

# Programación Orientada a Objetos

Objetos aspectos finales

CEIS

2023-2

# Agenda

## Desde los conceptos

- Clase

- Objetos

- Igualdad

## Modificadores

- De acceso

- De pertenencia

- De mutabilidad

## Tres conceptos

- Encapsulamiento

- Ocultación

- Sobrecarga

## Buenas prácticas

- .

## Opcionales

- Detalle código

- Dice

# Agenda

## Desde los conceptos

- Clase

- Objetos

- Igualdad

## Modificadores

- De acceso

- De pertenencia

- De mutabilidad

## Tres conceptos

- Encapsulamiento

- Ocultación

- Sobrecarga

## Buenas prácticas

- .

## Opcionales

- Detalle código

- Dice

# Clases

## Clase (en software)

Una **clase** define las características - atributos y métodos - que cada objeto que pertenece a la clase posee. Puede ser visto como un molde.

Clase =

```
public class Square {
```

```
}
```

# Clases

## Clase (en software)

Una **clase** define las características - atributos y métodos - que cada objeto que pertenece a la clase posee. Puede ser visto como un molde.

Clase = Atributos

```
public class Square {  
    private int size;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;  
  
    ...  
}
```

# Lenguaje. Java. Datos.

## Simples

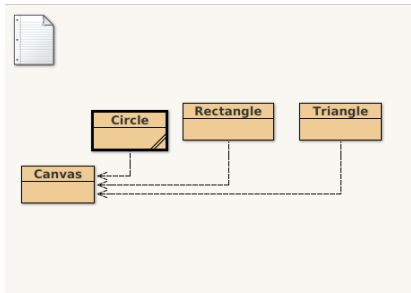
- For `byte`, from -128 to 127, inclusive
- For `short`, from -32768 to 32767, inclusive
- For `int`, from -2147483648 to 2147483647, inclusive
- For `long`, from -9223372036854775808 to 9223372036854775807, inclusive
- The floating-point types are `float` and `double`, which are conceptually associated with the single-precision 32-bit and double-precision 64-bit
- For `char`, from `'\u0000'` to `'\uffff'` inclusive, that is, from 0 to 65535
- `boolean` type represents a logical quantity with two possible values, indicated by the literals `true` and `false`

## Estructurador simple

```
int[] ai;           // array of int
short[][] as;       // array of array of short
Object[] ao;        // array of Object
```

# Lenguaje. Reuso.

## Propios. Shapes



## API. Java

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | M

java.lang

### Class Object

java.lang.Object

```
public class Object
```

Class Object is the root of the arrays, implement the methods

Since:  
JDK1.0

See Also:  
[Class](#)

#### Constructor Summary

Constructors
Constructor and Description

All Classes (Java Platform SE 8 x +)

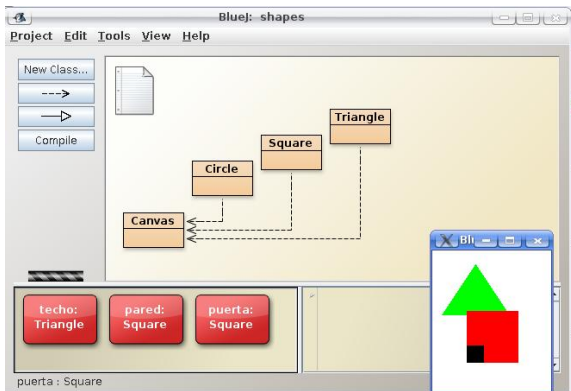
docs.oracle.com/javase/8/docs

All Classes

- AbstractAction
- AbstractAnnotationValueVisitor6
- AbstractAnnotationValueVisitor7
- AbstractAnnotationValueVisitor8
- AbstractBorder
- AbstractButton
- AbstractCellEditor
- AbstractChronology
- AbstractCollection
- AbstractColorChooserPanel
- AbstractDocument
- AbstractDocument.AttributeContext
- AbstractDocument.Content
- AbstractDocument.ElementEdit
- AbstractElementVisitor6
- AbstractElementVisitor7
- AbstractElementVisitor8
- AbstractExecutorService
- AbstractInterruptibleChannel
- AbstractLayoutCache
- AbstractLayoutCache.NodeDimensions
- AbstractList
- AbstractListModel

# Nueva Clase

Clase = Atributos



¿?

1. ¿Atributos Casa?



# Clases

## Clase (en software)

Una **clase** define las características - atributos y métodos - que cada objeto que pertenece a la clase posee. Puede ser visto como un molde.

