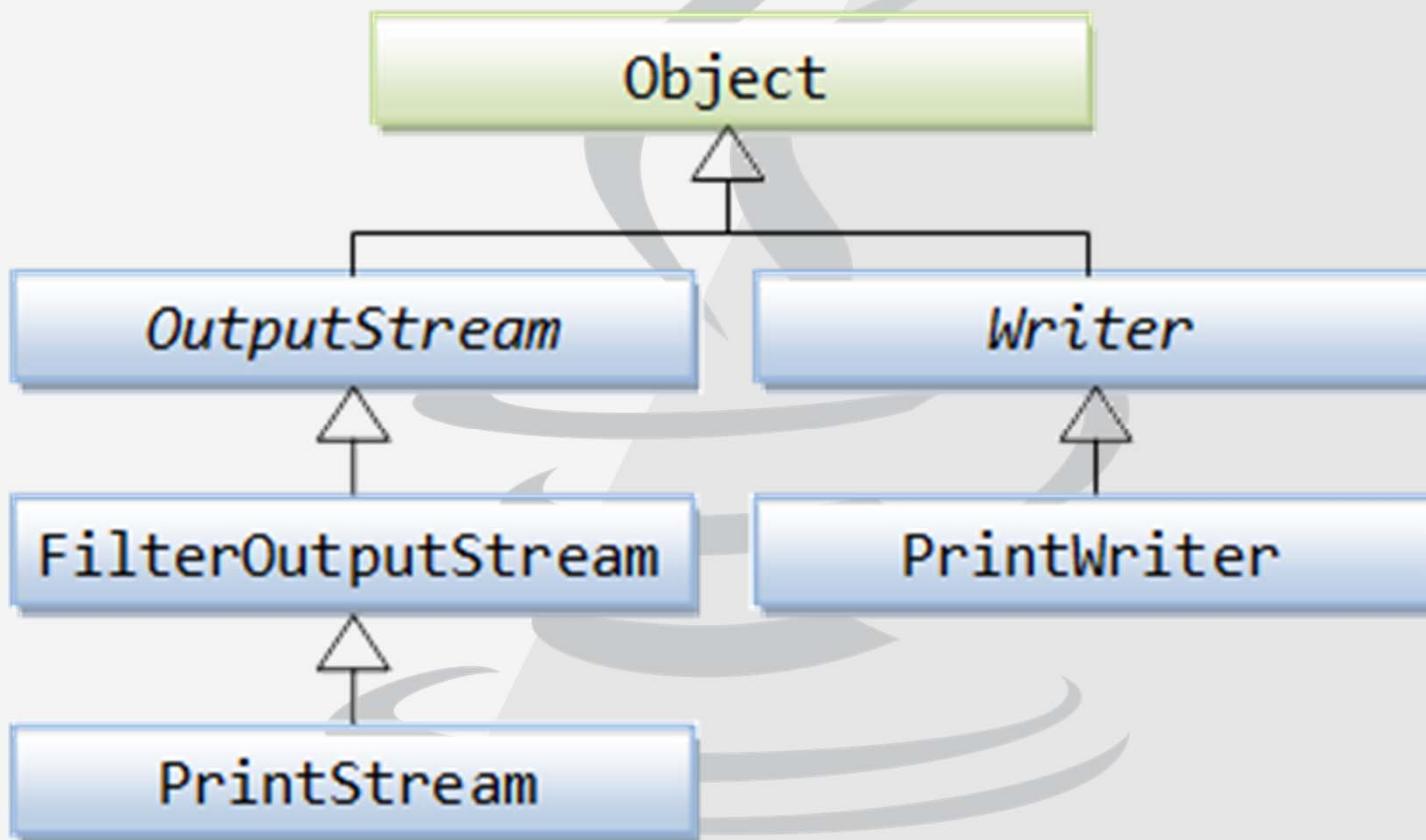
# Input/Output

```
public void addContact(Contact contact, File dest) throws IOException {  
    FileOutputStream fos = new FileOutputStream(dest, true);  
    BufferedOutputStream bos = new BufferedOutputStream(fos);  
    DataOutputStream out = new DataOutputStream(bos);  
  
    out.writeUTF(contact.getName());  
    out.writeInt(contact.getAge());  
    out.writeLong(contact.getCellPhone());  
    out.close();  
    bos.close();  
    fos.close();  
}  
  
public void retrieveContactByName (String name) throws IOException {  
    FileInputStream fis = new FileInputStream(Agenda.agendaFile);  
    BufferedInputStream bis = new BufferedInputStream(fis);  
    DataInputStream in = new DataInputStream(bis);  
  
    Contact contact;  
  
    while(in.available() > 0) {  
        contact = new Contact(in.readUTF(),in.readInt(),in.readLong());  
  
        if (contact.getName().equalsIgnoreCase(name))  
        {  
            System.out.println(contact.toString());  
        }  
    }  
}
```



PrintWriter

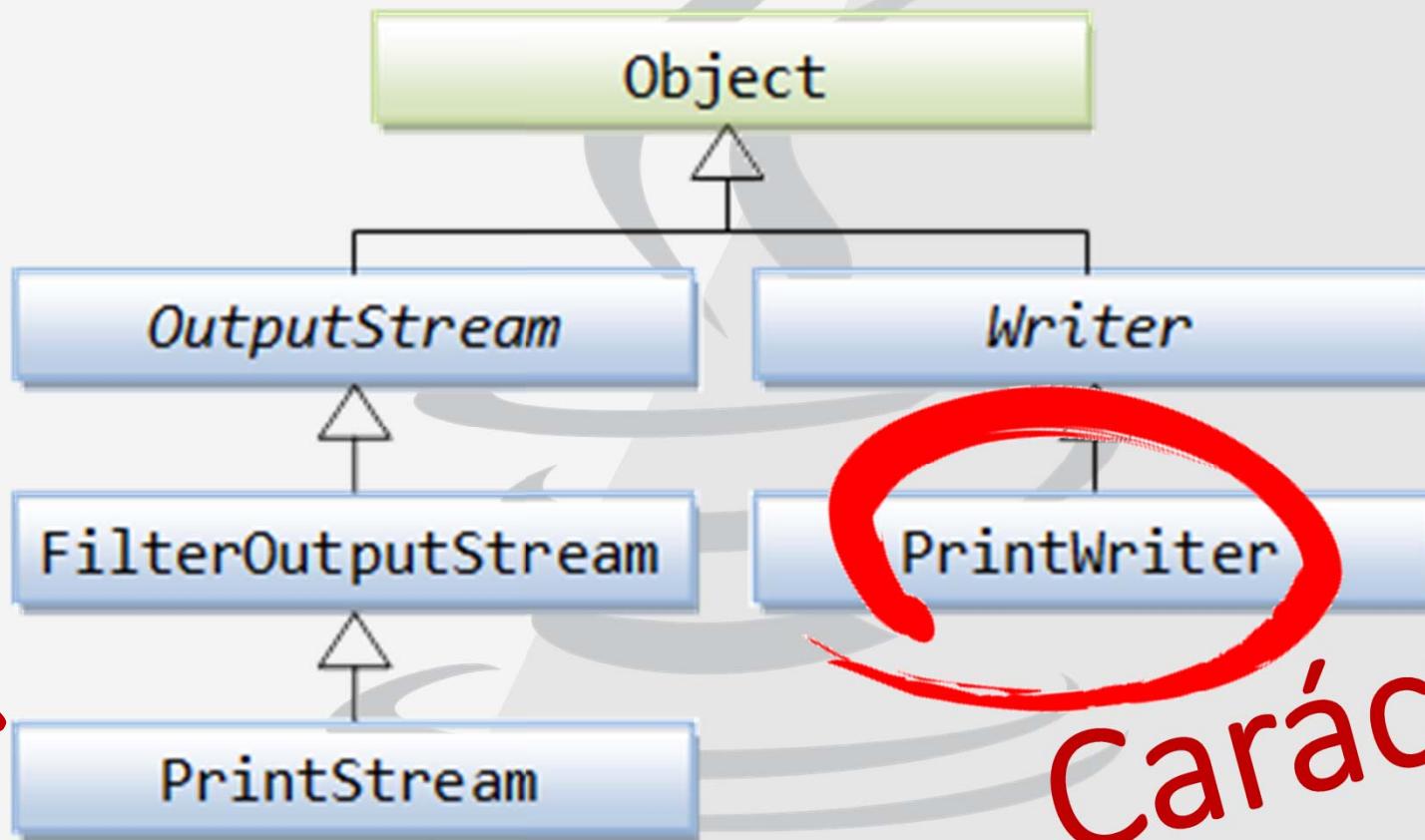
# Input/Output





PrintWriter

# Input/Output



Bytes

Carácteres

# Input/Output

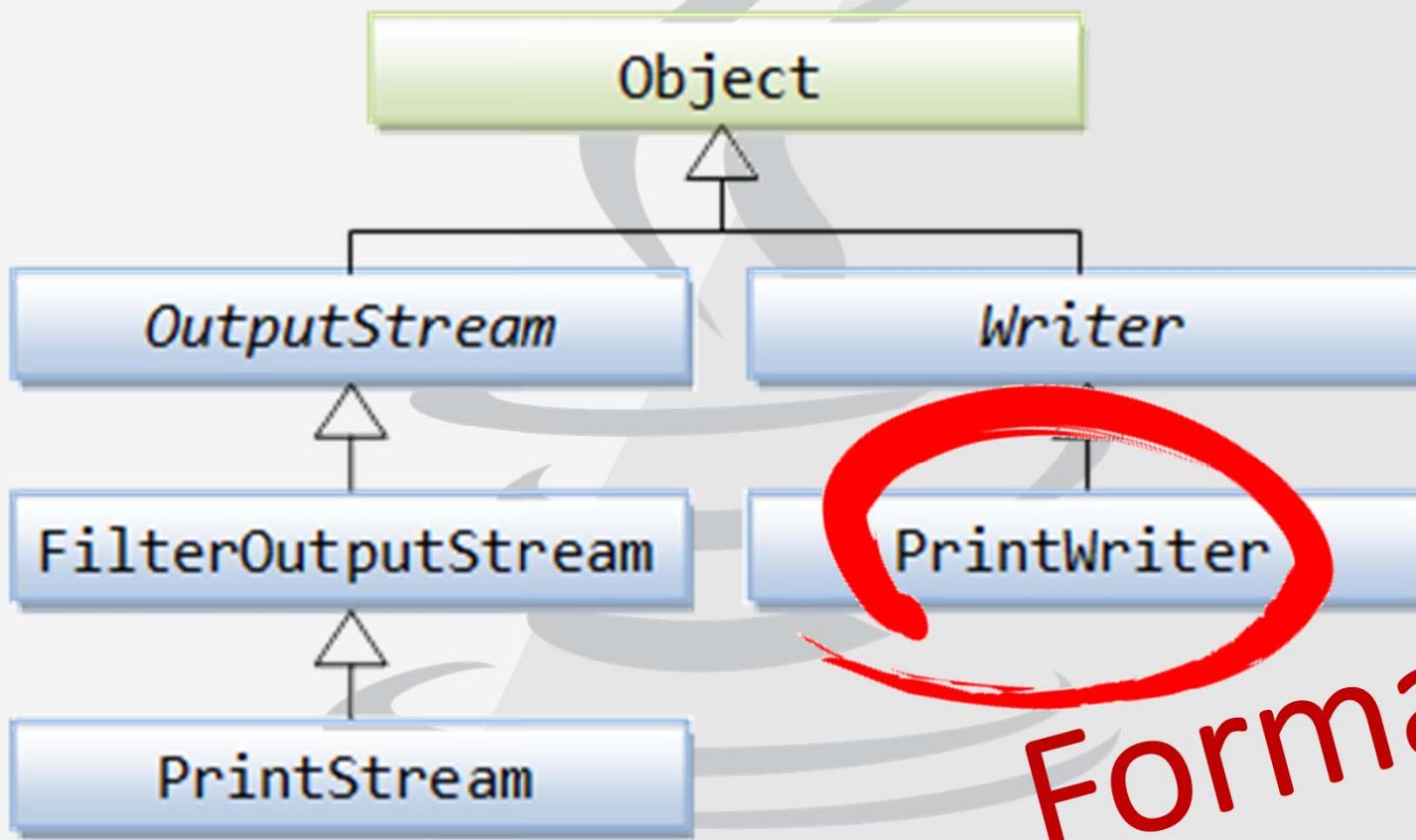
```
1. import java.io.FileWriter;
2. import java.io.PrintWriter;
3. import java.util.Random;
4. import java.io.IOException;

5. class PrintWriterTest {
6.
7.     public static void main(String[] args) throws IOException{
8.
9.         String inicio = "Cadena Inicio";
10.        Integer numeral = new Random().nextInt(100);
11.        String fin = "Cadena Fin";
12.
13.        FileWriter fw = new FileWriter("PrintWriterTest.txt");
14.        PrintWriter pw = new PrintWriter(fw);
15.
16.        pw.println(inicio);
17.        pw.println(numeral);
18.        pw.println(fin);
19.
20.        pw.close();
21.        fw.close();
22.    }
23.}
```



PrintWriter

# Input/Output



Formatting

# Input/Output

format

```
public PrintWriter format(String  
    Object... args) printf
```

Writes a formatted string to this writer using the specified format string and arguments. It automatically buffers the output buffer.

The locale always used is the one returned by the convenience method `getWriter()`. A call to `format(format, args)` is equivalent to `getWriter().format(format, args)`. This method will flush the output buffer.

Parameters:

format - A format string as described in Format string syntax.

args - Arguments referred to by the format specifiers in the format string. If there are more arguments than format specifiers, the extra arguments are ignored. The number of arguments must be at least one. The maximum number of arguments is limited by the maximum dimension of a Java array as defined by *The Java™ Virtual Machine Specification*.

Returns:

This writer

Throws:

`IllegalFormatException` - If a format string contains an illegal or maximal format specifier that is incompatible with the given arguments, insufficient arguments, or the format string for other legal conditions. No specification of all possible formats errors; see the Details section of the formatter class specification.

`NullPointerException` - If the format is null

Since:

1.5

**%b → Especifica un tipo booleano**

**%c → Especifica un tipo character**

**%d → Especifica un número entero**

**%f → Especifica un número decimal**

**%s → Especifica una cadena**

<http://docs.oracle.com/javase/6/docs/api/java/util/Formatter.html#syntax>

# Input/Output

```
1. import java.io.PrintWriter;  
  
2. class PrintWriterFormat {  
3.  
4.     public static void main(String[] args)  
5.     {  
6.         PrintWriter pw = new PrintWriter(System.out);  
7.  
8.         //0.5 is between 1 and 2  
9.         pw.format("%1.1f is between %d and %d%n", 0.5f, 1,  
10.            2);  
11.         //2 is bigger than 1  
12.         pw.printf("%2$d %3$s %1$d%n", 1, 2, "is bigger  
than");  
13.  
14.         pw.flush();  
15.     }  
16.}
```

# Input/Output

Flags

format

```
public PrintWriter format(String format, Object... args) printf
```

Writes a formatted string to this writer using the specified format string and arguments. If automatic flushing is enabled, calls to output buffer.

The locale always used is the one returned by the Locale method of the Locale class. A convenience method to write a formatted string to this writer using the specified format string and arguments. If automatic flushing is enabled, calls to output buffer.

Parameters:

format - A format string as described in the Locale class documentation.

An invocation of this method of the form `out.printf(format, args)` behaves in exactly the same way as the invocation

args - Arguments referenced by the format specifiers in the format string.

ignored. The number of arguments is limited by the maximum dimension of a Java array as defined in the Java Virtual Machine specification.

Parameters:

Returns:

This writer

Throws:

`IllegalFormatException` - If a format string contains an illegal syntax, a format specifier that is incompatible with the given arguments, insufficient arguments given the format string, or other illegal conditions. For specification of all possible formatting errors, see the Details section of the formatter class specification.

`Formatter` class specification.

`NullPointerException` - If the format string is null

Since:

1.5

**"-" Justificación a la izquierda**

"+" Incluye signo (+ o -)

**"0" Rellena con ceros**

**"," Separador local (i.e., the comma in 123,456)**

**"(" Parentesis para números negativos**

Throws:

`IllegalFormatException` - If a format string contains an illegal syntax, a format specifier that is incompatible with the given arguments, insufficient arguments given the format string, or other illegal conditions. For specification of all possible formatting errors, see the Details section of the formatter class specification.

`NullPointerException` - If the format string is null

Since:

1.5

# Input/Output

```
1. import java.io.PrintWriter;  
  
2. class PrintWriterFormat {  
3.  
4.     public static void main(String[] args)  
5.     {  
6.         PrintWriter pw = new PrintWriter(System.out);  
7.  
8.         int i1 = -123;  
9.         int i2 = 12345;  
10.  
11.         pw.printf(">%1$(7d< \n", i1);  
12.         pw.printf(">%0,7d< \n", i2);  
13.         pw.format(">%+-7d< \n", i2);  
14.         pw.printf(">%2$b + %1$5d< \n", i1, false);  
15.  
16.         pw.flush();  
17.     }  
18.}
```

# Input/Output

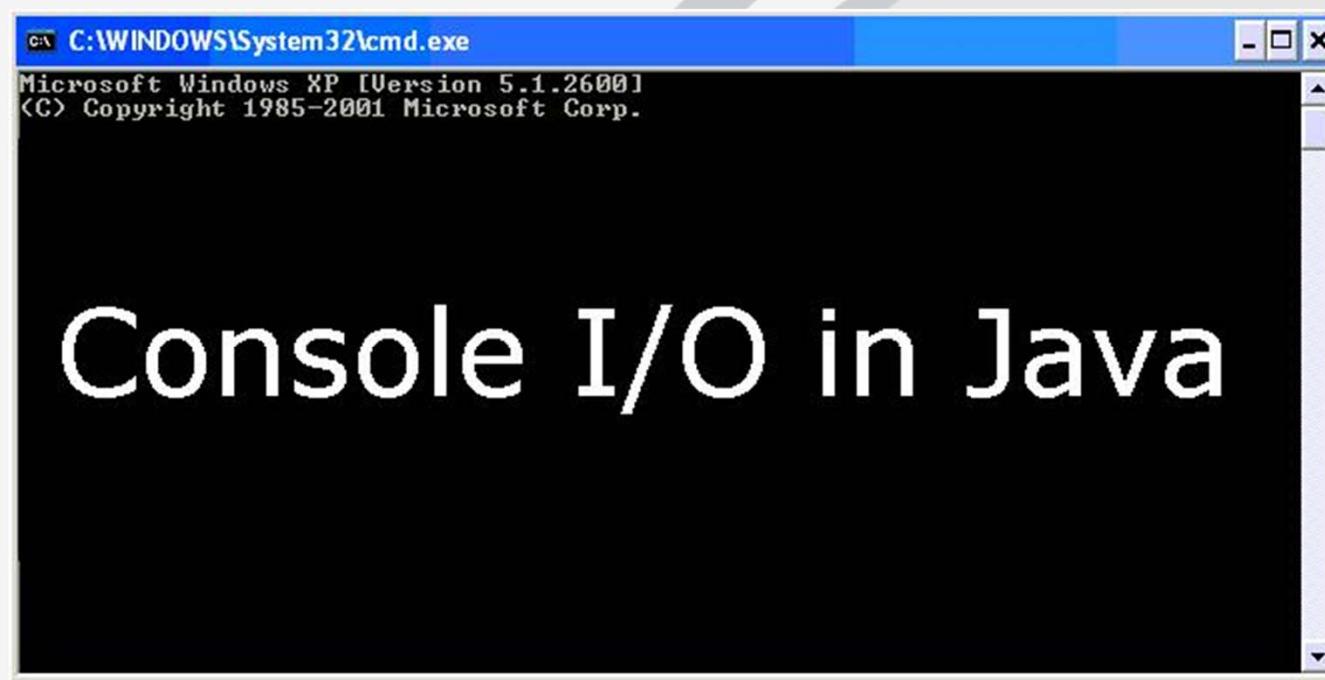
```
1. import java.io.PrintWriter;  
2. class PrintWriterFormat {  
3.     public static void main(String[] args)  
4.     {  
5.         PrintWriter pw = new PrintWriter(System.out);  
6.  
7.         int i1 = -123;  
8.         int i2 = 12345;  
9.         > (123)<  
10.        pw.printf(">%7d<", i1);  
11.        pw.printf(">%7d<", i2);  
12.        pw.format(">%+7d<", i2);  
13.        pw.printf(">%2$7d<", i1);  
14.        pw.printf(">%2$7d<", i1, false);  
15.        pw.println();  
16.    }  
17. }  
18.}
```

> (123)<  
>012.345<  
>+12345 <  
>false + -123<



Console

# Input/Output

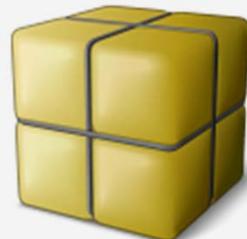


# Input/Output

```
1. import java.io.Console;  
2. class ConsoleTest {  
3.  
4.     public static void main(String[] args) {  
5.         //Get a Console  
6.         Console c = System.console();  
7.  
8.         //Read string  
9.  
10.        if(c != null)  
11.        {  
12.            String user = c.readLine("%s", "input string");  
13.  
14.            //Read password  
15.            char password[];  
16.            password = c.readPassword("%s", "input password");  
17.  
18.            c.writer().println("User: " + input2 + "Password (is secret)");  
19.        }  
20.    }  
21.}
```

# Formatting

java.text



java.util.Locale



# Formatting

5 8  
9 1 2



DecimalFormat

NumberFormat



## NumberFormat

# Input/Output

- ➔ *public static final NumberFormat getInstance()*, método de propósito general que usa el Locale por defecto.
- ➔ *public static NumberFormat getInstance(Locale loc)*, como el anterior pero utilizando un Locale concreto.
- ➔ *public static final NumberFormat getNumberInstance()* ,formatea y parsea números en el formato Locale por defecto.
- ➔ *public static NumberFormat getNumberInstance(Locale loc)*, formatea y parsea números en un formato Locale específico.
- ➔ *public static final NumberFormat getCurrencyInstance()*, devuelve un formato de moneda para el Locale por defecto.
- ➔ *public static NumberFormat getCurrencyInstance(Locale loc)*, devuelve un formato de moneda para unLocale concreto.



## NumberFormat

# Input/Output

→ *public static final NumberFormat getIntegerInstance()*, formateo de enteros.  
Los números decimales son redondeados usando el modo : 'half-even rounding'.

Input Number	Result of rounding input to one digit with the given rounding mode							UNNECESSARY
	UP	DOWN	CEILING	FLOOR	HALF_UP	HALF_DOWN	HALF_EVEN	
5.5	6	5	6	5	6	5	6	throw ArithmeticException
2.5	3	2	3	2	3	2	2	throw ArithmeticException
1.6	2	1	2	1	2	2	2	throw ArithmeticException
1.1	2	1	2	1	1	1	1	throw ArithmeticException
1.0	1	1	1	1	1	1	1	1
-1.0	-1	-1	-1	-1	-1	-1	-1	-1
-1.1	-2	-1	-1	-2	-1	-1	-1	throw ArithmeticException
-1.6	-2	-1	-1	-2	-2	-2	-2	throw ArithmeticException
-2.5	-3	-2	-2	-3	-3	-2	-2	throw ArithmeticException
-5.5	-6	-5	-5	-6	-6	-5	-6	throw ArithmeticException



## NumberFormat

# Input/Output

→ *public static final NumberFormat getIntegerInstance()*, formateo de enteros.  
Los números decimales son redondeados usando el modo : 'half-even rounding'.

Input Number	Result of rounding input to one digit with the given rounding mode							UNNECESSARY
	UP	DOWN	CEILING	FLOOR	HALF_UP	HALF_DOWN	HALF_EVEN	
5.5	6	5	6	5	6	5	6	throw ArithmeticException
2.5	3	2	3	2	3	2	2	throw ArithmeticException
1.6	2	1	2	1	2	2	2	throw ArithmeticException
1.1	2	1	2	1	1	1	1	throw ArithmeticException
1.0	1	1	1	1	1	1	1	1
-1.0	-1	-1	-1	-1	-1	-1	-1	-1
-1.1	-2	-1	-1	-2	-1	-1	-1	throw ArithmeticException
-1.0	-2	-1	-1	-2	-2	-2	-2	throw ArithmeticException
-2.5	-3	-2	-2	-3	-3	-2	-2	throw ArithmeticException
-5.5	-6	-5	-5	-6	-6	-5	-6	throw ArithmeticException

<https://docs.oracle.com/javase/6/docs/api/java/math/RoundingMode.html>



NumberFormat

# Input/Output

- ➔ *public static final NumberFormat getInstance()*, formateo de enteros.  
Los números decimales son redondeados usando el modo : ‘half-even rounding’.
- ➔ *public static NumberFormat getInstance(Locale loc)*, devuelve un formato de entero con un local específico.
- ➔ *public static final NumberFormat getInstance()*, formateo de porcentajes usando el Locale por defecto.
- ➔ *public static NumberFormat getInstance(Locale loc)*, formateo de porcentajes usando unLocale específico.



NumberFormat



# Input/Output

- *public final String format(long number)*
- *public final String format(double number)*



## NumberFormat

# Input/Output

→ *public Number parse (String source)*

### parse

```
public Number parse(String source,  
                    throws ParseException
```

Parses text from the beginning of the given string to produce a number. The method may not use the entire text of the given string.

See the [parse\(String, ParsePosition\)](#) method for more information on number parsing.

#### Parameters:

source - A String whose beginning should be parsed.

#### Returns:

A Number parsed from the string.

#### Throws:

ParseException - if the beginning of the specified string cannot be parsed.



## DecimalFormat



# Input/Output

```
1. double d = 1234567.437;
2. DecimalFormat one = new DecimalFormat("###,###,###.###");
3. System.out.println(one.format(d)); //1,234,567.437
4. DecimalFormat two = new DecimalFormat("000,000,000.00000");
5. System.out.println(two.format(d)); //001,234,567.43700
6. DecimalFormat three = new DecimalFormat("$#,###,###.##");
7. System.out.println(three.format(d)); //\$1,234,567.44
```



DateFormat

# Input/Output

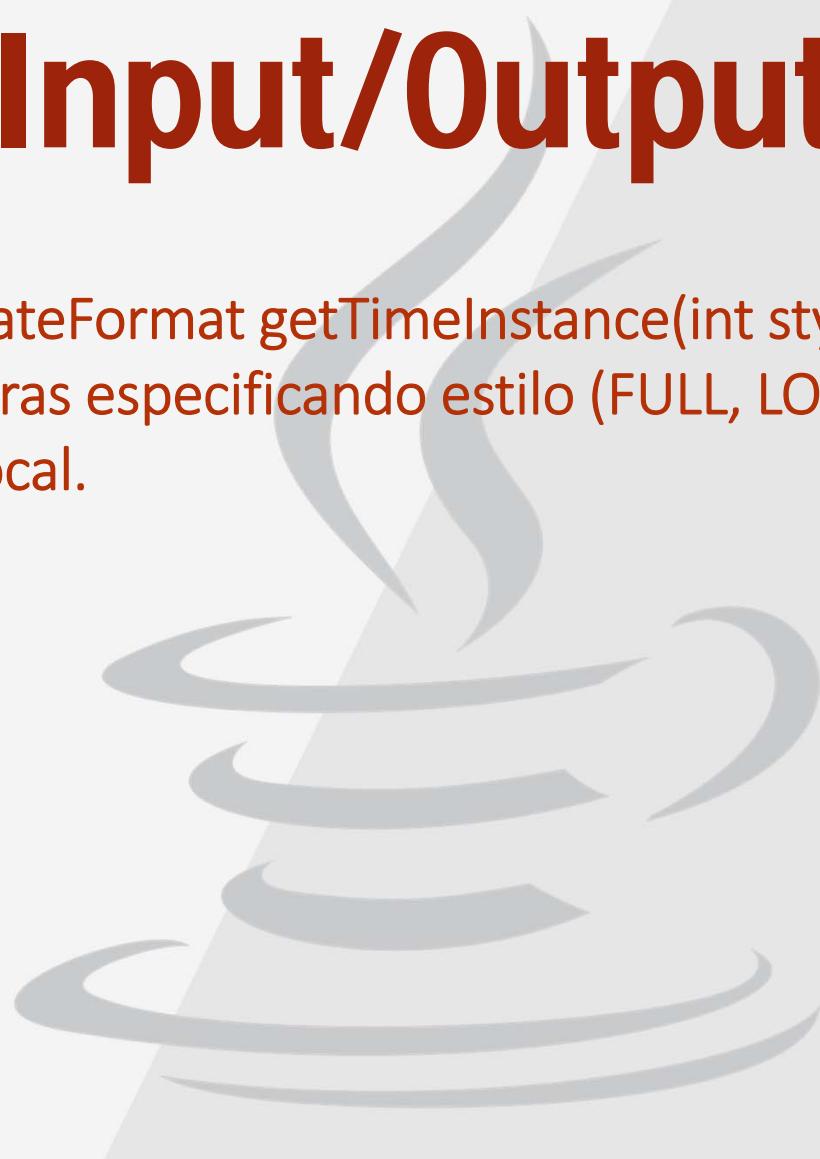
- ➔ public static final DateFormat getDateInstance(), formateo de fechas con el local por defecto.
- ➔ public static final DateFormat getDateInstance(int style, Locale loc), consigue un formateador de fecha especificando estilo (FULL, LONG, MEDIUM, y SHORT en clase DateFormat) y local.
- ➔ public static final DateFormat getTimeInstance() formateo de horas con el local por defecto.



DateFormat

# Input/Output

→ public static final DateFormat getTimelnstance(int style, Locale loc), consigue un formateador de horas especificando estilo (FULL, LONG, MEDIUM, y SHORT en clase DateFormat) y local.





## DateFormat

# Input/Output

- ➔ `public static final DateFormat getTimeInstance(int style, Locale loc)`, consigue un formateador de horas especificando estilo (FULL, LONG, MEDIUM, y SHORT en clase DateFormat) y local.
- ➔ `public static final DateFormat getDateTimelstance()` formateador de Fechas y horas usando el local por defecto.
- ➔ `public static final DateFormat getDateTimelstance(int dateStyle, int timeStyle, Locale loc)` consigue un formateador de fecha y hora especificando estilo (FULL, LONG, MEDIUM, y SHORT en clase DateFormat) y local.



DateFormat

# Input/Output

1. `DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT);`
2. `DateFormat full = DateFormat.getDateInstance(DateFormat.FULL);`
3. `Date d = new Date(444444444000L);`
4. `System.out.println(df.format(d)); //1/31/84`
5. `System.out.println(full.format(d)); //Tuesday, January 31, 1984`
6. `DateFormat dtf = DateFormat.getDateInstance(DateFormat.MEDIUM,DateFormat.FULL);`
7. `System.out.println(dtf.format(d)); //Jan 31, 1984 5:47:24 PM MST`
8. `DateFormat de = DateFormat.getDateInstance(DateFormat.MEDIUM,DateFormat.FULL,Locale.GERMANY);`
9. `System.out.println(de.format(d)); //31.01.1984 17.47 Uhr MST`



DateFormat

# Input/Output

## df.setLocale(...)

```
1. DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM, Locale.US);
2. DateFormat full = DateFormat.getDateInstance(DateFormat.FULL, Locale.US);
3. Date d = new Date(444444444000L);
4. System.out.println(df.format(d)); //1/31/84
5. System.out.println(full.format(d)); //Tuesday, January 31, 1984
6. DateFormat dtf = DateFormat.getTimeInstance(DateFormat.MEDIUM, DateFormat.US);
7. System.out.println(dtf.format(d)); //Jan 31, 1984 5:47:24 PM MST
8. DateFormat de = DateFormat.getDateTimeInstance(DateFormat.MEDIUM, DateFormat.US);
9. System.out.println(de.format(d)); //31.01.1984 17.47 Uhr MST
```



# Expresiones Regulares





Pattern

# Expresiones Regulares





Pattern

# Expresiones Regulares

```
1.String regex = "he";  
2.Pattern pattern = Pattern.compile(regex);  
3.Matcher m1 = pattern.matcher("hello");  
4.Matcher m2 = pattern.matcher("goodbye");  
5.if(m1.matches()) {  
6.    System.out.println("hello is a match");  
7.}  
8.if(m2.matches()) { //m2.matches = false  
9.    System.out.println("goodbye is a match");
```



Pattern

# Expresiones Regulares

```
1.String regex = ".ing";
2.Pattern pattern = Pattern.compile(regex);
3.String [] tests = {"ing", "ring", "trying", "running",
"beings"};
4.for(String test: tests) {
5.    Matcher m = pattern.matcher(test);
6.    if(m.matches()) {
7.        System.out.println(test + " matches " + regex);
8.    }
9.}
```

# Expresiones Regulares

Metacarácter	Descripción
.	(punto) Any character
*	Match the preceding character any number of times
+	Match the previous character one or more times
?	Match the previous character 0 or 1 times only
\d	A digit 0–9
\s	A whitespace character
\w	A word character (any lowercase or uppercase letter, the underscore character, or any digit)
[]	Match anything inside the square brackets for one character position once
()	Use parentheses for grouping together search expressions

# Expresiones Regulares

<http://www.regexr.com/>

The screenshot shows the RegExr v2.0 interface. On the left is a sidebar with links to Help, Reference, Cheatsheet, Examples, Community, and Favourites. Below this is a section about the tool's purpose: "RegExr is an online tool to learn, build, & test Regular Expressions (RegEx / RegExp)." It lists several features: Results update in real-time, Roll over a match or expression for details, Save & share expressions with others, Explore the Library for help & examples, Undo & Redo with Ctrl-Z / Y, and Search for & rate Community patterns.

The main area has tabs for Expression, Text, and Substitution. The Expression tab contains the regular expression `/([A-Z])\w+/g`, which matches 16 matches in the provided sample text. The Text tab contains the sample text: "Welcome to RegExr v2.0 by gskinner.com! Edit the Expression & Text to see matches. Roll over matches or the expression for details. Undo mistakes with ctrl-z. Save & Share expressions with friends or the Community. A full Reference & Help is available in the Library, or watch the video Tutorial." The Substitution tab is currently inactive.

# Expresiones Regulares

The screenshot shows the regex101.com web application interface. The main area displays a regular expression tester with the following components:

- REGULAR EXPRESSION:** A text input field containing the regex pattern `/gmixXsuUAJ`. To its left is a dropdown menu labeled "FLAVOR" with options "PCRE", "JS", and "PY".
- TEST STRING:** A text input field containing the string `insert your test string here`.
- EXPLANATION:** A panel stating, "An explanation of your regex will be automatically generated as you type." Below this, a preview window shows the result of the regex match.
- MATCH INFORMATION:** A panel stating, "Detailed match information will be displayed here automatically." Below this, a preview window shows the result of the regex match.
- QUICK REFERENCE:** A sidebar with a "FULL REFERENCE" link and a search icon. It lists several tokens and categories:
  - MOST USED TOKENS:** A single character o... [abc], A character except... [^abc]
  - CATEGORIES:** general tokens, anchors
- SUBSTITUTION:** A section at the bottom left with a dollar sign icon and a plus sign icon.

On the right side of the interface, there is a sidebar with the following information:

- by gskinner RegExr v1 GitHub Tutorial
- share save flags
- 16 matches
- o RegExr v2.0 by gskinner.com!
- Expression & Text to see matches. Roll over matches for details. Undo mistakes with ctrl-z. Save expressions with friends or the Community. A full & Help is available in the Library, or watch the tutorial.
- xt for testing:  
jklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ  
9 -+-,lgse\$&\*();\|<>"'  
.7 3.141 .6180 9,000 +42  
567 -+1-(800)-555-2468  
net bar.ba@test.co.uk  
com http://foo.co.uk/  
gexr.com/foo.html?q=bar

At the bottom left, the URL <https://regex101.com/> is displayed in blue text.

# Expresiones Regulares

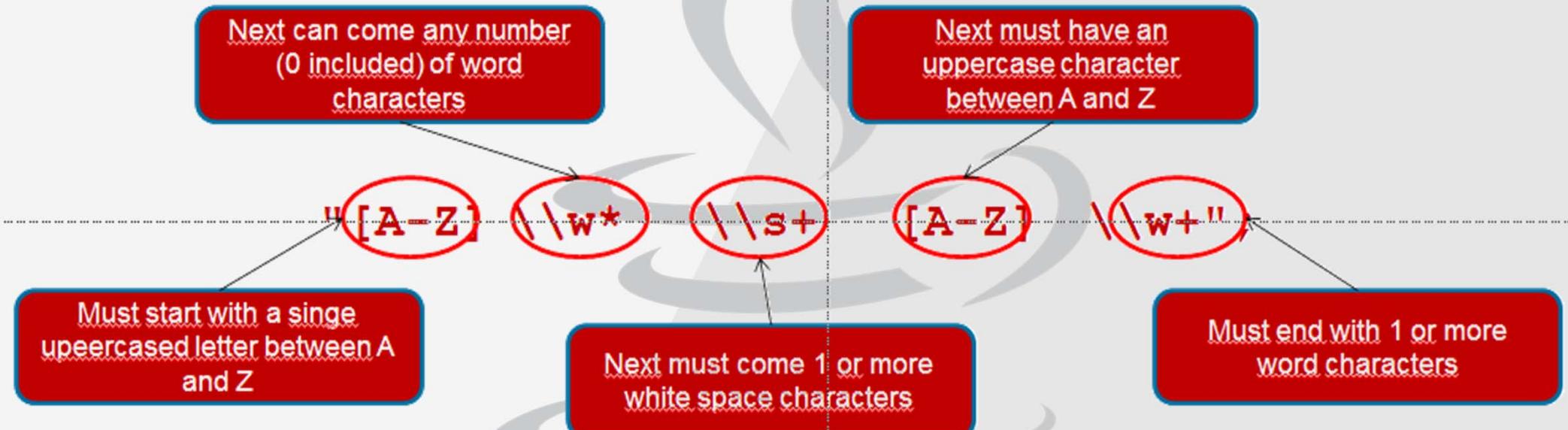
## Carácteres Predefinidos

- \d, which denotes a digit; equivalent to [0-9]
- \s, which denotes a whitespace character; equivalent to [ \t\n\x0B\f\r]
- \w, which denotes a word character; equivalent to [a-zA-Z\_0-9]

# Expresiones Regulares

```
1.String s = "[A-Z]\\w+\\s+[A-Z]\\w+";  
2.Pattern x = Pattern.compile(s);  
3.String [] names = {"John Doe", "JohnDoe",  
"John\tDoe", "John doe",  
4.        "J D", "J  D5"};  
5.for(String name: names) {  
6.    Matcher m = x.matcher(name);  
7.    if(m.matches()) {  
8.        System.out.println(name + " matches " + s);  
9.    }  
10.}
```

# Expresiones Regulares





# Expresiones Regulares

## String.split()

### split

```
public String[] split(String regex)
```

Splits this string around matches of the given [regular expression](#).

This method works as if by invoking the two-argument [split](#) method with the given expression and a limit argument of zero. Trailing empty strings are therefore not included in the resulting array.

