
P00 en Java

UNQ - PO2

Temario de las próximas dos clases

- Parte 1
 - Repaso de lenguajes de programación
 - Repaso de PO
 - Cómo Java implementa conceptualmente los conceptos vistos en el repaso
- Partes 2. Componentes de la plataforma Java
 - Lenguaje
 - JDK
 - JRE
- Instalación y Armado del ambiente ([video tutorial](#))
- [TP3 - Repaso POO e intro a Java](#)
- Pilares del lenguaje Java
- Arquitectura. ¿En que parte de la arquitectura iría lo que desarrollamos en esta materia?
- [TP4 - POO en Java](#)

PARTE 1 - REPASO LENGUAJES DE PROGRAMACIÓN, POO E INTRO A POO EN JAVA

Introducción a Java

Es un lenguaje de programación de propósito general desarrollado en 1991 y publicado en 1995.

Adelanto del final de la clase: jdk y Eclipse.



James Gosling

Objetivos originales de Java

- Orientado a objetos
- Simple
- Con soporte para concurrencia
- Con soporte para distribución
- Garbage collection
- Portabilidad
- Estáticamente tipado



James Gosling

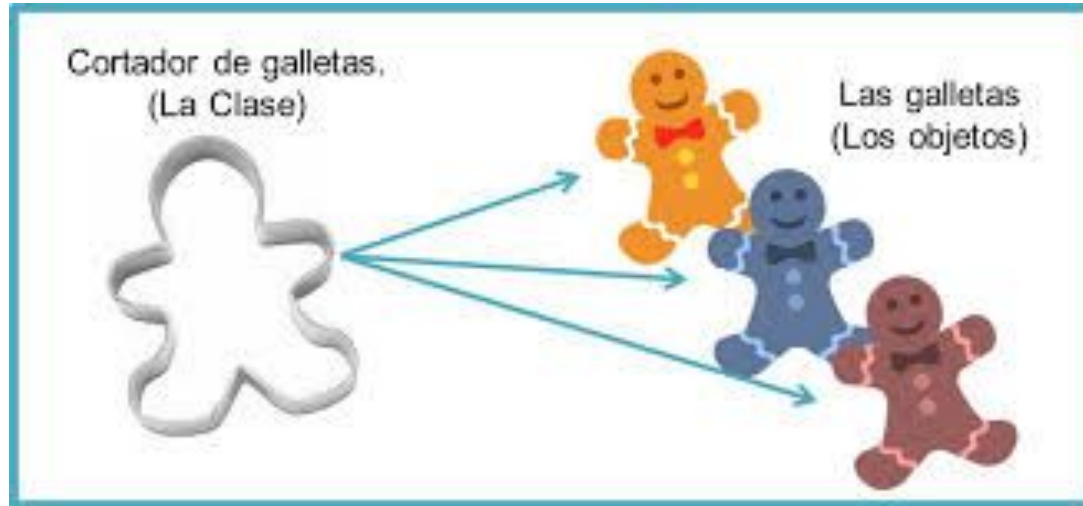
Objetivos originales del lenguaje

...orientado a objetos

- Clase: Molde
- Objeto: reúne estado y comportamiento
- Protocolo
- Encapsulamiento
- Herencia
- Polimorfismo
- Binding dinámico
- Abstracción



Repaso P00 - Clase

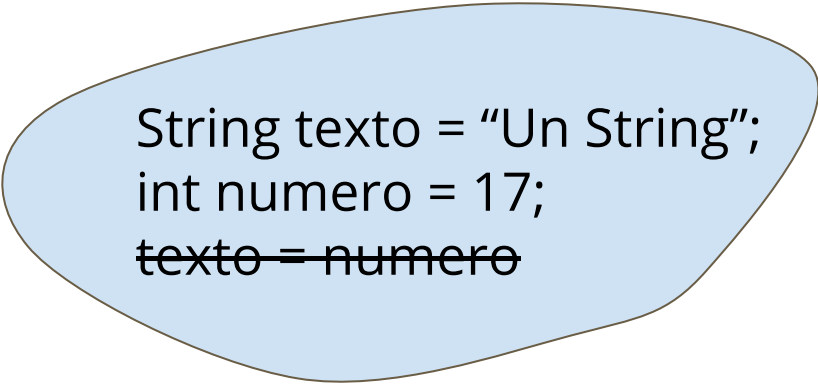


Java: Clases

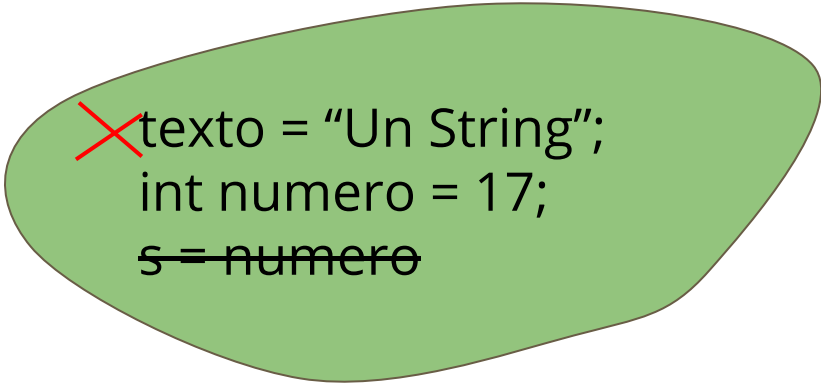
```
3 /**
4  * Representa un punto en el espacio
5  *
6  */
7  public class Punto {
8
9      //Estado: representado como variables de instancia
10     int x;
11     int y;
12
13
14     //Comportamiento representado con métodos
15     public int getX() {
16         return x;
17     }
18
19     public void setX(int x) {
20         this.x = x;
21     }
22     public int getY() {
23         return y;
24     }
25     public void setY(int y) {
26         this.y = y;
27     }
28
29     /**
30     * Retorno un punto nuevo con los valores absolutos de ambas componentes
31     * @return
32     */
33     public Punto abs() {
34
35     }
```


Objetivos originales del lenguaje

...estáticamente tipado



```
String texto = "Un String";  
int numero = 17;  
texto = numero
```

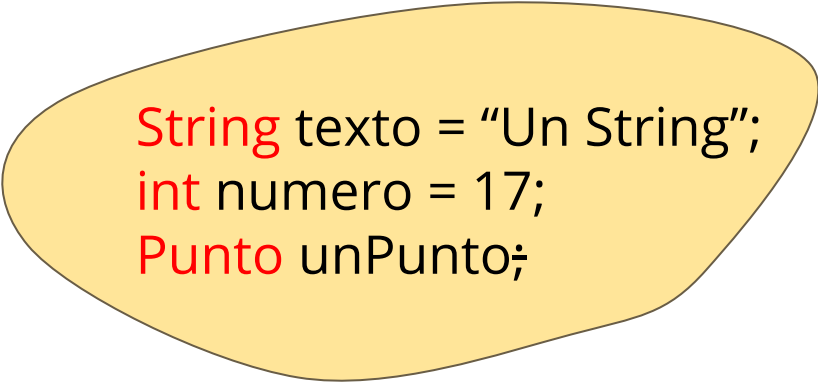


```
× texto = "Un String";  
int numero = 17;  
s = numero
```