Clase = Atributos + Métodos

```
public class Square{
    ...

    /**
     * Create a new square at default position with default color.
     */
    public Square(){
        ....
    }
    /**
     * Make this square visible. If it was already visible, do nothing.
     */
    public void makeVisible(){
        ..
    }
    /**
     * Move the square horizontally by 'distance' pixels.
     */
    public void moveHorizontal(int distance){

    }
    /**
     * Returns the length of this square.
     */
    public int size(){
```

# Lenguaje. Java. Instrucciones.

## Simples

```
short x = 3;  
x += 4.6;
```

```
int i = 2;  
int j = (i=3) * i;  
System.out.println(j);
```

## Condicionales

### 7.4 if Statements

```
if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
} else {  
    statements;  
}
```

### 7.8 switch Statements

```
switch (condition) {  
    case ABC:  
        statements;  
        /* falls through */  
    case DEF:  
        statements;  
        break;  
    default:  
        statements;  
        break;  
}
```

```
alpha = (aLongBooleanExpression) ? beta : gamma;
```

## Iterativas

### 7.5 for Statements

```
for (initialization; condition; update) {  
    statements;  
}
```

### 7.6 while Statements

```
while (condition) {  
    statements;  
}
```

### 7.7 do-while Statements

```
do {  
    statements;  
} while (condition);
```

# Lenguaje. Reuso.

## Square

### Class Square

[java.lang.Object](#)  
↳ **Square**

```
public class Square  
extends Object
```

A square that can be manipulated and that draws itself on a canvas.

**Version:**  
1.0 (15 July 2000)

**Author:**  
Michael Kolling and David J. Barnes

### Constructor Summary

[Square\(\)](#)  
Create a new square at default position with default color.

### Method Summary

```
void changeColor(String newColor)  
    Change the color.  
  
void changeSize(int newSize)  
    Change the size to the new size (in pixels).  
  
void makeInvisible()  
    Make this square invisible.  
  
void makeVisible()  
    Make this square visible.
```

## API. String

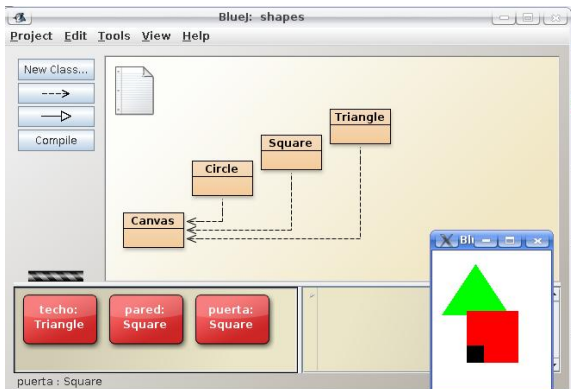
The screenshot shows the Java API documentation for the `String` class. The left sidebar lists various classes and interfaces, with `String` selected. The main content area displays the `String` class documentation, including a table of methods.

The `String` class represents character strings. The `String` class represents character strings.

Modifier and Type	Method and Description
char	<code>charAt(int index)</code> Returns the char value at the specified index.
int	<code>codePointAt(int index)</code> Returns the character (Unicode code point) at the specified index.
int	<code>codePointBefore(int index)</code> Returns the character (Unicode code point) before the specified index.
int	<code>codePointCount(int beginIndex, int endIndex)</code> Returns the number of Unicode code points in the specified range.
int	<code>compareTo(String anotherString)</code> Compares two strings lexicographically.
int	<code>compareToIgnoreCase(String anotherString)</code> Compares two strings lexicographically, ignoring case.
String	<code>concat(String str)</code> Concatenates the specified string to the end of this string.

# Nueva Clase

Clase = Atributos



¿?

1. ¿Casa slowMoveVertical(int distance) ?

# Clases- Tres vistas

## Diseño

Square
<ul style="list-style-type: none"><li>- size : int</li><li>- xPosition : int</li><li>- yPosition : int</li><li>- color : String</li><li>- isVisible : boolean</li></ul>
<ul style="list-style-type: none"><li>+ Square()</li><li>+ makeVisible() : void</li><li>+ makeInvisible() : void</li><li>+ moveRight() : void</li><li>+ moveLeft() : void</li><li>+ moveUp() : void</li><li>+ moveDown() : void</li><li>+ moveHorizontal(distance : int) : void</li><li>+ moveVertical(distance : int) : void</li><li>+ slowMoveHorizontal(distance : int) : void</li><li>+ slowMoveVertical(distance : int) : void</li><li>+ changeSize(newSize : int) : void</li><li>+ changeColor(newColor : String) : void</li><li>- draw() : void</li><li>- erase() : void</li></ul>

## Código

```
/**
 * A square that can be manipulated and that draws itself on a canvas.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 1.0 (15 July 2000)
 */

public class Square
{
    private int size;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new square at default position with default color.
     */
    public Square() {
        size = 50;
        xPosition = 60;
        yPosition = 50;
        color = "red";
        isVisible = false;
    }

    /**
     * Change the color. Valid colors are "red", "yellow", "blue", "green",
     * "magenta" and "black".
     */
    public void changeColor(String newColor) {
        color = newColor;
        draw();
    }
}
```

## Documentación

### Class Square

[Java class Square](#)  
[in Square](#)

public class Square

extends [Object](#)

A square that can be manipulated and that draws itself on a canvas.