The string "boo:and:foo", for example, yields the following results with these expressions:

Regex	Result
:	{ "boo", "and", "foo" }
o	{ "b", "", ":and:f" }

#### Parameters:

regex - the delimiting regular expression

#### Returns:

the array of strings computed by splitting this string around matches of the given regular expression

#### Throws:

[PatternSyntaxException](#) - if the regular expression's syntax is invalid

#### Since:

1.4

#### See Also:

[Pattern](#)



String.split()

# Expresiones Regulares

```
1.String data = "3035551212,123 Main  
St.\tDenver,CO:50431";  
2.String [] results =  
    data.split("[,;\\t]");  
3.for(String result : results) {  
4.    System.out.println(result);  
5.}
```



String.split()

# Expresiones Regulares

The output of the code is

3035551212

123 Main St.

Denver

CO

50431



Scanner

# Expresiones Regulares

```
Scanner s1 = new Scanner(System.in)
```

```
Scanner s1 = new Scanner(new  
File("myNumbers"))
```



Scanner

# Expresiones Regulares

```
while(b = s1.hasNext())
{
    s = s1.next(); defaultCount++;
}
```



Scanner

# Expresiones Regulares

```
while(b = s2.hasNext()) {  
    if (s2.hasNextInt()) {  
        int i = s2.nextInt(); intCount++;  
    } else if (s2.hasNextBoolean()) {  
        boolean b2 = s2.nextBoolean();  
        boolCount++;  
    } else {  
        s2.next(); defaultCount++;  
    }  
}
```



Scanner

public String next()

it returns the next token from the scanner

public String nextLine()

it moves the scanner position to the next line and returns the value as a string.

public byte nextByte()

it scans the next token as a byte.

public short nextShort()

it scans the next token as a short value.

public int nextInt()

it scans the next token as an int value

public long nextLong()

it scans the next token as a long value

public float nextFloat()

it scans the next token as a float value

public double nextDouble()

it scans the next token as a double value

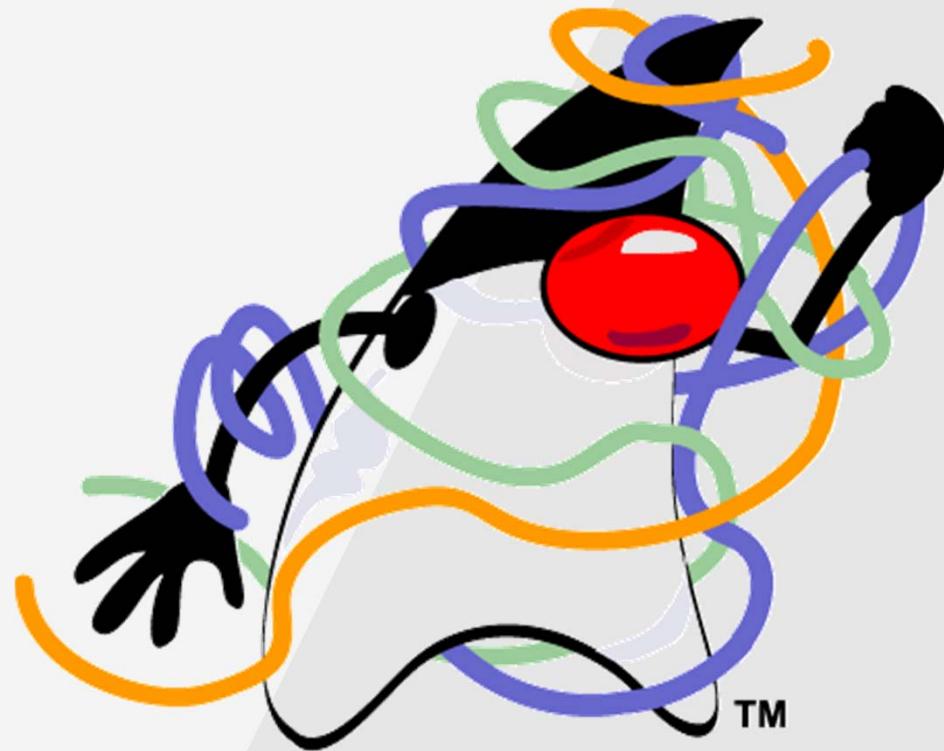




#4

# Concurrencia

# Concurrencia



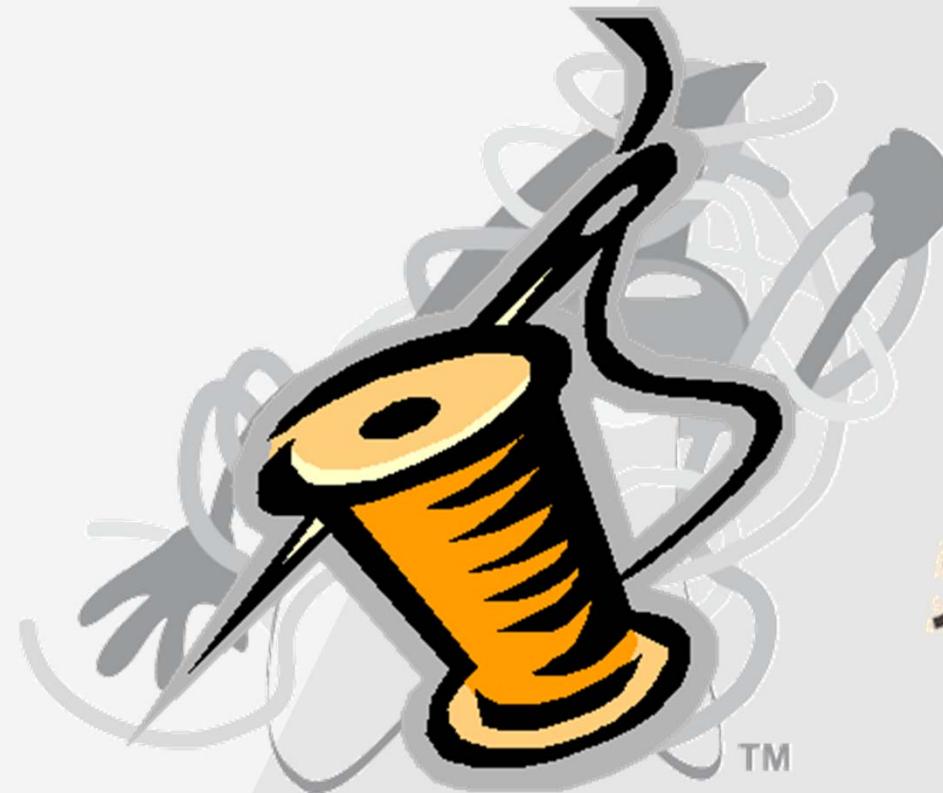
# Concurrencia

Threads



Runnable

# Concurrencia



JAVA public static  
void main  
(String args[])

# Concurrencia



# Concurrencia

```
1. package Concurrency;  
2. class MyRunnableClass implements Runnable{  
3.  
4.     public void run()  
5.     {  
6.         System.out.println("Entering in run() Method");  
7.         System.out.println("Doing things in run() Method");  
8.         System.out.println("Exciting in run() Method");  
9.     }  
10.    public static void main(String[] args)  
11.    {  
12.        MyRunnableClass myRunnable = new MyRunnableClass ();  
13.        Thread t = new Thread(myRunnable);  
14.        t.start();  
15.    }  
16.}
```

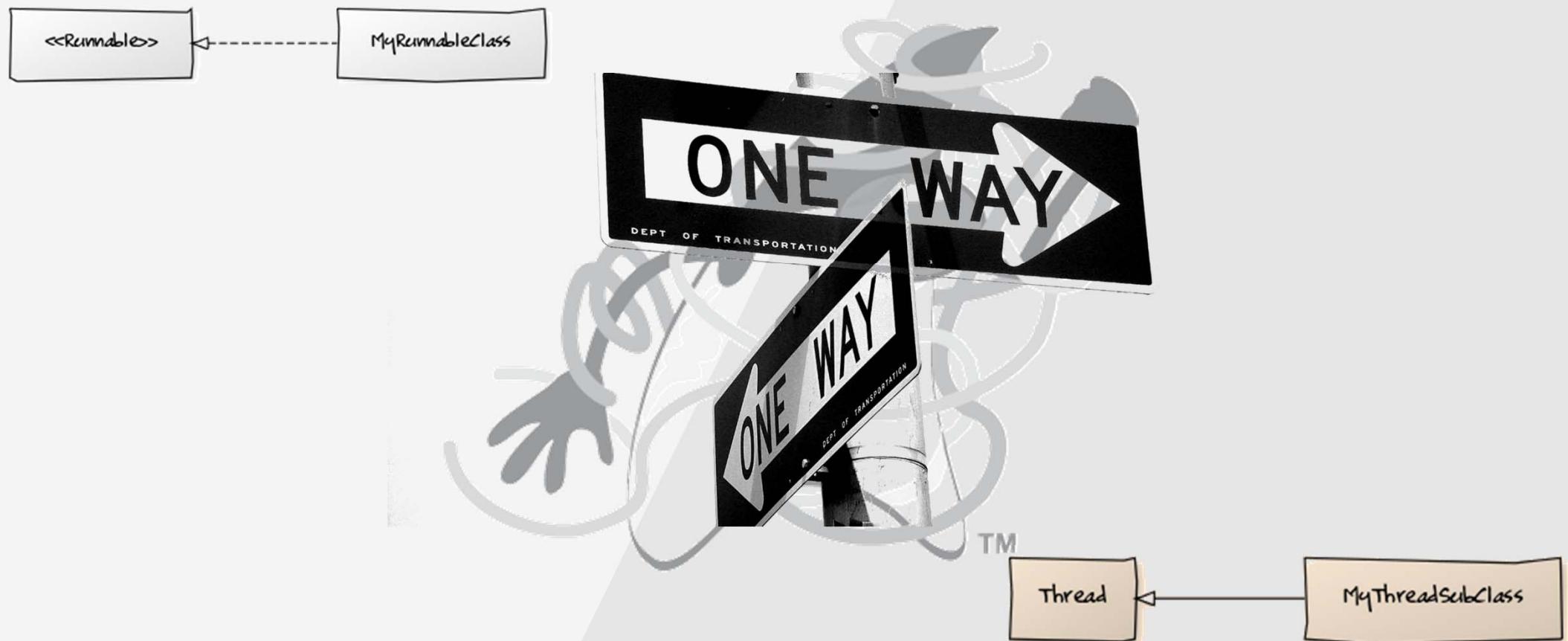
# Concurrencia

```
1. package Concurrency;  
2. class MyRunnableClass implements Runnable{  
3.     public void run()  
4.     {  
5.         System.out.println("Entering in run() Method");  
6.         System.out.println("Doing things in run() Method");  
7.         System.out.println("Exciting in run() Method");  
8.     }  
9.  
10.    public static void main(String[] args)  
11.    {  
12.        MyRunnableClass myRunnable = new MyRunnableClass ();  
13.        Thread t = new Thread(myRunnable);  
14.        t.start();  
15.    }  
16.}
```

# Concurrencia

```
1. package Concurrency;  
2. class MyRunnableClass implements Runnable{  
3.  
4.     public void run()  
5.     {  
6.         System.out.println("Entering in run() Method");  
7.         System.out.println("Doing things in run() Method");  
8.         System.out.println("Exciting in run() Method");  
9.     }  
10.    static void main(String[] args)  
11.    {  
12.        MyRunnableClass myRunnable = new MyRunnableClass();  
13.        Thread t = new Thread(myRunnable);  
14.        t.start();  
15.    }  
16.}
```

# Concurrencia



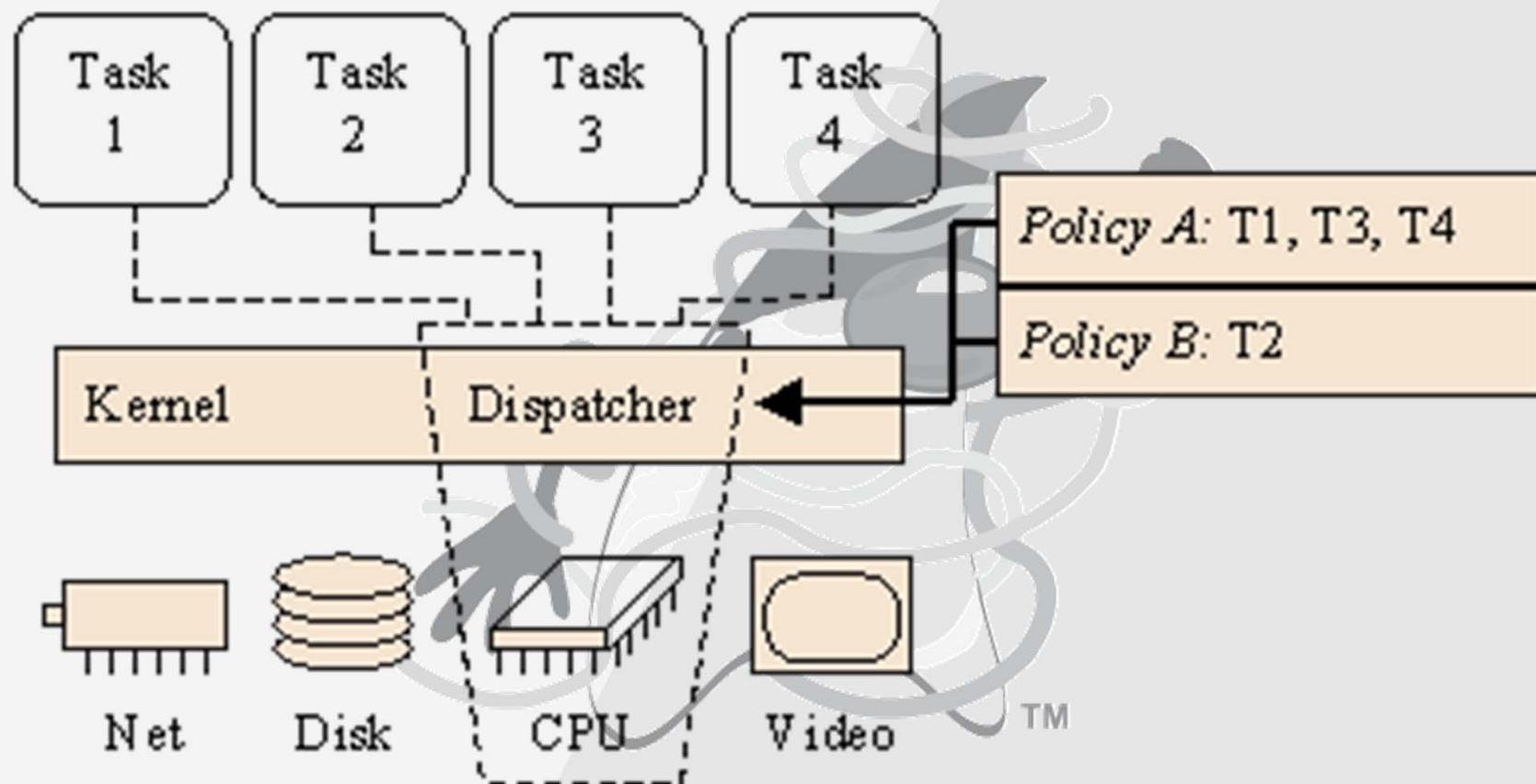
# Concurrencia

```
1. package Concurrency;  
2. class MyThreadSubClass extends Thread{  
3.  
4.     public MyThreadSubClass ()  
5.     {  
6.         System.out.println("Building MyThreadSubClass");  
7.  
8.  
9.         @Override  
10.        public void run()  
11.        {  
12.            System.out.println("run() Method @Overrided");  
13.        }  
14.  
15.        public static void main(String[] args) {  
16.  
17.            new MyThreadSubClass().start();  
18.        }  
19.    }
```

# Concurrencia

```
1. package Concurrency;  
2. class MyThreadSubClass extends Thread{  
3.  
4.     public MyThreadSubClass ()  
5.     {  
6.         System.out.println("Building MyThreadSubClass");  
7.     }  
8.  
9.     @Override  
10.    public void run()  
11.    {  
12.        System.out.println("run() Method @Overrided");  
13.    }  
14.  
15.    public static void main(String[] args) {  
16.        new MyThreadSubClass().start();  
17.    }  
18.  
19.}
```

# Concurrencia



# Concurrencia

# PREEMPTIVE



TM



# Concurrencia

*public Thread(Runnable target)*

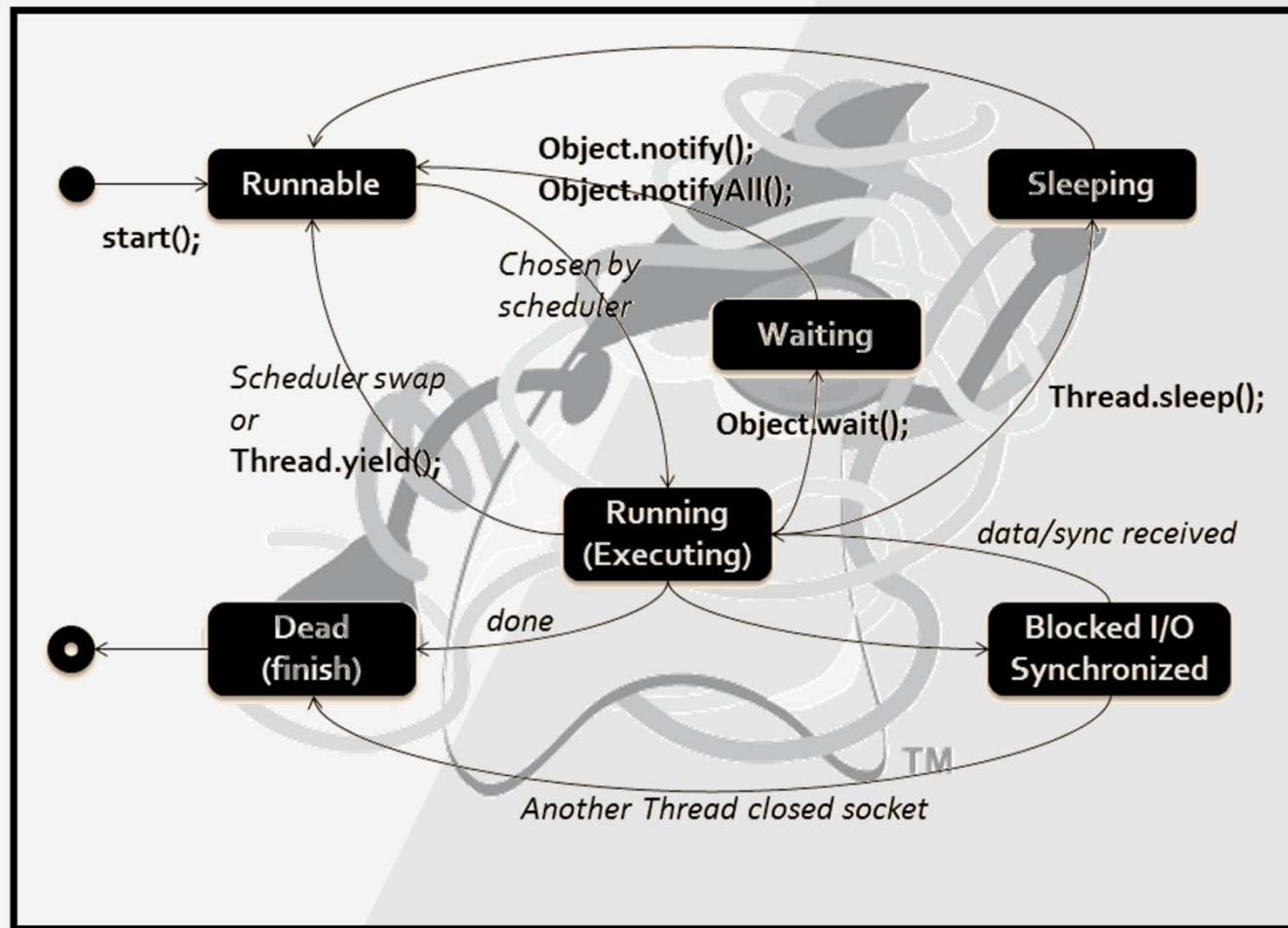
*public Thread(Runnable target, String name)*

*public Thread(ThreadGroup group, Runnable target)*

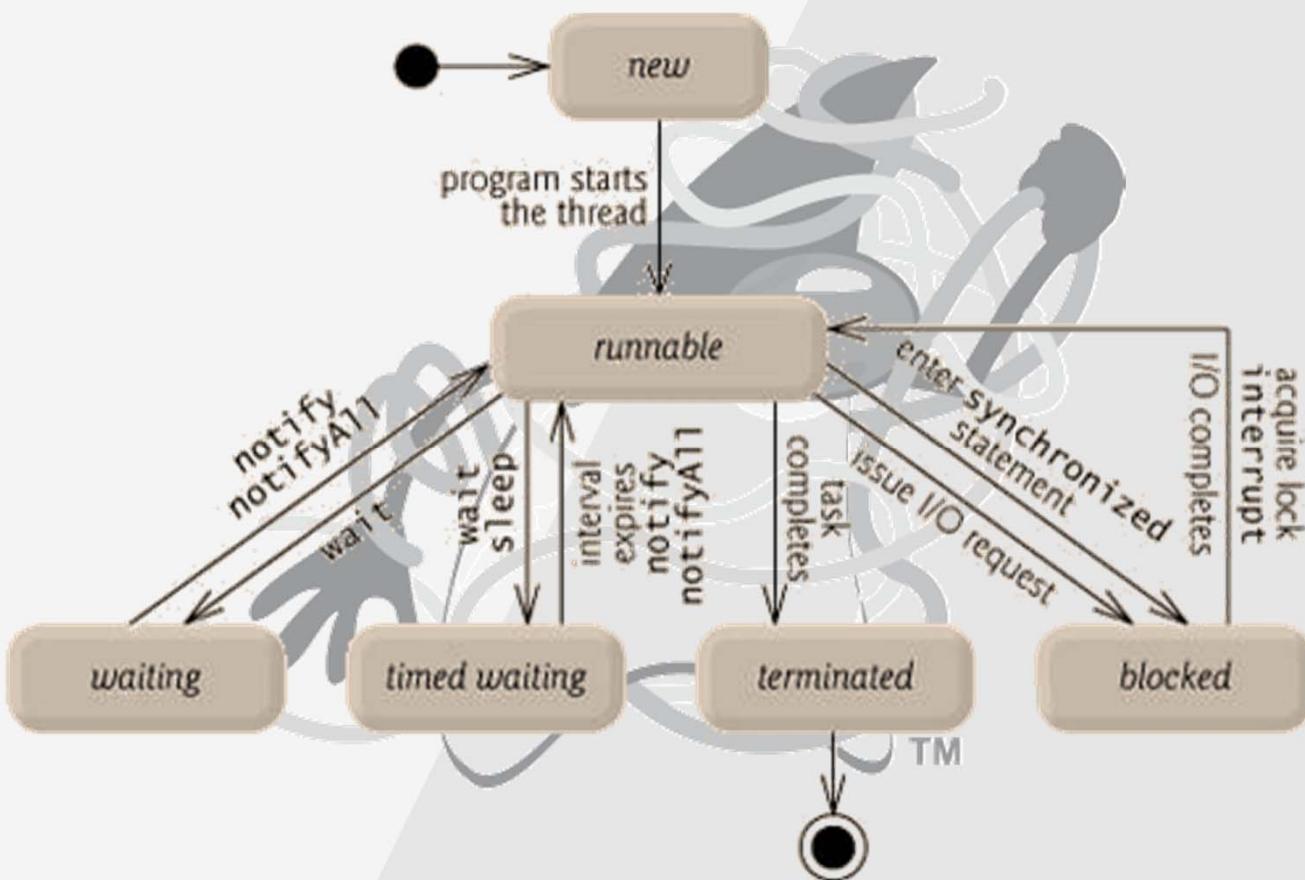
*public Thread(ThreadGroup group, Runnable target, String name)*

*public Thread(ThreadGroup group, Runnable target, String name, long stackSize)*

# Concurrencia



# Concurrencia



# Concurrencia

- NEW, El hilo ha sido instanciado pero aún no lanzado.
- RUNNABLE, El hilo se está ejecutando en la CPU o en espera del Scheduler de JAVA.
- BLOCKED, El hilo está esperando un desbloqueo de un lock ( Sincronización).

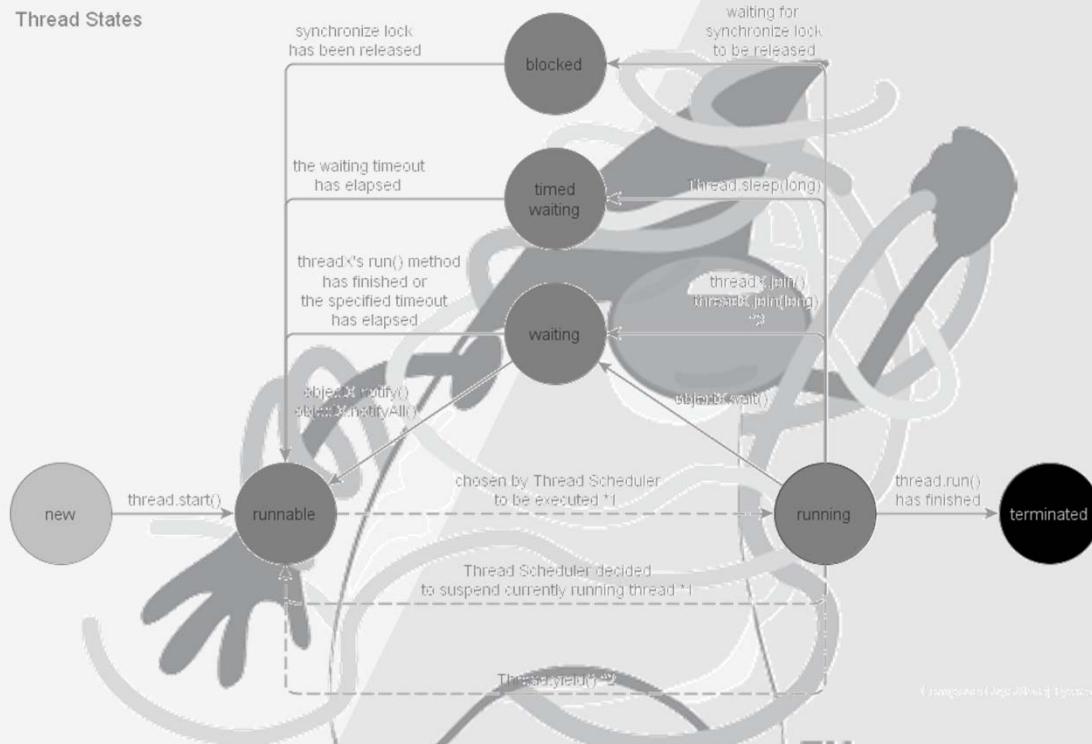
# Concurrencia

- WAITING, El hilo está esperando que otro hilo haga algo. ( Notify )
- TIMED\_WAITING, Similar a WAITING except que solo espera un notify o un join durante un tiempo.
- TERMINATED, El hilo termino su ejecución ( No puede ser relanzado ).

# Concurrencia

Method	Return Type	Description
currentThread( )	Thread	Returns an object reference to the thread in which it is invoked.
getName( )	String	Retrieve the name of the thread object or instance.
start( )	void	Start the thread by calling its run method.
run( )	void	This method is the entry point to execute thread, like the main method for applications.
sleep( )	void	Suspends a thread for a specified amount of time (in milliseconds).
isAlive( )	boolean	This method is used to determine the thread is running or not.
activeCount( )	int	This method returns the number of active threads in a particular thread group and all its subgroups.
interrupt( )	void	The method interrupt the threads on which it is invoked.
yield( )	void	By invoking this method the current thread pause its execution temporarily and allow other threads to execute.
join( )	void	This method and <code>join(long millisec)</code> Throws InterruptedException. These two methods are invoked on a thread. These are not returned until either the thread has completed or it is timed out respectively.

# Concurrencia



\*1 - the behaviour of Thread Scheduler is not guaranteed - you can't really know which thread will be chosen to be executed or when the Thread Scheduler decides to suspend (or if even will) one thread and start executing another one

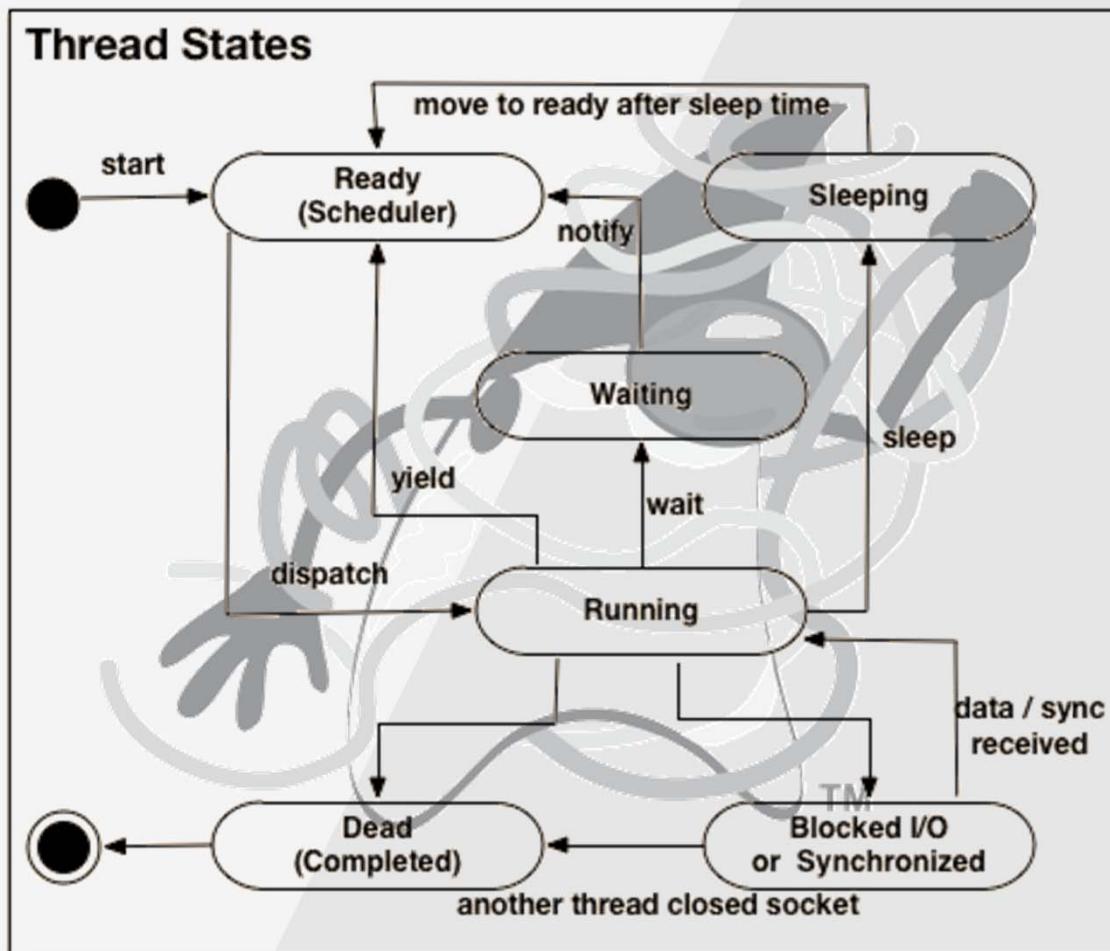
\*2 - although it might, there is no guarantee that Thread.yield() method will put current thread into runnable state

\*3 - the thread that executed the `threadX.join()` method will be waiting until the `threadX.run()` method will finish.

Keep in mind that the thread executing this command will be put on hold and not the thread on which the `.join()` method was executed (in this case it is `threadX`).

There is also a `.join(long)` method, that will keep current thread on hold only for maximum the "long" number of milliseconds or till the `run()` method of the `threadX` will end

# Concurrencia



# Concurrencia





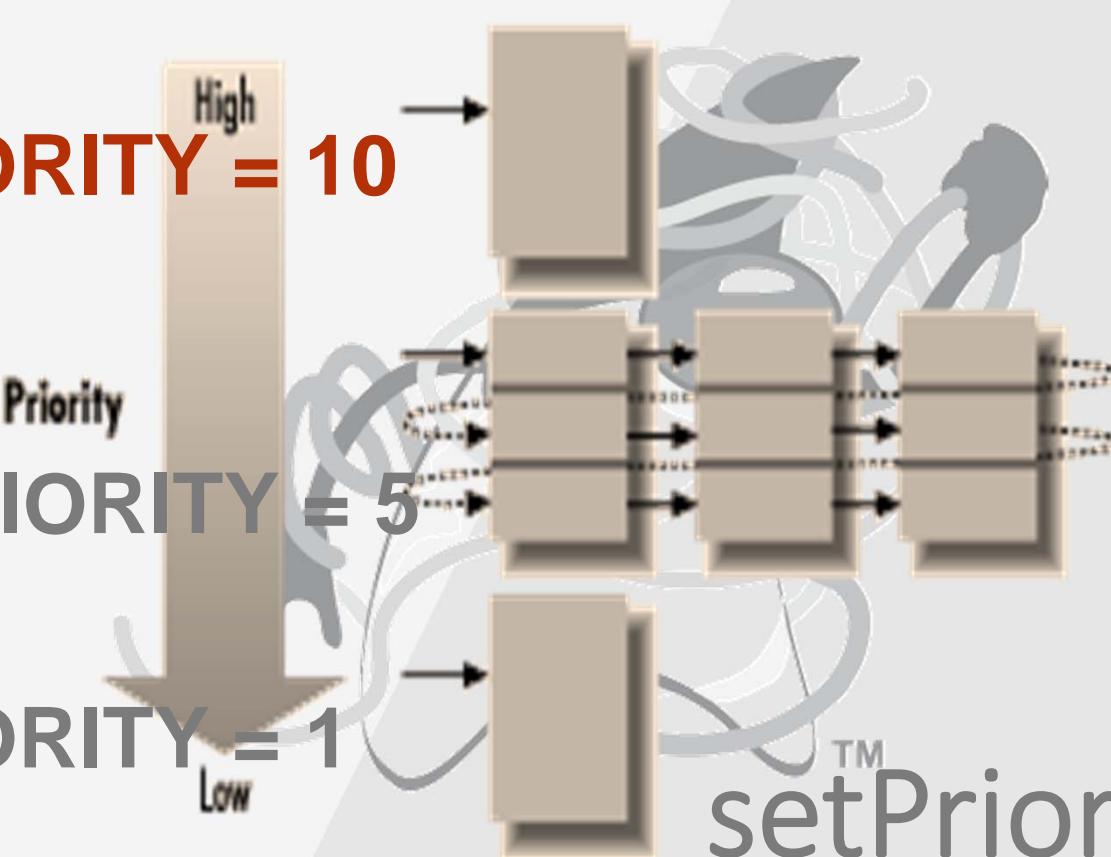
# Concurrencia

**MAX\_PRIORITY = 10**

Priority

**NORM\_PRIORITY = 5**

**MIN\_PRIORITY = 1**

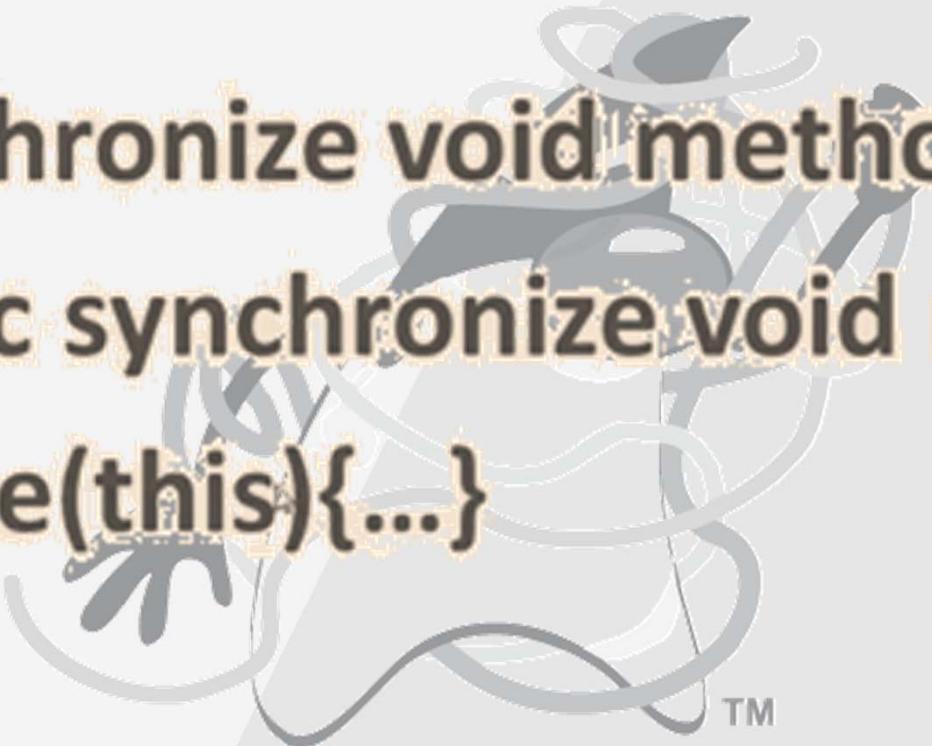


# Concurrencia



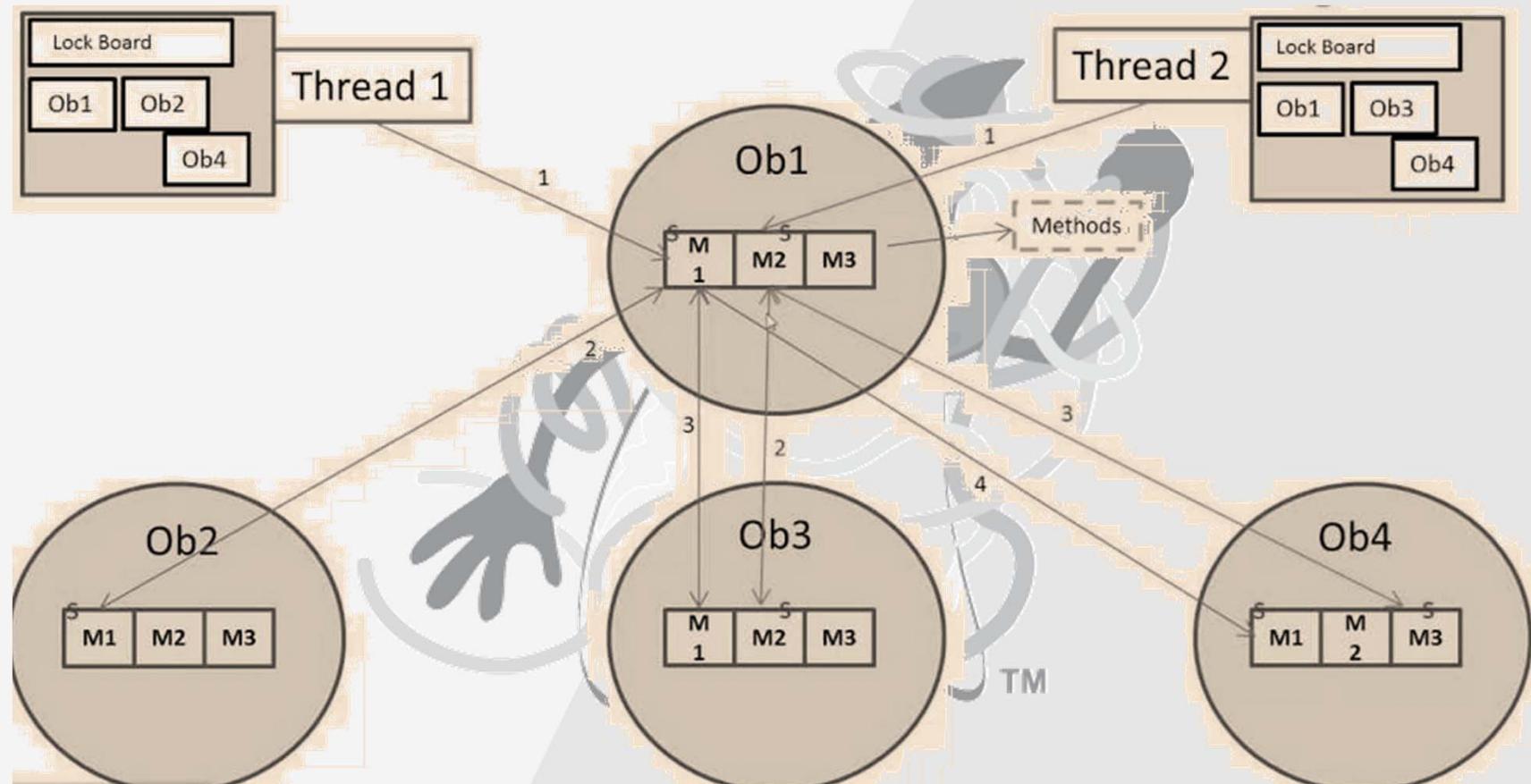
Synchronization

# Concurrencia

A faint watermark of the Java logo, featuring a stylized sun, a tree, and a figure, is centered behind the text.