Mostrar ejemplo en lenguaje dinámicamente tipado

Objetivos originales del lenguaje

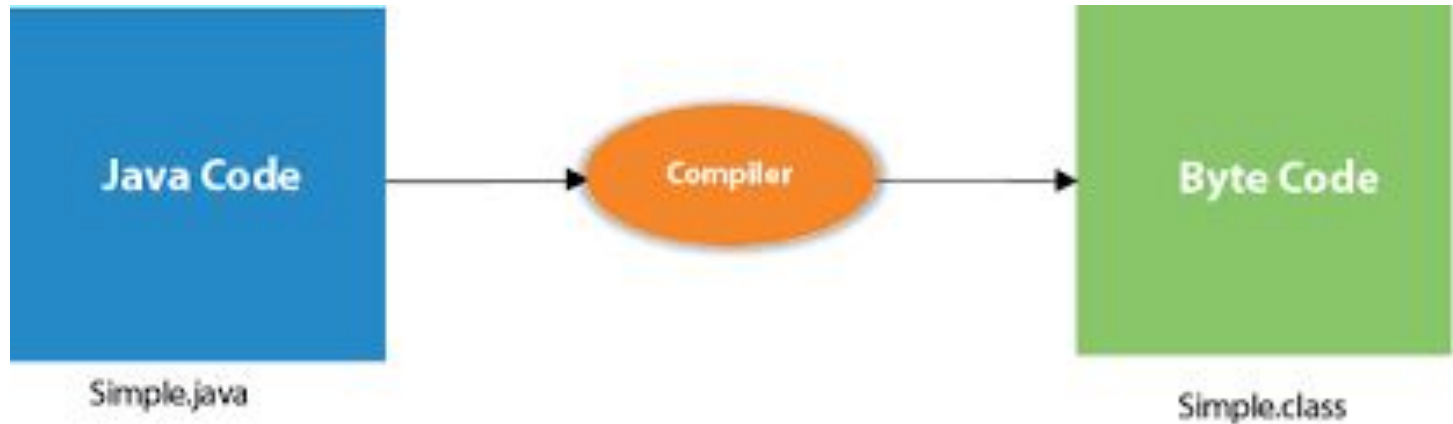
...Con qué puedo tipar mis variables?



```
String texto = "Un String";  
int numero = 17;  
Punto unPunto;
```

Objetivos originales del lenguaje

...Qué pasa si el tipo no coincide con el valor que asigno? Cuándo se chequea?



Paquetes

- Agrupa clases con características comunes.
- Convención para el Formato del nombre. De lo más general a lo más particular. Ejemplo: ar.edu.unq.po2....

You form a unique package name by first having (or belonging to an organization that has) an Internet domain name, such as `oracle.com`. You then reverse this name, component by component, to obtain, in this example, `com.oracle`, and use this as a prefix for your package names, using a convention developed within your organization to further administer package names. Such a convention might specify that certain package name components be division, department, project, machine, or login names.

- Se corresponde con folders en el file system.
- Tienen una estructura jerárquica
- Útil para manejar niveles de accesibilidad
- Evita colisión de nombres

Repaso P00 - P00 en Java

Instanciación de clases

Java: Instanciación de clases

```
3= /**
4  * Representa un punto en el espacio
5  *
6  */
7  public class Punto {
8
9      //Estado: representado como variables de instancia
10     int x;
11     int y;
12
13
14     //Constructores
15=    public Punto(int valorX, int valorY) {
16        this.setValues(valorX, valorY);
17    }
18
19
```

class name	Point
super class	Object
instance var	x y
class var	pi
class messages and methods	

```
newX:xvalue Y:yvalue | |
^ self new x: xvalue
  y: yvalue
```

x: anXNumber y: aYNumber

Set the x and y coordinate to anXNumber and aYNumber, respectively

```
Punto nuevoPunto = new Punto(-1,1);
nuevoPunto.abs();
```

```
aPoint <- Point newX:1 Y:2
```

Java: Instanciación de clases

```
3= /**
4  * Representa un punto en el espacio
5  *
6  */
7  public class Punto {
8
9      //Estado: representado como variables de instancia
10     int x;
11     int y;
12
13
14     //Constructores
15     public Punto(int valorX, int valorY) {
16         this.setValues(valorX, valorY);
17     }
18
19 }
```

```
Punto nuevoPunto = new Punto(-1,1);
nuevoPunto.abs();
```

class name	Point
super class	Object
instance var	x y
class var	pi
class messages and methods	

```
newX:xvalue Y:yvalue | |
^ self new x: xvalue
  y: yvalue
```

```
aPoint <- Point newX:1 Y:2
```

Constructores

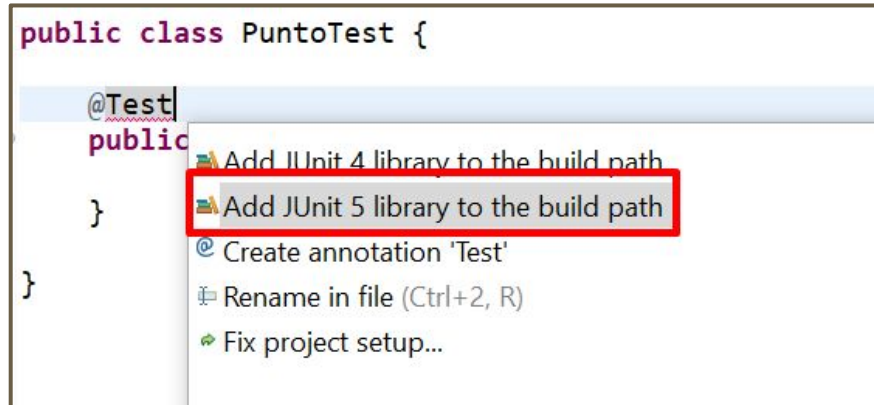
- ¿A toda clase le tengo que definir un constructor?
- ¿Puedo tener más de un constructor?
- Si defino nuevos constructores, sigo teniendo disponible el constructor por defecto?
- ¿Qué pasa si no le pongo el modificador public al constructor?
- ¿Como instancia una clase?

Tests

- Source folder hermano de src llamado test
- Misma estructura de paquetes que las clases
- Las clases de test llevan el mismo nombre que la clase que testean con la palabra Test al final.