#### Version:

1.0 (15 July 2000)

#### Author:

Michael Kolling and David J. Barnes

### Constructor Summary

[Square\(\)](#)

Create a new square at default position with default color.

### Method Summary

void [changeColor\(String newColor\)](#)

Change the color.

void [changeSize\(int newSize\)](#)

Change the size to the new size (in pixels).

void [makeInvisible\(\)](#)

Make this square invisible.

void [makeVisible\(\)](#)

Make this square visible.

# Agenda

## Desde los conceptos

Clase

Objetos

Igualdad

## Modificadores

De acceso

De pertenencia

De mutabilidad

## Tres conceptos

Encapsulamiento

Ocultación

Sobrecarga

## Buenas prácticas

.

## Opcionales

Detalle código

Dice

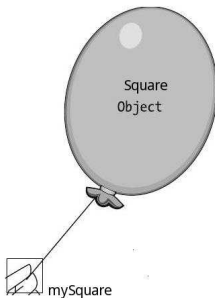
# Objetos

## Objeto (en software)

Un **objeto** es un artefacto software que representa una abstracción de un objeto del mundo real por medio de su estado (atributos) y comportamiento (métodos).

## Objeto

```
mySquare = new Square();
```



# Cuadrado

```
Square derecho=new Square();  
-->1  
Square izquierdo;  
-->2  
izquierdo=derecho;  
-->3  
izquierdo=new Square();  
-->4
```

```
public class Square {  
    private int size;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;  
  
    ...  
}
```

```
public class Square  
{  
    private int size;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;  
  
    /**  
     * Create a new square at default position with default color.  
     */  
    public Square()  
    {  
        size = 28;  
        xPosition = 68;  
        yPosition = 58;  
        color = "red";  
        isVisible = false;  
    }  
}
```

¿Cómo está la memoria? 1,2,3



# Agenda

## Desde los conceptos

Clase

Objetos

**Igualdad**

## Modificadores

De acceso

De pertenencia

De mutabilidad

## Tres conceptos

Encapsulamiento

Ocultación

Sobrecarga

## Buenas prácticas

.

## Opcionales

Detalle código

Dice

# La igualdad

==

- ▶ Son iguales si tienen el mismo valor
- ▶ En referencias (direcciones), esto implica que apunten al mismo sitio

## *equals*

- ▶ Son iguales si, para nosotros, tienen el mismo valor
- ▶ Normalmente, es necesario definirlo de manera adecuada
- ▶ Si no se define, por defecto es ==

---

### **equals**

```
public boolean equals(Object obj)
```

Indicates whether some other object is "equal to" this one.

The equals method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x == y` has the value `true`).

#### **Parameters:**

`obj` - the reference object with which to compare.

#### **Returns:**

`true` if this object is the same as the `obj` argument; `false` otherwise.

# La igualdad

## Cuadrados con $c1 == c2$

```
/**
 * A square that can be manipulated and that draws itself on a canvas.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 1.0 (15 July 2000)
 */
public class Square
{
    private int size;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new square at default position with default color.
     */
    public Square() {
        size = 30;
        xPosition = 60;
        yPosition = 50;
        color = "red";
        isVisible = false;
    }

    /**
     * Change the color. Valid colors are "red", "yellow", "blue", "green",
     * "magenta" and "black".]
     */
    public void changeColor(String newColor) {
        color = newColor;
        draw();
    }
}
```

```
Square derecho = new Square();
Square izquierdo = new Square();
```

¿?

1. ¿ Cuándo dos cuadrados son iguales?

# La igualdad

## Cuadrados con $c1 == c2$

```
/**
 * A square that can be manipulated and that draws itself on a canvas.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 1.0 (15 July 2000)
 */
public class Square
{
    private int size;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new square at default position with default color.
     */
    public Square() {
        size = 30;
        xPosition = 60;
        yPosition = 50;
        color = "red";
        isVisible = false;
    }

    /**
     * Change the color. Valid colors are "red", "yellow", "blue", "green",
     * "magenta" and "black".
     */
    public void changeColor(String newColor) {
        color = newColor;
        draw();
    }
}
```

```
Square derecho = new Square();
Square izquierdo = new Square();
```

¿?

1. ¿ Cuándo dos cuadrados son iguales?
2. ¿ Es posible que dos cuadrados recién creados sean iguales? ¿Cómo?
3. ¿Es posible lograr que esos dos objetos sean iguales? ¿Cómo?
4. ¿Es posible lograr que las dos variables sean iguales? ¿Cómo?