**public synchronize void method(){ }**  
**public static synchronize void method{...}**  
**synchronize(this){...}**

# Concurrencia



# Práctica: “Productores / Consumidores”



# Práctica: “Productores / Consumidores”

```
1. package Concurrency;  
2. public class Almacen {  
3.  
4.     private int unidades;  
5.  
6.     public Almacen(int unidades) {  
7.         this.unidades = unidades;  
8.     }  
9.     public synchronized void addUnidad(){  
10.        this.unidades++;  
11.    }  
12.  
13.    public synchronized void getUnidad() throws NoUnidadesException{  
14.        if (this.unidades > 0)  
15.        {  
16.            this.unidades--;  
17.        }  
18.        else  
19.        {  
20.            throw new NoUnidadesException();  
21.        }  
22.    }  
23.  
24.    public Integer getExistencias(){  
25.        return this.unidades;  
26.    }  
27. }
```

# Práctica: “Productores / Consumidores”

```
1. package Concurrency;  
2. class NoUnidadesException extends Exception  
3. {  
4.     private final String NO_UNIDDES_MESSAGE = "El Almacén No  
    Dispone de Más Unidades";  
5.  
6.     public String getMessage()  
7.     {  
8.         return this.NO_UNIDDES_MESSAGE;  
9.     }  
10.}
```

# Práctica: “Productores / Consumidores”

```
1. package Concurrency;  
2. public class Productor implements Runnable{  
3.  
4.     private Almacen almacen;  
5.     private String nombre;  
6.  
7.     public Productor (Almacen almacen, String nombre){  
8.         this.almacen = almacen;  
9.         this.nombre= nombre;  
10.    }  
11.  
12.    public void putUnidad(){  
13.        this.almacen.addUnidad();  
14.    }  
15.  
16.    public void run(){  
17.        System.out.println("Productor: " + this.nombre + " llevando mercancia");  
18.  
19.        this.putUnidad();  
20.  
21.        System.out.println("Productor: " + this.nombre + " notificando entrega");  
22.    }  
23.}
```

# Práctica: “Productores / Consumidores”

```
1. package Concurrency;  
2. public class Consumidor implements Runnable{  
3.  
4.     private Almacen almacen;  
5.     private String nombre;  
6.  
7.     public Consumidor (Almacen almacen, String nombre){  
8.         this.almacen = almacen;  
9.         this.nombre= nombre;  
10.    }  
11.  
12.    public void consumir() throws NoUnidadesException{  
13.        this.almacen.getUnidad();  
14.    }  
15.  
16.    public void run() {  
17.        System.out.println("Consumidor: " + this.nombre + " dirigiendose al almacen");  
18.  
19.        try  
20.        {  
21.            this.consumir();  
22.        }  
23.        catch(NoUnidadesException nuex)  
24.        {  
25.            System.out.println(nuex.getMessage());  
26.        }  
27.  
28.        System.out.println("Consumidor: " + this.nombre + " unidad obtenida");  
29.    }  
30. }
```

# Práctica: “Generic Queue”

```
1. package recursividad.test;  
2. import java.util.LinkedList;  
3. import java.util.Queue;  
4. public class BlockingQueue<T> {  
5.     private Queue<T> queue = new LinkedList<T>();  
6.     private int capacity;  
7.     public BlockingQueue(int capacity) {  
8.         this.capacity = capacity;  
9.     }  
10.    public synchronized void put(T element) throws InterruptedException {  
11.        while(queue.size() == capacity) {  
12.            wait();  
13.        }  
14.        queue.add(element);  
15.        notify(); // notifyAll() for multiple producer/consumer threads  
16.    }  
17.    public synchronized T take() throws InterruptedException {  
18.        while(queue.isEmpty()) {  
19.            wait();  
20.        }  
21.        T item = queue.remove();  
22.        notify(); // notifyAll() for multiple producer/consumer threads  
23.        return item;  
24.    }  
25.}
```

# Práctica: “Generic Queue”



# Práctica: “Productores / Consumidores”

```
1. package Concurrency;
2. import java.util.Random;
3. import java.util.List;
4. import java.util.ArrayList;
5.
6. public class StartProductoresConsumidores {
7.     private static final int UNIDADES_ALMACEN = 10;
8.
9.     public static void main(String[] args) {
10.
11.         Almacen almacen = new Almacen(StartProductoresConsumidores.UNIDADES_ALMACEN);
12.
13.         Random generador = new Random();
14.
15.         Integer numeroProductores = generador.nextInt(5);
16.         System.out.println("Generados -> " + numeroProductores + " productores");
17.         Integer numeroConsumidores = generador.nextInt(5);
18.         System.out.println("Generados -> " + numeroConsumidores + " consumidores");
19.
20.         List <Runnable> hilos = new ArrayList<Runnable>();
21.
22.         for (int i=0; i < numeroProductores; i++){
23.             hilos.add(new Productor(almacen, "#" + Integer.valueOf(i+1).toString()));
24.             System.out.println("Agregado Productor");
25.         }
26.
27.         for (int i=0; i < numeroConsumidores; i++){
28.             hilos.add(new Consumidor(almacen, "#" + Integer.valueOf(i+1).toString()));
29.             System.out.println("Agregado Consumidor");
30.         }
31.
32.         while (hilos.size() > 0){
33.             int numeroHilo = generador.nextInt(hilos.size());
34.
35.             System.out.println("Extrayendo hilo del pool (" + numeroHilo + " de " + hilos.size() + ")");
36.             Thread hilo = new Thread(hilos.get(numeroHilo));
37.
38.             System.out.println("Lanzando hilo");
39.             hilo.start();
40.
41.             System.out.println("Quitando hilo del pool");
42.             hilos.remove(numeroHilo);
43.         }
44.
45.         System.out.println("Estado Final del Almacen: " + almacen.getExistencias() + " existencias");
46.     }
47. }
```

# Concurrencia

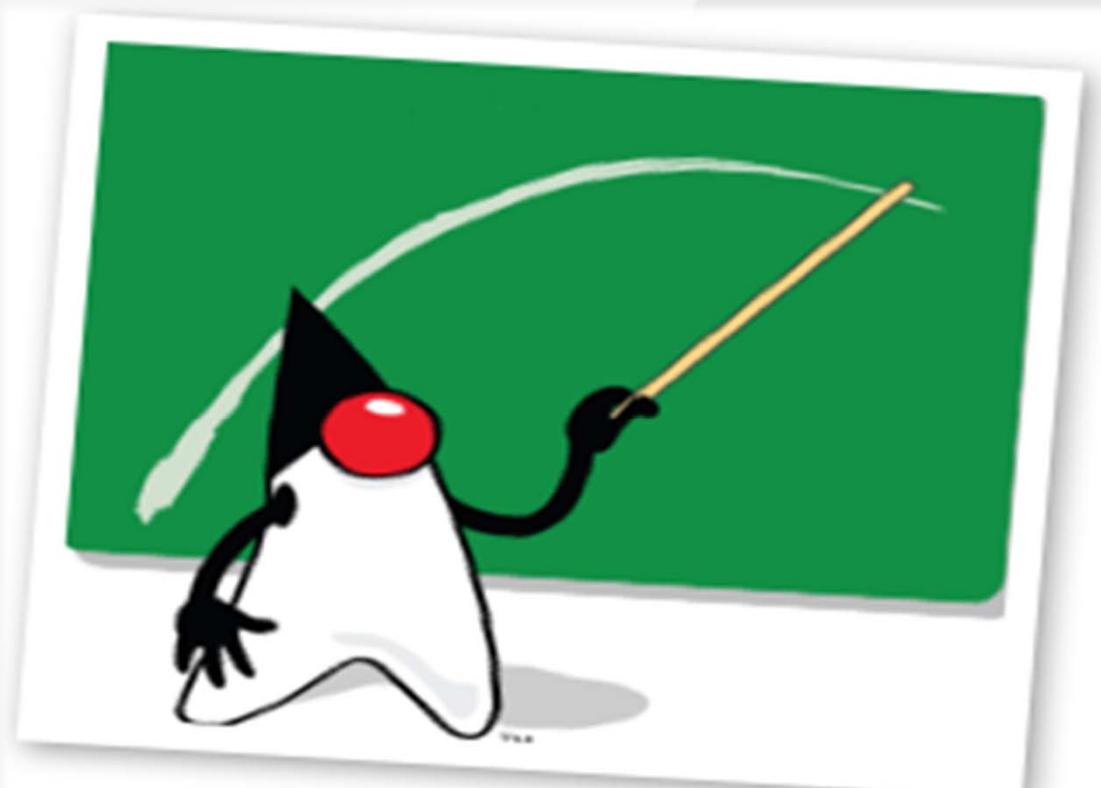




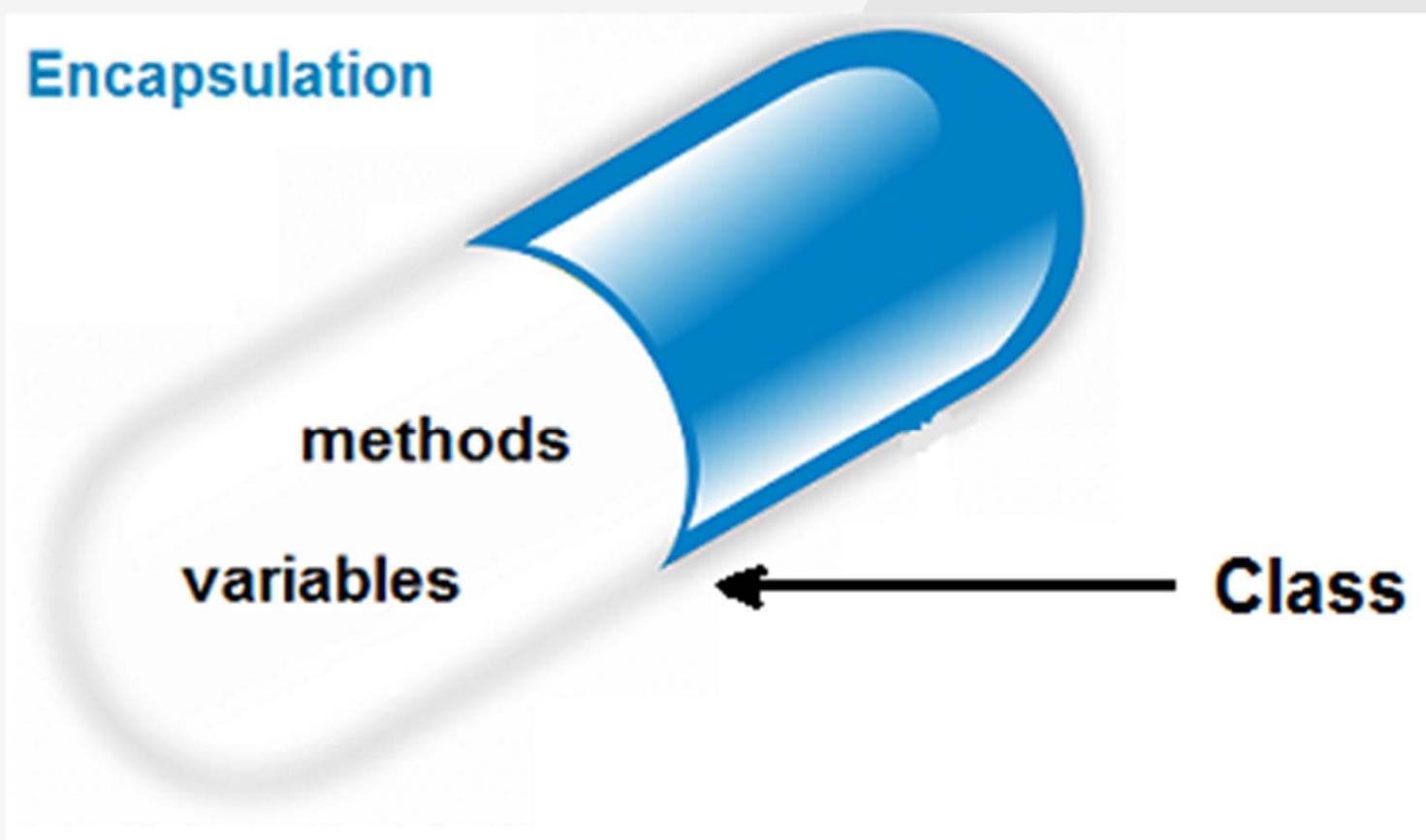
#5

# Conceptos Orientación Objetos

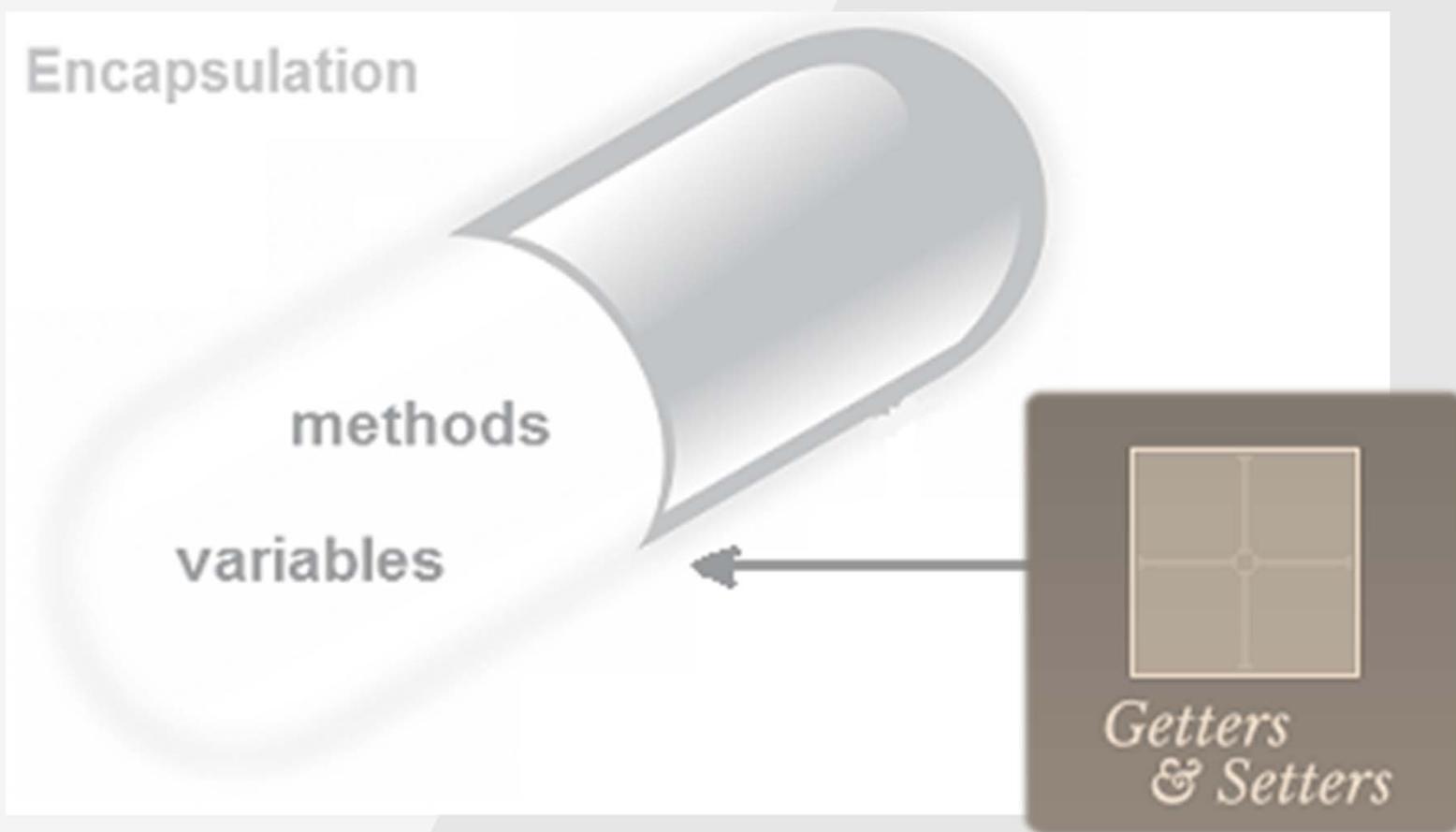
# CONCEPTOS 00



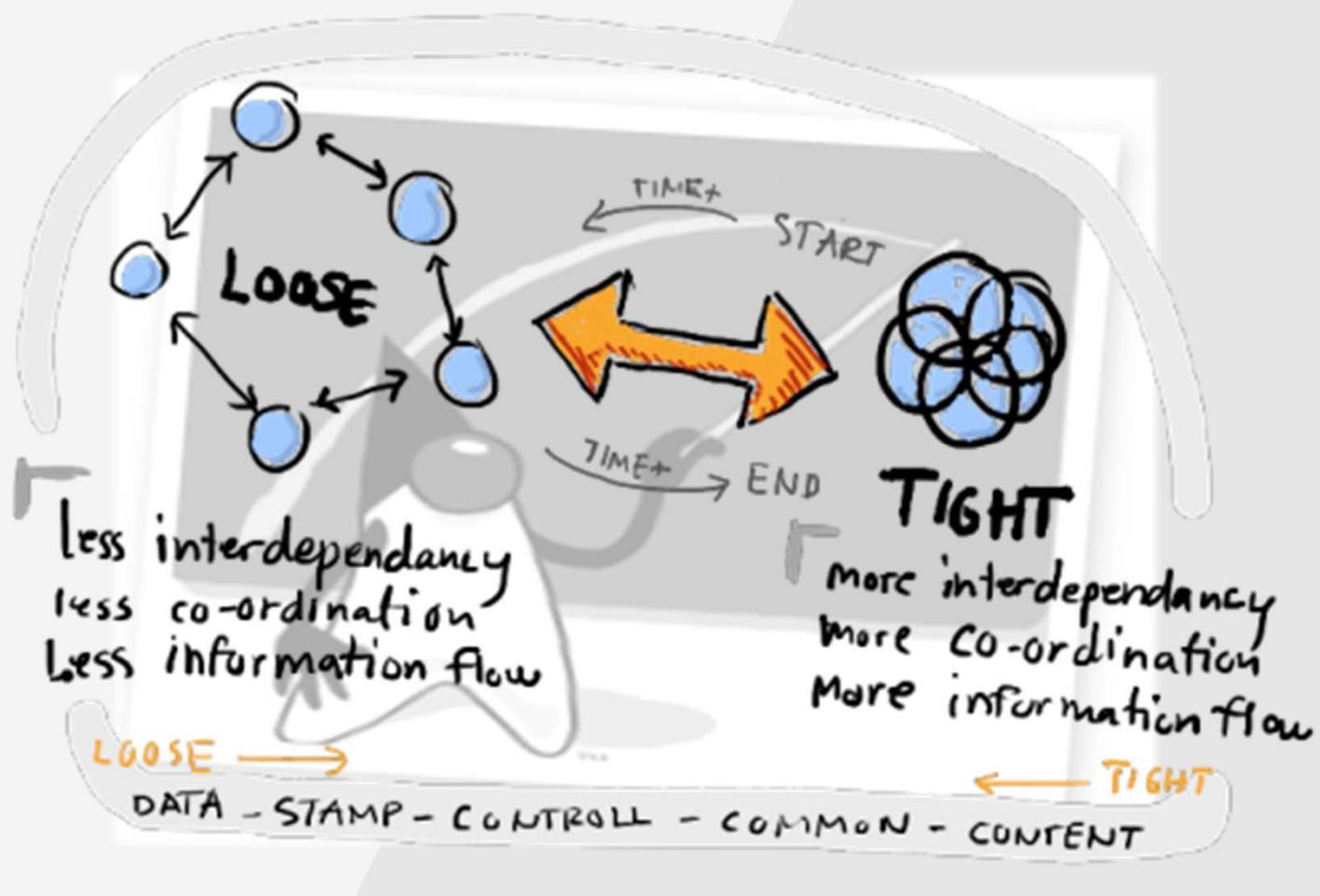
# CONCEPTOS OO



# CONCEPTOS 00



# CONCEPTOS 00



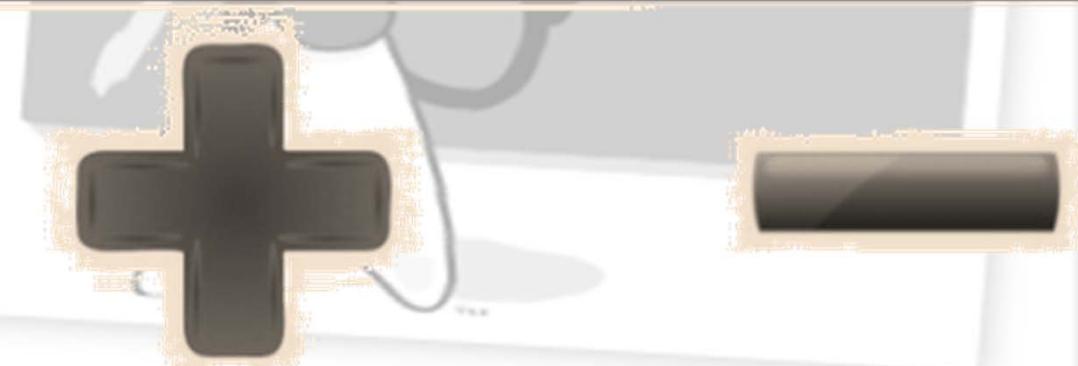
# CONCEPTOS 00

10NE  
1.thing

# CONCEPTOS 00

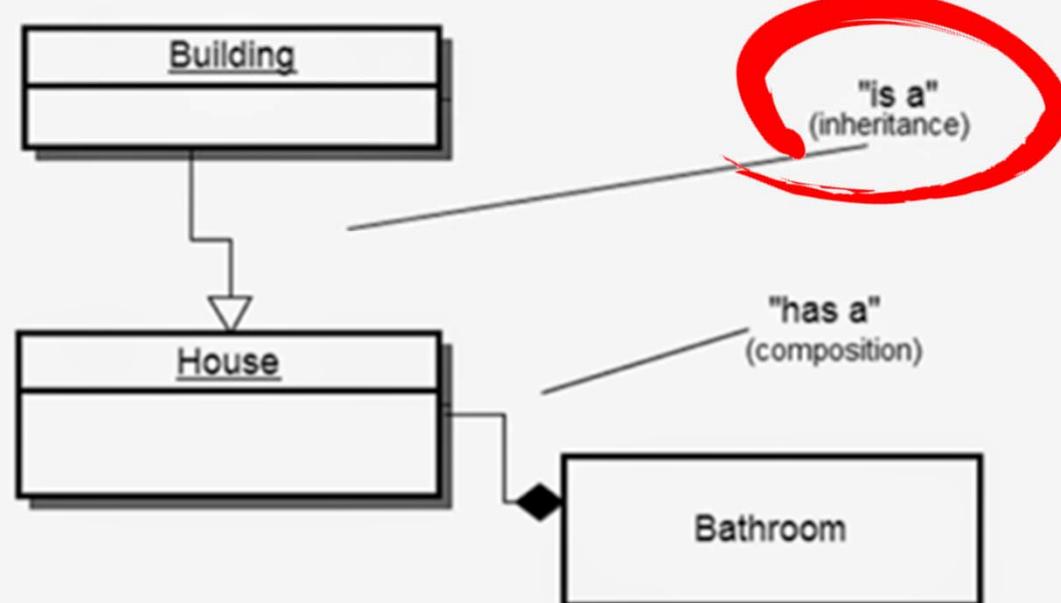
Alta cohesión

Bajo acoplamiento



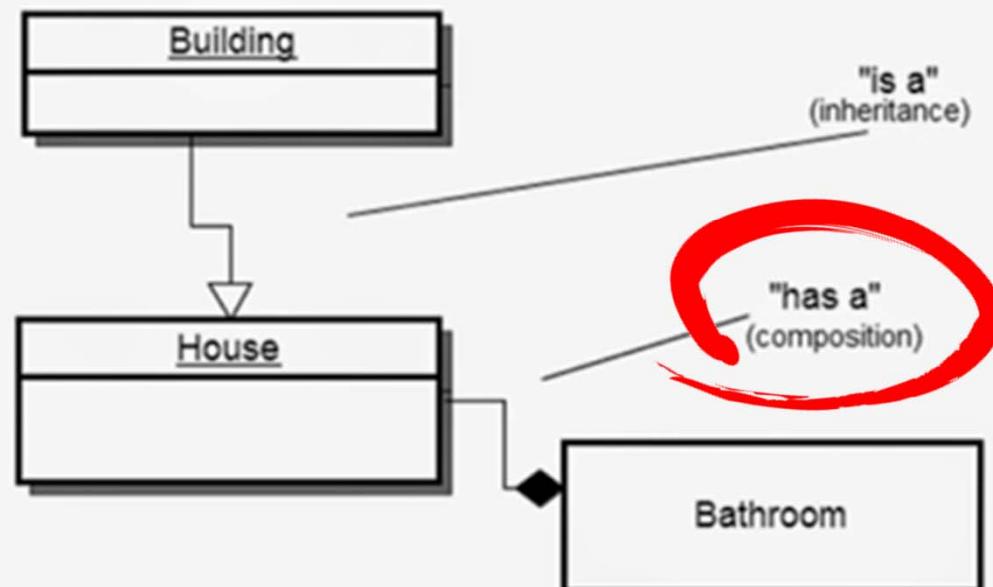
# CONCEPTOS 00

## Composition vs. Inheritance



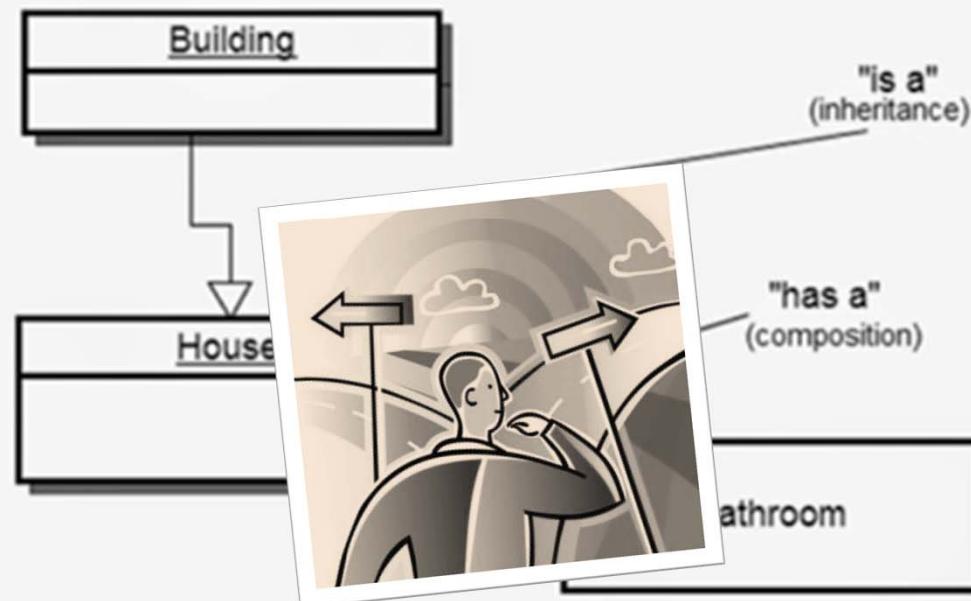
# CONCEPTOS 00

## Composition vs. Inheritance



# CONCEPTOS 00

## Composition vs. Inheritance

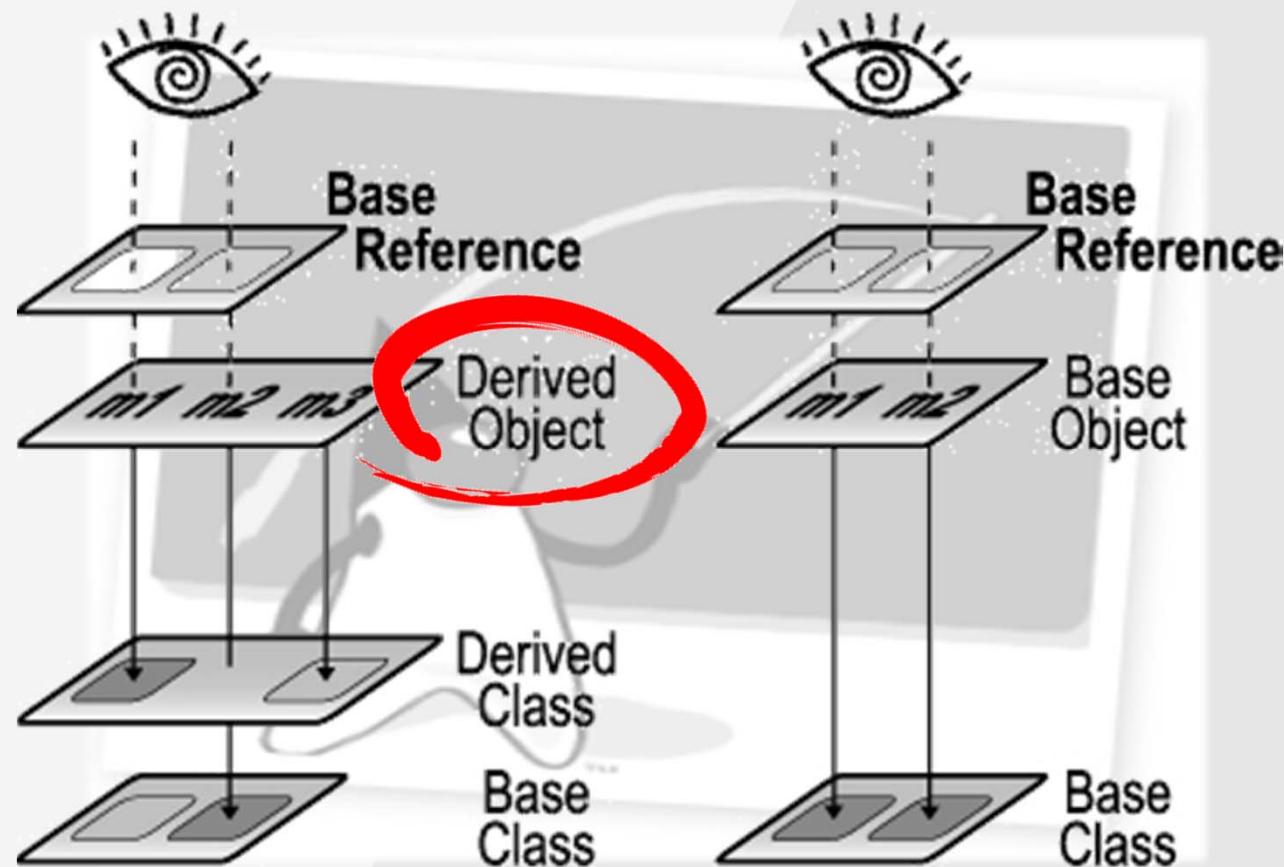


# CONCEPTOS 00

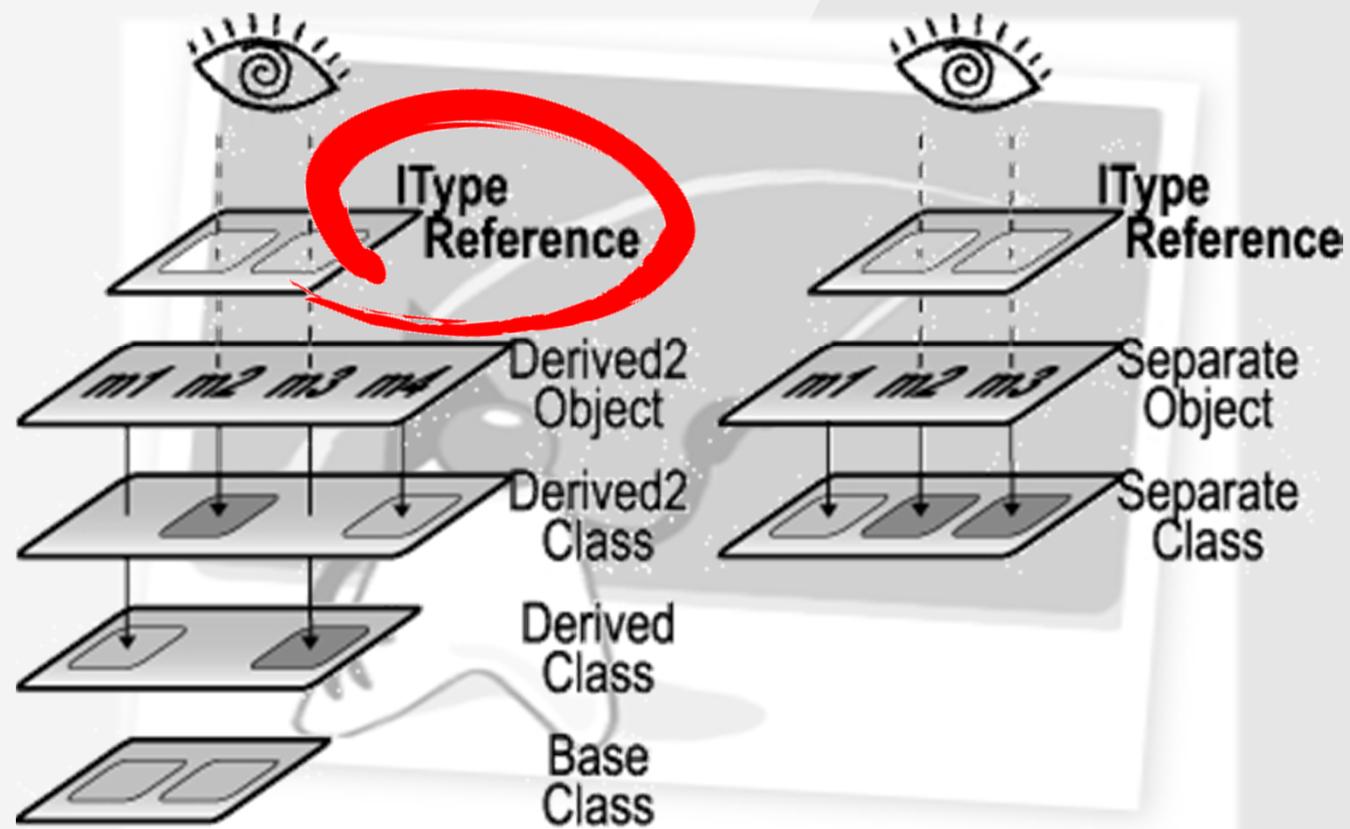
polymorph



# CONCEPTOS OO



# CONCEPTOS OO



# CONCEPTOS 00



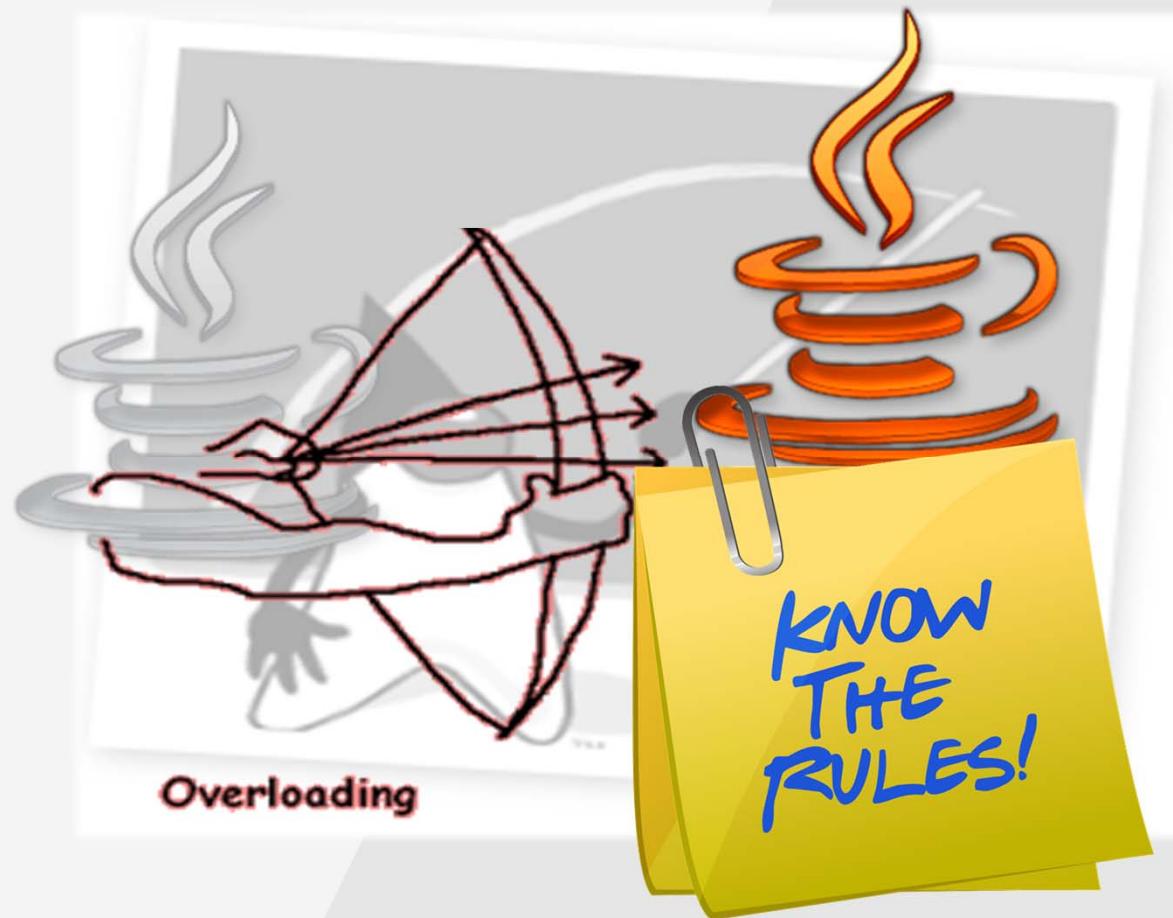
# CONCEPTOS 00

- La lista de argumentos debe coincidir exactamente con el método a sobrescribir. ( Sino estaremos agregando un método sobrecargado )
- El retorno debe ser del mismo tipo que el original o una subclase del mismo.
- El nivel de acceso del método no puede ser más restrictivo.
- Solo pueden reescribirse métodos heredados.
- Se puede lanzar cualquier “unchecked” exception ( RuntimeException ) aunque no estén declaradas en el original.