Tests

- Los métodos que hacen las validaciones se deben marcar con `@Test`. La primera vez importar la librería de JUnit 5 (Jupiter)



Tests

- El framework de test (JUnit Jupiter) provee métodos de validación. Se llaman asserts.
- Son métodos estáticos que se pueden importar utilizando `import static org.junit.jupiter.api.Assertions.*;`
- Por ahora les va a alcanzar con `assertTrue`, `assertEquals`.

Tests

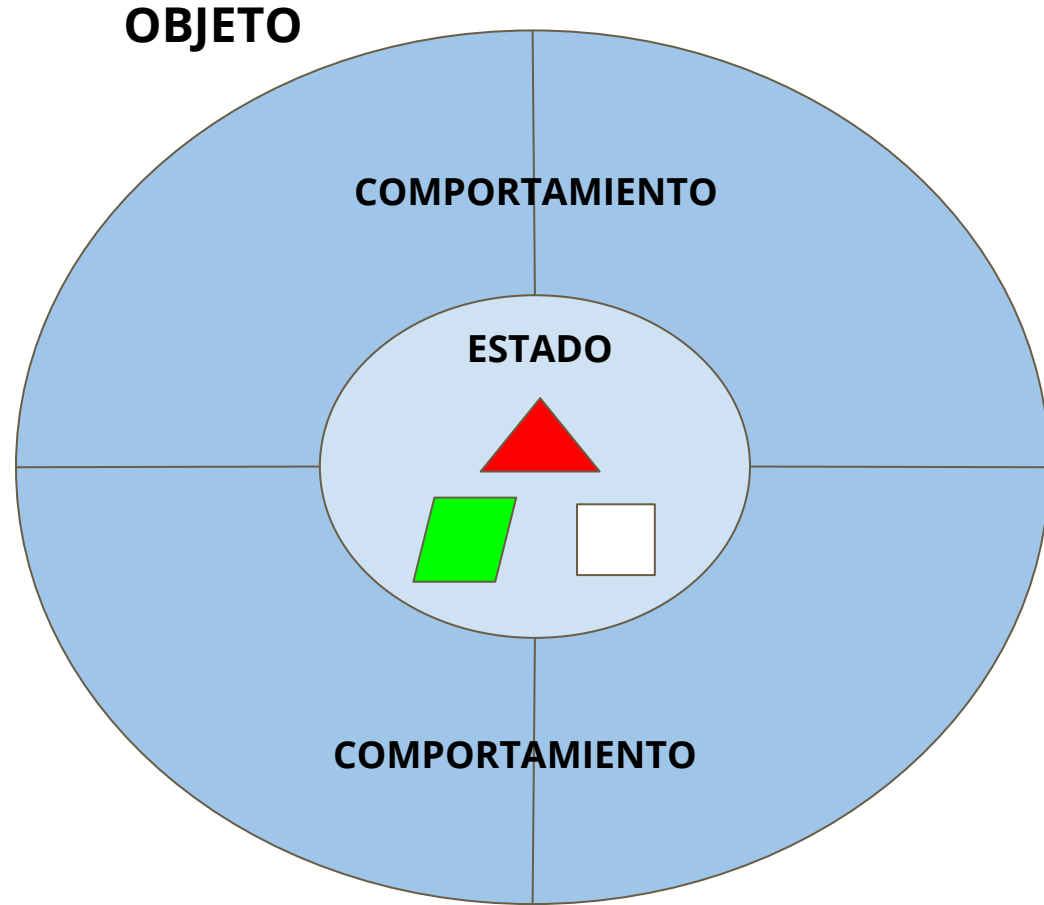
- Los assert pueden llevar opcionalmente un mensaje de error como parámetro. Sea claro con los mensajes.
- No puede conocer el orden de ejecución. Cada test debe ser programado de forma independiente a los demás.

Tests - Fixtures

- Métodos marcados con `@Before` se ejecutan antes de iniciar los tests. Los `@After` luego de correr todos los tests.
- Métodos marcados con `@BeforeEach` se ejecuta antes de cada test. Idem los `@AfterEach`.
- Promoveremos el uso de TDD.

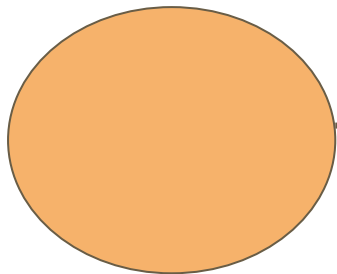
¿Mensaje y método es lo mismo?