# La igualdad

## Definiendo `c1.equals(c2)`

```
/**
 * A square that can be manipulated and that draws itself on a canvas.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 1.0 (15 July 2000)
 */
public class Square
{
    private int size;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new square at default position with default color.
     */
    public Square() {
        size = 30;
        xPosition = 60;
        yPosition = 50;
        color = "red";
        isVisible = false;
    }

    /**
     * Change the color. Valid colors are "red", "yellow", "blue", "green",
     * "magenta" and "black".
     */
    public void changeColor(String newColor) {
        color = newColor;
        draw();
    }
}
```

```
Square derecho = new Square();
Square izquierdo = new Square();
```

¿?

1. ¿ Cuándo queremos que dos cuadrados sean iguales?

# La igualdad

## Definiendo `c1.equals(c2)`

```
/**
 * A square that can be manipulated and that draws itself on a canvas.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 1.0 (15 July 2000)
 */
public class Square
{
    private int size;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new square at default position with default color.
     */
    public Square() {
        size = 30;
        xPosition = 60;
        yPosition = 50;
        color = "red";
        isVisible = false;
    }

    /**
     * Change the color. Valid colors are "red", "yellow", "blue", "green",
     * "magenta" and "black".
     */
    public void changeColor(String newColor) {
        color = newColor;
        draw();
    }
}
```

```
Square derecho = new Square();
Square izquierdo = new Square();
```

¿?

1. ¿ Cuándo queremos que dos cuadrados sean iguales?
2. ¿ Es posible que dos cuadrados recién creados sean iguales? ¿Cómo?
3. ¿Es posible lograr que esos dos objetos sean iguales? ¿Cómo?
4. ¿Es posible lograr que las dos variables sean iguales? ¿Cómo?

# La igualdad

## Definiendo `c1.equals(c2)`

```
/**
 * A square that can be manipulated and that draws itself on a canvas.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 1.0 (15 July 2000)
 */
public class Square
{
    private int size;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new square at default position with default color.
     */
    public Square() {
        size = 30;
        xPosition = 60;
        yPosition = 50;
        color = "red";
        isVisible = false;
    }

    /**
     * Change the color. Valid colors are "red", "yellow", "blue", "green",
     * "magenta" and "black".
     */
    public void changeColor(String newColor) {
        color = newColor;
        draw();
    }
}
```

```
Square derecho = new Square();
Square izquierdo = new Square();
```

¿?

1. ¿ Cuándo queremos que dos cuadrados sean iguales?
2. ¿ Es posible que dos cuadrados recién creados sean iguales? ¿Cómo?
3. ¿Es posible lograr que esos dos objetos sean iguales? ¿Cómo?
4. ¿Es posible lograr que las dos variables sean iguales? ¿Cómo?
5. ¿Qué debemos implementar? Implementemos. En Square.

# Agenda

## Desde los conceptos

Clase

Objetos

Igualdad

## Modificadores

De acceso

De pertenencia

De mutabilidad

## Tres conceptos

Encapsulamiento

Ocultación

Sobrecarga

## Buenas prácticas

.

## Opcionales

Detalle código

Dice



# De acceso *(public|private)*


## Barreras

Client code may only access features that are declared to be public (typically, methods signatures).

**Public:**

Attributes:

Methods :

signatures() {  }

The internal details of method code bodies are effectively private.

**Private:**

Attributes:

Methods

... as are most attributes.

# De acceso (public|private)

## Circle

```
public class Circle{  
  
    public double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;
```

¿?

1. ¿ Quién puede usar PI? ¿Cómo?

# De acceso (public|private)

## Circle

```
public class Circle{  
    public double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;
```

¿?

1. ¿ Quién puede usar PI? ¿Cómo?
2. ¿ Quién puede usar diameter? ¿Cómo?

# De acceso (public|private)

## Circle

```
public class Circle{  
  
    public double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;
```

¿?

1. ¿ Quién puede usar PI? ¿Cómo?
2. ¿ Quién puede usar diameter? ¿Cómo?
3. Si queremos que otro componente consulte y modifique a diameter ? ¿Qué hacemos?

# De acceso *(public|private)*

## Circle

```
public Circle(){
    diameter = 30;
    xPosition = 20;
    yPosition = 15;
    color = "blue";
    isVisible = false;
}

public void makeVisible(){
    isVisible = true;
    draw();
}

public void makeInvisible(){
    erase();
    isVisible = false;
}

private void draw(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color,
            new Ellipse2D.Double(xPosition, yPosition,
                diameter, diameter));
        canvas.wait(10);
    }
}

private void erase(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.erase(this);
    }
}
```

¿?

1. ¿ Quién puede usar makeVisible? ¿Cómo?