# CONCEPTOS 00

- No se pueden lanzar nuevas “checked exceptions” salvo que sean subclases de la excepción original lanzada.
- No se puede sobreescibir un método final.
- No se puede sobreescibir un método estático

# CONCEPTOS OO



# CONCEPTOS 00

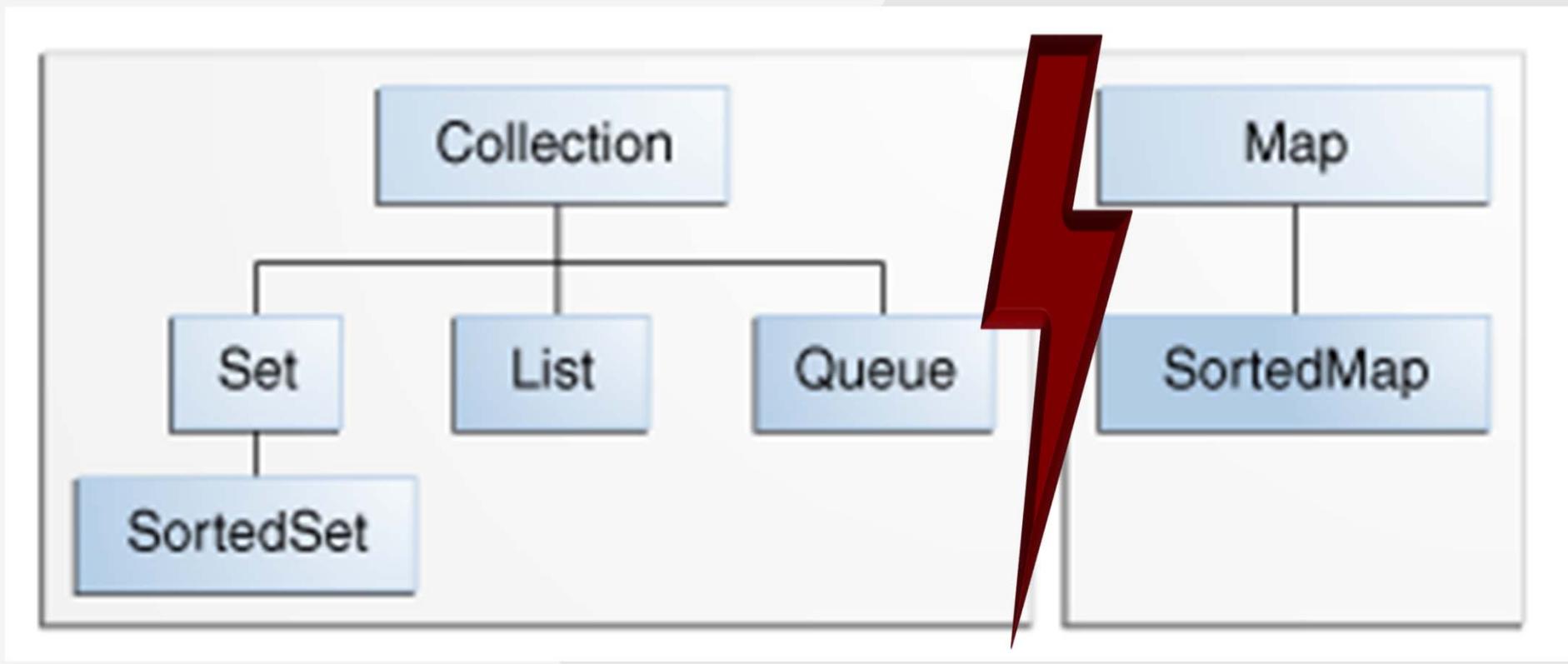
- ➔ Un método sobrecargado debe cambiar la lista de argumentos.
- ➔ Un método sobrecargado puede cambiar el tipo de retorno.
- ➔ Un método sobrecargado puede cambiar el modificador de acceso.
- ➔ Un método sobrecargado puede lanzar nuevas excepciones o ampliar las ya existentes.
- ➔ Un método sobrecargado puede darse a nivel de clase o subclase.



#6

# Colecciones y Genéricos

# COLECCIONES



# COLECCIONES



# COLECCIONES

## <<Interface>> Collection<E>

```
+add( in e:E ):Boolean  
+addAll( in c:Collection<? extends E> ):Boolean  
+clear()  
+contains( in o:Object ):Boolean  
+containsAll( in c:Collection<?> ):Boolean  
+isEmpty():Boolean  
+iterator():Iterator<E>  
+remove( in o:Object ):Boolean  
+removeAll( in c:Collection<?> ):Boolean  
+retainAll( in c:Collection<?> ):Boolean  
+size():Integer  
+toArray():Object[]  
+toArray<T>( in a[0..*]:T ):T[]
```

## <<Interface>> Collection<E>

```
+boolean add( E e )  
+boolean addAll( Collection<? extends E> c )  
+void clear()  
+boolean contains( Object o )  
+boolean containsAll( Collection<?> c )  
+boolean isEmpty()  
+Iterator<E> iterator()  
+boolean remove( Object o )  
+boolean removeAll( Collection<?> c )  
+boolean retainAll( Collection<?> c )  
+int size()  
+Object[] toArray()  
+<T> T[] toArray( T[] a )
```

# COLECCIONES

```
public interface Collection {  
    // Basic Operations  
    int size();  
    boolean isEmpty();  
    boolean contains(Object element);  
    boolean add(Object element);      // Optional  
    boolean remove(Object element);  // Optional  
    Iterator iterator();  
  
    // Bulk Operations  
    boolean containsAll(Collection c);  
    boolean addAll(Collection c);     // Optional  
    boolean removeAll(Collection c); // Optional  
    boolean retainAll(Collection c); // Optional  
    void clear();                  // Optional  
  
    // Array Operations  
    Object[] toArray();  
    Object[] toArray(Object a[]);  
}
```



# COLECCIONES

<<Interface>>  
Collection<E>

```
+add( in E):Boolean  
+addAll( in c:Collection<E>):Boolean  
+clear()  
+contains( in o:Object):Boolean  
+containsAll( in c:Collection<?> ):Boolean  
+isEmpty():Boolean  
+iterator():Iterator  
+remove( in o:Object ):Boolean  
+removeAll( in c:Collection<?> ):Boolean  
+retainAll( in c:Collection<?> ):Boolean  
+size():Integer  
+toArray():Object[]..  
+toArray<T>( in a[0..*]:T ):T[]..
```

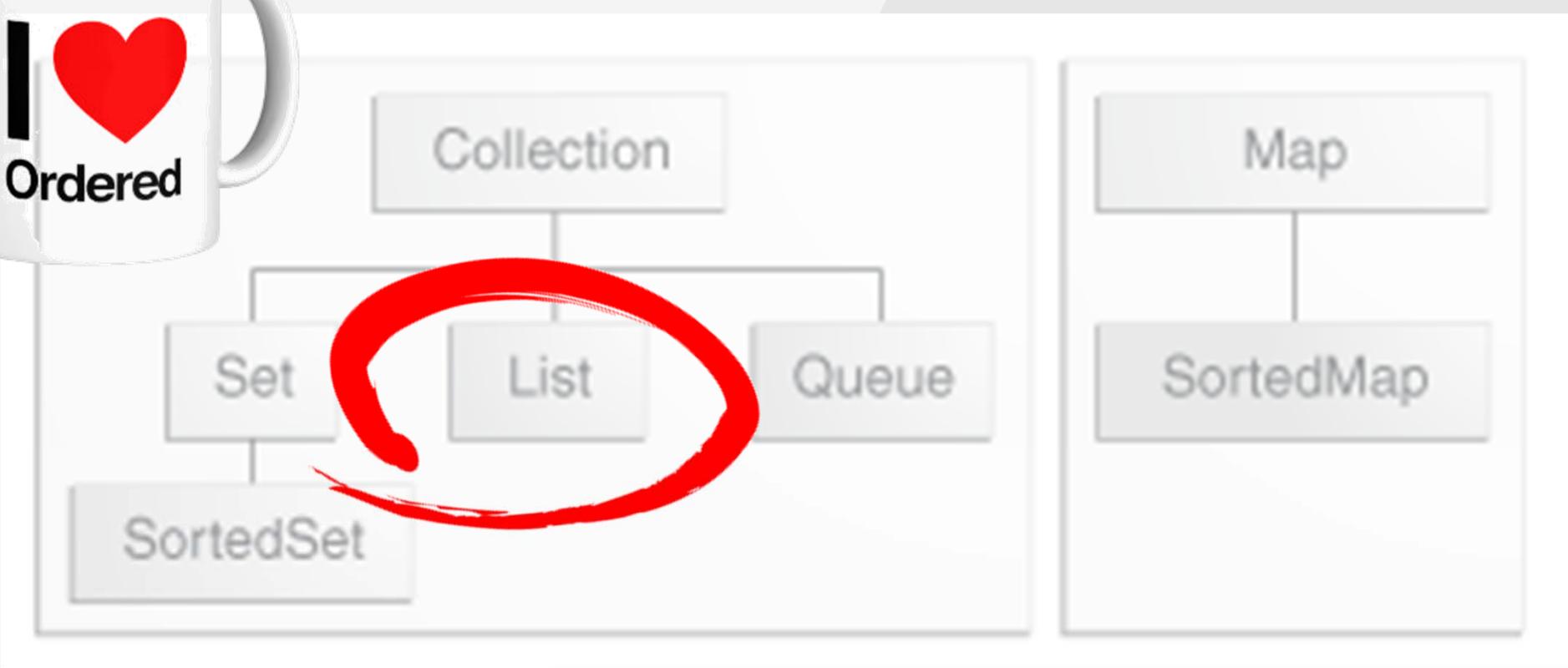
<<Interface>>  
Collection<E>

```
+boolean add( E e )  
+boolean addAll( Collection<? extends E> c )  
+void clear()  
+boolean contains( Object o )  
+boolean containsAll( Collection<?> c )  
+boolean isEmpty()  
+Iterator<E> iterator()  
+boolean remove( Object o )  
+void removeAll( Collection<?> c )  
+boolean retainAll( Collection<?> c )  
+int size()  
+Object[] toArray()  
+T[] toArray( T[] a )
```

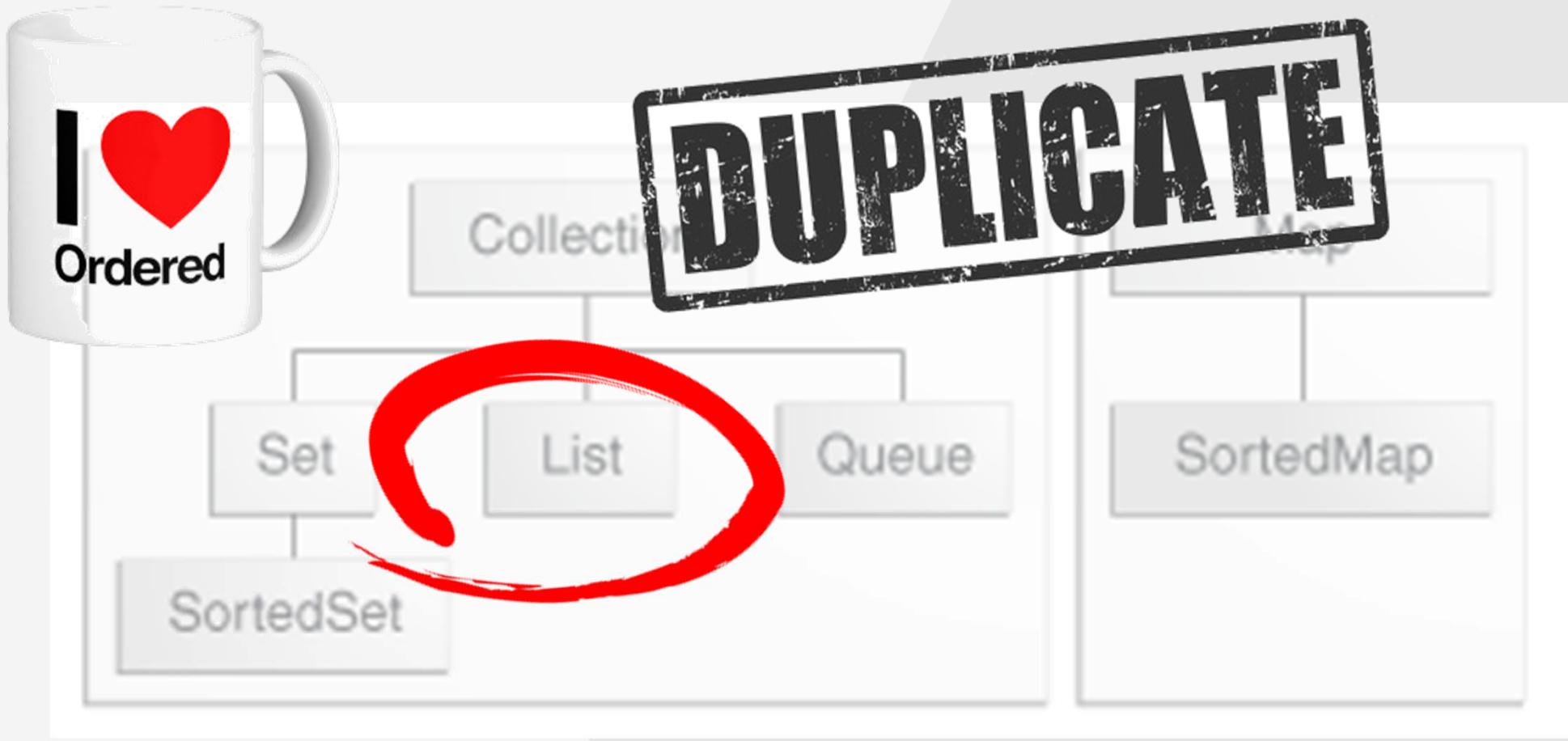
# COLECCIONES



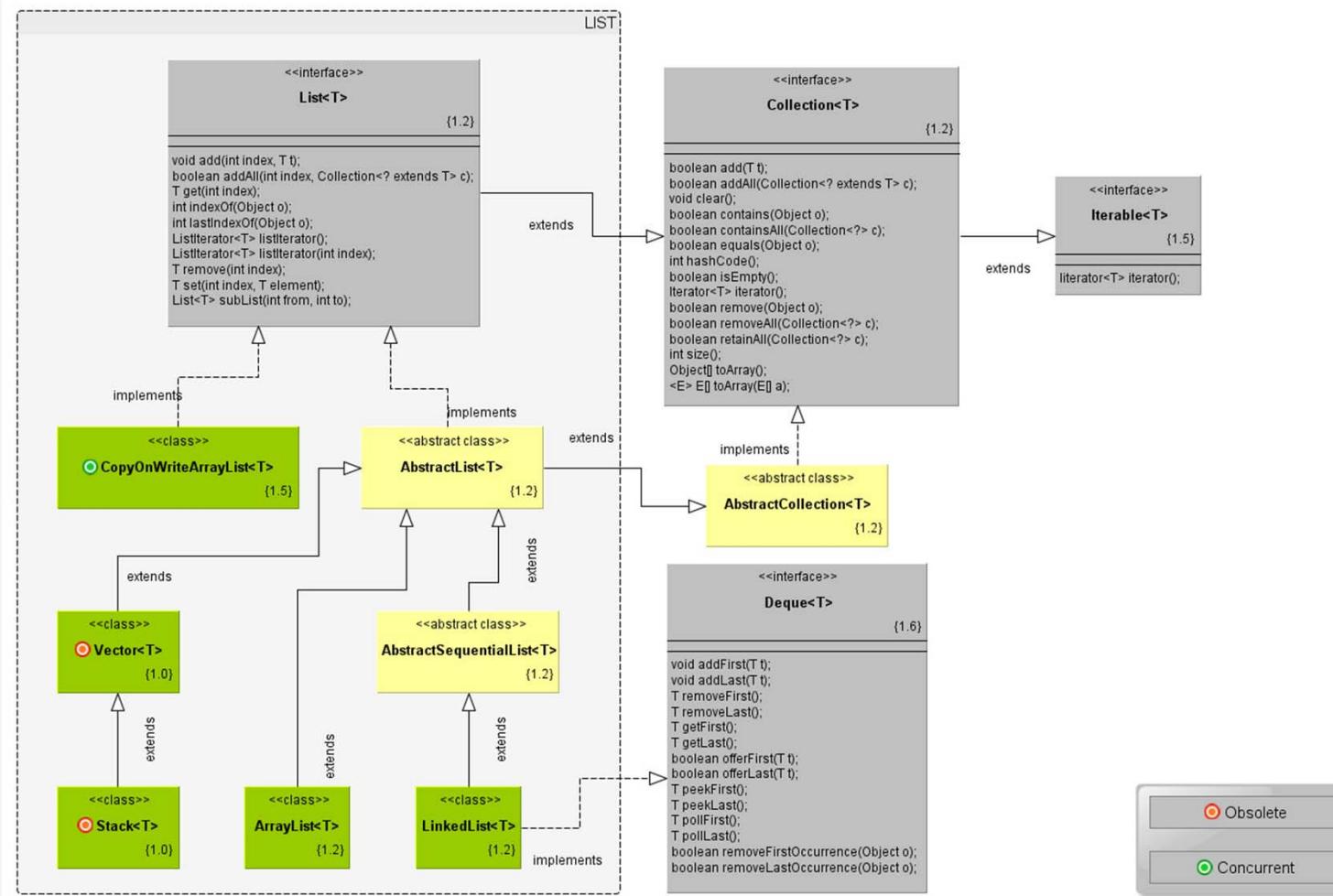
# COLECCIONES



# COLECCIONES



# COLECCIONES



# COLECCIONES

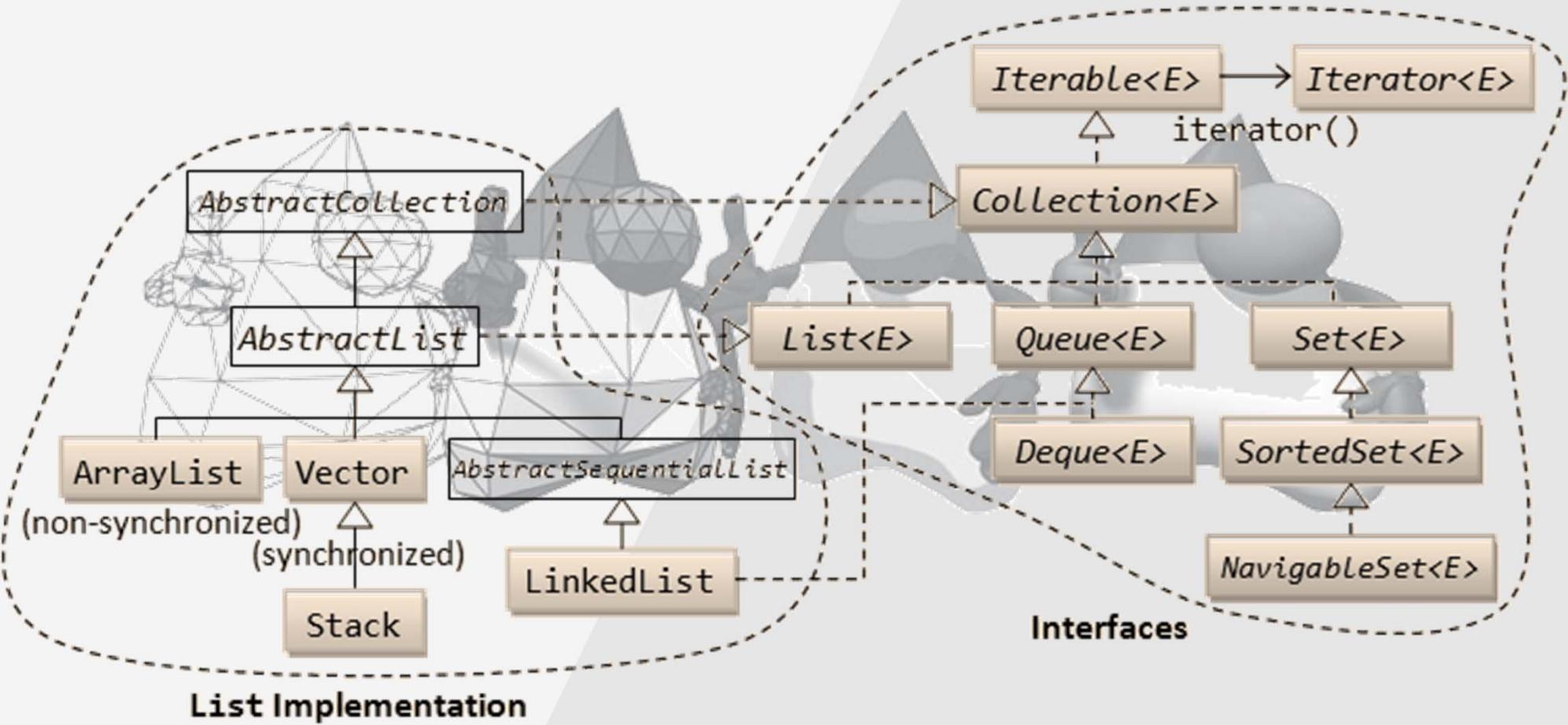
```
public interface List extends Collection {  
    // Positional Access  
    Object get(int index);  
    Object set(int index, Object element); // Optional  
    void add(int index, Object element); // Optional  
    Object remove(int index); // Optional  
    abstract boolean addAll(int index, Collection c);  
                                // Optional  
  
    // Search  
    int indexOf(Object o);  
    int lastIndexOf(Object o);  
  
    // Iteration  
    ListIterator listIterator();  
    ListIterator listIterator(int index);  
  
    // Range-view  
    List subList(int from, int to);  
}
```

# COLECCIONES

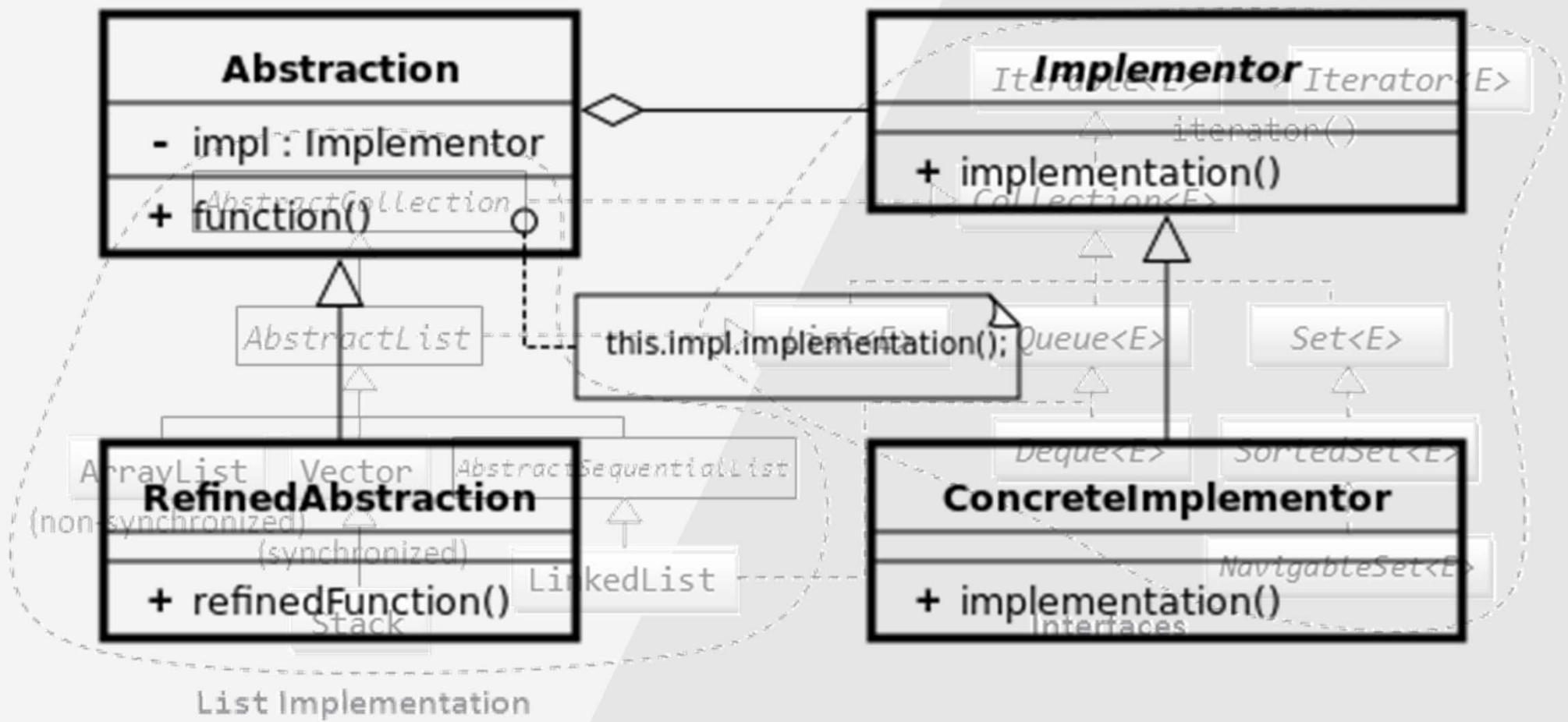
```
public interface ListIterator extends Iterator {  
    boolean hasNext();  
    Object next();  
  
    boolean hasPrevious();  
    Object previous();  
  
    int nextIndex();  
    int previousIndex();  
  
    void remove();          // Optional  
    void set(Object o);    // Optional  
    void add(Object o);    // Optional  
}
```