Repaso P00

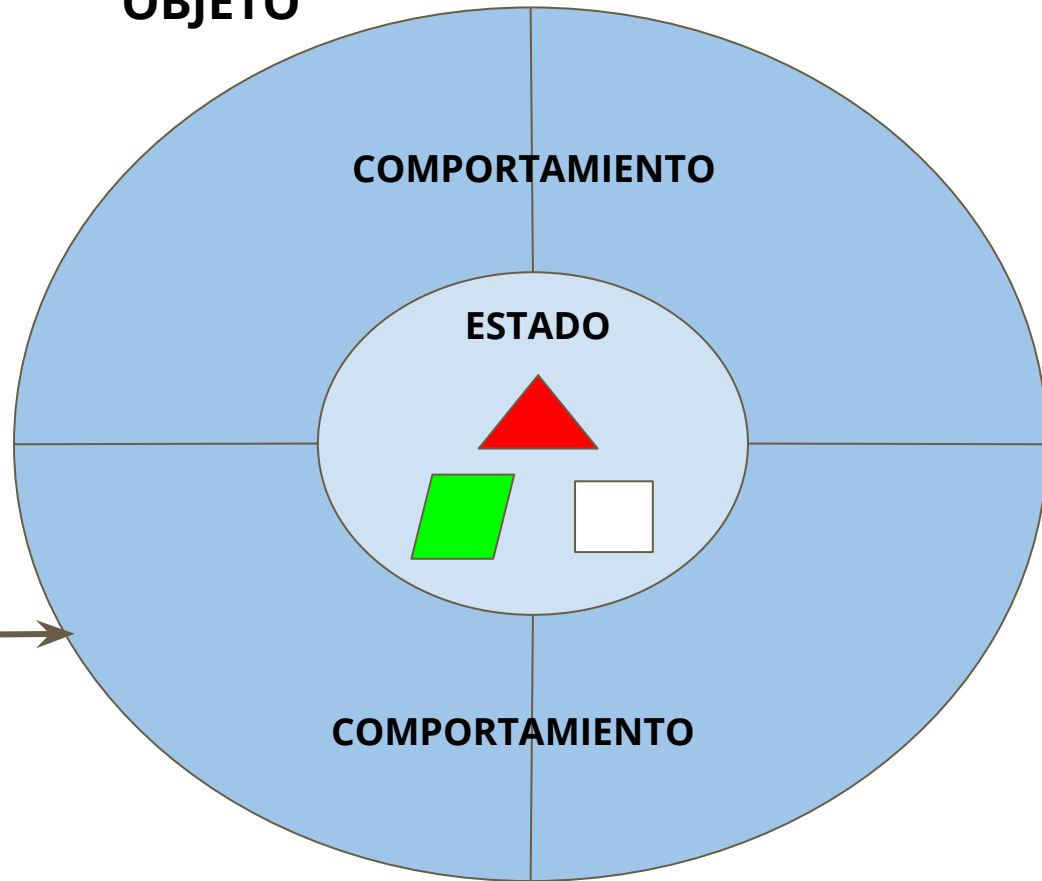


Repaso P00

OBJETO



OBJETO



Repaso P00

OBJETO

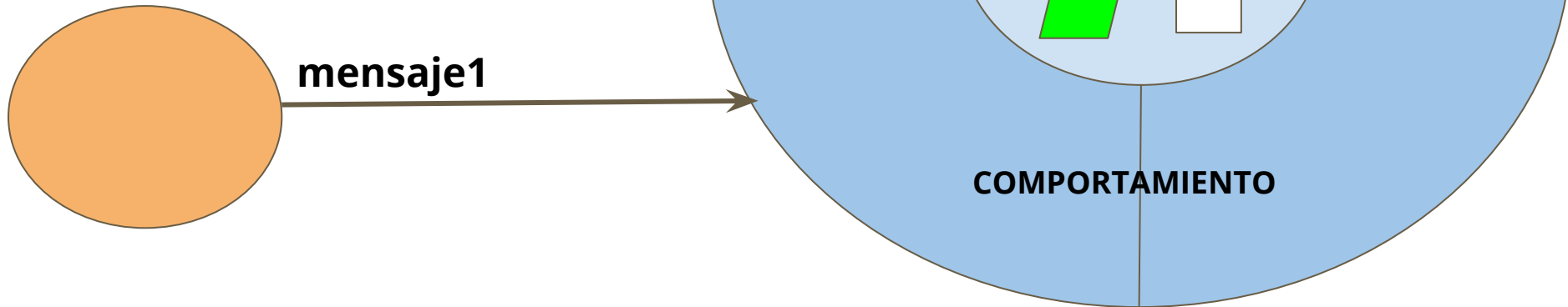
mensaje1

OBJETO

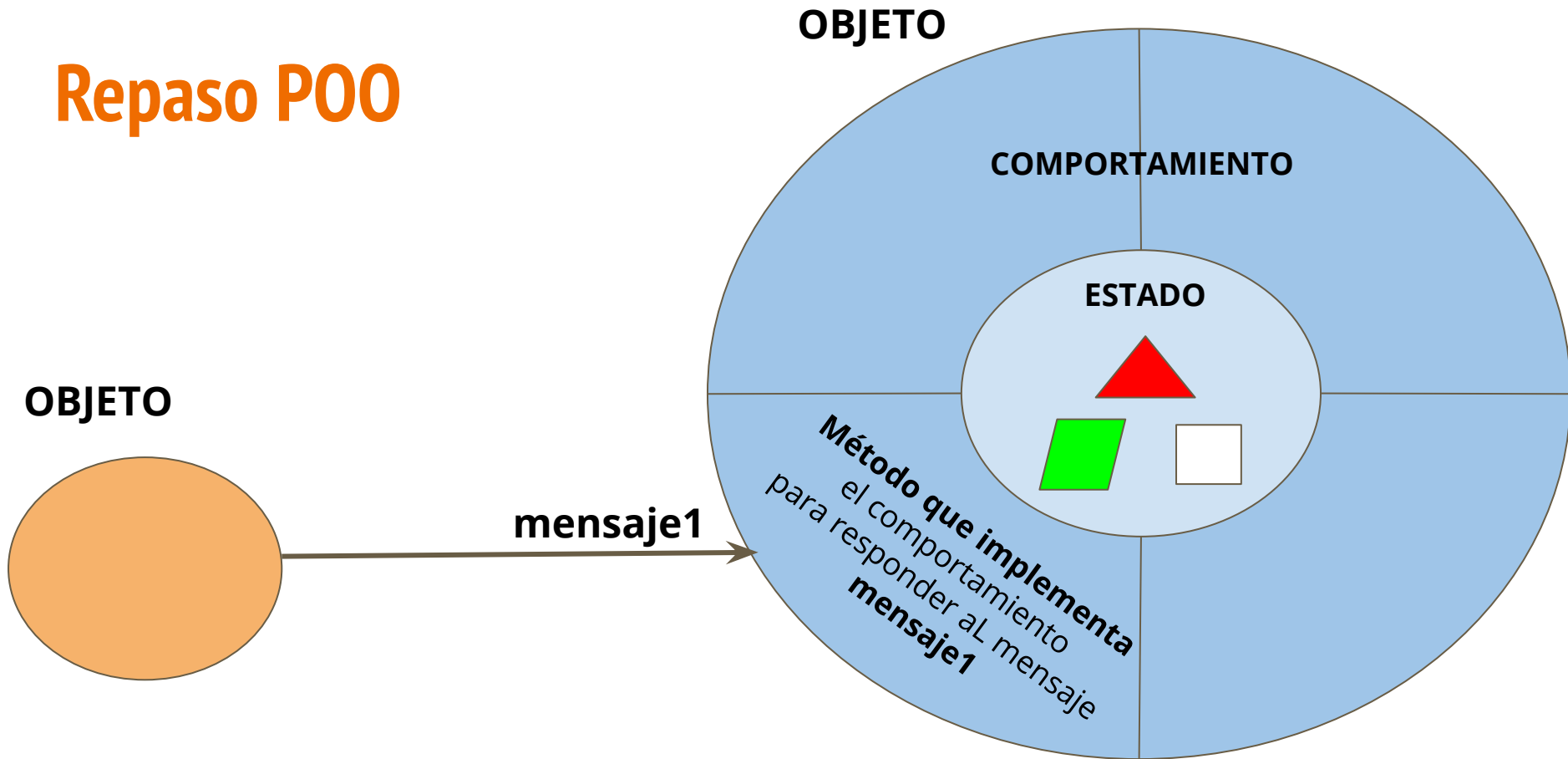
COMPORTAMIENTO

ESTADO

COMPORTAMIENTO

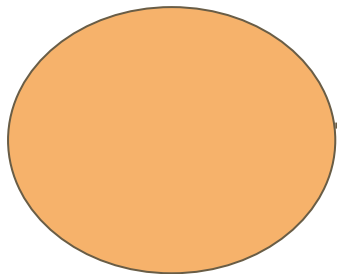


Repaso P00

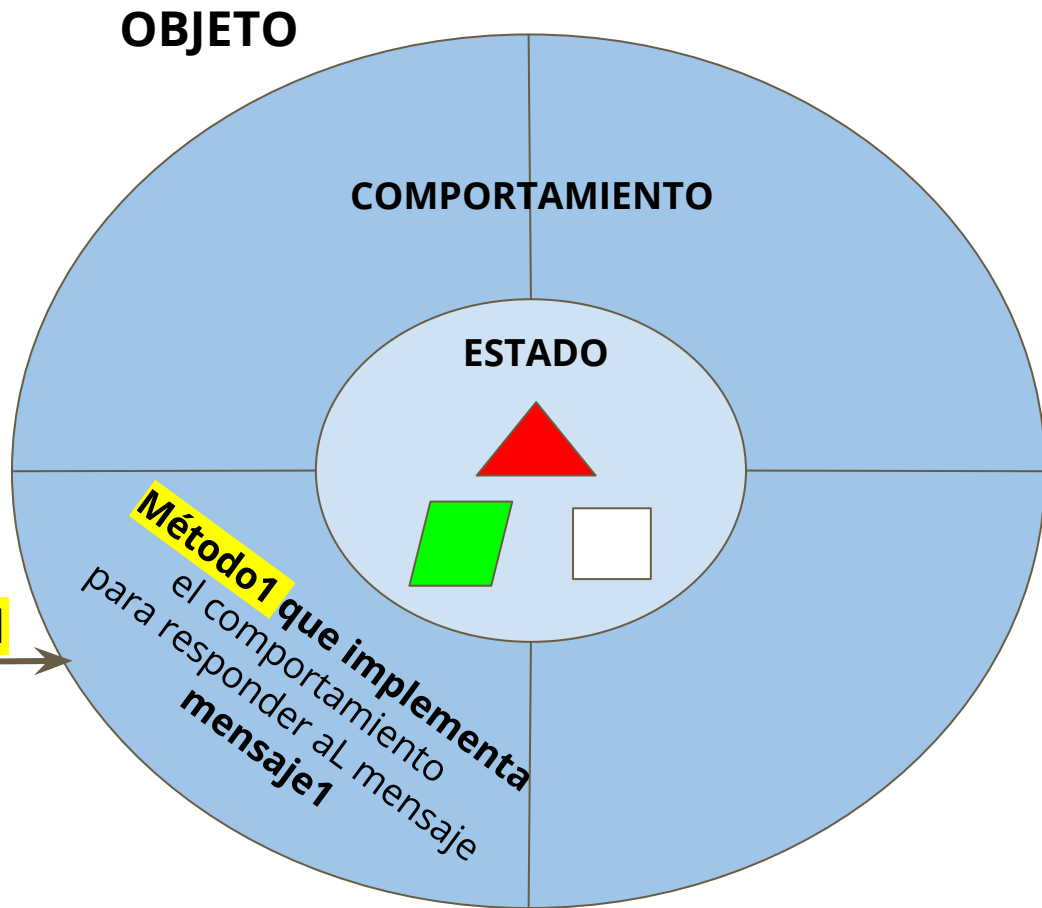


Repaso P00

OBJETO



mensaje1



Repaso P00

Encapsulamiento

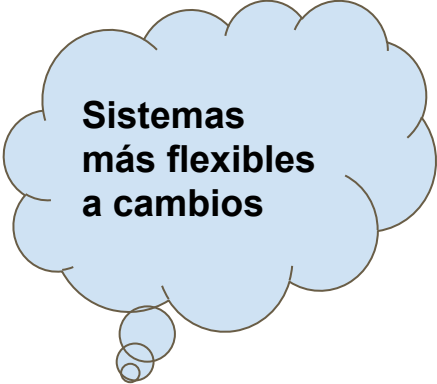
¿Qué busco con el encapsulamiento?

Repaso P00

Encapsulamiento

- Promover sistemas más flexibles al cambio
- Minimizar el riesgo de estados internos inconsistentes

Java: Herramientas para el Encapsulamiento

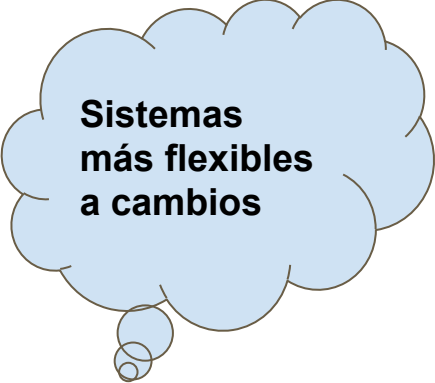


**Sistemas
más flexibles
a cambios**

Java: Herramientas para el Encapsulamiento

CLASE
PuntoColeoreado

```
3  /**
4   * Representa un punto colerado en el espacio
5   *
6   */
7  public class PuntoColoreado {
8
9      //Estado: representado como variables de instancia
10     public int x;
11     public int y;
12     //Color en inglés
13     public String color;
14 }
```



**Sistemas
más flexibles
a cambios**

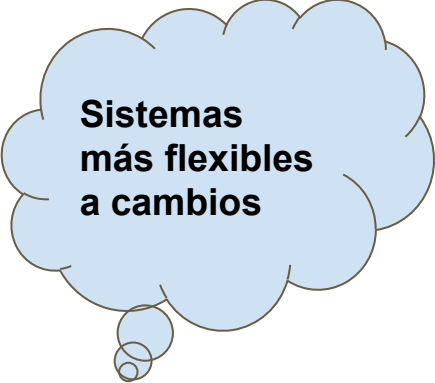