# De acceso *(public|private)*

## Circle

```
public Circle(){
    diameter = 30;
    xPosition = 20;
    yPosition = 15;
    color = "blue";
    isVisible = false;
}

public void makeVisible(){
    isVisible = true;
    draw();
}

public void makeInvisible(){
    erase();
    isVisible = false;
}

private void draw(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color,
            new Ellipse2D.Double(xPosition, yPosition,
                diameter, diameter));
        canvas.wait(10);
    }
}

private void erase(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.erase(this);
    }
}
```

¿?

1. ¿ Quién puede usar makeVisible? ¿Cómo?
2. ¿ Quién puede usar draw? ¿Cómo?

# De acceso *(public|private)*

## Circle

```
public Circle(){
    diameter = 30;
    xPosition = 20;
    yPosition = 15;
    color = "blue";
    isVisible = false;
}

public void makeVisible(){
    isVisible = true;
    draw();
}

public void makeInvisible(){
    erase();
    isVisible = false;
}

private void draw(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color,
            new Ellipse2D.Double(xPosition, yPosition,
                diameter, diameter));
        canvas.wait(10);
    }
}

private void erase(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.erase(this);
    }
}
```

¿?

1. ¿ Quién puede usar makeVisible? ¿Cómo?
2. ¿ Quién puede usar draw? ¿Cómo?
3. Si queremos que otro componente use draw ? ¿Qué hacemos?

# De acceso *(public|private)*

## Circle

```
public Circle(){
    diameter = 30;
    xPosition = 20;
    yPosition = 15;
    color = "blue";
    isVisible = false;
}

public void makeVisible(){
    isVisible = true;
    draw();
}

public void makeInvisible(){
    erase();
    isVisible = false;
}

private void draw(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color,
            new Ellipse2D.Double(xPosition, yPosition,
                diameter, diameter));
        canvas.wait(10);
    }
}

private void erase(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.erase(this);
    }
}
```

¿?

1. ¿Quién puede usar makeVisible? ¿Cómo?
2. ¿Quién puede usar draw? ¿Cómo?
3. Si queremos que otro componente use draw ? ¿Qué hacemos?
4. ¿Por qué no necesitamos un draw público?



# De acceso *(public|private)*

## Circle

```
public Circle(){
    diameter = 30;
    xPosition = 20;
    yPosition = 15;
    color = "blue";
    isVisible = false;
}

public void makeVisible(){
    isVisible = true;
    draw();
}

public void makeInvisible(){
    erase();
    isVisible = false;
}

private void draw(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color,
            new Ellipse2D.Double(xPosition, yPosition,
                diameter, diameter));
        canvas.wait(10);
    }
}

private void erase(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.erase(this);
    }
}
```

¿?

1. ¿ Quién puede usar makeVisible? ¿Cómo?
2. ¿ Quién puede usar draw? ¿Cómo?
3. Si queremos que otro componente use draw ? ¿Qué hacemos?
4. ¿Por qué no necesitamos un draw público?
5. ¿Por qué es necesario que draw sea privado?

# Agenda

## Desde los conceptos

Clase

Objetos

Igualdad

## Modificadores

De acceso

**De pertenencia**

De mutabilidad

## Tres conceptos

Encapsulamiento

Ocultación

Sobrecarga

## Buenas prácticas

.

## Opcionales

Detalle código

Dice

# De pertenencia (static|)

## Circle

```
public class Circle{  
  
    public double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;
```

¿?

1. ¿Qué problema tenemos? ¿Qué más debe ser PI?

# De pertenencia (static|)

## Circle

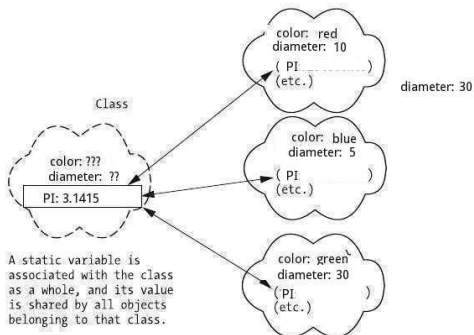
```
public class Circle{  
    public static double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;
```

¿?

1. ¿Qué problema tenemos? ¿Qué más debe ser PI?

# De pertenencia (static|.)

## Ubicación



## Atributos - Métodos

# De pertenencia (static|.)

```
public class Circle{  
    public static double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;  
  
    public Circle(){  
        diameter = 30;  
        xPosition = 20;  
        yPosition = 15;  
        color = "blue";  
        isVisible = false;  
    }  
}
```

¿?

1. ¿Qué hacemos si queremos que todos los círculos sean siempre del mismo color, así cambien de color?

# De pertenencia (static|.)

```
public class Circle{  
    public static double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;  
  
    public Circle(){  
        diameter = 30;  
        xPosition = 20;  
        yPosition = 15;  
        color = "blue";  
        isVisible = false;  
    }  
}
```

¿?