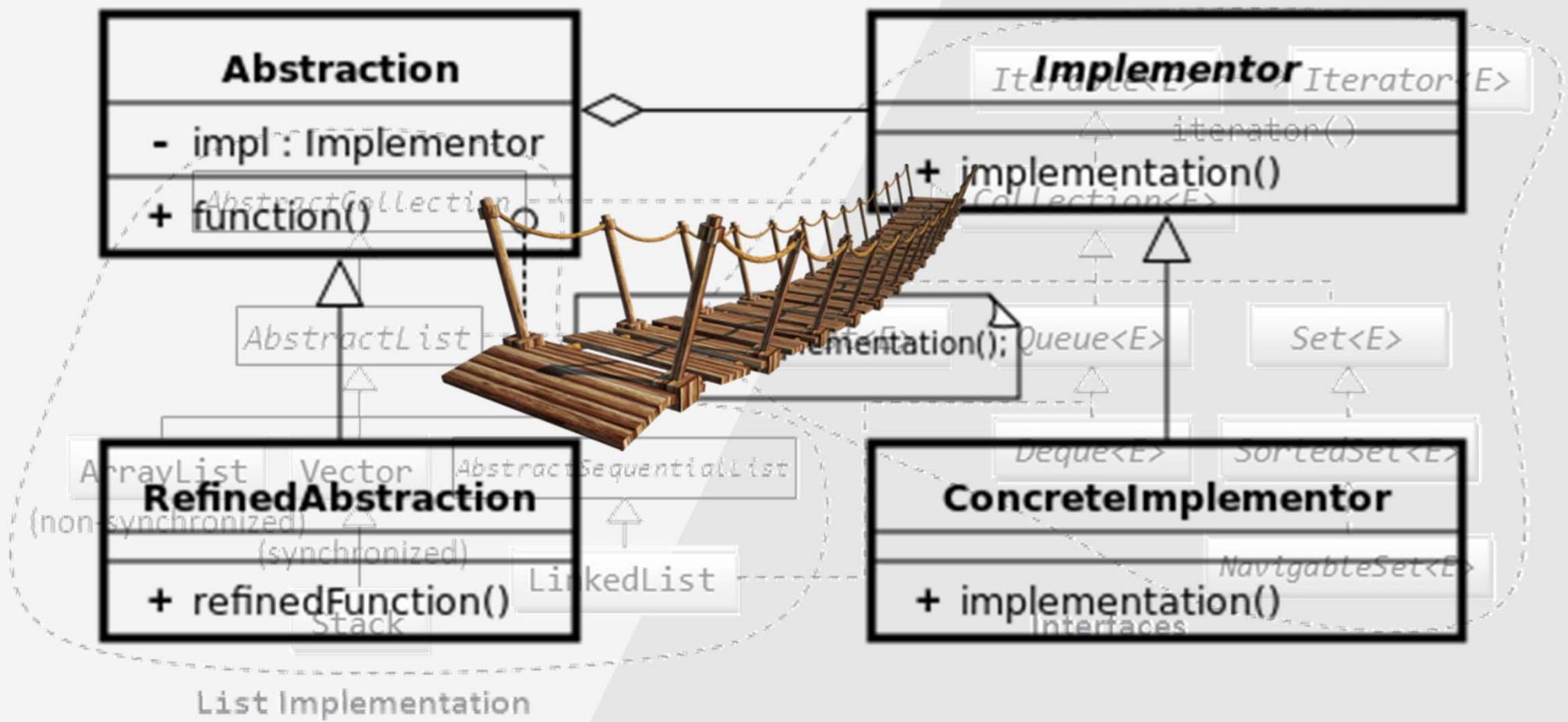
# COLECCIONES



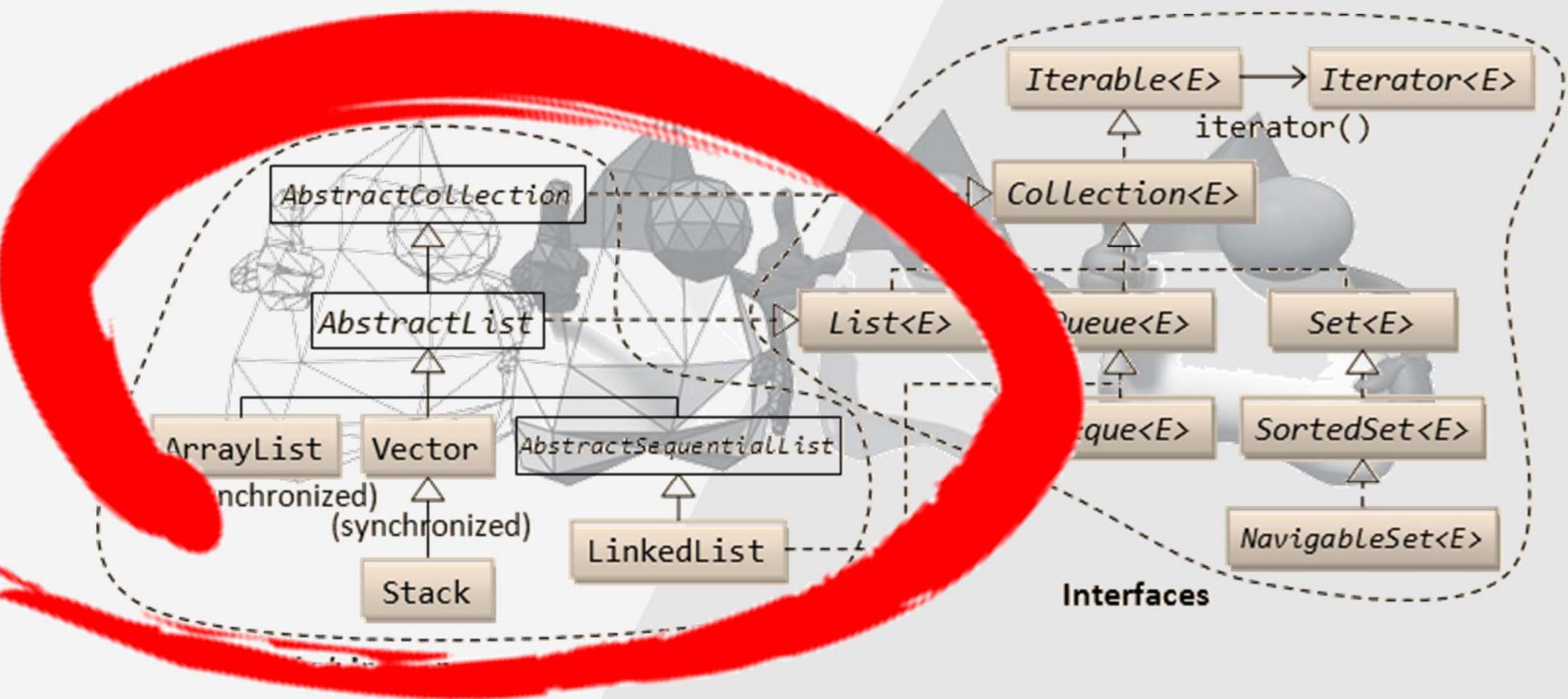
# COLECCIONES



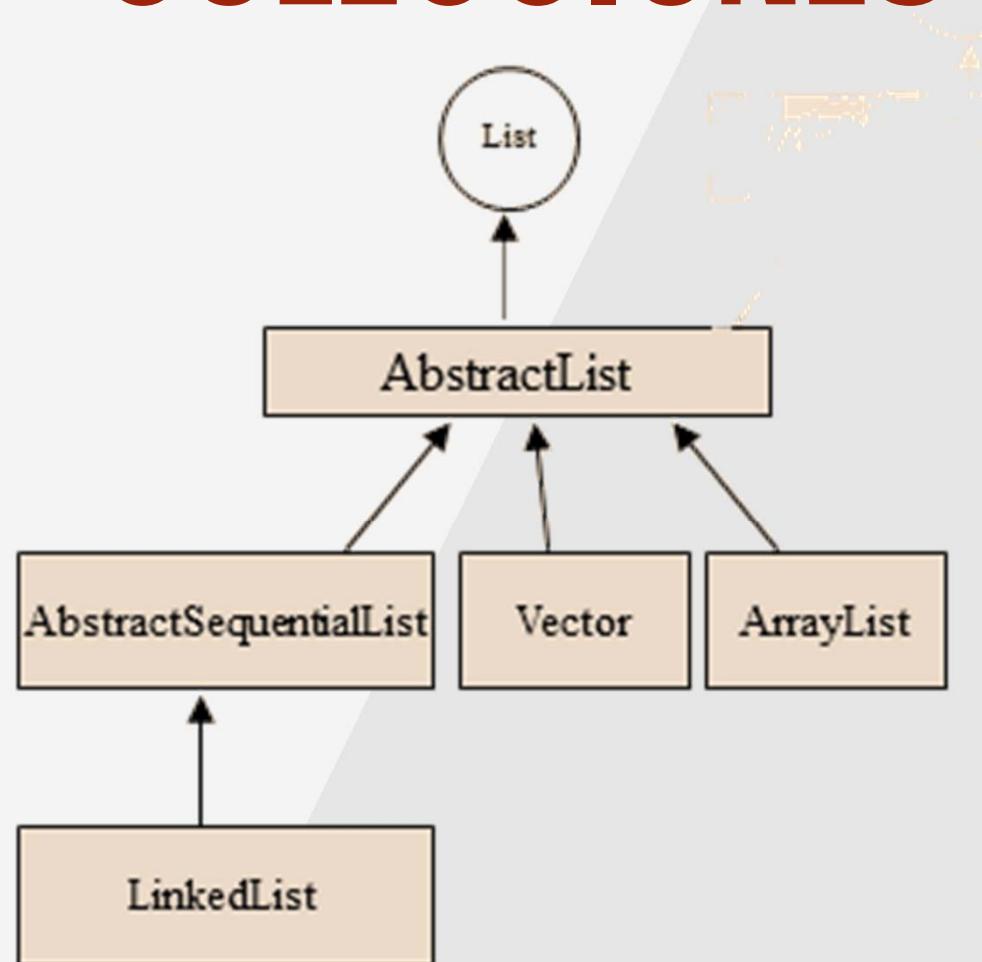
# COLECCIONES



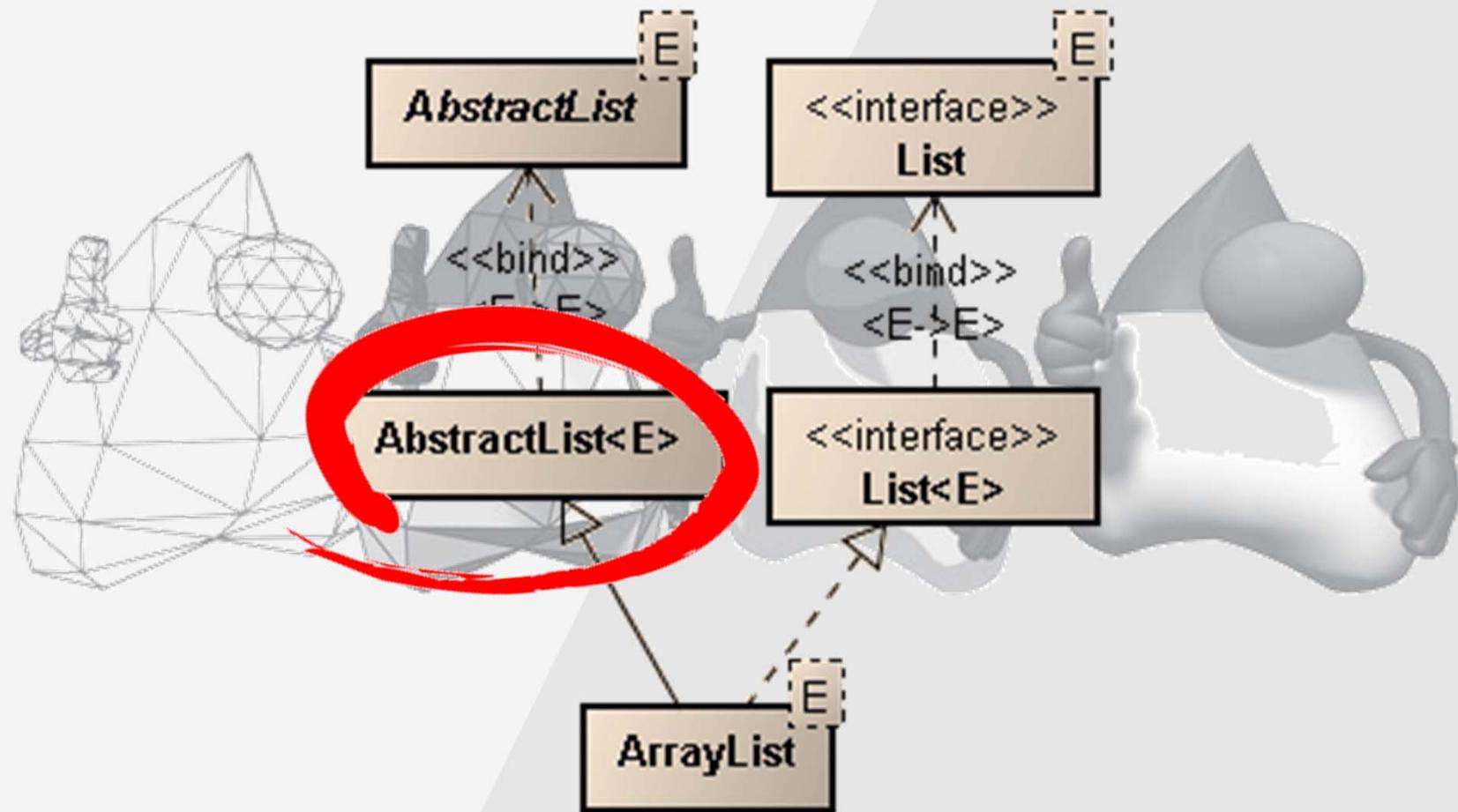
# COLECCIONES



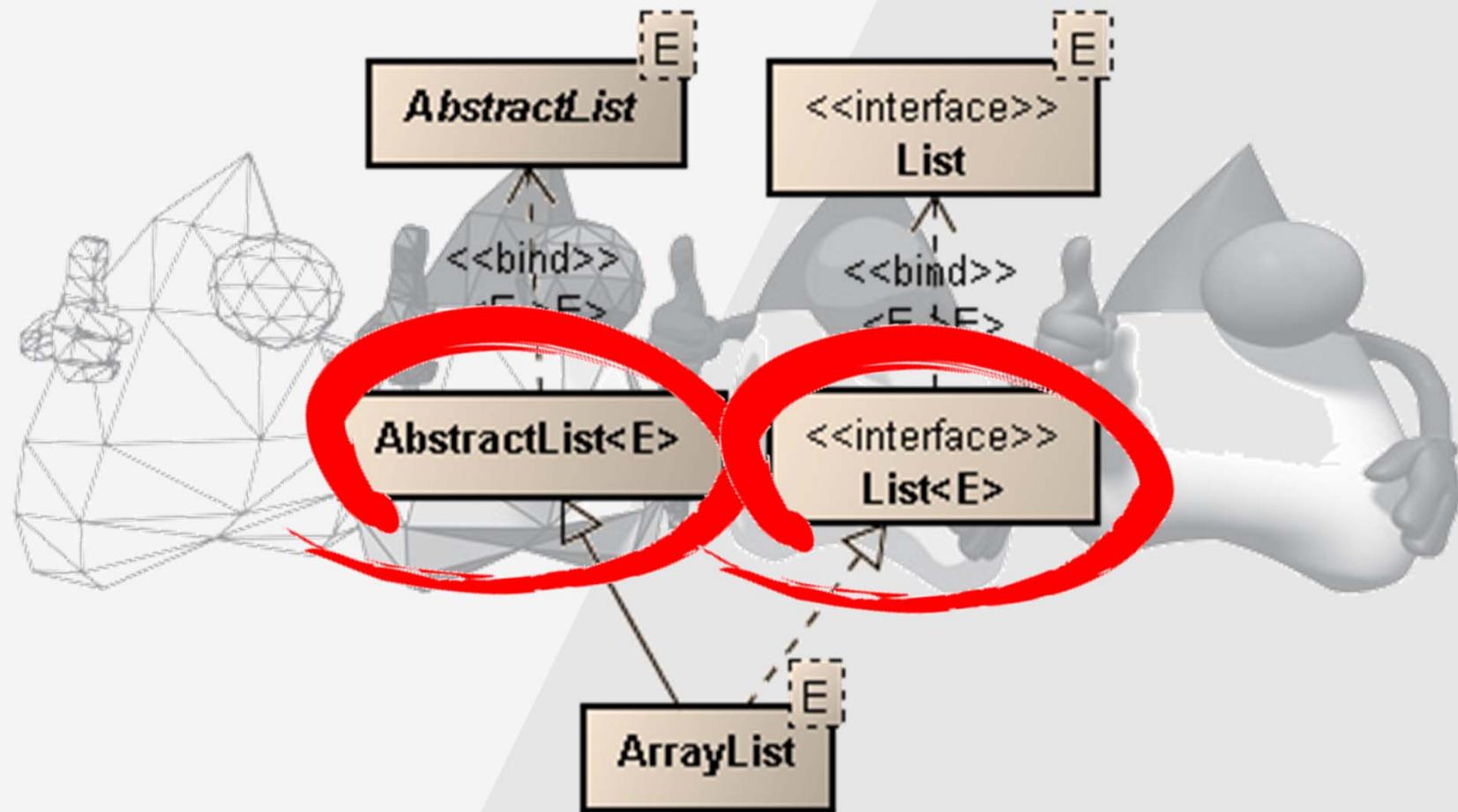
# COLECCIONES



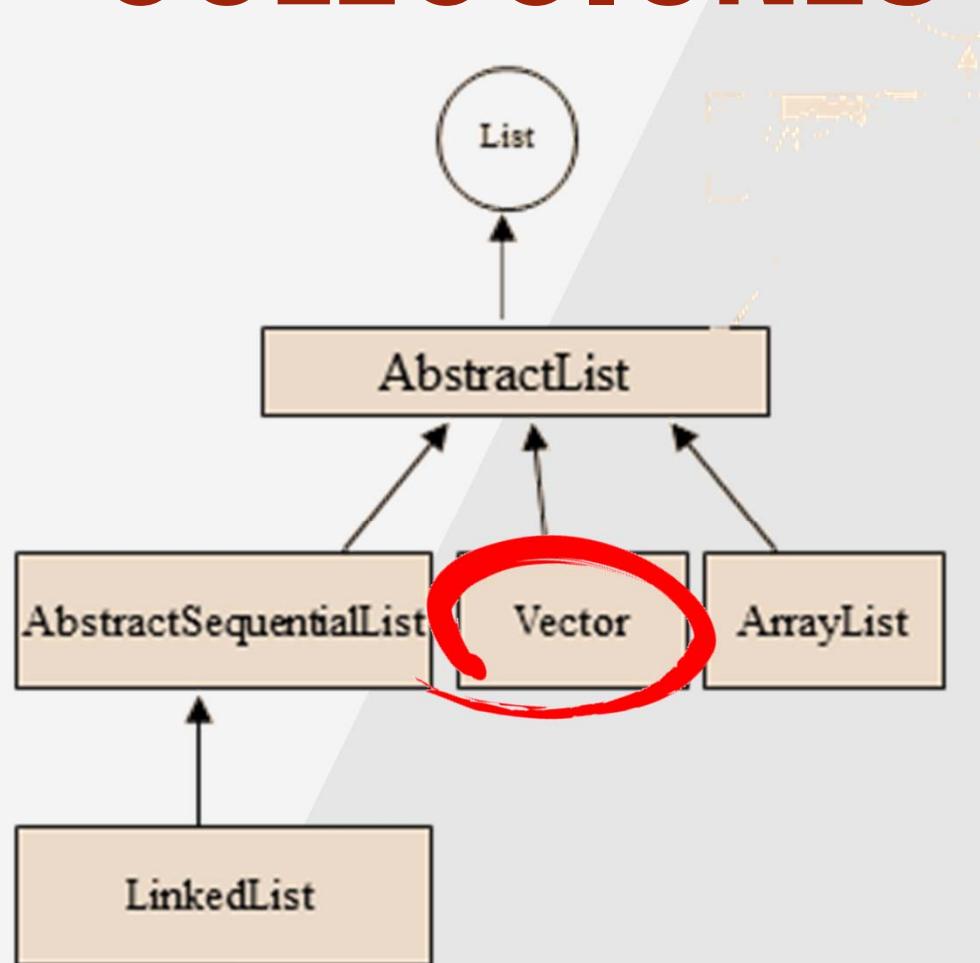
# COLECCIONES



# COLECCIONES



# COLECCIONES



# COLECCIONES

## elementAt

```
public E elementAt(int index)
```

Returns the component at the specified index.

This method is ic

Parameters:

index - an in

Returns:

the compone

Throws:

ArrayIndexC

## removeElementAt

```
public void removeElementAt(int index)
```

Deletes the component at the specified index. The index is shifted downward to have an index of 1.

The index must be a value greater than or

This method is identical in functionality to the remove(int) method. This method returns the old value that was stored at the specified index.

Parameters:

index - the index of the object to remo

Throws:

ArrayIndexOutOfBoundsException - i

## insertElementAt

```
public void insertElementAt(E obj,  
                           int index)
```

Inserts the specified object as a component in this vector at the specified index. Each component in this vector with an index greater or equal to the specified index is shifted upward to have an index one greater than the value it had previously.

The index must be a value greater than or equal to 0 and less than or equal to the current size of the vector. (If the index is equal to the current size of the vector, the new element is appended to the Vector.)

This method is identical in functionality to the add(int, E) method (which is part of the List interface). Note that the add method reverses the order of the parameters, to more closely match array usage.

Parameters:

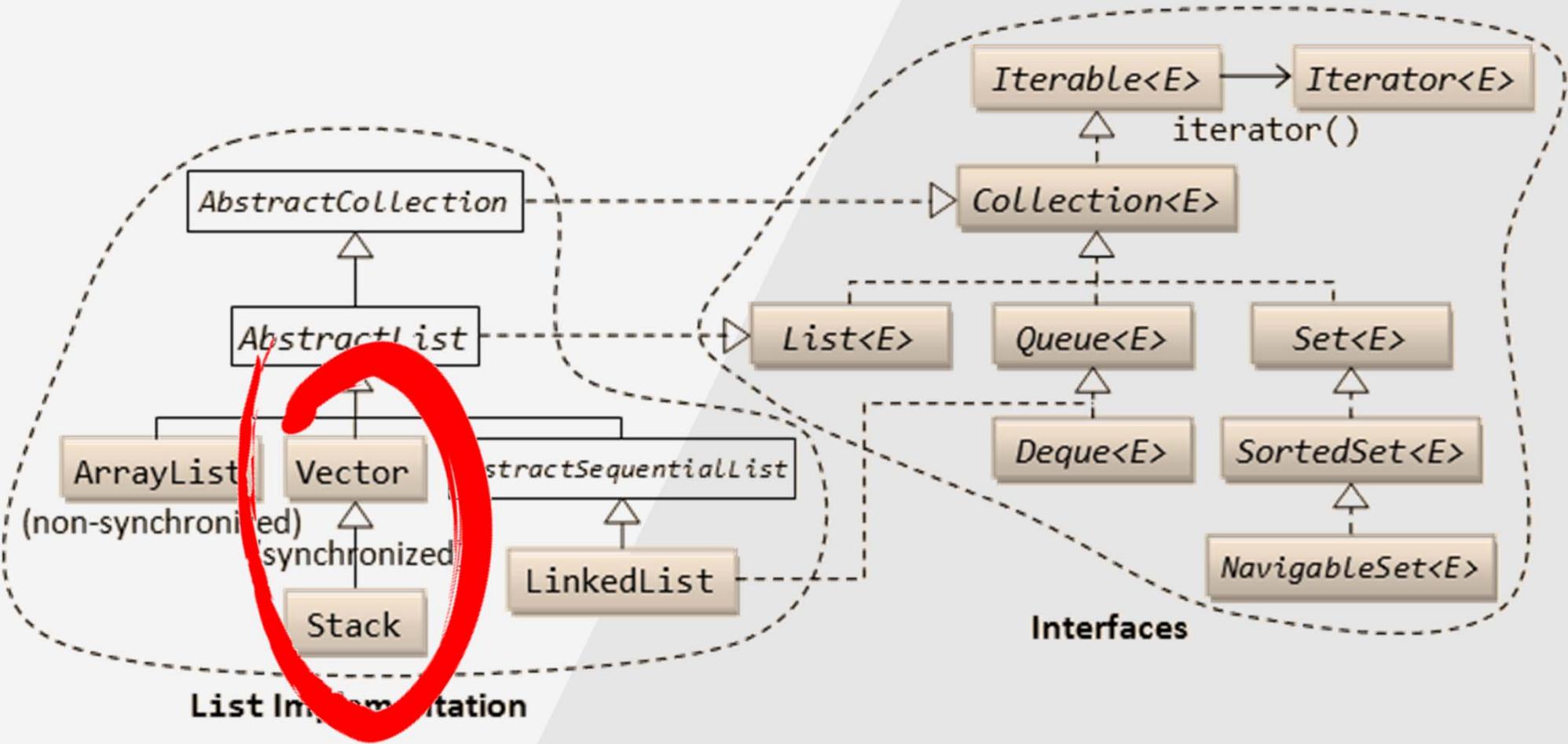
obj - the component to insert

index - where to insert the new component

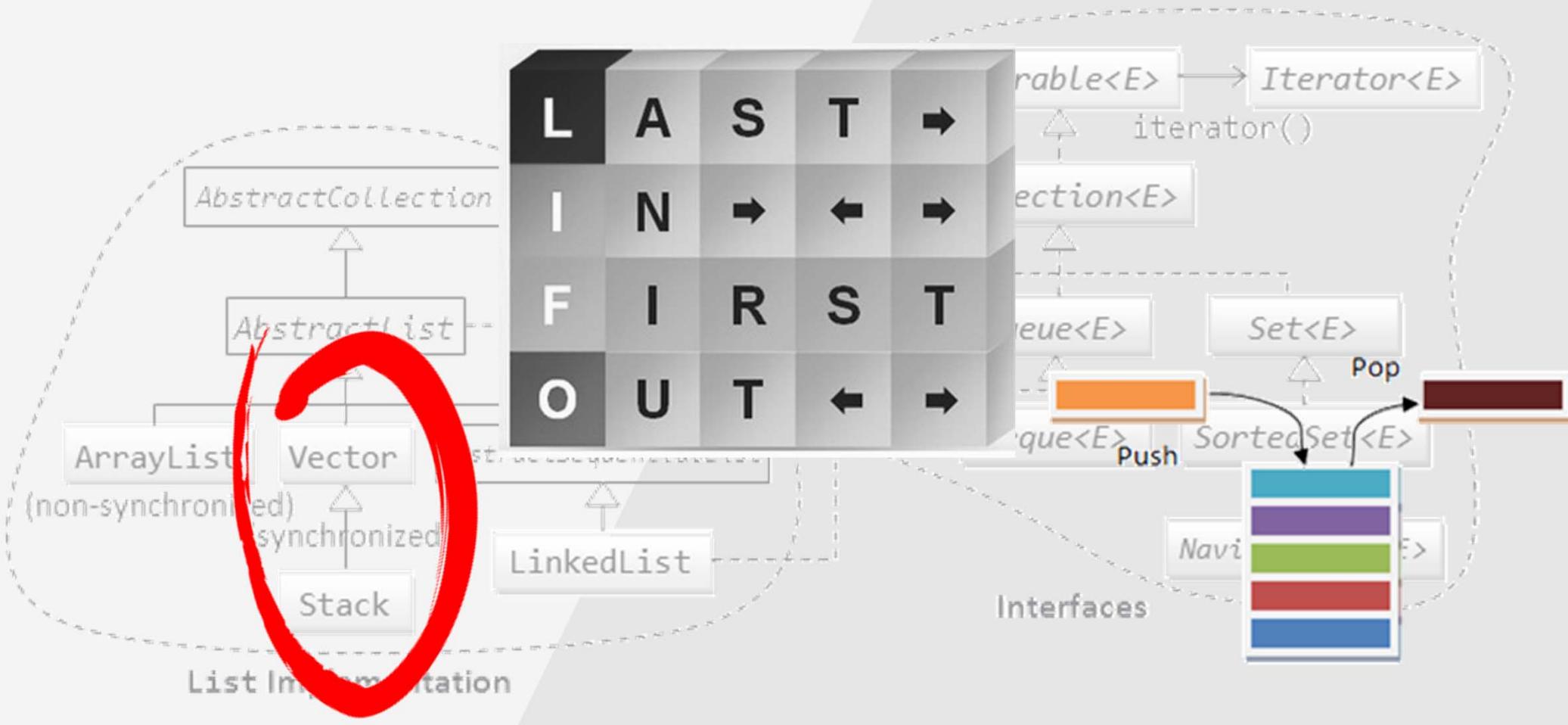
Throws:

ArrayIndexOutOfBoundsException - if the index is out of range (index < 0 || index > size())

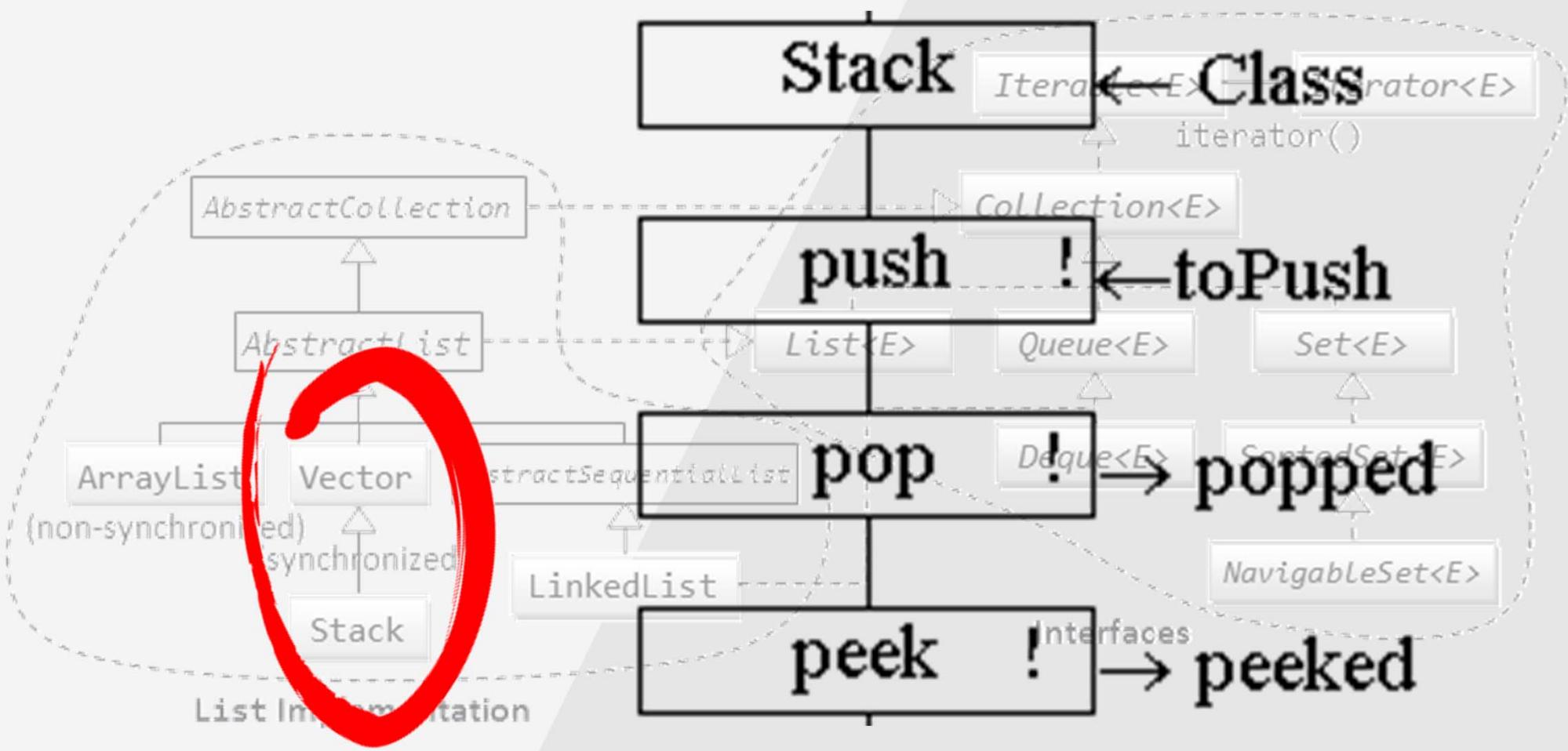
# COLECCIONES



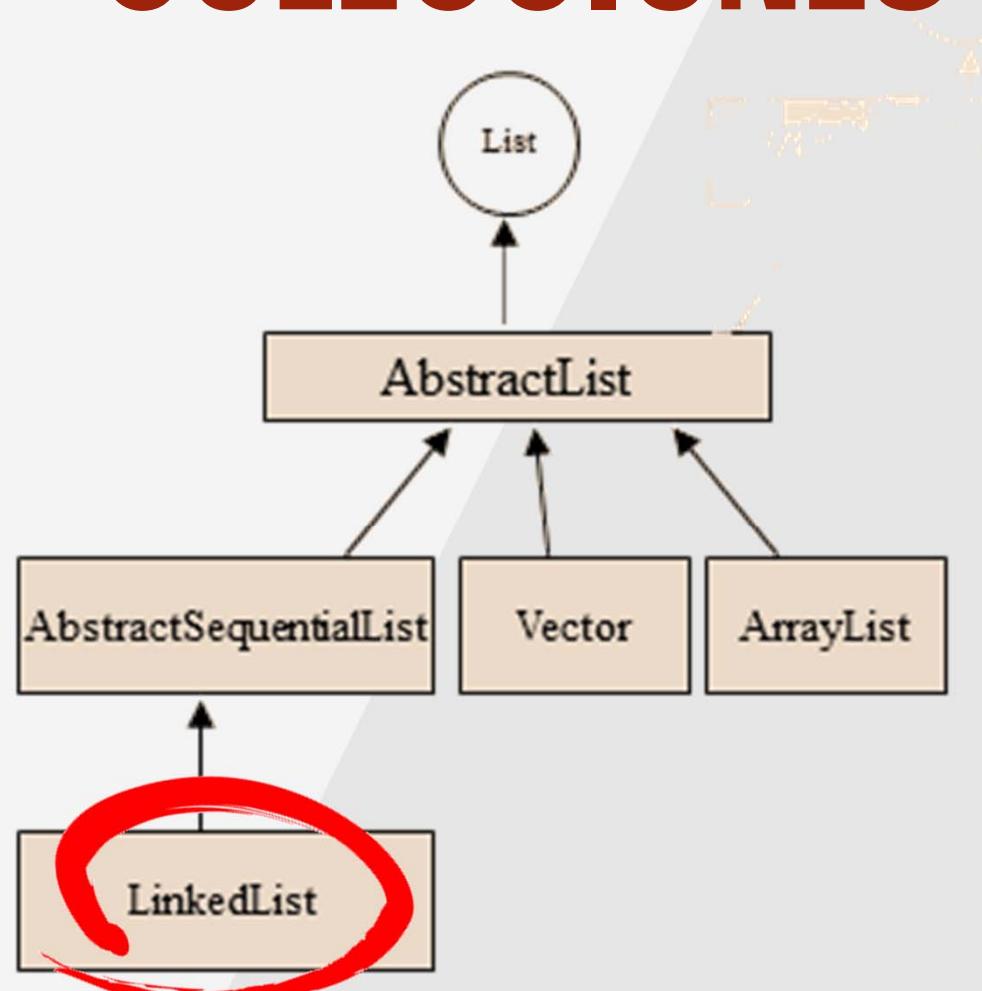
# COLECCIONES



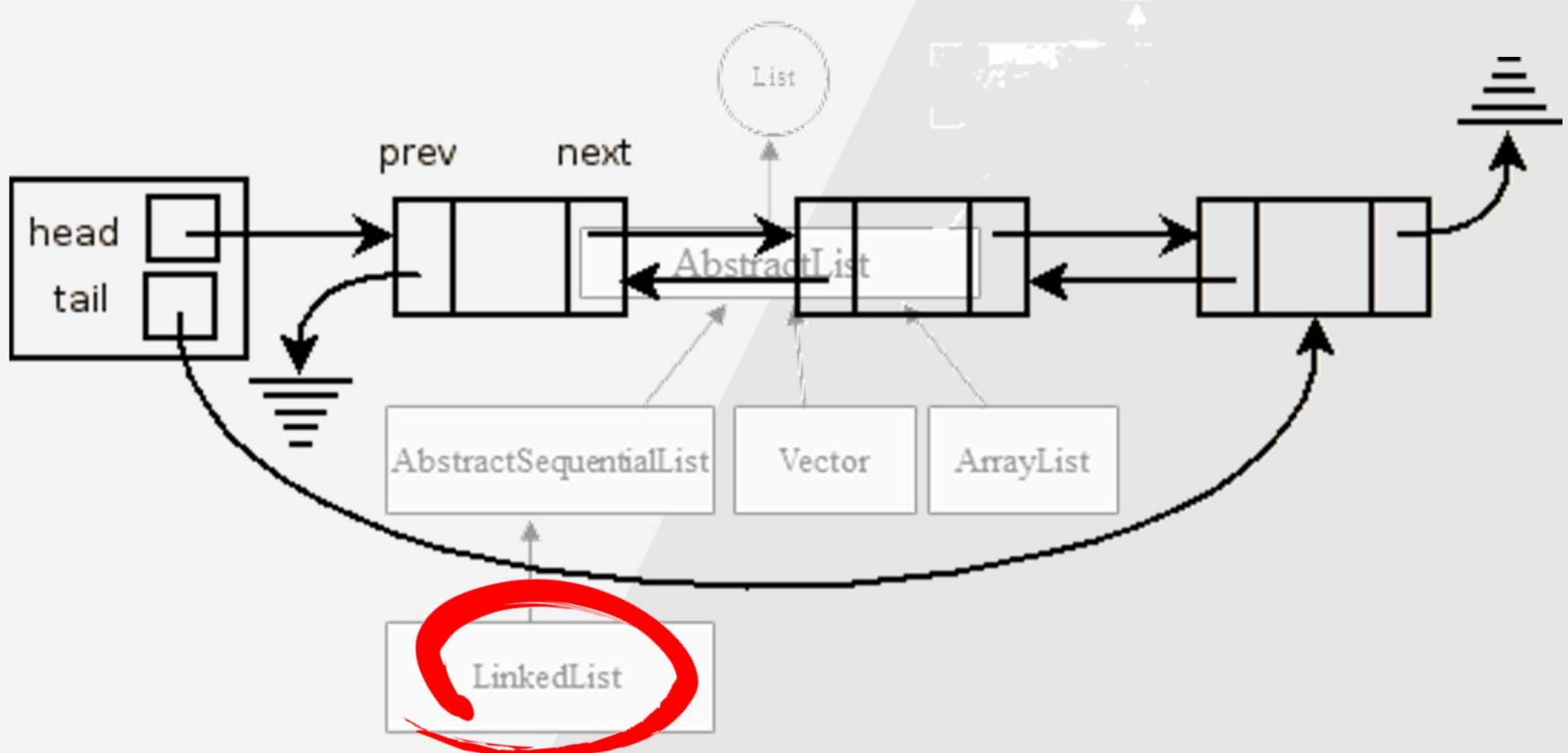
# COLECCIONES



# COLECCIONES



# COLECCIONES



# COLECCIONES

## getFirst

```
public E getFirst()
```

Returns the first element in this list.

### Specified by:

getFirst in interface Deque<E>

### Returns:

the first element in this list

### Throws:

NoSuchElementException - if this list is empty

## getLast

```
public E getLast()
```

Returns the last element in this list.

### Specified by:

getLast in interface Deque<E>

### Returns:

the last element in this list

### Throws:

NoSuchElementException - if this list is empty

## removeLast

```
public E removeLast()
```

Removes and returns the last element from this list.

### Specified by:

removeLast in interface Deque<E>

### Returns:

the last element from this list

### Throws:

NoSuchElementException - if this list is empty

## addLast

```
public void addLast(E e)
```

Appends the specified element to the end of this list.

This method is equivalent to add(E).

### Specified by:

addLast in interface Deque<E>

### Parameters:

e - the element to add

## addFirst

```
public void addFirst(E e)
```

Inserts the specified element at the beginning of this list.