Java: Herramientas para el Encapsulamiento

CLASE PuntoColeoreado

```
public class PuntoColoreado {  
  
    private int x;  
    private int y;  
  
    //Color en inglés. Por ejemplo: red, black  
    public String color;  
  
    public PuntoColoreado(int x, int y) {  
        super();  
        this.x = x;  
        this.y = y;  
    }  
}
```

CLASE ClientePuntoColoreado

```
public void rompeEncapsualmientoDePuntoColoreado() {  
  
    PuntoColoreado puntoRojo = new PuntoColoreado(10,10);  
    puntoRojo.color="Red";  
  
}
```



Sistemas
más flexibles
a cambios

Java: Herramientas para el Encapsulamiento

CLASE ClientePuntoColoreado

```
public void rompeEncapsulamientoDePuntoColoreado2() {  
    PuntoColoreado puntoMarron = new PuntoColoreado(10,10);  
    puntoMarron.color="Brown";  
}
```

CLASE ClientePuntoColoreado2

```
public void rompeEncapsulamientoDePuntoColoreado3() {  
    PuntoColoreado puntoAzul = new PuntoColoreado(10,10);  
    puntoAzul.color="Blue";  
}
```

CLASE ClientePuntoColoreado3

```
public void rompeEncapsulamientoDePuntoColoreado() {  
    PuntoColoreado puntoRojo = new PuntoColoreado(10,10);  
    puntoRojo.color="Red";  
}
```