1. ¿Qué hacemos si queremos que todos los círculos sean siempre del mismo color, así cambien de color?
2. Implementemos un método para conocer el total de círculos creados. llámelo total

# De pertenencia (static|.)

```
public class Circle{  
    public static double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;  
  
    public Circle(){  
        diameter = 30;  
        xPosition = 20;  
        yPosition = 15;  
        color = "blue";  
        isVisible = false;  
    }  
}
```

¿?

1. ¿ Quién más no es de objeto?



# El método de clase

## Creador

- ▶ Java nos ofrece automáticamente un creador sin parámetros para todas las clases

Un método así:

```
public _____ Student() {  
    NO return type!  
}
```

*constructor name must match  
class name, followed by  
comma-separated list of formal  
parameters enclosed in ()*

El cuerpo de un creador no tiene retorno

Pero si definimos cualquier creador lo perdemos

# Agenda

## Desde los conceptos

Clase

Objetos

Igualdad

## Modificadores

De acceso

De pertenencia

**De mutabilidad**

## Tres conceptos

Encapsulamiento

Ocultación

Sobrecarga

## Buenas prácticas

.

## Opcionales

Detalle código

Dice

# Mutabilidad (final)

¿?

```
public class Circle{  
    public static double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;
```

¿?

1. ¿Qué problema nos queda? ¿Qué más debería ser PI?

# Mutabilidad (final)

¿?

```
public class Circle{  
    public static final double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;
```

¿?

1. ¿Qué problema nos queda? ¿Qué más debería ser PI?

# Mutabilidad (final)

## Atributos

```
public class Example {  
    // Assign values to static final variables/final attributes at the  
    // same time that we declare them.  
    public static final int x = 1;  
    private final int y = 2;  
    // etc.
```

# Mutabilidad (final)

## Atributos

```
public class Example {  
    // Assign values to static final variables/final attributes at the  
    // same time that we declare them.  
    public static final int x = 1;  
    private final int y = 2;  
    // etc.
```

## Variables locales

```
public void someMethod() {  
    // Even a local variable may be declared to be final.  
    final int z;  
  
    // However, whereas we ARE permitted to assign a local  
    // final variable a value in a method separately from its  
    // declaration ...  
    z = 3;
```

# Mutabilidad (final)

```
public class Circle{  
    public static double PI=3.1416;  
  
    private int diameter;  
    private int xPosition;  
    private int yPosition;  
    private String color;  
    private boolean isVisible;  
  
    public Circle(){  
        diameter = 30;  
        xPosition = 20;  
        yPosition = 15;  
        color = "blue";  
        isVisible = false;  
    }  
}
```

¿?

1. ¿Qué hacemos si queremos que los círculos no cambien de tamaño una vez han sido creados?

# Agenda

## Desde los conceptos

- Clase

- Objetos

- Igualdad

## Modificadores

- De acceso

- De pertenencia

- De mutabilidad

## Tres conceptos

- Encapsulamiento**

- Ocultación

- Sobrecarga

## Buenas prácticas

- .

## Opcionales

- Detalle código

- Dice

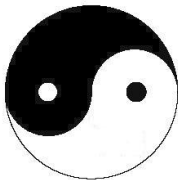


# Encapsulamiento

# Encapsulamiento

## Concepto

El *encapsulamiento* (*encapsulation*) es un término formal que se refiere al mecanismo que reúne el estado y el comportamiento de un objeto en una única unidad lógica



```
public class Square {  
  
}  
}
```

# Agenda

## Desde los conceptos

Clase

Objetos

Igualdad

## Modificadores

De acceso

De pertenencia

De mutabilidad

## Tres conceptos

Encapsulamiento

**Ocultación**

Sobrecarga

## Buenas prácticas

.

## Opcionales

Detalle código

Dice

# Ocultación de información

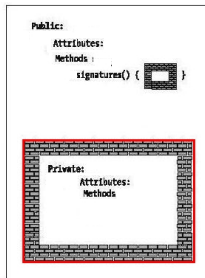
# Ocultación de información

## Concepto

La ocultación de información (*information hiding*) es una buena práctica de programación en la cual se oculta cómo los servicios se realizan y los datos que se mantienen internamente para soportar los servicios.

## Student

Client code may only access features that are declared to be public (typically, methods signatures).



The internal details of method code bodies are effectively private.

... as are most attributes.

# Agenda

## Desde los conceptos

- Clase

- Objetos

- Igualdad

## Modificadores

- De acceso

- De pertenencia

- De mutabilidad

## Tres conceptos

- Encapsulamiento

- Ocultación

- Sobrecarga**

## Buenas prácticas

- .

## Opcionales

- Detalle código

- Dice

# Sobrecarga

# Sobrecarga

## Concepto

La sobrecarga (*overloading*) es un mecanismo de los lenguajes que permite que dos o más métodos de la misma clase tengan el mismo nombre si tienen argumentos diferentes.