### Specified by:

addFirst in interface Deque<E>

### Parameters:

e - the element to add

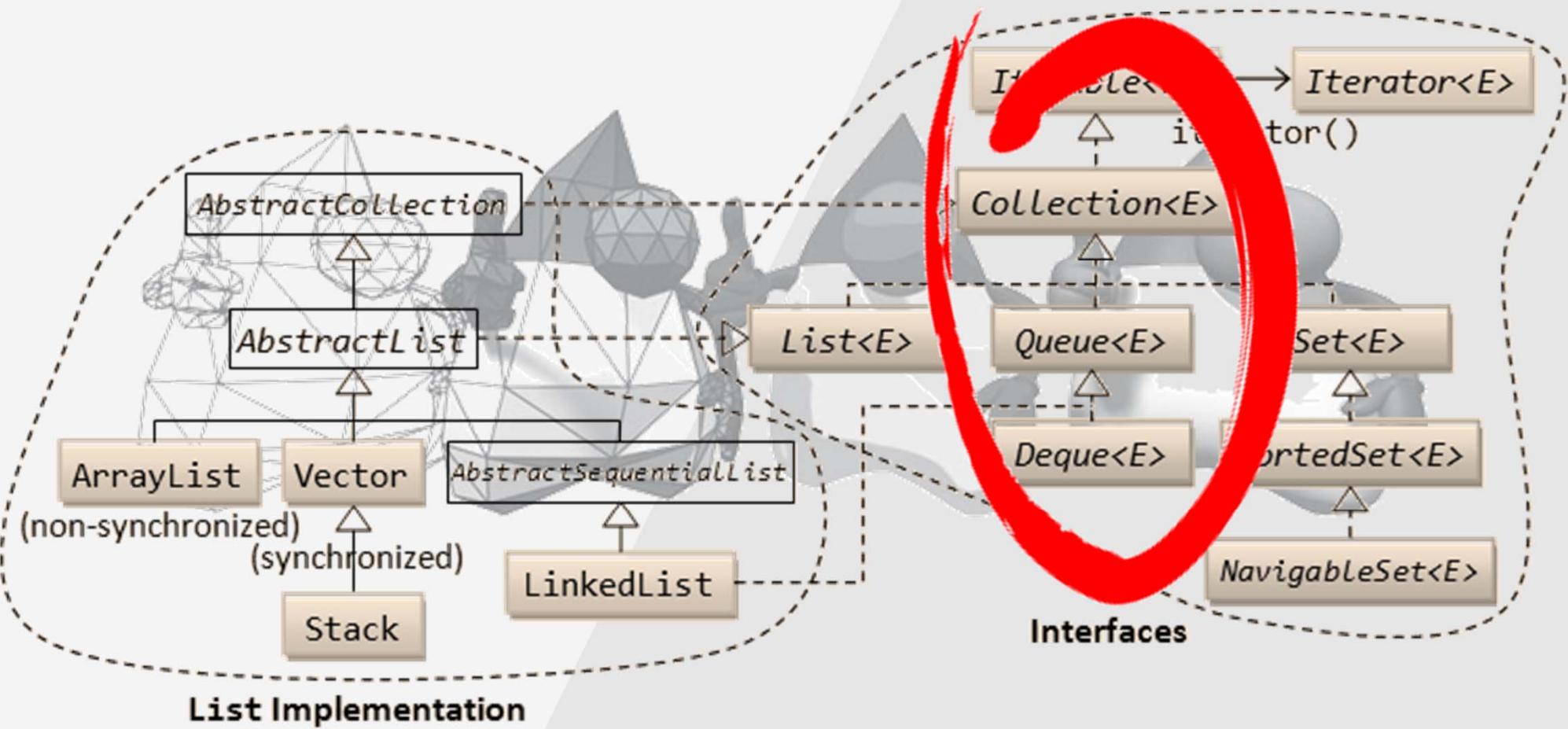
AbstractSeq

LinkedList

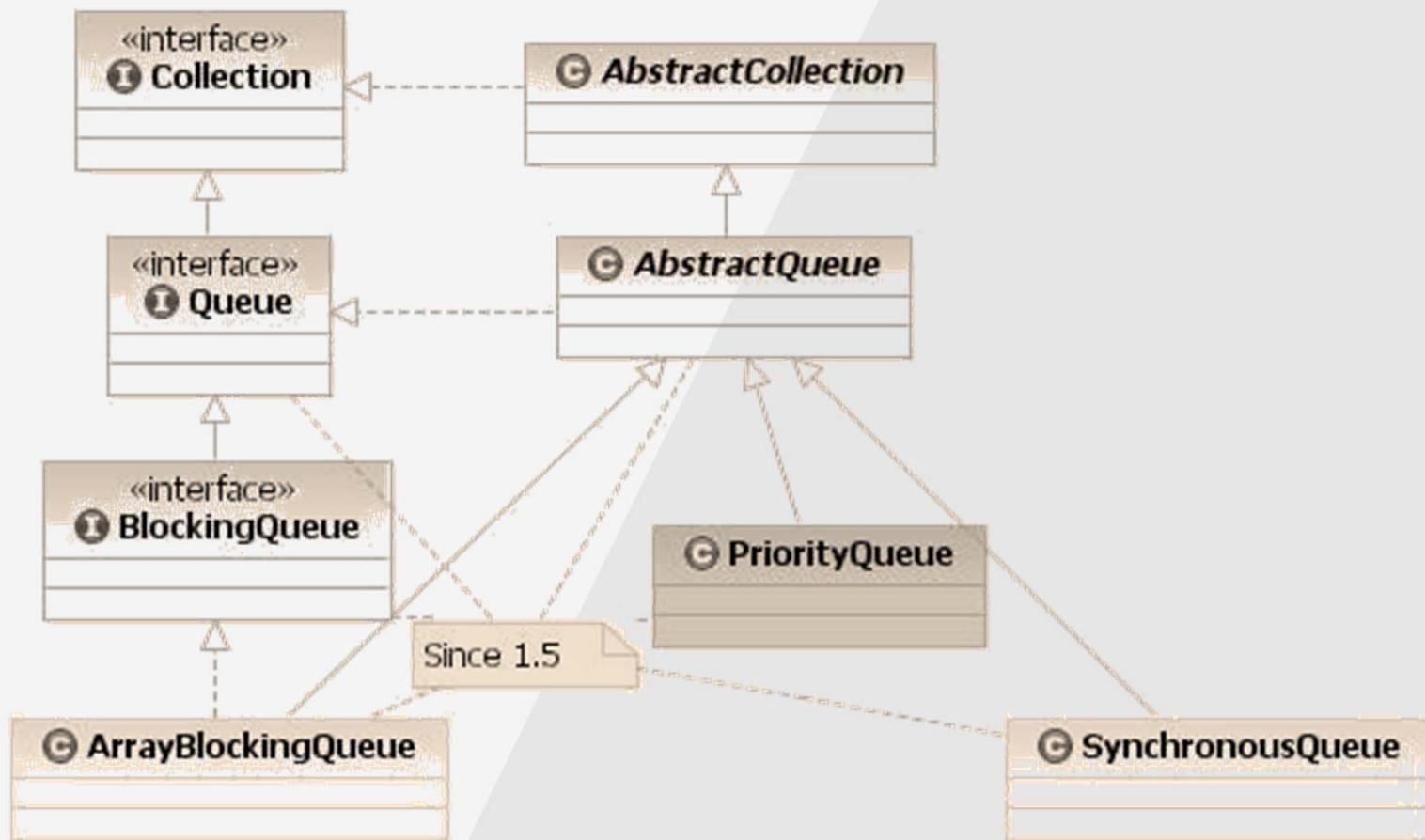
# COLECCIONES



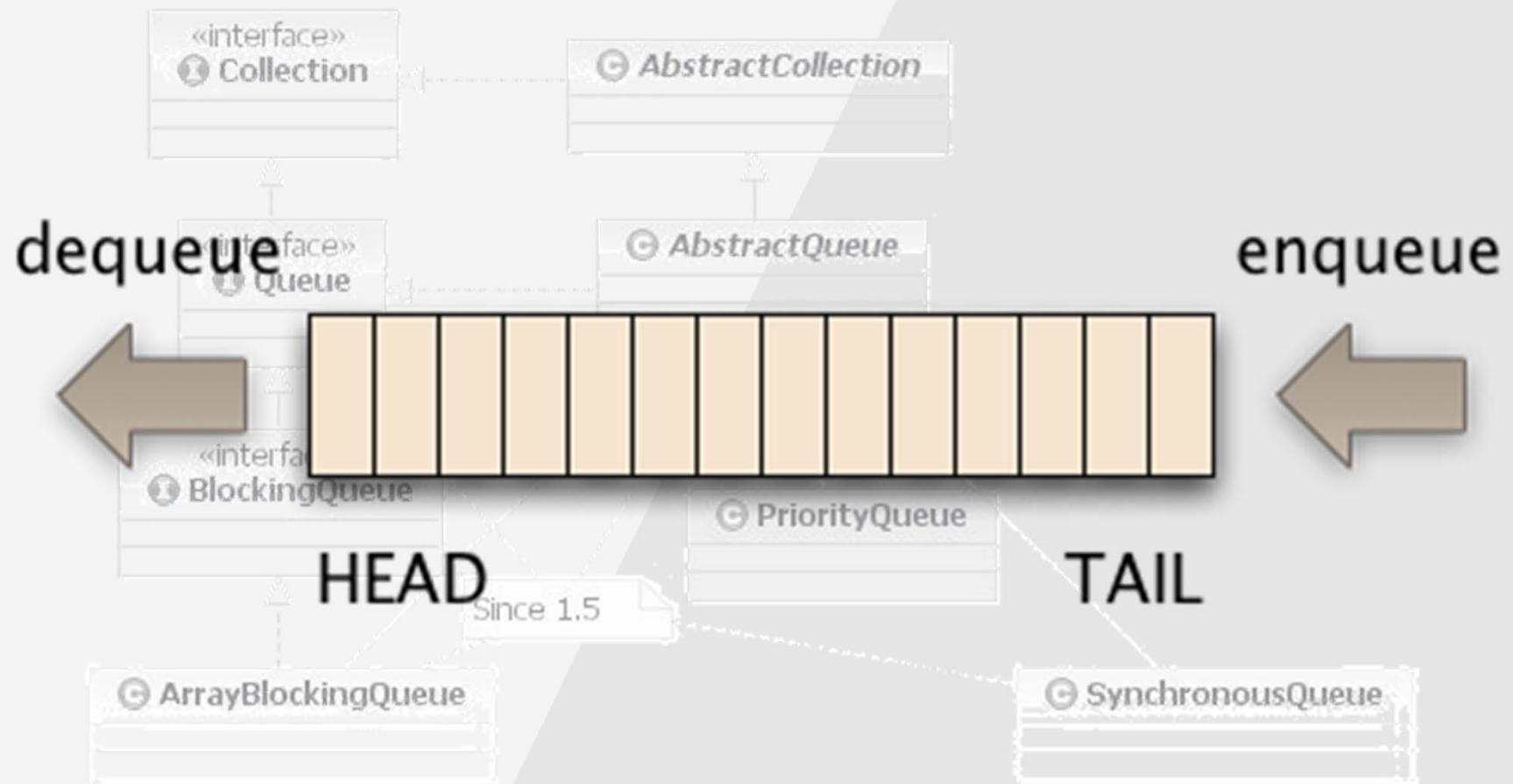
# COLECCIONES



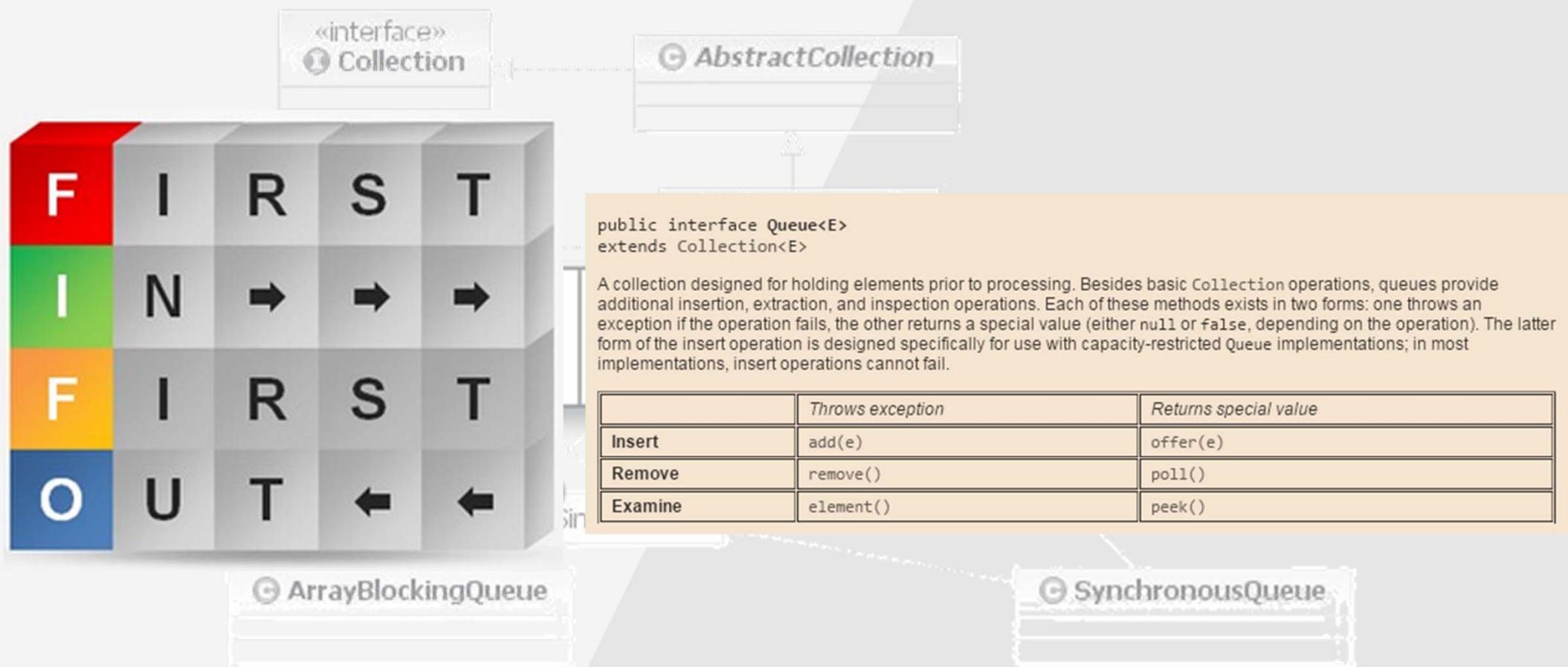
# COLECCIONES



# COLECCIONES

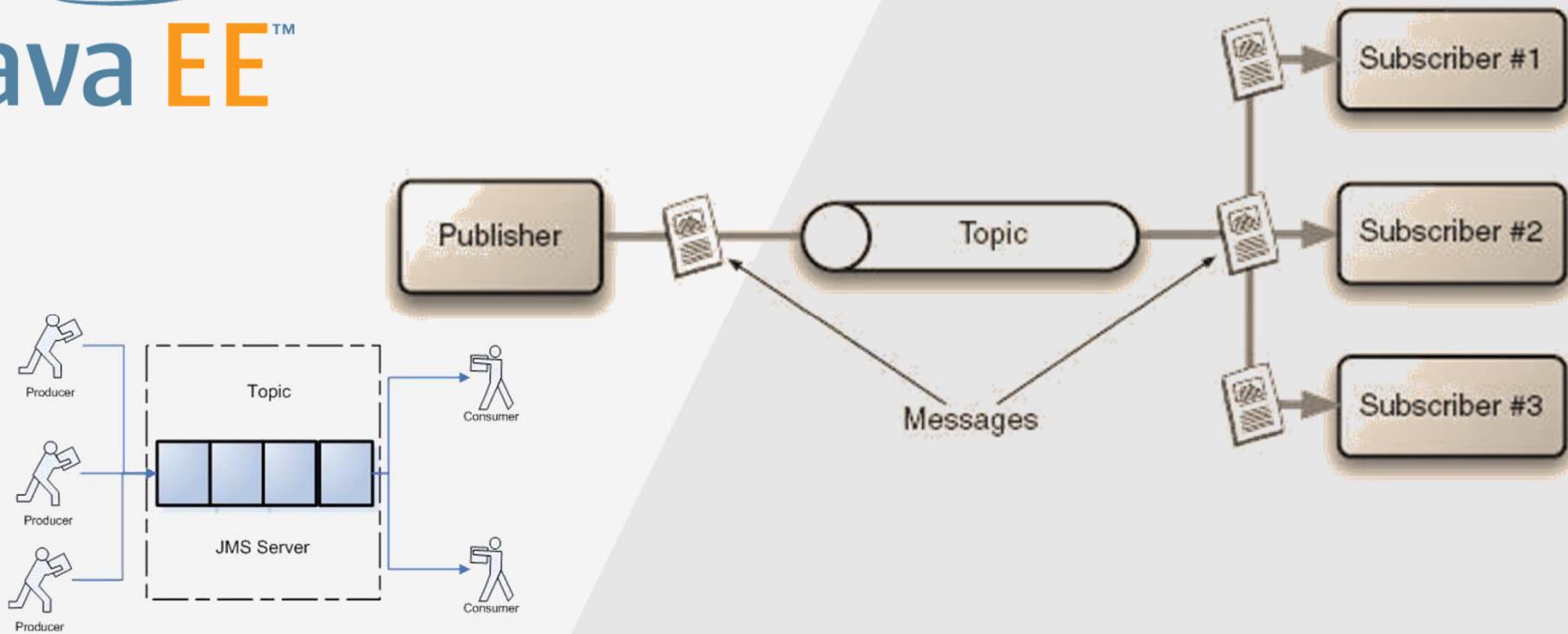


# COLECCIONES





# COLECCIONES



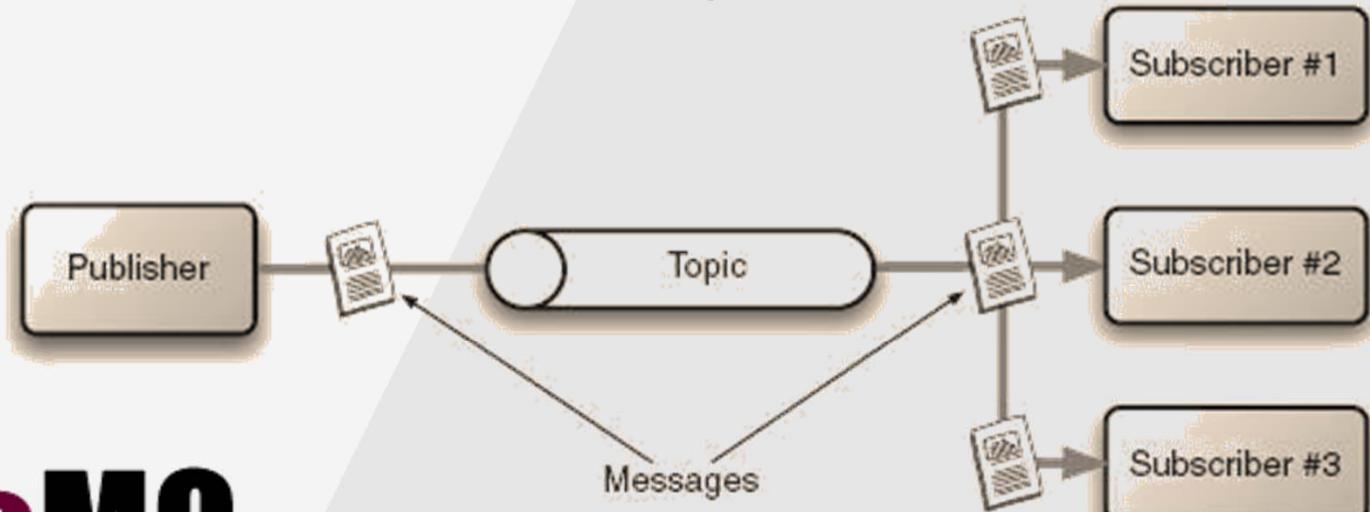
# COLECCIONES



Java EE™

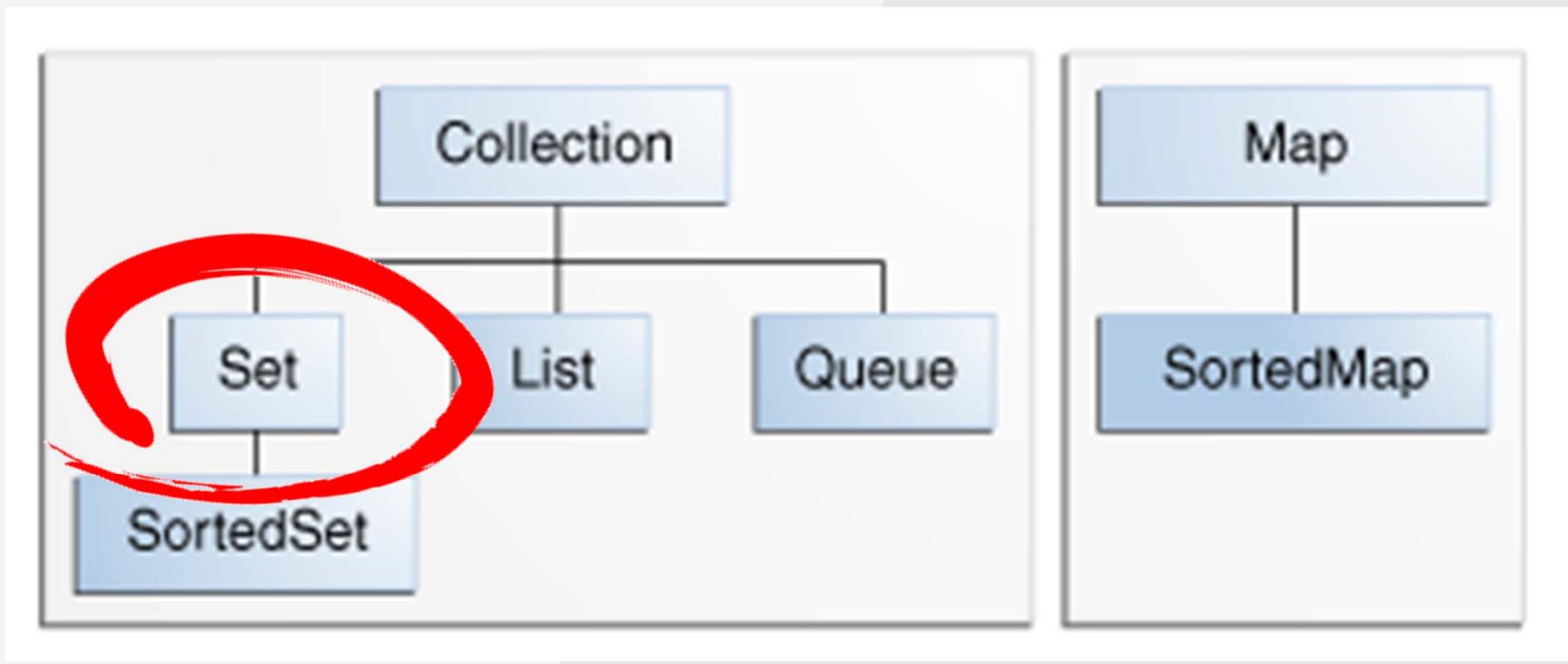


HornetQ

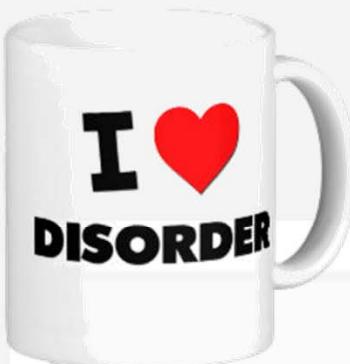


ActiveMQ

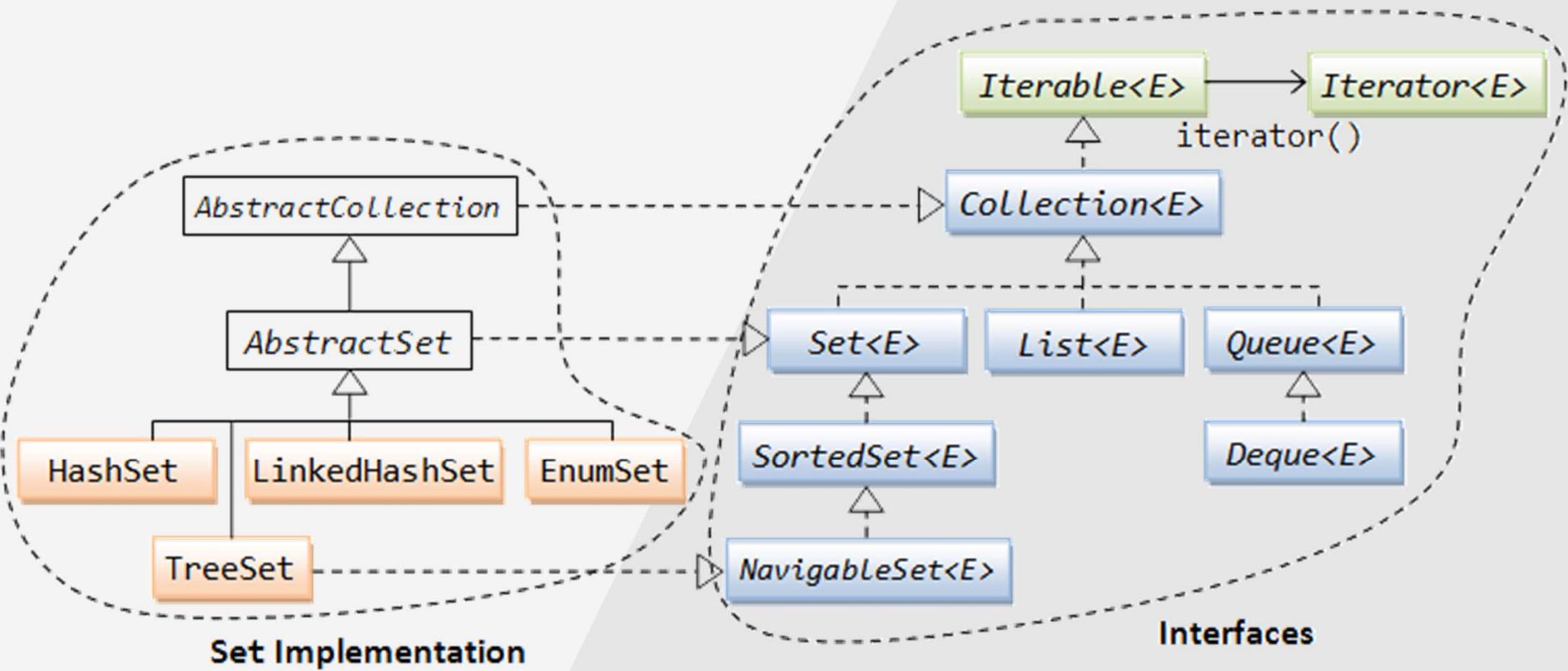
# COLECCIONES



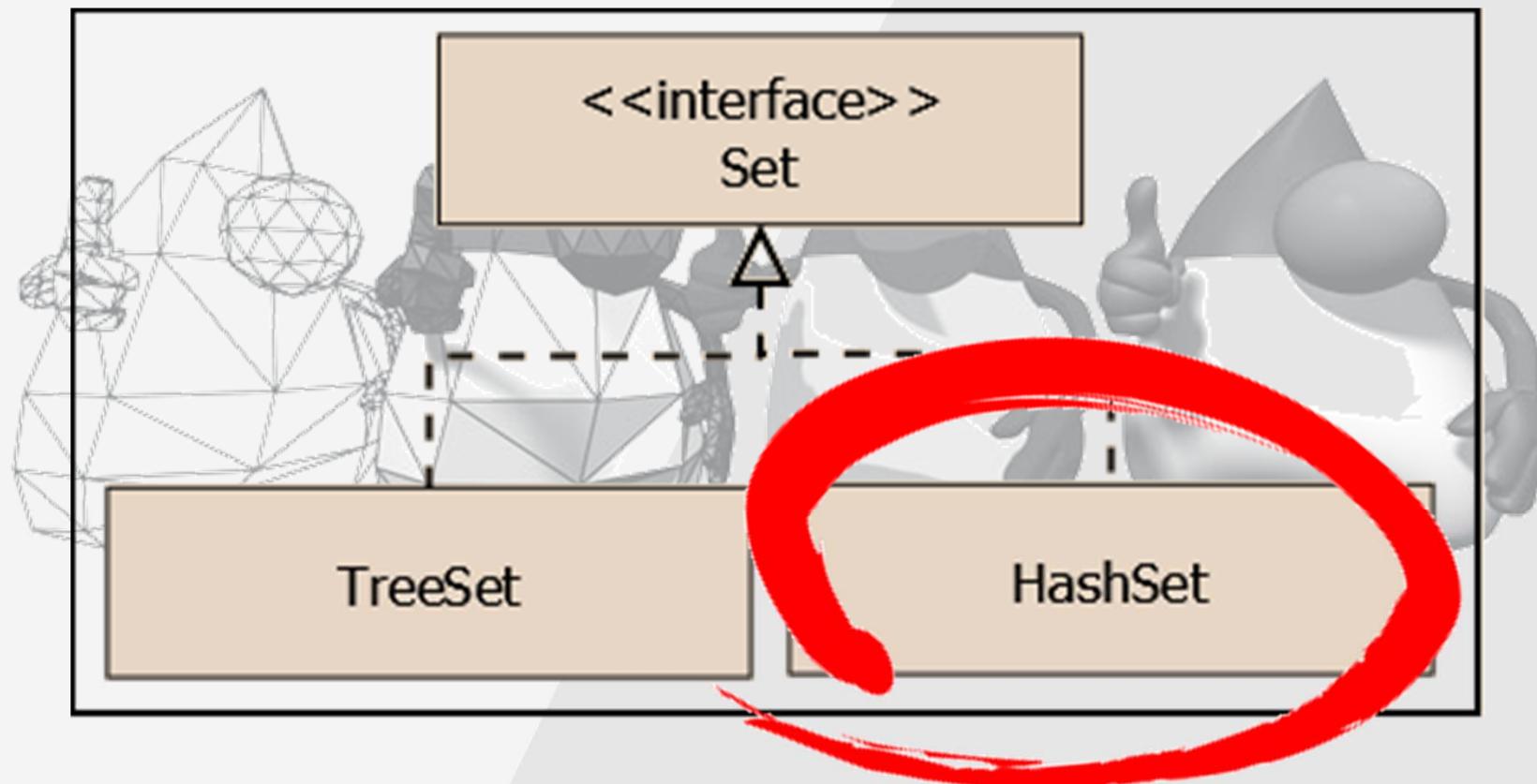
# COLECCIONES



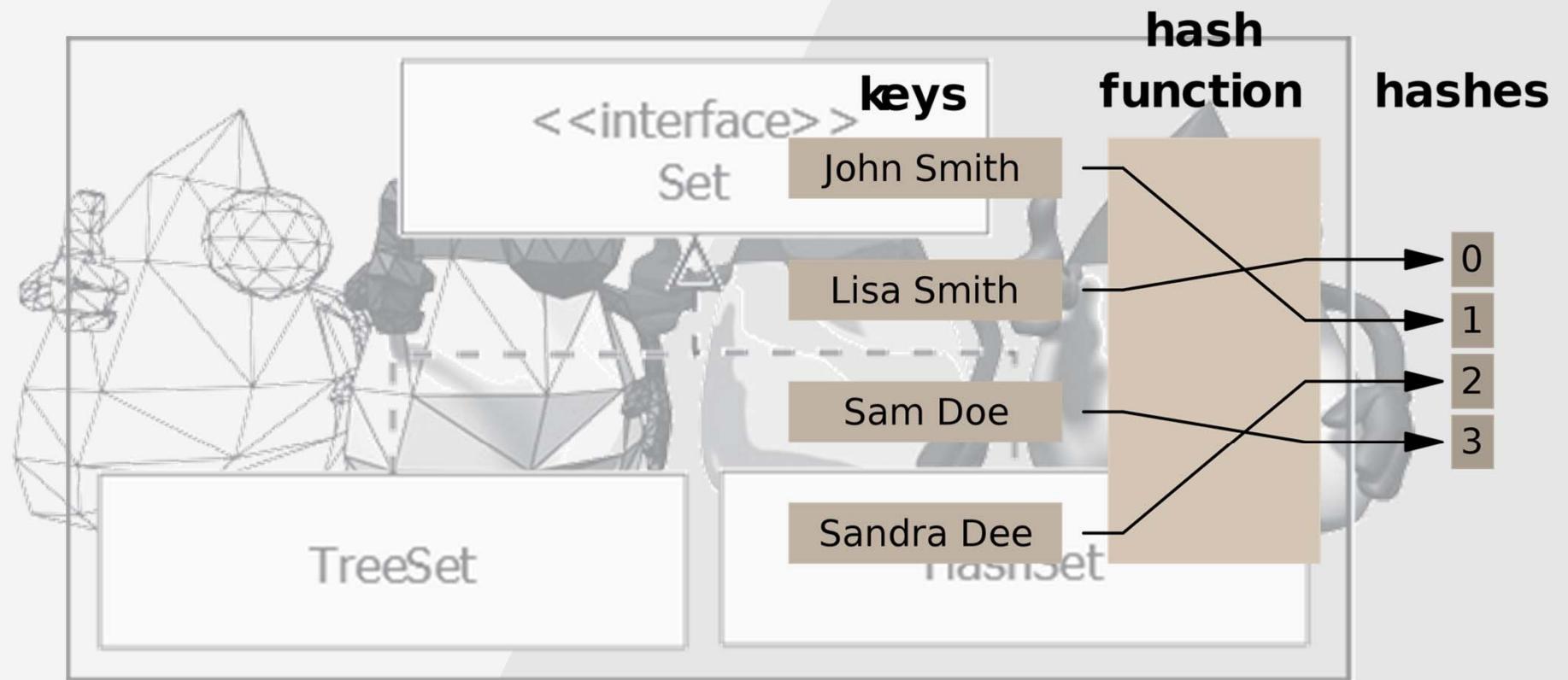
# COLECCIONES



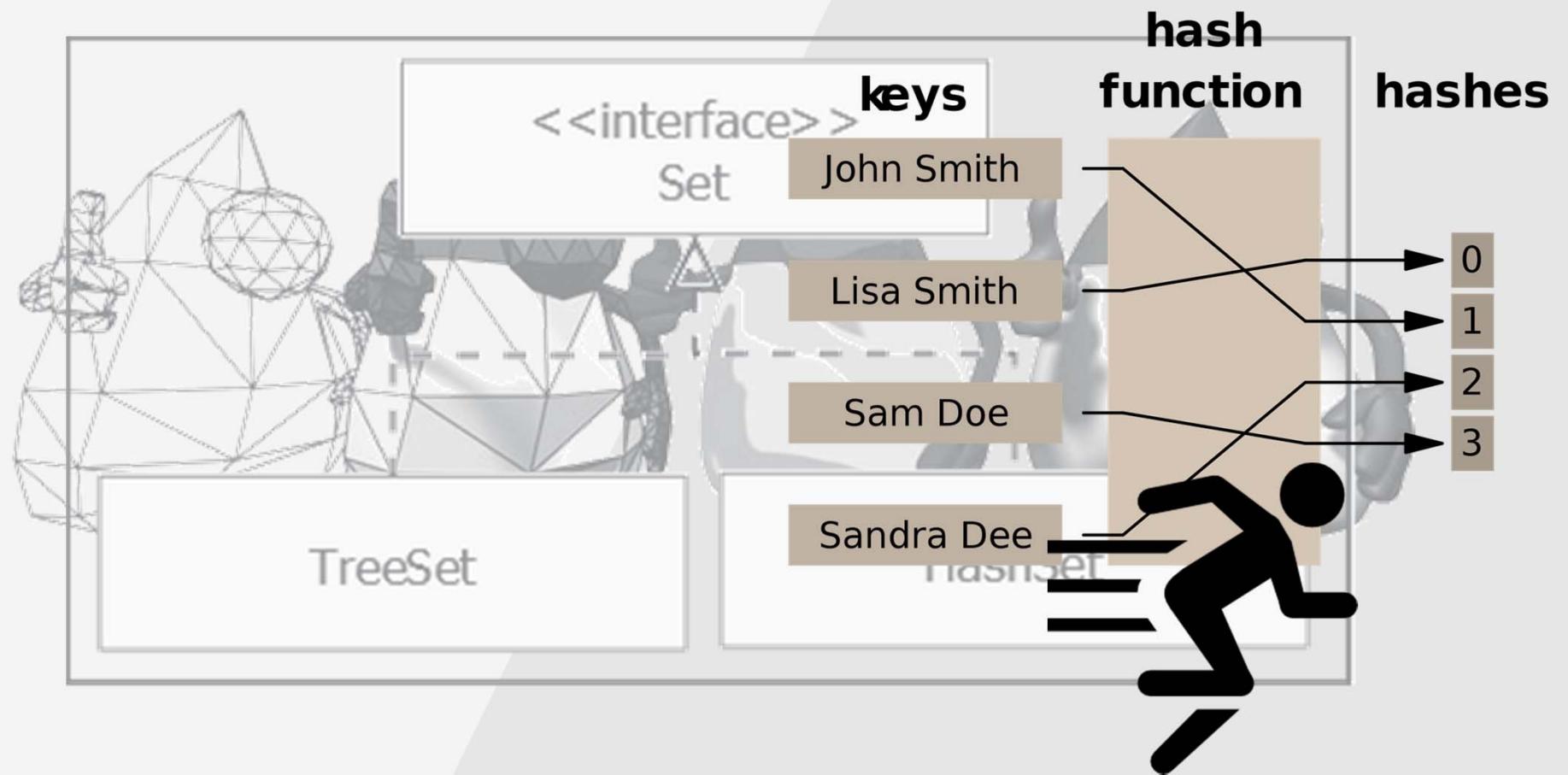
# COLECCIONES



# COLECCIONES

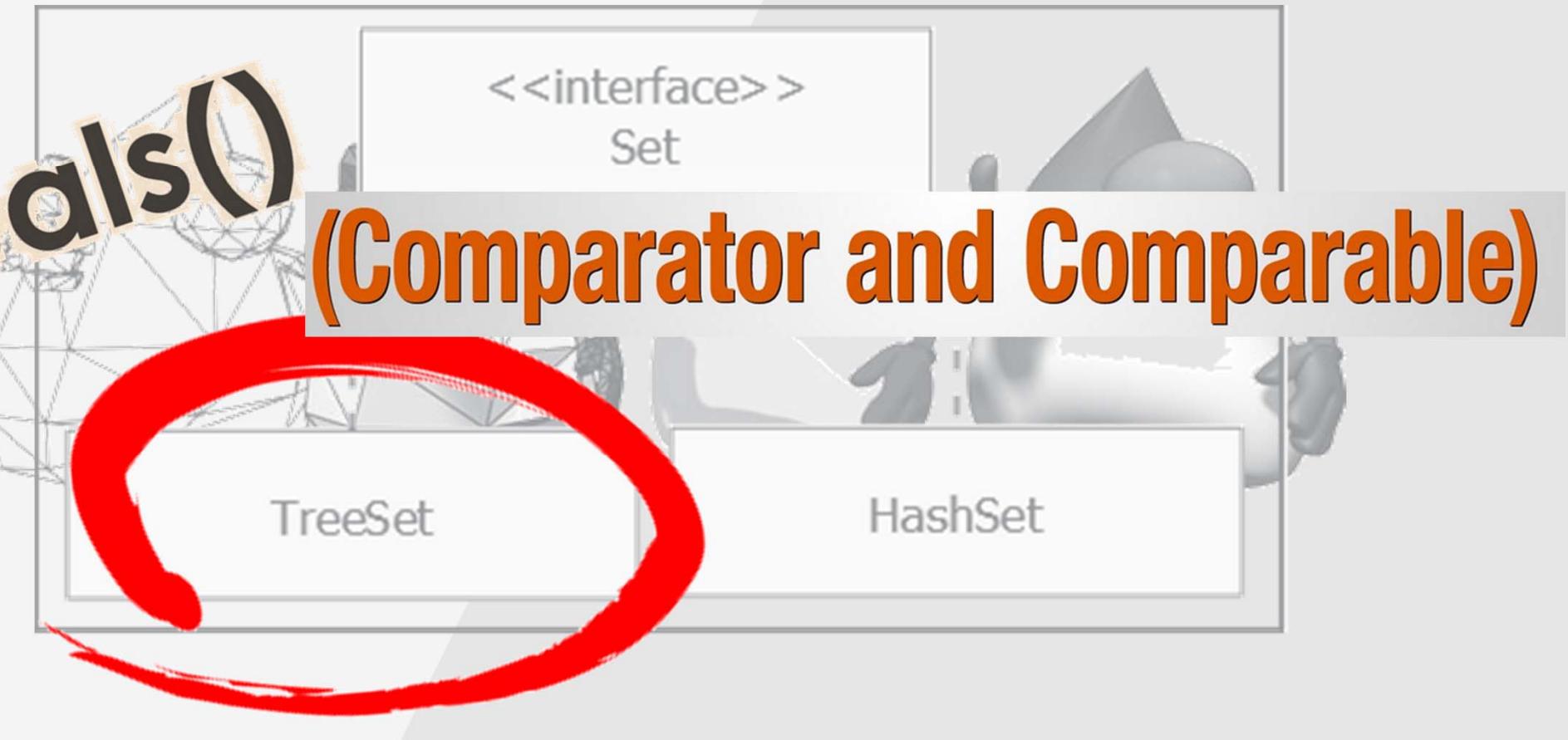


# COLECCIONES



# COLECCIONES

equals()



# COLECCIONES

compareTo

int compareTo(T o)

Interface Comparable<T>



# COLECCIONES

compareTo

int



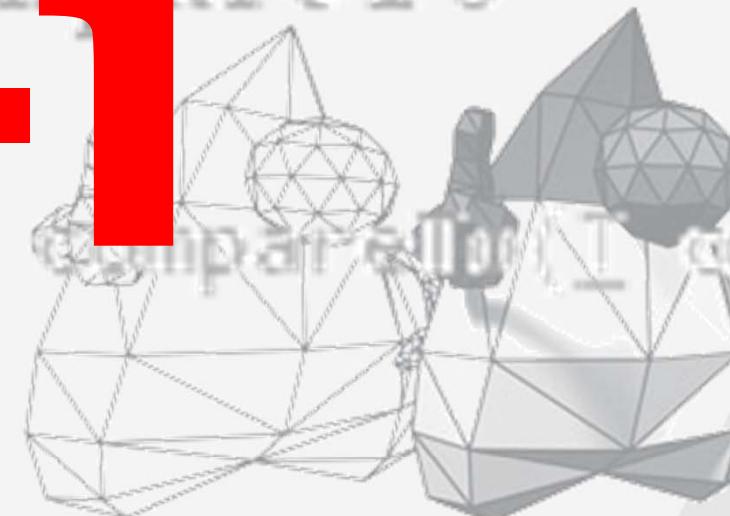
Interface Comparator<T>

# COLECCIONES

compareTo

-1

int

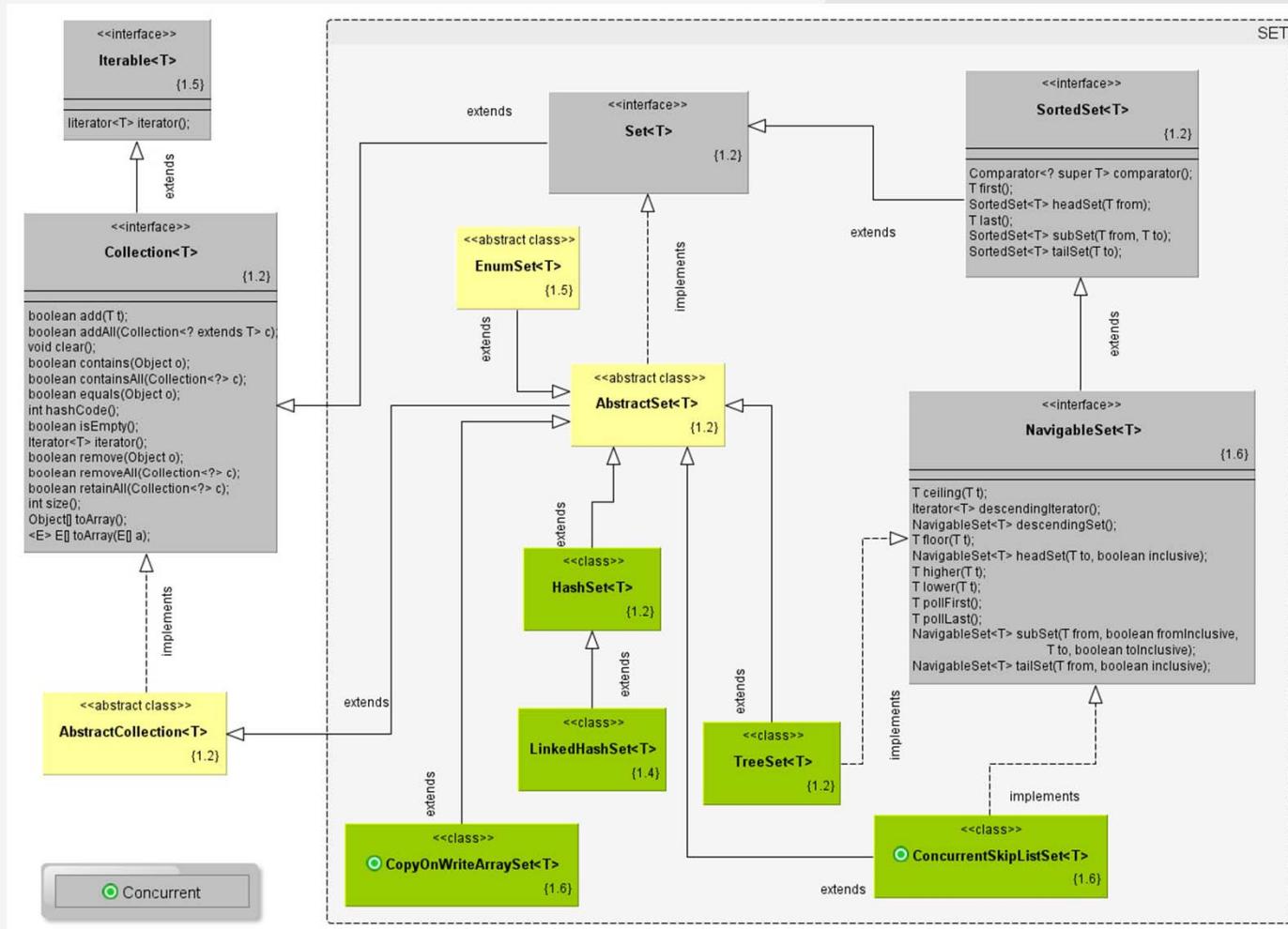


0

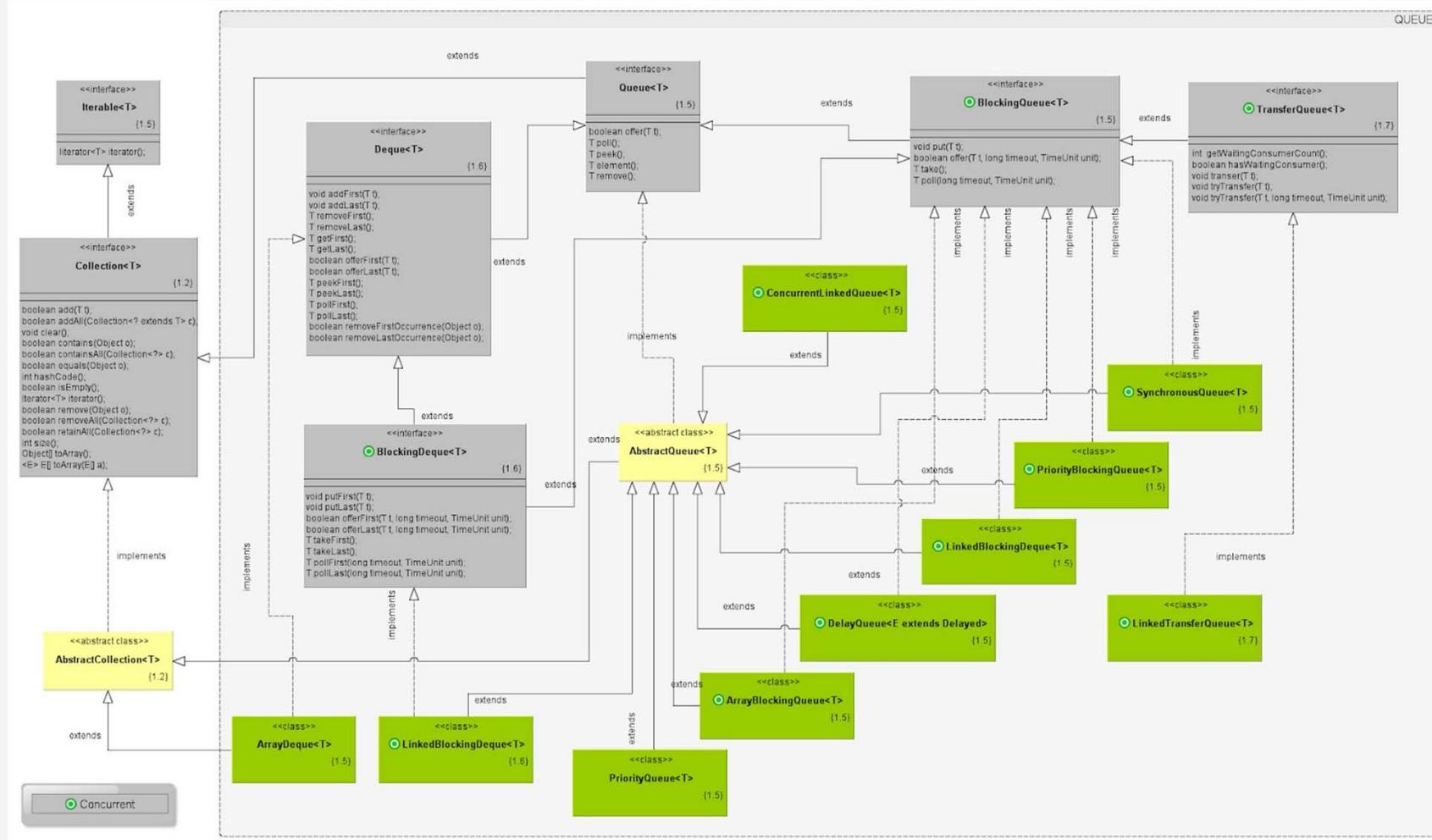


+1

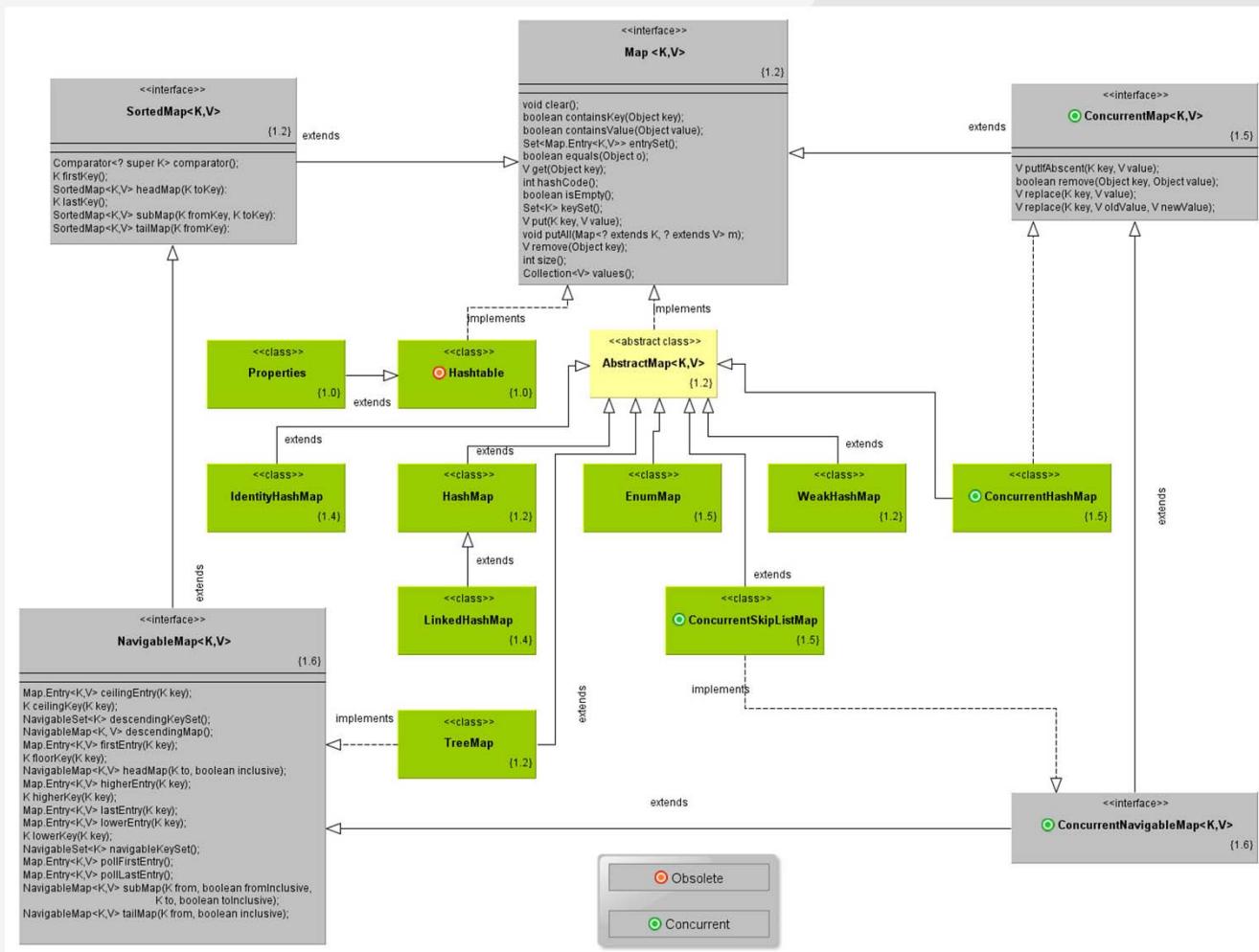
# COLECCIONES



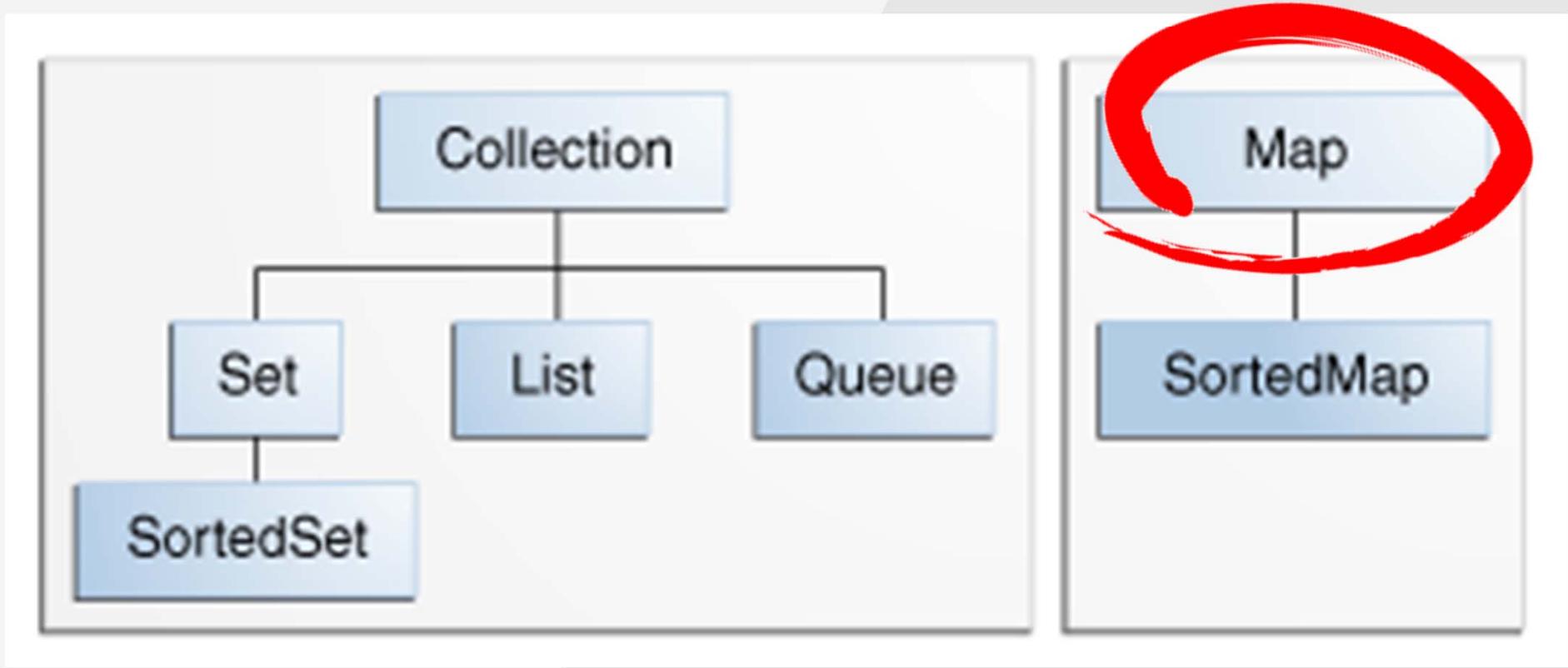
# COLECCIONES



# COLECCIONES



# COLECCIONES



# COLECCIONES

## <<Interface>> Map<K,V>

```
+clear()
+containsKey(in key:Object):Boolean
+containsValue( in value:Object ):Boolean
+entrySet():Set<Map.Entry<K, V>>
+get( Object key ):V
+isEmpty():Boolean
+keySet():Set<K>
+put( in key: K, in value:V ):V
+putAll( in m:Map<? extends K, ? extends V> )
+remove( in key:Object ):V
+size():Integer
+values():Collection<V>
```