Java: Herramientas para el Encapsulamiento

```
3= /**
4  * Representa un punto colerado en el espacio
5  *
6  */
7  public class PuntoColoreado {
8
9      //Estado: representado como variables de instancia
10     public int x;
11     public int y;
12     //Color en inglés
13     public String color;
14 }
```



```
4
3= /**
4  * Representa un punto colerado en el espacio
5  *
6  */
7  public class PuntoColoreado {
8
9      //Estado: representado como variables de instancia
10     public int x;
11     public int y;
12     //RGB en Notación Hexadecimal. Ejemplo: #ffffff
13     public String color;
14 }
```

Java: Herramientas para el Encapsulamiento

```
public void rompeEncapsulamientoDePuntoColoreado2() {  
    PuntoColoreado puntoMarron = new PuntoColoreado(10,10);  
    puntoMarron.color="Brown";  
}
```



```
public void rompeEncapsulamientoDePuntoColoreado3() {  
    PuntoColoreado puntoAzul = new PuntoColoreado(10,10);  
    puntoAzul.color="Blue";  
}
```

```
public void rompeEncapsulamientoDePuntoColoreado() {  
    PuntoColoreado puntoRojo = new PuntoColoreado(10,10);  
    puntoRojo.color="Red";  
}
```



Java: Herramientas para el Encapsulamiento

```
3= /**
4  * Representa un punto colerado en el espacio
5  *
6  */
7  public class PuntoColoreado {
8
9      //Estado: representado como variables de instancia
10     private int x;
11     private int y;
12     //Color en inglés
13     private String color;
14
15=    public void setColor(String color) {
16        this.color = color;
17    }
18
19=    public String getColor() {
20        return color;
21    }
22 }
```

Java: Herramientas para el Encapsulamiento

CLASE ClientePuntoColoreado

```
public void rompeEncapsulamientoDePuntoColoreado() {  
    PuntoColoreado puntoRojo = new PuntoColoreado(10,10);  
    puntoRojo.color;  
}
```

 The field PuntoColoreado.color is not visible
2 quick fixes available:
[Change visibility of 'color' to 'package'](#)
[Replace puntoRojo.color with getter](#)
Press 'F2' for focus

```
3 /**  
4  * Representa un punto coloreado en el espacio  
5  *  
6  */  
7 public class PuntoColoreado {  
8  
9     //Estado: representado como variables de instancia  
10    private int x;  
11    private int y;  
12    //Color en inglés  
13    private String color;  
14  
15    public void setColor(String color) {  
16        this.color = color;  
17    }  
18  
19    public String getColor() {  
20        return color;  
21    }  
22 }
```

```
public void rompeEncapsulamientoDePuntoColoreado() {  
    PuntoColoreado puntoRojo = new PuntoColoreado(10,10);  
    puntoRojo.setColor("Red");  
}
```

Java: Herramientas para el Encapsulamiento

```
public void rompeEncapsulamientoDePuntoColoreado2() {  
    PuntoColoreado puntoMarron = new PuntoColoreado(10,10);  
    puntoMarron.setColor("Brown");  
}
```

```
public void rompeEncapsulamientoDePuntoColoreado3() {  
    PuntoColoreado puntoAzul = new PuntoColoreado(10,10);  
    puntoAzul.setColor("Blue");  
}
```

```
public void rompeEncapsulamientoDePuntoColoreado() {  
    PuntoColoreado puntoRojo = new PuntoColoreado(10,10);  
    puntoRojo.setColor("Red");  
}
```

Java: Herramientas para el Encapsulamiento

```
1 package ar.edu.unq.po2.tp1;
2
3 /**
4  * Representa un punto colerado en el espacio
5  *
6  */
7 public class PuntoColoreado {
8
9     //Estado: representado como variables de instancia
10    private int x;
11    private int y;
12    //RGB en Notación Hexadecimal. Ejemplo:
13    private String color;
14
15    public void setColor(String color) {
16        this.color = fromEnglishToHex(color);
17    }
18 }
```

Java: Herramientas para el Encapsulamiento

```
public void rompeEncapsulamientoDePuntoColoreado2() {  
    PuntoColoreado puntoMarron = new PuntoColoreado(10,10);  
    puntoMarron.setColor("Brown");  
}
```



```
public void rompeEncapsulamientoDePuntoColoreado3() {  
    PuntoColoreado puntoAzul = new PuntoColoreado(10,10);  
    puntoAzul.setColor("Blue");  
}
```

```
public void rompeEncapsulamientoDePuntoColoreado() {  
    PuntoColoreado puntoRojo = new PuntoColoreado(10,10);  
    puntoRojo.setColor("Red");  
}
```



Repaso POO: Ocultar estructura interna

- Alcanza con disponer las variables privadas? No! El diseño también debe ser acorde.



Evaluación de protocolos de una clase



Sea la clase Rectángulo con 4 variables de instancia (esquinaSuperiorIzquierda, esquinaSuperiorDerecha, esquinaInferiorIzquierda y esquinaInferiorDerecha). Elija uno de los protocolos presentados a continuación y justifique su elección. Recuerde que el protocolo es el conjunto de mensajes que entiende una clase o tipo.

Encuesta

**Minimizar el
riesgo de
estados
internos
inconsistentes**

Opción 1)

```
Class Rectangulo>>new  
  
Rectangulo>>esquinaSuperiorIzquierda: unPunto  
  
Rectangulo>>esquinaSuperiorDerecha: unPunto  
  
Rectangulo>>esquinaInferiorIzquierda: unPunto  
  
Rectangulo>>esquinaInferiorDerecha: unPunto
```

Opcion 2)

```
Class Rectangulo>>newEnOrigenEsquinaSuperiorIzquierda: unPunto alto: unNumero ancho:unNumero  
  
Rectangulo>>reubicarEsquinaSuperiorIzquierdaEn: unPunto  
  
Rectangulo>>ancho: unNumero  
  
Rectangulo>>alto: unNumero
```

Repaso POO: Ocultar estructura interna

- Alcanza con disponer las variables privadas? No! El diseño también debe ser acorde.



Evaluación de protocolos de una clase



Sea la clase Rectángulo con 4 variables de instancia (esquinaSuperiorIzquierda, esquinaSuperiorDerecha, esquinaInferiorIzquierda y esquinaInferiorDerecha). Elija uno de los protocolos presentados a continuación y justifique su elección. Recuerde que el protocolo es el conjunto de mensajes que entiende una clase o tipo.

Encuesta

Minimizar el
riesgo de
estados
internos
inconsistentes

Opción 1)

```
public class RectanguloNoEncapsulado
```

```
Class Rectangulo>>new
```

```
Rectangulo>>esquinaSuperiorIzquierda: unPunto
```

```
Rectangulo>>esquinaSuperiorDerecha: unPunto
```

```
Rectangulo>>esquinaInferiorIzquierda: unPunto
```

```
Rectangulo>>esquinaInferiorDerecha: unPunto
```

Opcion 2)

```
public class Rectangulo
```

```
Class Rectangulo>>newEnOrigenEsquinaSuperiorIzquierda: unPunto alto: unNumero ancho:unNumero
```

```
Rectangulo>>reubicarEsquinaSuperiorIzquierdaEn: unPunto
```

```
Rectangulo>>ancho: unNumero
```

```
Rectangulo>>alto: unNumero
```

Repaso POO: Ocultar estructura interna

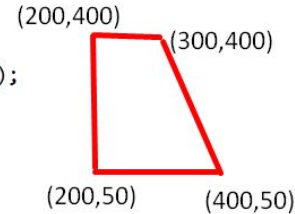
Alcanza con disponer las variables privadas? No! El diseño también debe ser acorde.