## Student

```
void print(String fileName) { ... // version #1  
void print(int detailLevel) { ... // version #2  
void print(int detailLevel, String fileName) { ... // version #3  
int print(String reportTitle, int maxPages) { ... // version #4  
boolean print() { ... // version #5
```



# Sobrecarga

## Concepto

La sobrecarga (*overloading*) es un mecanismo de los lenguajes que permite que dos o más métodos de la misma clase tengan el mismo nombre si tienen argumentos diferentes.

## Student

```
void print(String fileName) { ... // version #1  
void print(int detailLevel) { ... // version #2  
void print(int detailLevel, String fileName) { ... // version #3  
int print(String reportTitle, int maxPages) { ... // version #4  
boolean print() { ... // version #5
```

¿Podemos invertir el orden de los argumentos en 4?

# Agenda

## Desde los conceptos

- Clase

- Objetos

- Igualdad

## Modificadores

- De acceso

- De pertenencia

- De mutabilidad

## Tres conceptos

- Encapsulamiento

- Ocultación

- Sobrecarga

## Buenas prácticas

- .

## Opcionales

- Detalle código

- Dice

# Cambio

## Student

---

### The “Before” Code

---

```
public class Student {  
    // We have an explicit  
    // age attribute.  
    private int age;  
  
    public int getAge() {  
        return age;  
    }  
  
    // etc.  
}
```

---

### The “After” Code

---

```
import java.util.Date;  
  
public class Student {  
    // We replace age with  
    // birthDate.  
    private Date birthDate;  
  
    public int getAge() {  
        // Compute the age on demand  
        // (pseudocode).  
        return system date - birthDate;  
    }  
  
    // etc.  
}
```

---

¿Por qué cambiar?

# Cambio

## Student

The "Before" Code	The "After" Code
<pre>public class Student {     // We have an explicit     // age attribute.     private int age;      public int getAge() {         return age;     }      // etc. }</pre>	<pre>import java.util.Date;  public class Student {     // We replace age with     // birthDate.     private Date birthDate;      public int getAge() {         // Compute the age on demand         // (pseudocode).         return system date - birthDate;     }      // etc. }</pre>

¿?

1. ¿Impacto del cambio usuarios de Student?

# Cambio

## Student

The "Before" Code	The "After" Code
<pre>public class Student {     // We have an explicit     // age attribute.     private int age;      public int getAge() {         return age;     }      // etc. }</pre>	<pre>import java.util.Date;  public class Student {     // We replace age with     // birthDate.     private Date birthDate;      public int getAge() {         // Compute the age on demand         // (pseudocode).         return system date - birthDate;     }      // etc. }</pre>

¿?

1. ¿Impacto del cambio usuarios de Student?

¿COMO LO LOGRAMOS?

# Cambio

## Student

The "Before" Code	The "After" Code
<pre>public class Student {     // We have an explicit     // age attribute.     private int age;      public int getAge() {         return age;     }      // etc. }</pre>	<pre>import java.util.Date;  public class Student {     // We replace age with     // birthDate.     private Date birthDate;      public int getAge() {         // Compute the age on demand         // (pseudocode).         return system date - birthDate;     }      // etc. }</pre>

¿?

1. ¿Impacto del cambio usuarios de Student?  
¿COMO LO LOGRAMOS?
2. ¿Impacto del cambio en clase Student?

# Cambio

## Student

The "Before" Code	The "After" Code
<pre>public class Student {     // We have an explicit     // age attribute.     private int age;      public int getAge() {         return age;     }      // etc. }</pre>	<pre>import java.util.Date;  public class Student {     // We replace age with     // birthDate.     private Date birthDate;      public int getAge() {         // Compute the age on demand         // (pseudocode).         return system date - birthDate;     }      // etc. }</pre>

¿?

1. ¿Impacto del cambio usuarios de Student?  
¿COMO LO LOGRAMOS?
2. ¿Impacto del cambio en clase Student?  
¿DE QUE DEPENDERÍA?

# Buenas prácticas

1. Atributos

2. Métodos

3. Atributos-Métodos



# Buenas prácticas

## 1. Atributos

Los atributos normalmente son privados

## 2. Métodos

Los métodos normalmente son públicos

## 3. Atributos-Métodos

Los atributos se manejan con métodos especiales

# Buenas prácticas

## 1. Atributos

Los atributos normalmente son privados

## 2. Métodos

Los métodos normalmente son públicos

## 3. Atributos-Métodos

Los atributos se manejan con métodos especiales

`deme(get))` o `es(is)` y `cambie(set)`

Las condiciones de validez se centralizan, si es posible, en el `cambie`

# Buenas prácticas

## 1. Atributos

Los atributos normalmente son privados

Las constantes pueden ser publicas

## 2. Métodos

Los métodos normalmente son públicos

Pero, hay métodos que no se necesitan publicar (son privados)

## 3. Atributos-Métodos

Los atributos se manejan con métodos especiales

`deme(get))` o `es(is)` y `cambie(set)`

Únicamente se ofrecen cuando se necesitan

Las condiciones de validez se centralizan, si es posible, en el `cambie`

# Agenda

## Desde los conceptos

- Clase

- Objetos

- Igualdad

## Modificadores

- De acceso

- De pertenencia

- De mutabilidad

## Tres conceptos

- Encapsulamiento

- Ocultación

- Sobrecarga

## Buenas prácticas

- .