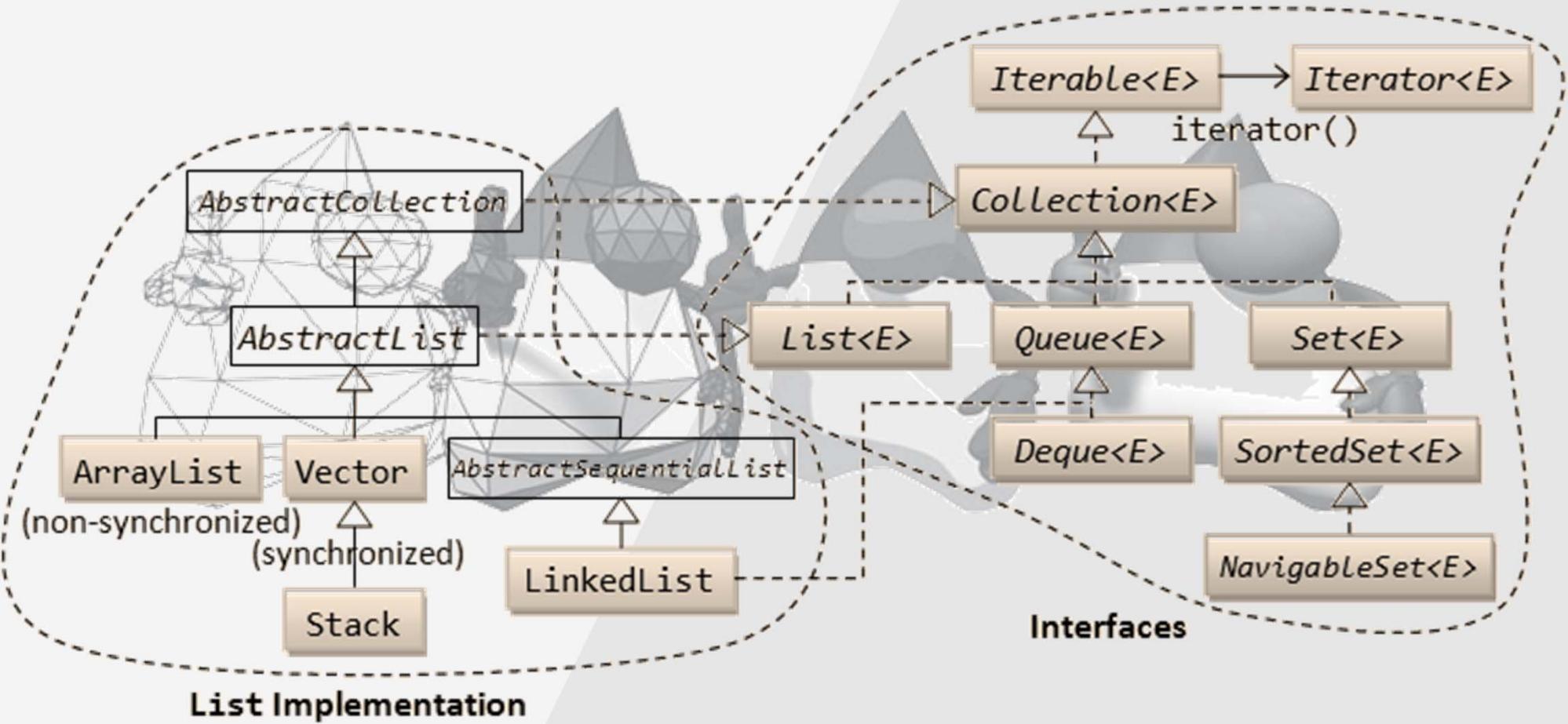
## <<Interface>> Map<K, V>

```
+ void      clear()
+ boolean   containsKey( Object key )
+ boolean   containsValue ( Object value )
+ Set<Map.Entry<K, V>> entrySet()
+ V         get ( Object key )
+ boolean   isEmpty ()
+ Set<K>   keySet()
+ V         put ( K key , V value )
+ void      putAll( Map<? extends K, ? extends V> m )
+ V         remove( Object key )
+ int       size()
+ Collection<V> values()
```

# COLECCIONES

```
public interface Map {  
    // Basic Operations  
    Object put(Object key, Object value);  
    Object get(Object key);  
    Object remove(Object key);  
    boolean containsKey(Object key);  
    boolean containsValue(Object value);  
    int size();  
    boolean isEmpty();  
    // Bulk Operations  
    void putAll(Map t);  
    void clear();  
    // Collection Views  
    public Set keySet();  
    public Collection values();  
    public Set entrySet();  
    // Interface for entrySet elements  
    public interface Entry {  
        Object getKey();  
        Object getValue();  
        Object setValue(Object value);  
    }  
}
```

# COLECCIONES



# COLECCIONES

=EQUALS=

*public boolean equals(Object o)*

# COLECCIONES

*reflexivo* →  $x.equals(x) == true$

=EQUALS=

# COLECCIONES

$x.equals(y) == y.equals.(x)$  ← simétrico

=EQUALS=

# COLECCIONES

*reflexivo*

*simétrico*

=EQUALS=

*transitivo*     $x.equals(y) == true ;$   
                     $y.equals(z) == true ; \rightarrow x.equals(z) == true ;$

# COLECCIONES

*reflexivo*

*simétrico*

=EQUALS=

`x.equals(y) == true;`

*OR*

`x, y = cte; ← consistente`

`x.equals(y) == false;`

# COLECCIONES

=EQUALS=

*y = null; ➔ x.equals(y) == false;*

# COLECCIONES



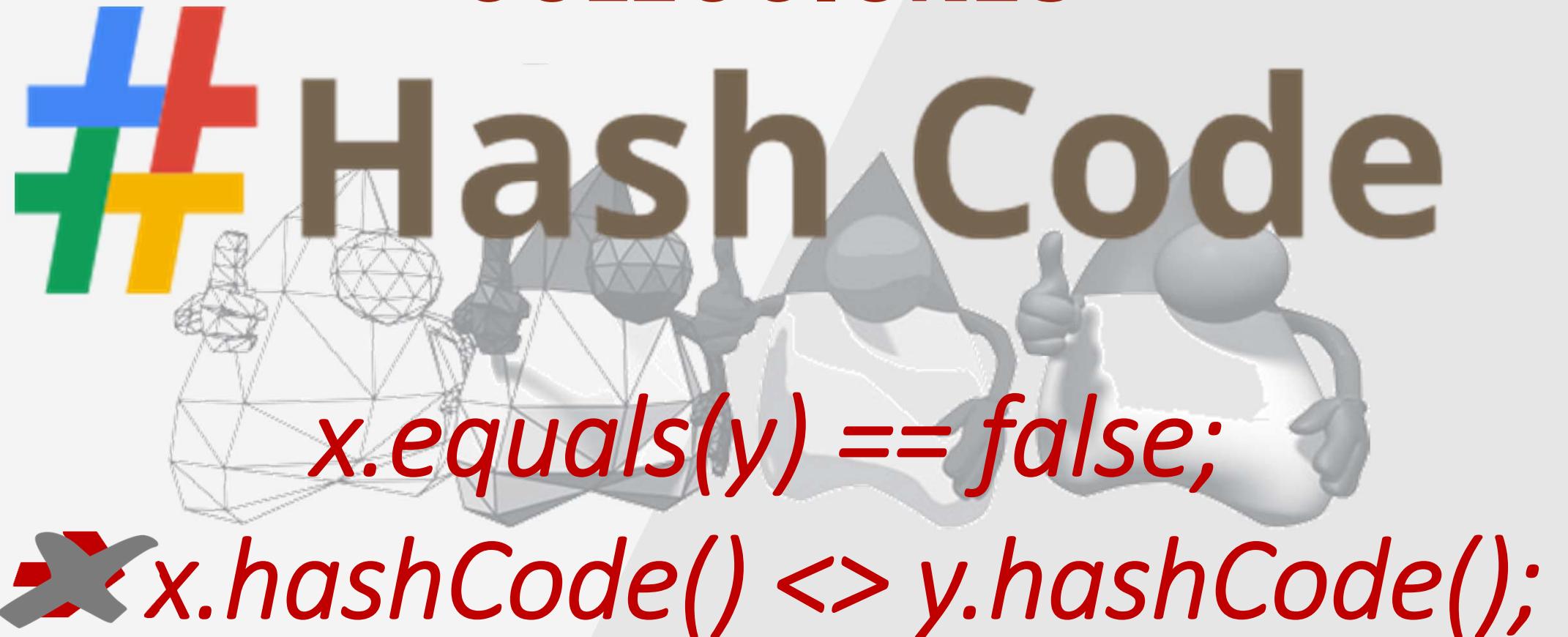
# COLECCIONES



*x.equals(y) == true;*

→ *x.hashCode() == y.hashCode();*

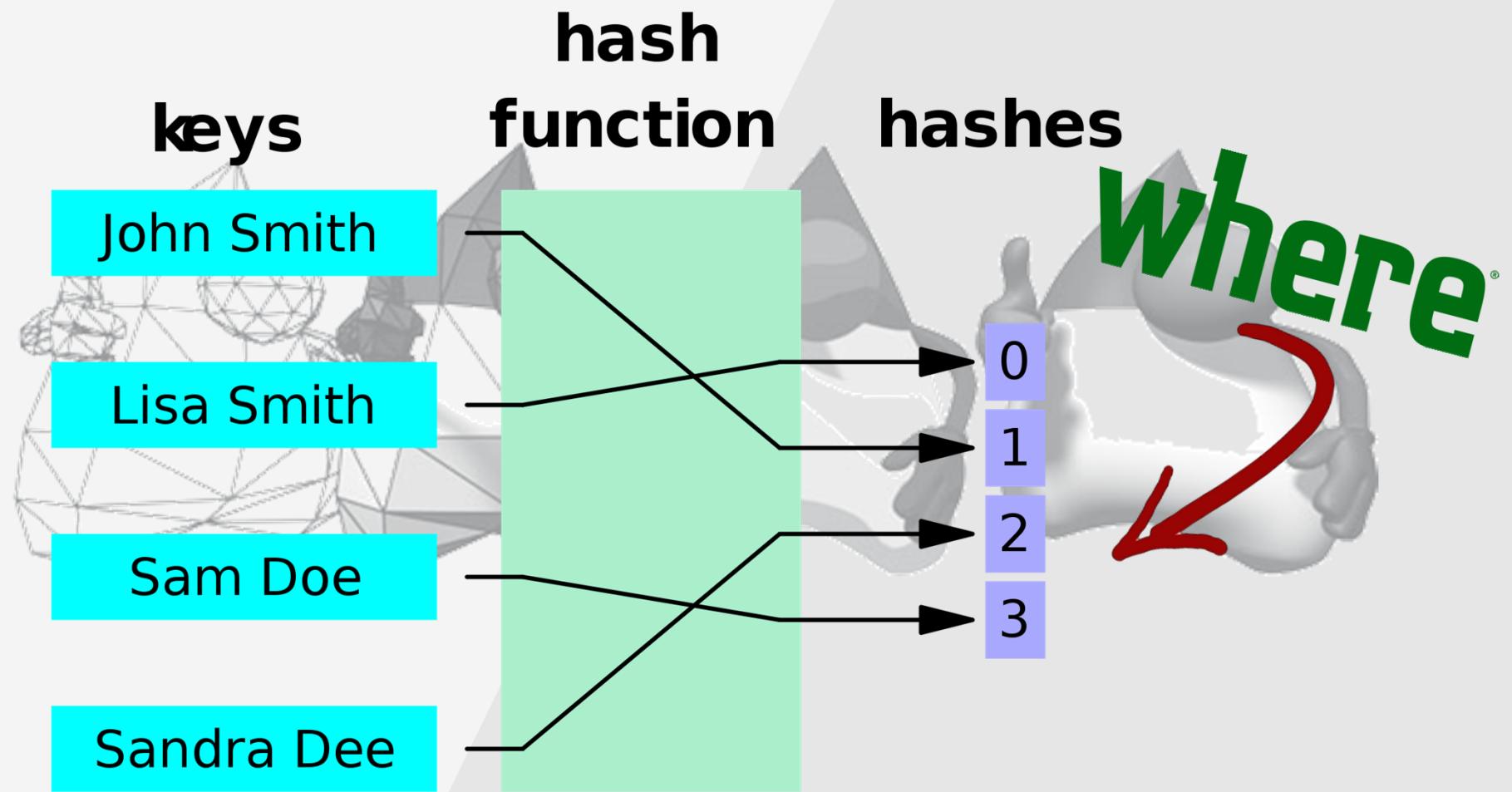
# COLECCIONES



# COLECCIONES



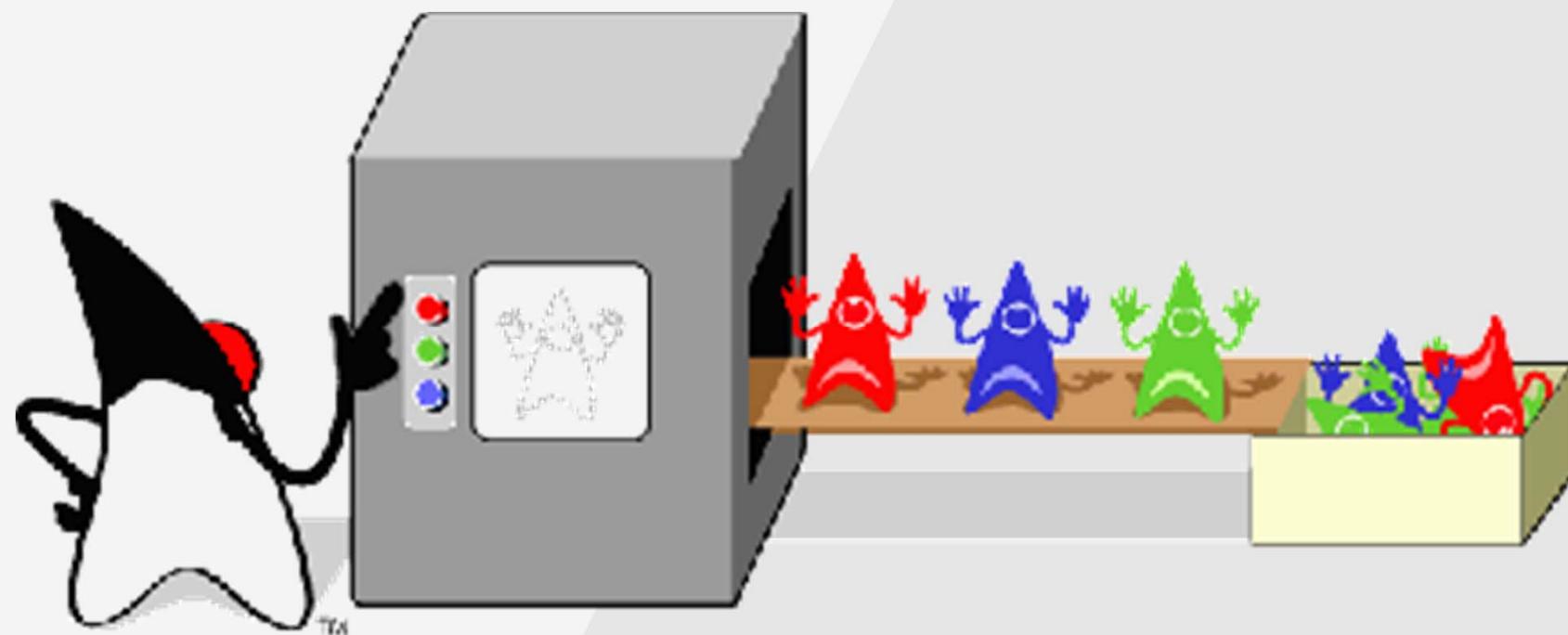
# COLECCIONES



# COLECCIONES



# GENÉRICOS



# GENÉRICOS

```
1.ArrayList list = new ArrayList();
2.list.add("java");
3.list.add(new Date());
4.for (Object o : list)
5.{}
6.System.out.println(o.toUpperCase());
7.
```

RUNTIME  
EXCEPTION !!!

# GENÉRICOS

## COMPILATION

```
1.ArrayList<String> list = new  
    ArrayList<String>(); ERROR !!!  
2.list.add("java");  
3.list.add(new Date()); for (String s : list)  
4.{  
5.System.out.println(s.toUpperCase());  
6.}
```

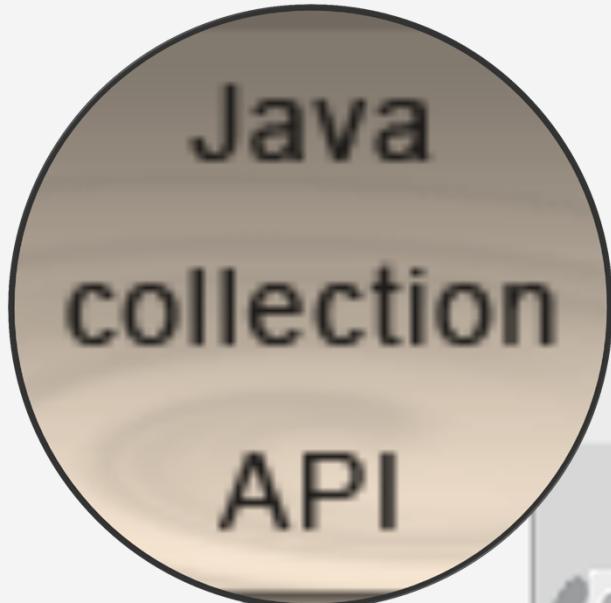
# GENÉRICOS

*Clases Genéricas  
Interfaces Genéricos*

*Métodos Genéricos*

*Tipos Limitados Genéricos*

*Desconocidos Genéricos*



# Clases Genéricas

```
1. public class Cupboard<T> {  
2.     private T item;  
3.     public Cupboard(T item) {  
4.         System.out.println("Cupboard for " +  
item.getClass());  
5.         this.item = item;  
6.     }  
7.     public T getItem() {  
8.         return item;  
9.     }  
10.}
```

# Clases Genéricas

```
1. public class Cupboard<T> {  
2.     private T item;  
3.     public Cupboard(T item) {  
4.         System.out.println("Cupboard for " +  
item.getClass());  
5.         this.item = item;  
6.     }  
7.     public T getItem() {  
8.         return item;  
9.     }  
10.}
```

# Interfaces Genéricas

```
1. public interface Breakable <T>
2. {
3.     public void doBreak (T t);
4. }
```

# Interfaces Genéricas

```
1. public interface Breakable<T>
2. {
3.     public void doBreak(T t);
4. }
```

# Interfaces Genéricas

```
1. public class Glass implements Breakable<String> {  
2.     public void doBreak(String message) {  
3.         System.out.println("Breaking a Glass: " +  
        message);  
4.     }  
5. }  
  
6. class Dish<U> implements Breakable <U> {  
7.     public void doBreak(U u) {  
8.         System.out.println("Breaking " + u.toString());  
9.     }  
10.}
```

# Interfaces Genéricas

```
1. public class Glass implements Breakable<String> {  
2.     public void doBreak(String message) {  
3.         System.out.println("Breaking a Glass: " + message);  
4.     }  
5. }
```

```
6. class Dish<U> implements Breakable<U> {  
7.     public void doBreak(U u) {  
8.         System.out.println("Breaking " + u.toString());  
9.     }  
10.}
```

```
11. List<Dish<String>> dishes = new ArrayList<Dish<String>>();
```



# Interfaces Genéricas

```
1. public class Glass implements Breakable<String> {  
2.     public void doBreak(String message) {  
3.         System.out.println("Breaking a Glass: " + message);  
4.     }  
5. }
```

```
6. class Dish<U> implements Breakable<U> {  
7.     public void doBreak(U u) {  
8.         System.out.println("Breaking " + u.toString());  
9.     }  
10.}
```

```
11. List<Dish<String>> dishes = new ArrayList<Dish<String>>();
```



# Métodos Genéricos

```
1. public class Box {  
2.     public static <T> void ship(T item) {  
3.         System.out.println("Shipping " + item.toString());  
4.     }  
5.     public static void main(String [] args) {  
6.         Box.ship("a String object"); //Shipping a String object  
7.         Box.ship(args); //Shipping [Ljava.lang.String;@3e25a5  
8.         Box.ship(new Box()); //Shipping This is a Box  
9.     }  
10.    public String toString() {  
11.        return "This is a Box";  
12.    }  
13.}
```

# Métodos Genéricos

```
1. public class Box {  
2.     public static <T> void ship(T item) {  
3.         System.out.println("Shipping " + item.toString());  
4.     }  
5.     public static void main(String [] args) {  
6.         Box.ship("a String object"); //Shipping a String object  
7.         Box.ship(args); //Shipping [Ljava.lang.String;@3e25a5  
8.         Box.ship(new Box()); //Shipping This is a Box  
9.     }  
10.    public String toString() {  
11.        return "This is a Box";  
12.    }  
13.}
```

# Métodos Genéricos

```
1. public class Box {  
2.     public static <T> void ship(T item) {  
3.         System.out.println("Shipping " + item.toString());  
4.     }  
5.     public static void main(String[] args) {  
6.         Box.ship("a String object"); //Shipping a String object  
7.         Box.ship(args); //Shipping [Ljava.lang.String;@3e25a5  
8.         Box.ship(new Box()); //Shipping his is a Box  
9.     }  
10.    public String toString() {  
11.        return "This is a Box";  
12.    }  
13.}
```

# Métodos Genéricos

```
1. public class Box {  
2.     public static <T> void ship(T item) {  
3.         System.out.println("Shipping " + item.toString());  
4.     }  
5.     public static void main(String [] args) {  
6.         Box.<String>ship("a String object");  
7.         Box.<String []>ship(args);  
8.         Box.<Box>ship(new Box());  
9.     }  
10.    public String toString() {  
11.        return "This is a Box";  
12.    }  
13.}
```

# Métodos Genéricos

```
1. public class Box {  
2.     public static <T> void ship(T item) {  
3.         System.out.println("Shipping " + item.toString());  
4.     }  
5.     public static void main(String [] args) {  
6.         Box.<String>ship("a String object");  
7.         Box.<String []>ship(args);  
8.         Box.<Box>ship(new Box());  
9.     }  
10.    public String toString() {  
11.        return "This is a Box";  
12.    }  
13.}
```



# Tipos Limitados Genéricos

```
1. public class Hello <T extends List> { }
```

# Tipos Limitados Genéricos

1. `public class Hello<T extends List> { }`

2. `Hello<ArrayList> h1 = new Hello<ArrayList>();`

3. `Hello<Stack> h2 = new Hello<Stack>();`

# Desconocidos Genéricos

→ ?, *ilimitado*



# Desconocidos Genéricos

→ ?, ilimitado

→ ? extends type,

acotado por extensión



# Desconocidos Genéricos

→ ?, ilimitado

→ ? extends type,

acotado por extensión

→ ? super type,

acotado por herencia

# Desconocidos Genéricos

```
1. public static void printList(List<?> list)
2. {
3.     for(Object x : list)
4.     {
5.         System.out.println(x.toString());
6.     }
7. }
```

# Desconocidos Genéricos

```
ArrayList <Double> list = new ArrayList <Double>();
```

# Desconocidos Genéricos

```
ArrayList <Double> list = new ArrayList <Double>();
```

```
ArrayList <Number> notvalid = new ArrayList <Double>();
```



# Desconocidos Genéricos

```
ArrayList <Double> list = new ArrayList <Double>();  
ArrayList <Number> notval = new ArrayList <Double>();
```



# Desconocidos Genéricos

```
ArrayList <Double> list = new ArrayList <Double>();
```

```
ArrayList <Number> notvalid = new ArrayList <Double>();
```

```
ArrayList <? extends Number> list2 = new ArrayList <Double>();
```



# Desconocidos Genéricos

```
ArrayList <? super IOException> alist1 = new ArrayList <Exception>();  
ArrayList <? super IOException> alist2 = new ArrayList <IOException>();
```

# Desconocidos Genéricos

```
ArrayList <? super IOException> alist1 = new ArrayList <Exception>();  
ArrayList <? super IOException> alist2 = new ArrayList <IOException>();
```



# Desconocidos Genéricos

```
ArrayList <? super IOException> alist1 = new ArrayList <Exception>();  
  
ArrayList <? super IOException> alist2 = new ArrayList <IOException>();
```



# Desconocidos Genéricos

```
ArrayList <? super IOException> alist1 = new ArrayList <Exception>();  
  
ArrayList <? super IOException> alist2 = new ArrayList <IOException>();  
  
ArrayList <? super IOException> alist3 =  
    new ArrayList <FileNotFoundException>();
```



# Desconocidos Genéricos

```
ArrayList<Exception> alist1 = new ArrayList<Exception>();
```

```
ArrayList<? super IOException> alist2 = new  
ArrayList<IOException>();
```

```
ArrayList<? super IOException> alist3 = new ArrayList<IOException>();
```

# Convenciones

- ➔ *E, para un elemento*
- ➔ *K, claves de mapas*
- ➔ *V, valores de mapas*
- ➔ *N, números*
- ➔ *T, tipos de datos genéricos*

# Convenciones

- ➔ *List<E>*
- ➔ *Map<K,V>*
- ➔ *public <N> N getNumber();*
- ➔ *public <T> T trasnform(T myType);*

# Ordenación



# Ordenación

```
1.import java.util.*;  
2.class TestUnSort {  
3.    public static void main(String[] args) {  
4.        ArrayList<String> stuff = new ArrayList<String>();  
5.        stuff.add("Denver");  
6.        stuff.add("Boulder");  
7.        stuff.add("Vail");  
8.        stuff.add("Aspen");  
9.        stuff.add("Telluride");  
10.       System.out.println("unsorted " + stuff);  
11.    }  
12.}
```

unsorted [Denver, Boulder, Vail, Aspen, Telluride]

# Ordenación

```
1. import java.util.*;  
2. class TestSort {  
3.     public static void main(String[] args) {  
4.         ArrayList<String> stuff = new ArrayList<String>();  
5.         stuff.add("Denver");  
6.         stuff.add("Boulder");  
7.         stuff.add("Vail");  
8.         stuff.add("Aspen");  
9.         stuff.add("Telluride");  
10.        Collections.sort(stuff);  
11.        System.out.println("sorted  " + stuff);  
12.    }  
13.} sorted  [Aspen, Boulder, Denver, Telluride, Vail]
```

# Ordenación



# Ordenación

*Collections.sort(List<T> list,  
Comparator<? super T> c)*

## Method Summary

int	<u><a href="#">compare(T o1, T o2)</a></u> Compares its two arguments for order.
boolean	<u><a href="#">equals(Object obj)</a></u> Indicates whether some other object is "equal to" this comparator.

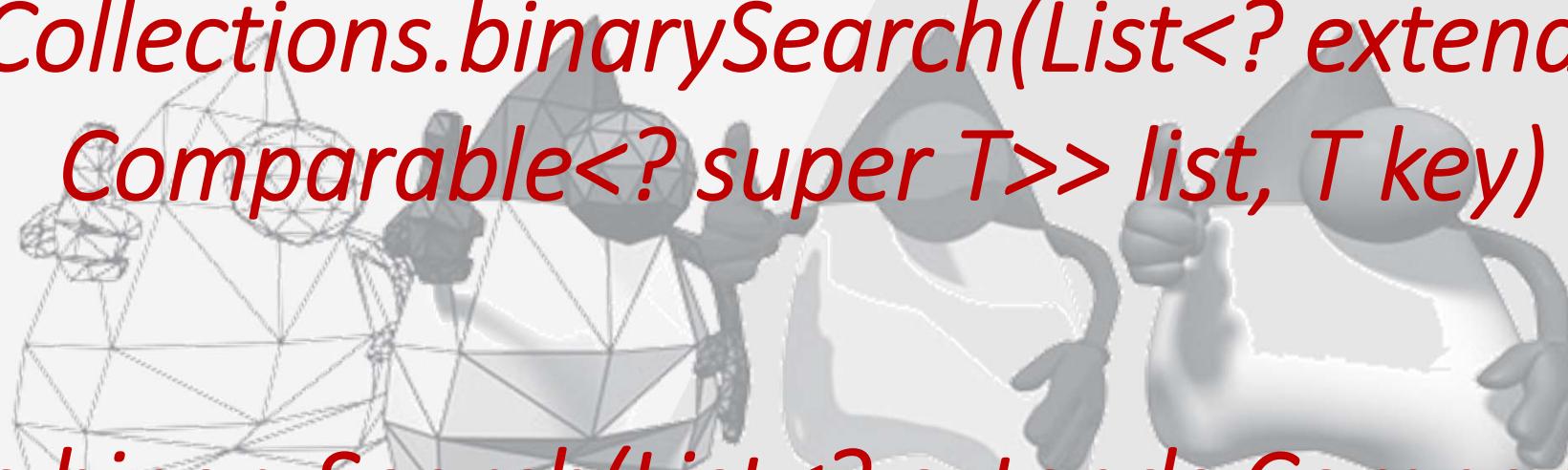
# Ordenación

*Arrays.sort(arrayToSort)*

*Arrays.sort(arrayToSort, Comparator)*

# Búsquedas

*Collections.binarySearch(List<? extends Comparable<? super T>> list, T key)*



*Arrays.binarySearch(List<? extends Comparable<? super T>> list, T key)*

# Búsquedas

.1



*Collections.binarySearch(List<? extends Comparable<? super T>> list, T key)*

*Arrays.binarySearch(List<? extends Comparable<? super T>> list, T key)*

# Búsquedas

> -1



# INDEX

ArrayLists.binarySearch(List<? extends Comparable<?>, T key))  
Collections.binarySearch(List<? extends Comparable<?>, T key))  
Arrays.binarySearch(List<? extends Comparable<?>, T key))

# Búsquedas

*La Colección o el Array  
debe estar ordenado  
antes de buscar, sino el  
resultado de  
*binarySearch* es  
impredecible.*

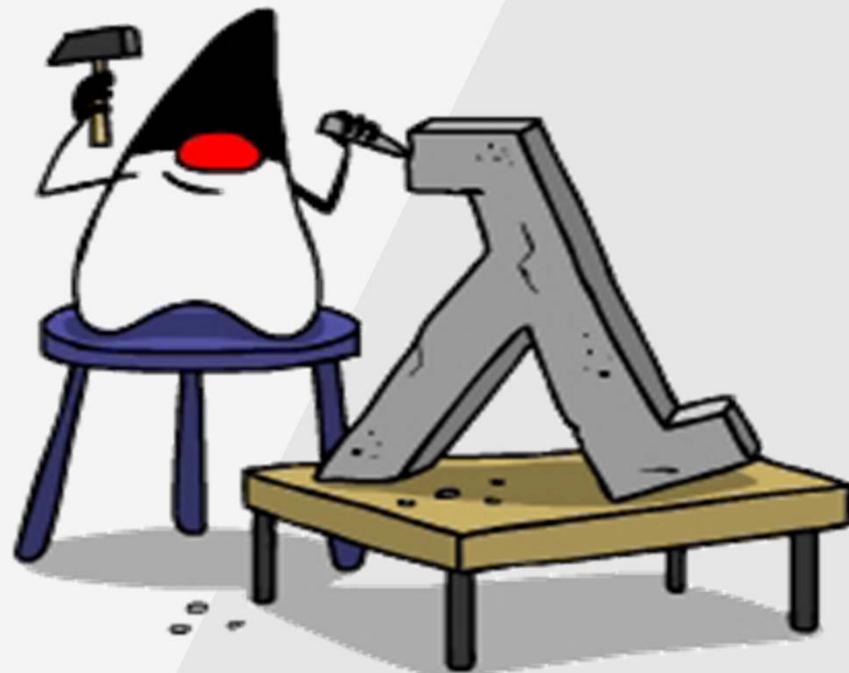




#7

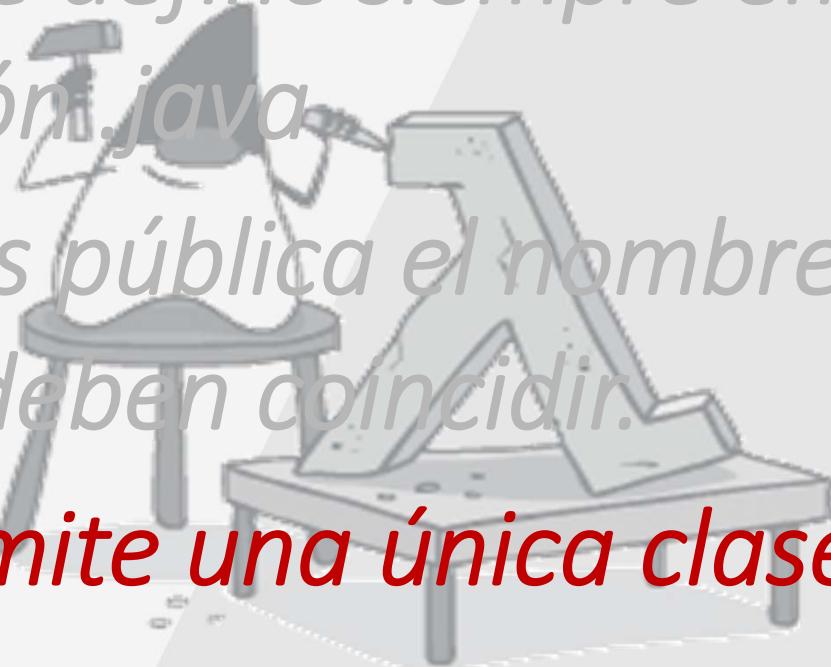
# Fundamentos

# FUNDAMENTOS



# Escribiendo Clases

- Una clase se define siempre en un fichero con extensión .java
- Si la clase es pública el nombre de la clase y del fichero deben coincidir.
- Sólo se permite una única clase pública por fichero.