OPCION1

```
public void dibujarRectanguloNoEncapsulado(Graphics g) {  
    RectanguloNoEncapsulado r = new RectanguloNoEncapsulado();  
    r.setEsquinaSuperiorDerecha(new Punto(150,400));  
    r.setEsquinaInferiorDerecha(new Punto(150,50));  
    r.setEsquinaSuperiorIzquierda(new Punto(50,400));  
    r.setEsquinaInferiorIzquierda(new Punto(50,50));  
    r.dibujar(g);  
}
```

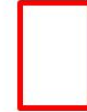


```
public void dibujarMALRectanguloNoEncapsulado(Graphics g) {  
    RectanguloNoEncapsulado r = new RectanguloNoEncapsulado();  
    r.setEsquinaSuperiorDerecha(new Punto(300,400));  
    r.setEsquinaInferiorDerecha(new Punto(400,50));  
    r.setEsquinaSuperiorIzquierda(new Punto(200,400));  
    r.setEsquinaInferiorIzquierda(new Punto(200,50));  
    r.dibujar(g);  
}
```



OPCION2

```
public void dibujarRectanguloEncapsulado(Graphics g) {  
    Rectangulo r = new Rectangulo(new Punto(450,400),350,100);  
    r.dibujar(g);  
}
```

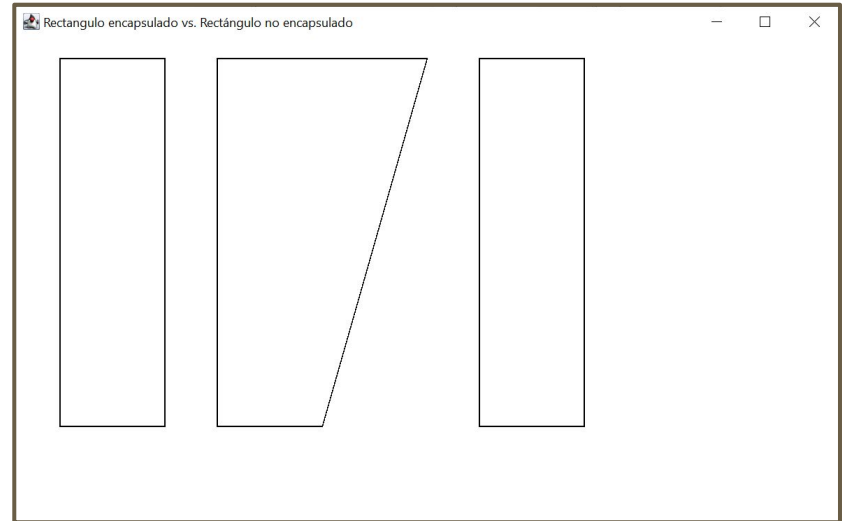


```
public void dibujarMALRectanguloEncapsulado(Graphics g) {  
    //NO PUEDO!!!  
}
```

Repaso P00: Ocultar estructura interna

- Alcanza con disponer las variables privadas? No! El diseño también debe ser acorde.

```
public void dibujar(Graphics g){
```



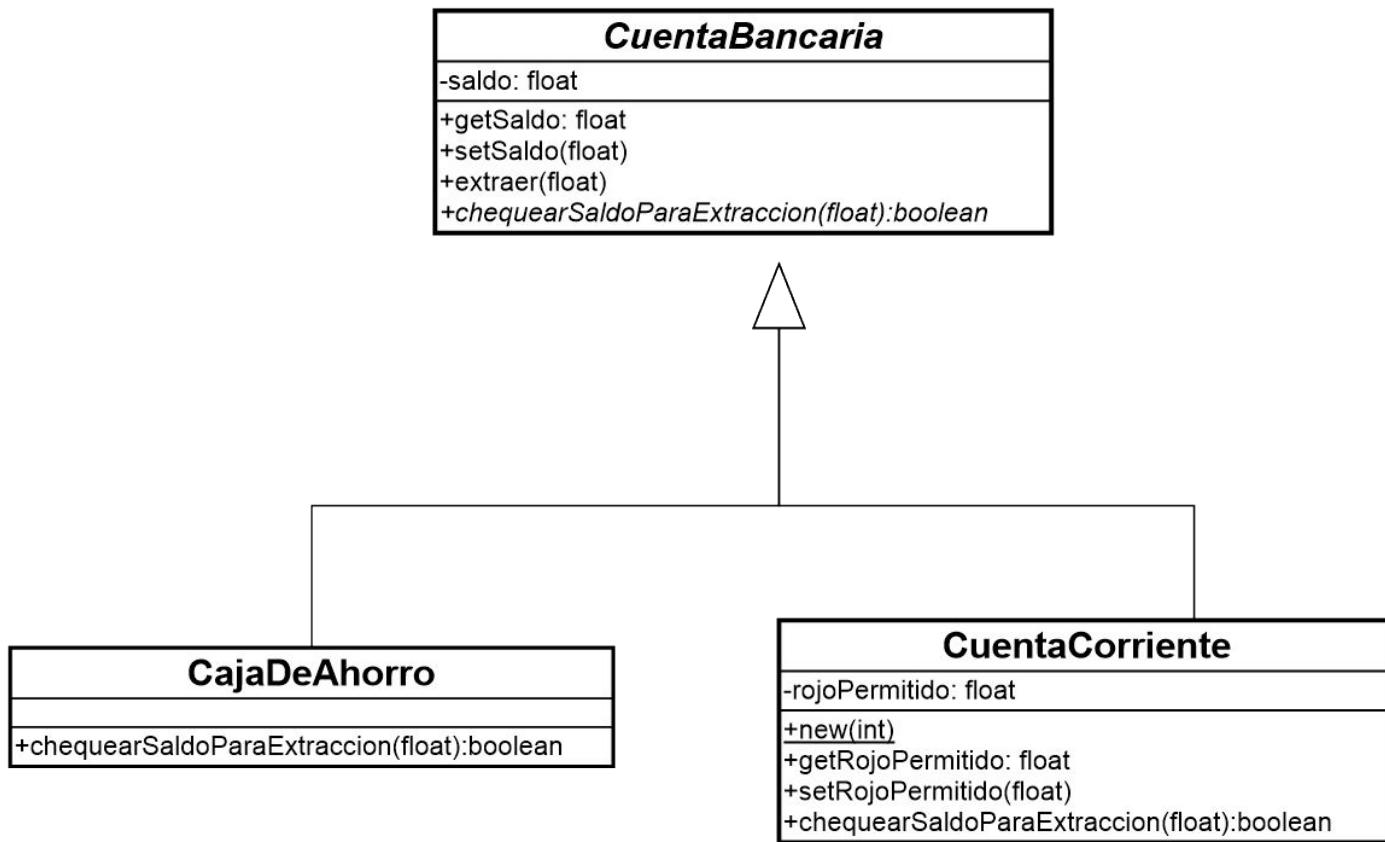
Repaso P00: Ocultar estructura interna

¿Qué formas de representar el rectángulo encontraron?