## Opcionales

- Detalle código

- Dice

# Lenguaje. Tipos básicos.

¿Tipos básicos? ¿Diferentes tipos?

# Lenguaje. Tipos básicos.

- For byte, from -128 to 127, inclusive
- For short, from -32768 to 32767, inclusive
- For int, from -2147483648 to 2147483647, inclusive
- For long, from -9223372036854775808 to 9223372036854775807, inclusive
- The floating-point types are `float` and `double`, which are conceptually associated with the single-precision 32-bit and double-precision 64-bit
- For `char`, from `'\u0000'` to `'\uffff'` inclusive, that is, from 0 to 65535
- `boolean` type represents a logical quantity with two possible values, indicated by the literals `true` and `false`

¿Tipos básicos? ¿Diferentes tipos?

# Lenguaje. Estructurador simple.

¿Para estructurar?

## Lenguaje. Estructurador simple.

```
int[] ai;           // array of int
short[][] as;       // array of array of short
Object[] ao,        // array of Object
```

¿Para estructurar?



# Lenguaje. Condicionales.

¿Condicionales?

# Lenguaje. Condicionales.

## 7.4 if Statements

```
if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
} else {  
    statements;  
}
```

## 7.8 switch Statements

```
switch (condition) {  
    case ABC:  
        statements;  
        /* falls through */  
    case DEF:  
        statements;  
        break;  
    default:  
        statements;  
        break;  
}
```

```
alpha = (aLongBooleanExpression) ? beta : gamma;
```

¿Condicionales?

# Lenguaje. Iterativas.

¿Iterativas?

# Lenguaje. Iterativas.

## 7.5 for Statements

```
for (initialization; condition; update) {  
    statements;  
}
```

## 7.6 while Statements

```
while (condition) {  
    statements;  
}
```

## 7.7 do-while Statements

```
do {  
    statements;  
} while (condition);
```

¿Iterativas?

# Agenda

## Desde los conceptos

Clase

Objetos

Igualdad

## Modificadores

De acceso

De pertenencia

De mutabilidad

## Tres conceptos

Encapsulamiento

Ocultación

Sobrecarga

## Buenas prácticas

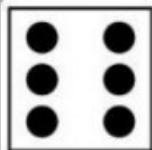
.

## Opcionales

Detalle código

Dice

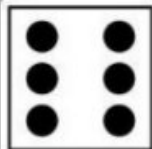
# Dice



Dice
+ Dice() + value() : int + makeVisible() : void + makeInvisible() : void + moveHorizontal(distance : int) : void + roll() : void

1. clase=atributos+..

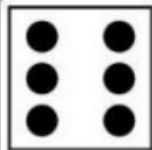
# Dice



Dice
+ Dice() + value() : int + makeVisible() : void + makeInvisible() : void + moveHorizontal(distance : int) : void + roll() : void

1. `clase=atributos+métodos`
2. `makeInvisible` sólo hay círculos donde es necesario

# Dice



Dice
+ Dice() + value() : int + makeVisible() : void + makeInvisible() : void + moveHorizontal(distance : int) : void + roll() : void

1. clase=atributos+métodos
2. roll import java.util.Random



# Dice

## Constructor Summary

### Random()

Creates a new random number generator.

## Method Summary

<code>boolean</code>	<code>nextBoolean()</code> Returns the next pseudorandom, uniformly distributed <code>boolean</code> value from this random number generator's sequence.
<code>void</code>	<code>nextBytes(byte[] bytes)</code> Generates random bytes and places them into a user-supplied byte array.
<code>double</code>	<code>nextDouble()</code> Returns the next pseudorandom, uniformly distributed <code>double</code> value between 0.0 and 1.0 from this random number generator's sequence.
<code>float</code>	<code>nextFloat()</code> Returns the next pseudorandom, uniformly distributed <code>float</code> value between 0.0 and 1.0 from this random number generator's sequence.
<code>int</code>	<code>nextInt()</code> Returns the next pseudorandom, uniformly distributed <code>int</code> value from this random number generator's sequence.
<code>int</code>	<code>nextInt(int n)</code> Returns a pseudorandom, uniformly distributed <code>int</code> value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.
<code>long</code>	<code>nextLong()</code> Returns the next pseudorandom, uniformly distributed <code>long</code> value from this random number generator's sequence.