# Escribiendo Clases

- Las clases java compiladas generan archivos en bytecode con extensión .class
- Aunque haya varias clases definidas en un mismo fuente .java se generan diferentes archivos compilados de extensión .class, uno por clase definida.

# Paquetes

- Un package es una unidad organizativa de agrupación de clases.
- Un package además tiene propiedades de ámbito ( scope ) para las clases.
- El package de un fuente siempre debe figurar en la primera linea del fichero.

# Paquetes

- Un package es el prefijo del nombre completo de una clase.
- Un package genera directorios para la clase que se compila.

# Paquetes

- *Clases no declaradas dentro de un paquete forman parte de un paquete sin nombre o default.*
- *Las clases que no forman parte de un paquete no pueden ser importadas.*



# Paquetes

- La palabra reservada *import* se usa para importar una clase determinada.
- Se permite utilizar el comodín (\*), que permite importar la totalidad de clases desde un determinado punto de la estructura del paquete.

# Paquetes

*Aunque se utilice el comodín (\*), solo se incorporan las clases que son utilizadas por la clase que importa, no obstante el importar específicamente la clase utilizada se considera una buena práctica.*



# Paquetes

→ *El parámetro –d a la hora de compilar sirve para designar el directorio de compilación donde se generarán los .class*

```
javac -d c:\myproject\build .\es\hp\payroll\Employee.java →  
c:\myproject\build\es\hp\payroll\Employee.class
```

# Classpath

- *El classpath hace referencia a la ruta donde los archivos compilados (bytecode) son guardados.*
- *La ruta del classpath, se almacena en una variable de entorno denominada CLASSPATH.*

# Classpath

→ *Ejemplo classpath:*

Si la clase es.hp.payroll.Employee se encuentra en la ruta:

c:\myproject\build\es\hp\payroll\Employee.class

La variable classpath necesitará incluir la ruta:

c:\myproject\build\ para incluir la clase.

# Classpath

→ *La variable CLASSPATH puede incluir tantos directorios como sea preciso, cada uno irá separado por ; o por :, según el sistema operativo.*



```
Set CLASSPATH = "c:\Projects\workspace\MyFirstApp\build";
c:\myproject\build;c:\tomcat\lib\servlet.jar;.;
```

# Classpath

→ *La variable de CLASSPATH puede incluir tanto rutas donde se encuentran clases compiladas como rutas que especifican archivos jar concretos.*

# Classpath

- *Es posible especificar el classpath a través de linea de comando, utilizando el flag -classpath.*
- *Este flag sobreescribe para la ejecución la variable de entorno CLASSPATH.*

# Classpath



*El nombre del paquete de la clase es parte del nombre de la clase y por tanto NO debe ser incluido en la ruta del classpath.*

# Ejecución

- *Para ejecutar inicialmente una clase es necesario que esta provea un método main.*
- *La estructura del metodo main es la siguiente: public static void main (args [] )*

# Ejecución

- Para ejecutar una clase se utiliza el archivo ejecutable `java.exe`.
- Al invocar `java.exe` no se debe especificar la extensión de la clase (`.class`), ya que es presupuesta.

# Ejecución

→ *java.exe escanea el contenido de CLASSPATH para buscar las clases.*

→ *Si la variable CLASSPATH, no está definida se presupone el directorio actual de trabajo. “.”*

# JARS

- Las clases compiladas ( bytecode ), pueden ser almacenadas en archivos comprimidos con extensión .jar
- Estos archivos .jar no necesitan ser descomprimidos para ser utilizados.

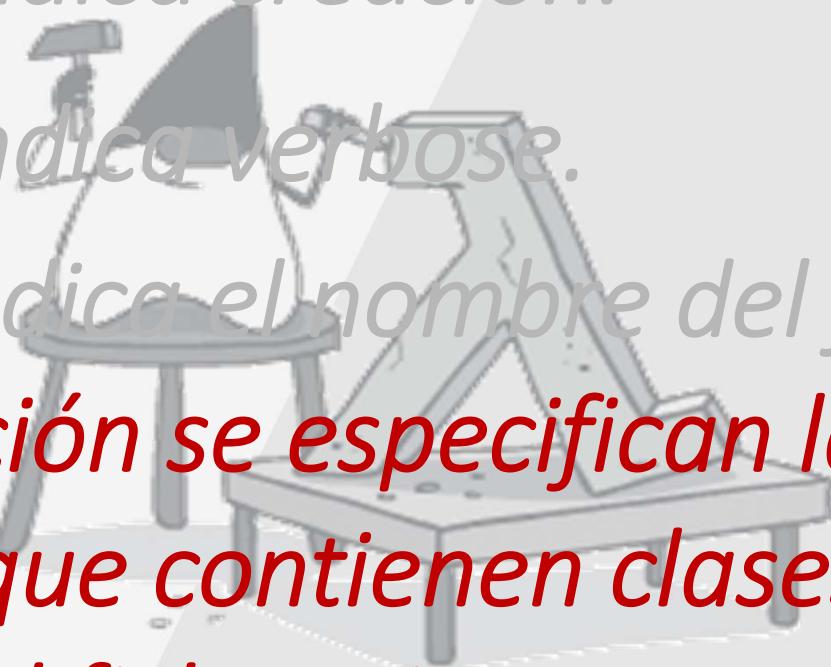
# JARS

- *La herramienta java.exe es la encargada de gestionar archivos jar*
- *Veamos un ejemplo de cómo añadir un directorio que contiene clases a un archivo jar:*

```
C:\myproject\build>jar -cvf myproject.jar .
```

# JARS

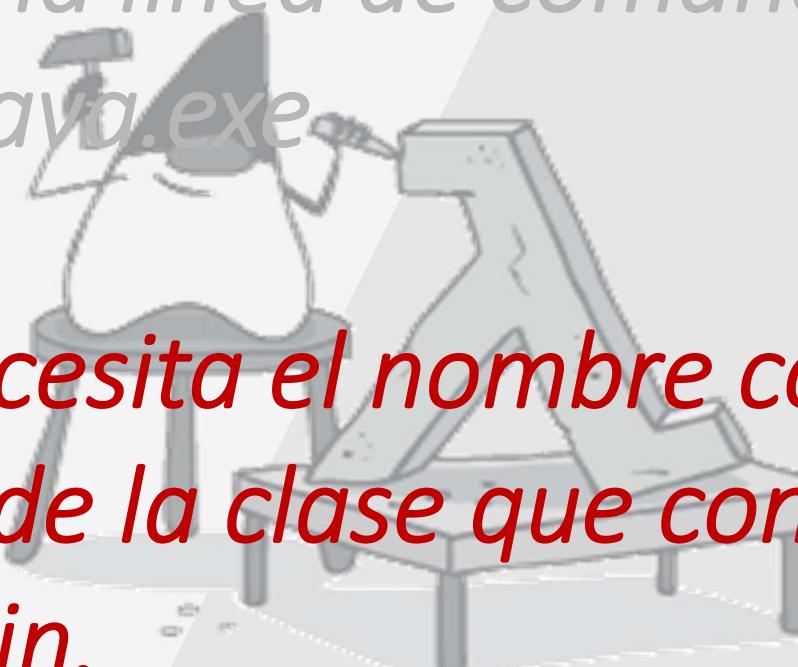
- El flag *-c*, indica creación.
- El flag *-v*, indica verbose.
- El flag *-f*, indica el nombre del fichero jar.
- A continuación se especifican las clases o directorios que contienen clases que se agregarán al fichero jar.



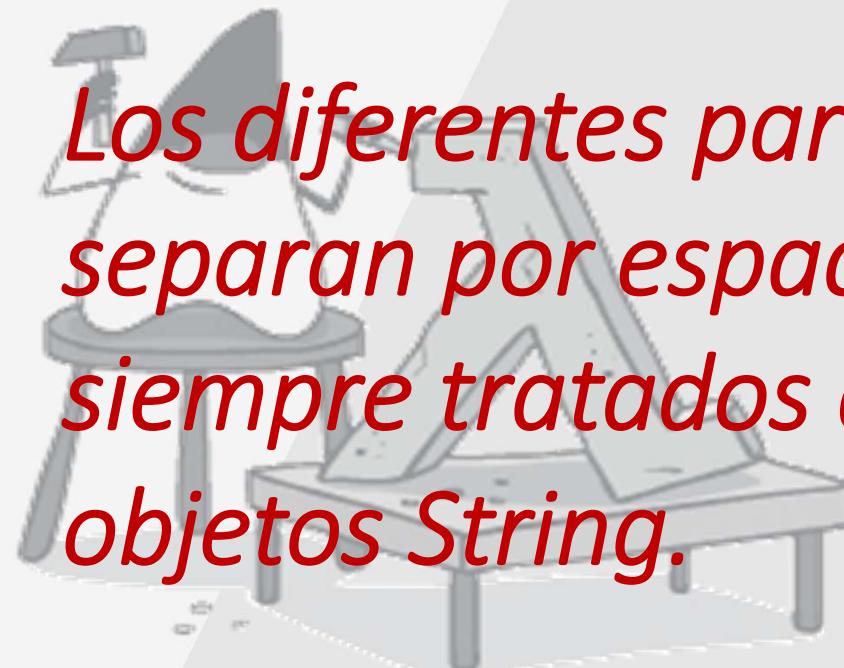
# Linea de Comando

→ A través de la linea de comando se invoca la aplicación `java.exe`

→ `java.exe` necesita el nombre completo cualificado de la clase que contiene un método `main`.



# Linea de Comando

A grayscale illustration of a telescope mounted on a tripod, pointing upwards. The telescope has a large objective lens and a smaller eyepiece lens. The tripod legs are extended downwards.

*Los diferentes parámetros se separan por espacio y son siempre tratados como objetos String.*

# Primitivos VS Referencias

- Los tipos primitivos son realmente variables tipadas, que almacenan valores. ( int, double, etc..)
- Las referencias son otro tipo de variables también tipadas que “apuntan” a un objeto.

# Primitivos VS Referencias

→ Una referencia apuntará a un objeto bien por ser asignada a otra referencia o bien mediante un método constructor.



# Primitivos VS Referencias



*Las referencias equivalen a punteros en otros lenguajes, pero con la salvedad que en java NO es posible manipular directamente la memoria.*

# Primitivos VS Referencias

→ *En java los arrays son realmente objetos, por tanto se disponen de 8 tipos diferentes de arrays, uno por cada tipo primitivo más uno adicional para almacenar Objects.*

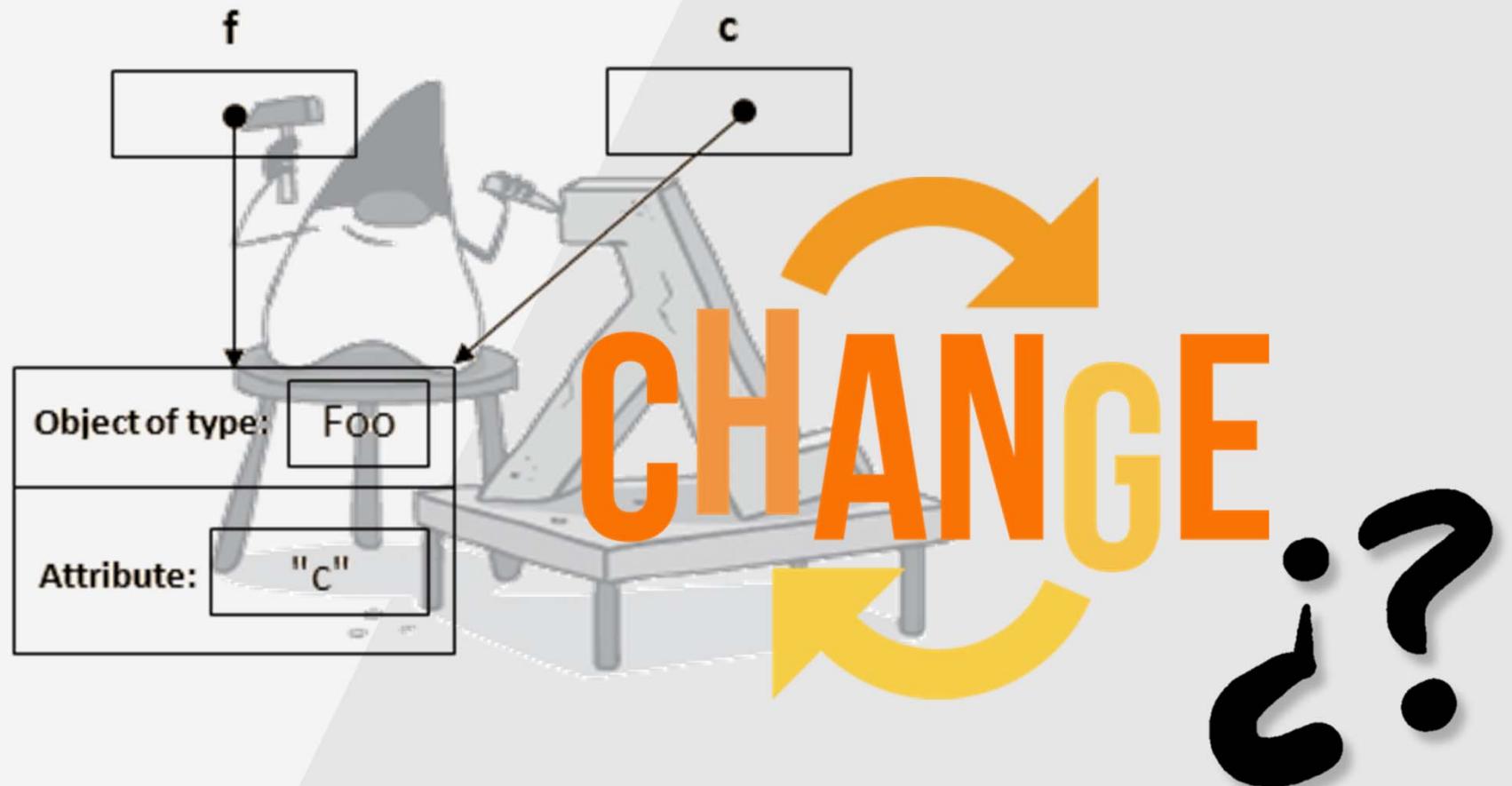
# Primitivos VS Referencias

- Existe un tipo de dato especial denominado *null*, que se utiliza para indicar que una referencia no apunta a nada.
- Las variables no pueden apuntar a null.
- No es posible crear tipos de datos null.

# Paso de Parámetros

- En java siempre los parámetros se pasan por valor, no existe el paso por referencia.
- Tanto los primitivos como las referencias originales permanecen inalteradas. Se pasa siempre una copia.

# Paso de Parámetros



# Paso de Parámetros

→ Los objetos en un método se pueden alterar porque la copia de la referencia contiene el mismo valor ( posición de memoria ) que la referencia original, por tanto acceden al mismo objeto.

# Paso de Parámetros

- También la devolución de valores ( return ), se realiza siempre por valor.
- Las llamadas entre métodos dejan los parámetros en una pila, y a la hora de volver de la llamada son desapilados.

# Garbage Collector

- Los objetos una vez creados se alojan en una zona especial de memoria denominada heap.
- Este heap tiene un tamaño máximo predefinido en función del entorno de ejecución y puede desbordarse.

# Garbage Collector

→ *El garbage collector es un proceso java que se ejecuta en todas las JVM, cuya función es destruir los objetos que ya no van a ser usados, es decir, aquellos que no están apuntados por referencia alguna.*

# Garbage Collector

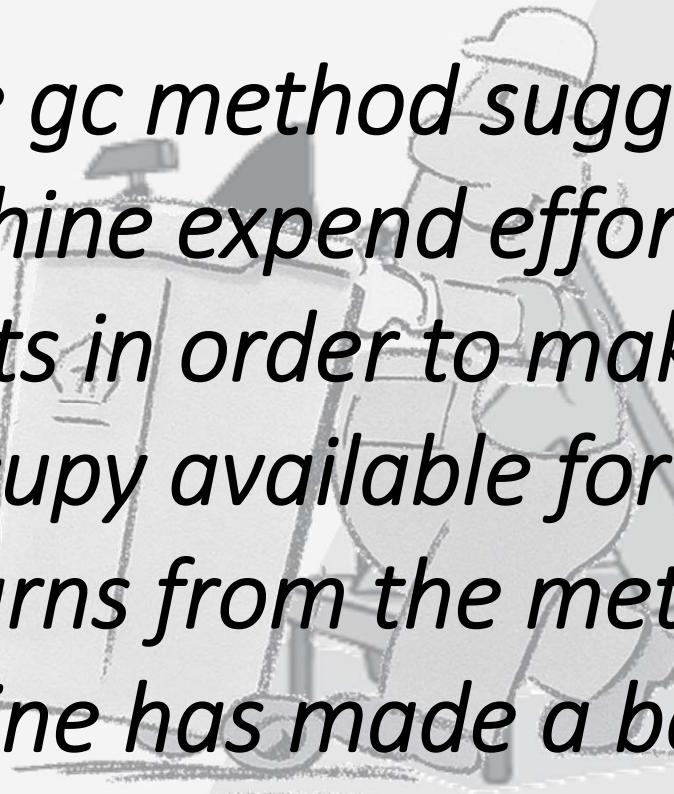
→ *No es posible saber cuando pasará el garbage collector para liberar memoria, ya que la especificación de la JVM, no lo indica, sólo especifica que cuando sea un objeto seleccionado por este la memoria debe ser liberada.*

# Garbage Collector

System.gc()



# Garbage Collector

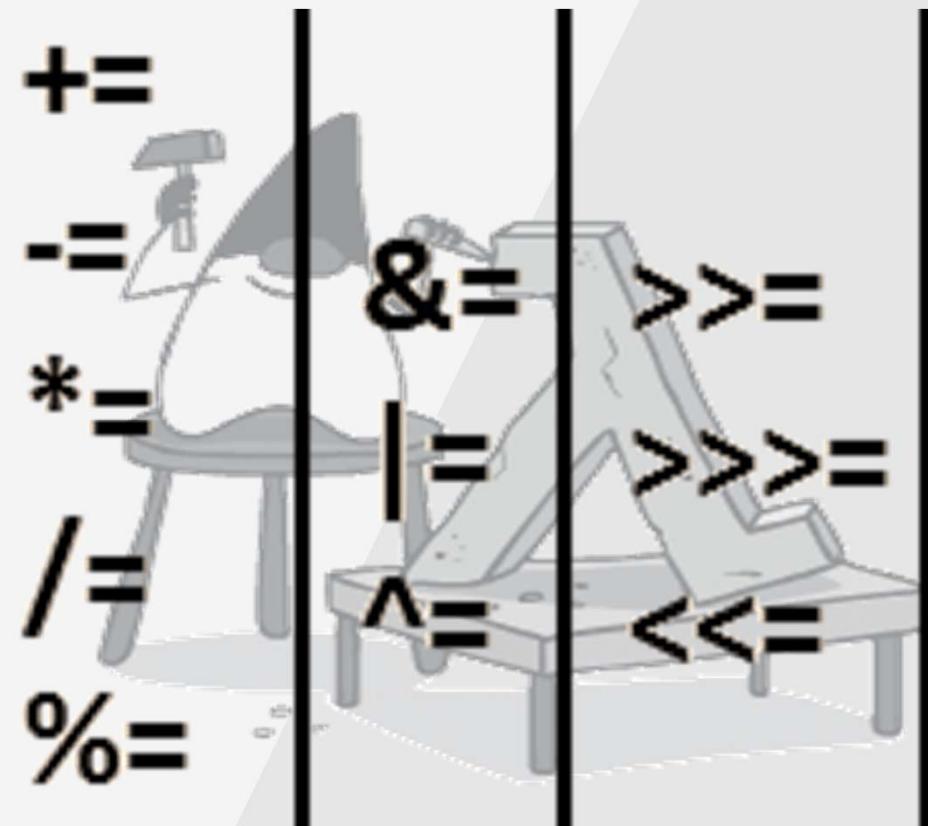


*“Calling the gc method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made a best effort to reclaim space from all discarded objects.”*

# Garbage Collector

- Antes de liberar memoria en relación a un objeto garbage collector invoca siempre al método de `object finalize();`
- Es una buena práctica que cuando se sobreescribe `finalize` se invoque a `super.finalize()`, pero se debe declarar `Throwable`.

# Operadores



# Operadores



# Operadores

*Post-increment/post-decrement*

*Pre-increment/pre-decrement*

*Unary operators*

*Multiplication/division/modulus*

*Addition/subtraction*

*Shift operators*

*Relational operators*

*Equal to/not equal to*

*Bitwise AND, exclusive OR, inclusive OR*

*Logical AND, OR*

*Ternary operator*

*Assignment operators*

*expression++, expression--*

*++expression, --expression*

*+, -, ~, !*

*\**

*%*

*<, >, >>*

*<<, >>, >>>*

*<<=, >>=, >>>=*

*, instanceof*

*- &, ^, |*

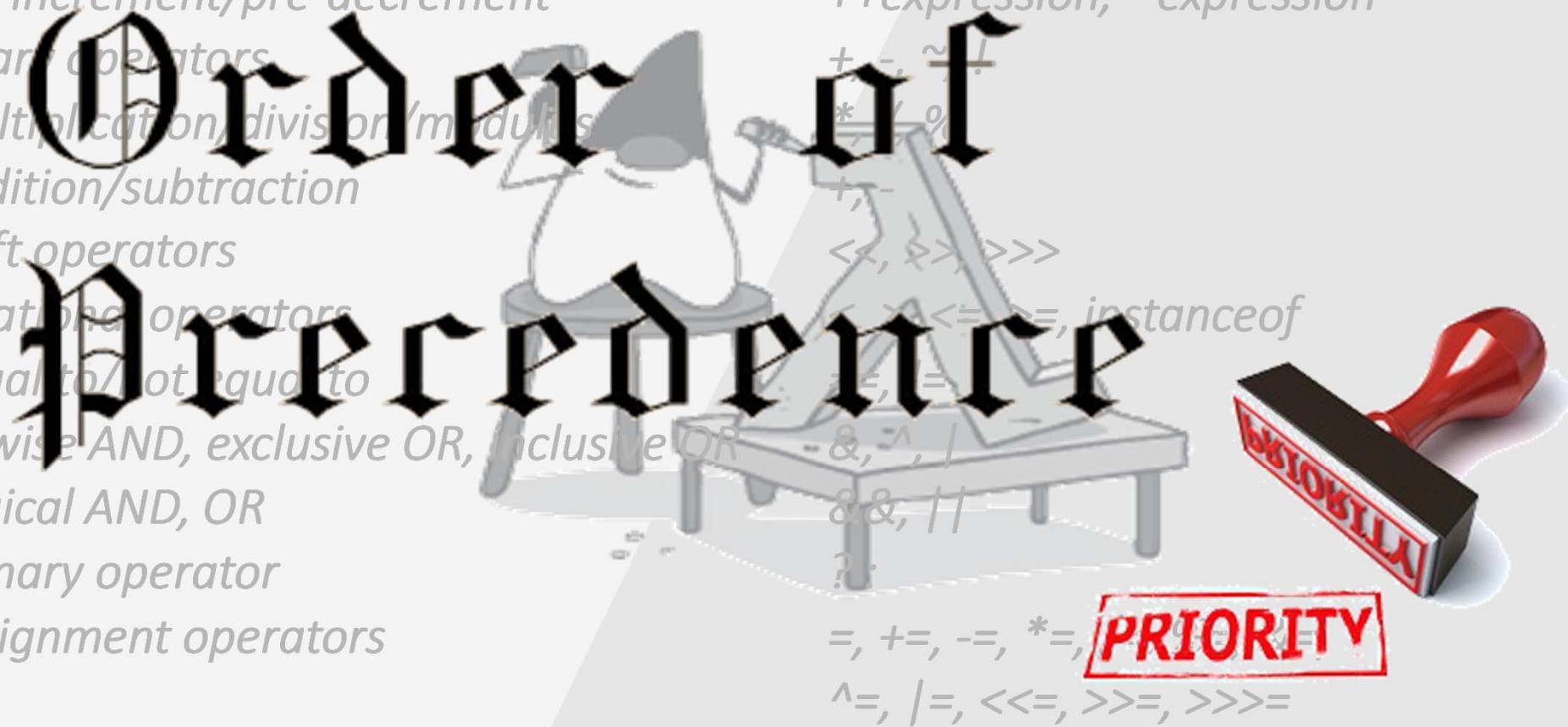
*&&, ||*

*? :*

*=, +=, -=, \*=,*

*PRIORITY*

*^=, /=, <<=, >>=, >>>=*



# Operadores

*Post-increment/post-decrement*

*Pre-increment/pre-decrement*

*Unary operators*

*Multiplication/division/modulus*

*Addition/subtraction*

*Shift operators*

*Relational operators*

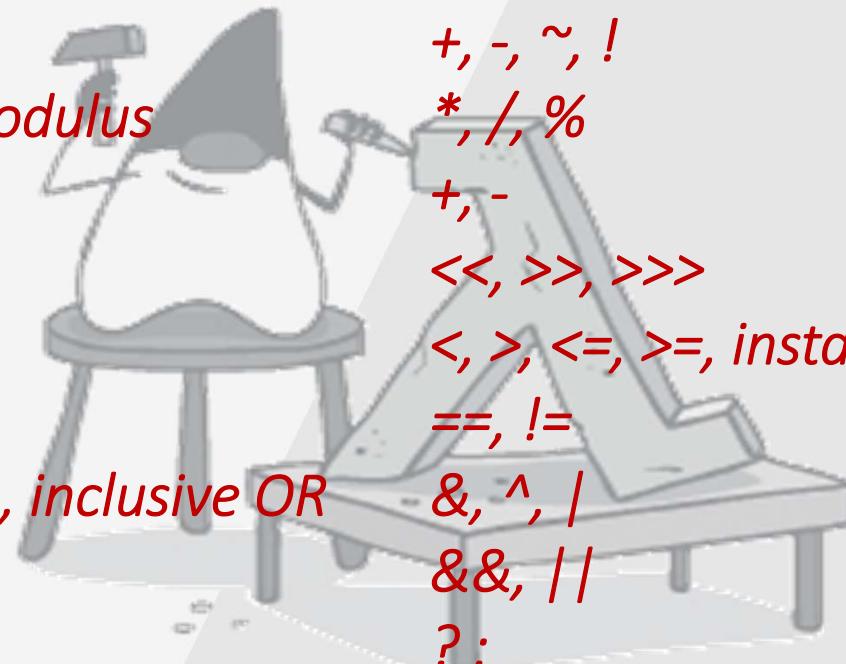
*Equal to/not equal to*

*Bitwise AND, exclusive OR, inclusive OR*

*Logical AND, OR*

*Ternary operator*

*Assignment operators*



*expression++, expression--*

*++expression, --expression*

*+, -, ~, !*

*\*, /, %*

*+, -*

*<<, >>, >>>*

*<, >, <=, >=, instanceof*

*==, !=*

*&, ^, |*

*&&, ||*

*? :*

*=, +=, -=, \*=, /=, %=, &=,*

*^=, |=, <<=, >>=, >>>=*

# Operadores

Post-increment  
Pre-increment  
Unary operators  
Multiplication  
Addition  
Shift operators  
Relational operators  
Equal to  
Bitwise operators  
Logical operators  
Ternary operators  
Assignment operators

## Examen

*Assignment operators:*

=, += and -=

*Arithmetic operators:*

+, -, \*, /, %, ++, and --

*Relational operators:*

<, <=, >, >=, ==, and !=

*The instance operator:*

instanceof

*Bitwise and logical operators:*

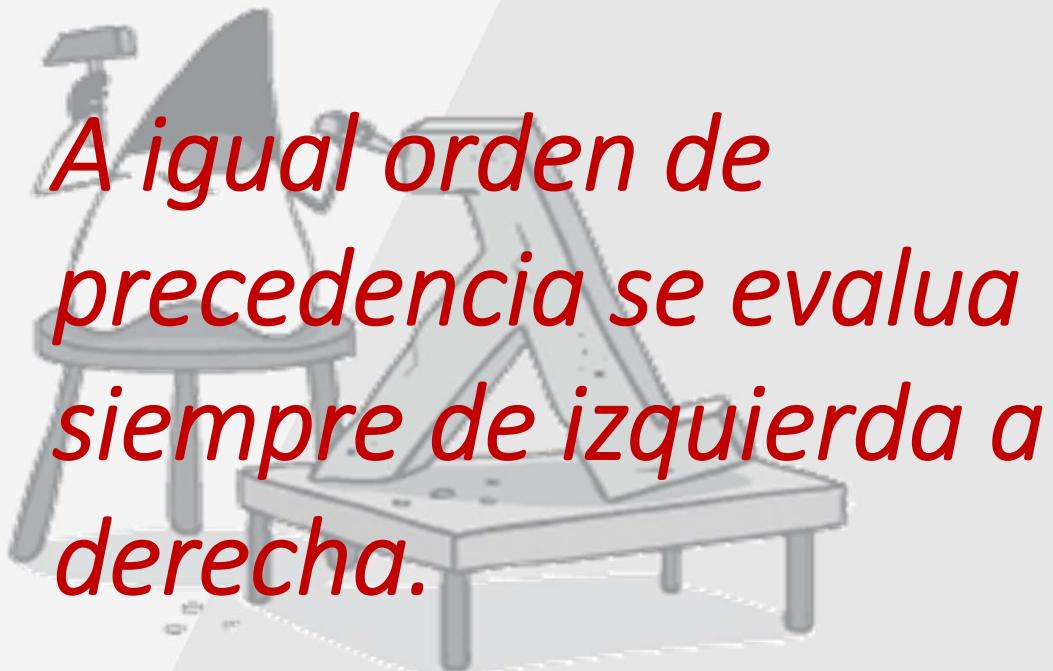
&, |, ^, !, &&, and ||

*The ternary operator*

(?:)

, , , , , , , , ,  
^=, |=, <<=, >>=, >>>=

# Operadores



*A igual orden de precedencia se evalua siempre de izquierda a derecha.*

# Operadores

*“Corto-circuito” de operadores, si la evaluación de operador determina el resultado conjunto de una operación, se para de evaluar.*





# ORACLE

The screenshot shows the Oracle University homepage. At the top, there's a navigation bar with links for "Training Search", "Training", "Certification", "Community", "Cart", "Help", and a phone number "1.800.529.0165". The main banner features the text "Secure Your Database Now" and "Develop Value-Max Oracle Expertise". It includes a large red "ORACLE" logo and a call-to-action button "Act Now". Below the banner, there are four main service offerings: "Take Database Security Courses", "Develop Trusted Training Expertise.", "New HR & Talent Mgmt Subscriptions.", and "Cloud Learning Subscription". The central part of the page has a large "UNIVERSITY" logo. At the bottom, there are three sections: "Get Training", "Get Certified", and "Get Connected".

<https://education.oracle.com>

# ORACLE

**ORACLE® UNIVERSITY**

Welcome Noel ( Sign Out | Account )   [Request Information](#)   United States ▾

Training Search   Training ▾ Certification ▾ Community ▾  Help ▾ [1.800.529.0165](#)

[Home](#)

## Secure Your Database Now Develop Valuable Oracle Expertise

Learn how to protect data against increasing insider and external threats, while ensuring regulatory compliance.

[Act Now ▶](#)



**Take Database Security Courses**  
Reduce Risk, Protect data.

**Explore Oracle E-Business Suite**  
Develop Trusted Training Expertise.

**Oracle Cloud Learning Subscriptions**  
New HR & Talent Mgmt Subscriptions.

**Oracle Learning Subscription**  
Learning Simplified.

**Oracle Business Intelligence Training:**  
Gain Visibility into Your Business Performance

**PeopleSoft Training**  
Use applications built for your critical information

**Oracle Hyperion Training**  
Unlock new financial insights

**Oracle Cloud Learning Subscriptions**  
24/7 Cloud Learning delivered by Oracle experts

**Get Training**  
Find training related to specific products or technology.  
[Training by Product or Technology](#)  
[Why Oracle University?](#)  
[View all Products](#)

**Get Certified**  
Find the training and exams you need to get certified.  
[Certification by Product or Technology](#)  
[Why Get Oracle Certified?](#)  
[View all Certifications](#)  
[New! Certification Finder](#)

**Get Connected**  
Want to stay informed?  
Connect with Oracle University to learn more about local special offers and packages:  
[Subscribe to Oracle University Newsletters](#)  
[Check out the OU Blog](#)  
[Create Your Free Personalized Catalog](#)  
[Join Oracle University Communities](#)

# ORACLE

Training Courses    Certification    Learning Paths    Packages & Offers

## Java EE Certification

Click on the boxes below to learn detailed requirements for achieving each certification. [?](#)

Professional	Oracle Certified Professional, Java EE 5 Web Component Developer	Oracle Certified Professional, Java EE 5 Business Component Developer	Oracle Certified Professional, Java EE 5 Web Services Developer
Master	Oracle Certified Master, Java EE 5 Enterprise Architect	Oracle Certified Master, Java EE 6 Enterprise Architect	

Certified Expert	Oracle Certified Expert, Java EE 6 Web Component Developer	Oracle Certified Expert, Java Platform, Enterprise Edition 6 Enterprise JavaBeans Developer	Oracle Certified Expert, Java EE 6 Web Services Developer
	Oracle Certified Expert, Java EE 6 Java Persistence API Developer		Oracle Certified Expert, Java EE 6 JavaServer Faces Developer

The Oracle Certified Expert certification program grants credentials that recognize competency in specific technologies, architectures or domains not currently covered in the path-based Oracle Certifications.

# EXAMEN

**PEARSON** ALWAYS LEARNING

**PEARSON VUE**

Contact us Language ▾  🔍

[Home](#) | [For test takers](#) | [For test owners](#) | [Become a test center](#) | [About Pearson VUE](#)



Schedule or  
Re-schedule an exam.  
Locate a test center.

[Test takers - get started now!](#)

< >

• • • •

[Recent news](#)  
Majority of Indian employers compromise when hiring talent, reveals new survey

[View all news](#)

**Test takers: schedule an exam and more**



[get started now!](#)

At the [Test taker site](#) find information about your testing program, test center locations, FAQs,

**You and Pearson VUE**  
Making a world of difference



*Test anywhere. Succeed everywhere.*

[Test takers](#) | [Test owners](#) | [Test centers](#)

**Pearson VUE named MCP Practice Test Provider**



Visit [mindhub](#) to purchase a Microsoft Official Practice Test

[Read the press release](#)

# EXAMEN

**PEARSON** ALWAYS LEARNING

**PEARSON VUE**

Contact us  Language  Search 

[Home](#) | [For test takers](#) | [For test owners](#) | [Become a test center](#) | [About Pearson VUE](#)



Schedule or  
Re-schedule an exam.  
Locate a test center.

 [Test takers - get started now!](#)

Recent news

Test takers: schedule an exam and more

You and Pearson VUE

Making a world of difference

Pearson VUE named MCP Practice Test Provider

Visit mindhub to purchase a Microsoft Official Practice Test



At the [Test taker site](#) find information about your testing program, test center locations, FAQs,

[Test takers](#) | [Test owners](#) | [Test centers](#)

[Read the press release](#)

**<http://home.pearsonvue.com/>**

# EXAMEN

## Certification Exam Preparation Seminar: Java SE 6 Programmer

Schedule/Purchase	Training Formats 	Price	Duration	Course Materials	Language
 View Details	Training On Demand	US\$ 480	1 Days	English	English
 Close					

Training On Demand

Product Version	Price	Instruction Language	Materials
 Buy Now Java SE6	US\$ 480	English	English
 Close			

**Suggested Prerequisites:**

- ▶ Foundational knowledge of Java Programmer

# EXAMEN

## What You Will Learn

## Audience

## Course Topics

## Course Objectives

## Related Training

This video seminar – Certification Exam Preparation Seminar: Java SE 6 Programmer will help you significantly in your preparation to take any of the following Oracle Certification exams:

- ▶ [Java Standard Edition 5 Programmer \(Exam # 1Z0-853\)](#)
- ▶ [Java Standard Edition 6 Programmer \(1Z0-851\)](#)

Become ready to take and pass one of the required exams for Oracle Certified Professional, Java SE 6 or SE 5 Programmer Certification with this comprehensive Training On Demand Exam 'prep' seminar. Led by one of our Oracle Java expert instructors, this fast-paced video-based seminar will help increase your confidence by providing you with exclusive tips and strategies to prepare you to take the certification exam.

This Exam Preparation Seminar is intended to help those with a solid foundation in Java Programming. It provides a thorough review of the exam objectives and help students understand the breadth of topics and skills required to pass the exams listed above.

Exam Preparation Seminars are fast-paced video-based reviews that include primarily slides, lecture and in some cases simple demos. Exam Prep Seminars do not include: an Oracle University eKit, expert video, labs, student environment, simulations, student Q&A. Additionally this seminar does not meet the Hands-on Course requirement (if applicable to your certification track).

The seminar may be accessed for repeated review as needed during subscription term.



**Save On Certification  
Value Packages**

# EXAMEN

What You Will Learn

Audience

Course Topics

Course Objectives

Related Training

- ▶ Inner and Nested Classes
- ▶ Review Key Java Language Rules
- ▶ Compiler and CLASSPATH
- ▶ Packages
- ▶ Exceptions and Errors
- ▶ Casting References
- ▶ Flow Control
- ▶ Collections
- ▶ The I/O API
- ▶ Internationalization
- ▶ Threads

# EXAMEN

Exam Number :	<b>1Z0-851</b>	Duration:	<b>150 minutes</b>
Associated Certifications:	<a href="#"><b>Oracle Certified Professional, Java SE 6 Programmer</b></a>	Number of Questions:	<b>60</b>
Exam Product Version:	<b>Java SE,</b>	Passing Score	<b>61%</b> <a href="#"><u>View passing score policy</u></a>
Exam Price:	<b>US\$ 245</b> <a href="#"><u>More on exam pricing</u></a>	Validated Against:	<b>This exam has been validated against SE 5 and SE 6.</b>
		format:	<b>Multiple Choice</b>

# EXAMEN

Exam Number :	1Z0-851	Duration:	150 minutes
Associated Certifications:	<a href="#">Oracle Certified Professional, Java SE 6 Programmer</a>	Number of Questions:	60
Exam Product Version:	Java SE,	Passing Score	61% <a href="#">View passing score policy</a>
Exam Price:	US\$ 245 <a href="#">More on exam pricing</a>	Validated Against:	This exam has been validated against SE 5 and SE 6.
		format:	Multiple Choice

# EXAMEN

Exam Number :	1Z0-851	Duration:	150 minutes
Associated Certifications:	<a href="#">Oracle Certified Professional, Java SE 6 Programmer</a>	Number of Questions:	60
Exam Product Version:	Java SE,	Passing Score	61% <a href="#">View passing score policy</a>
Exam Price:	US\$ 245 <a href="#">More on exam pricing</a>	Validated Against:	This exam has been validated against SE 5 and SE 6.
		format:	Multiple Choice

# EXAMEN

Exam Number :	1Z0-851	Duration:	150 minutes
Associated Certifications:	<a href="#">Oracle Certified Professional, Java SE 6 Programmer</a>	Number of Questions:	60
Exam Product Version:	Java SE,	Passing Score	61% <a href="#">View passing score policy</a>
Exam Price:	US\$ 245 <a href="#">More on exam pricing</a>	Validated Against:	<a href="#">This exam has been validated against SE 5 and SE 6.</a>
		format:	Multiple Choice

# Próxima Parada...