Repaso P00

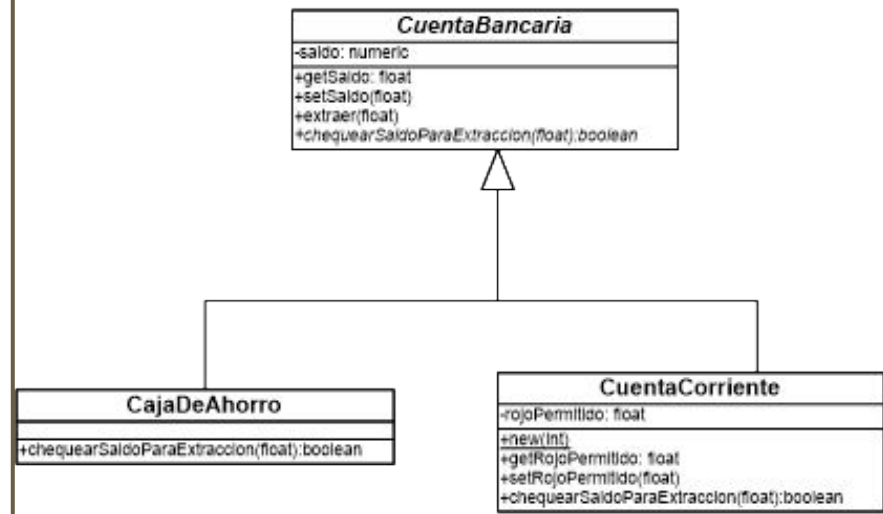
Herencia

P00 en Java: Herencia



P00 en Java: Herencia

Ejercicio 4 - Opción 4



```
package ar.edu.unq.po2.tp1.herencia;

public class CajaDeAhorro extends CuentaBancaria{

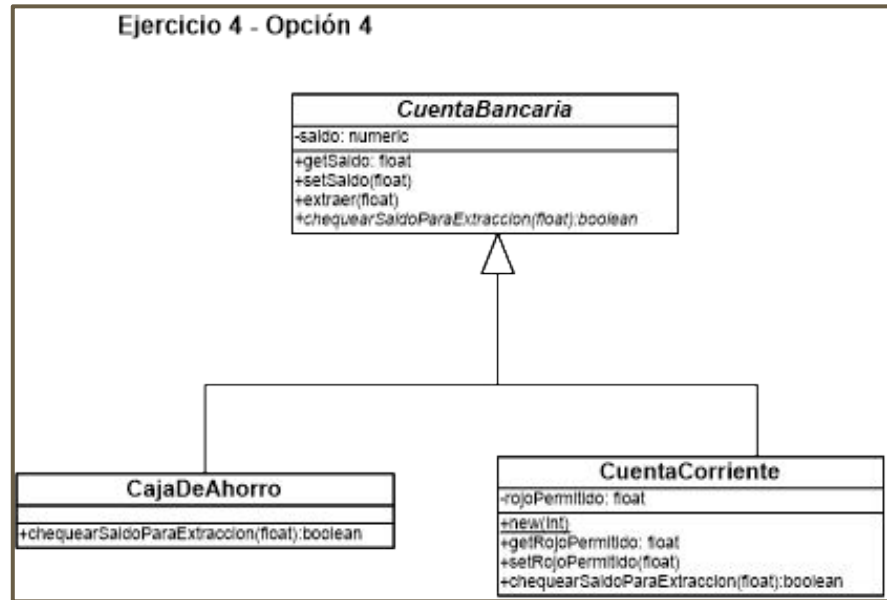
    @Override
    boolean chequearSaldoParaExtraccion(float unMonto) {
        return this.getSaldo()>=unMonto;
    }

}
```

¿Es obligatorio el override?

P00 en Java: Herencia

Ejercicio 4 - Opción 4



```
package ar.edu.unq.po2.tp1.herencia;

public class CajaDeAhorro extends CuentaBancaria{

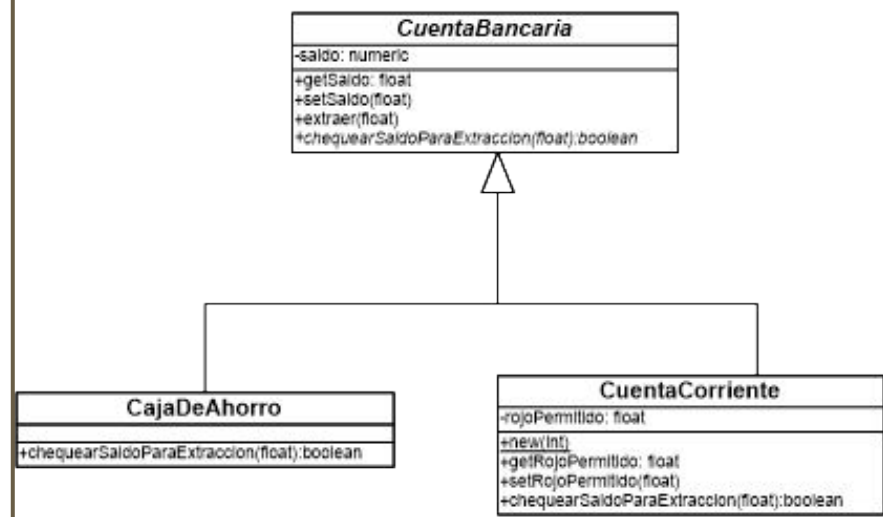
    @Override
    boolean chequearSaldoParaExtraccion(float unMonto) {
        return this.getSaldo()>=unMonto;
    }

}
```

¿Es obligatorio el override?

P00 en Java: Herencia

Ejercicio 4 - Opción 4



```
package ar.edu.unq.po2.tp1.herencia;

public class CuentaCorriente extends CuentaBancaria{

    private int rojoPermitido;

    @Override
    boolean chequearSaldoParaExtraccion(float unMonto) {
        return this.getSaldo()+this.getRojoPermitido()>=unMonto;
    }

    protected int getRojoPermitido() {
        return rojoPermitido;
    }

    protected void setRojoPermitido(int rojoPermitido) {
        this.rojoPermitido = rojoPermitido;
    }

    ...
}
```

P00 en Java: Herencia

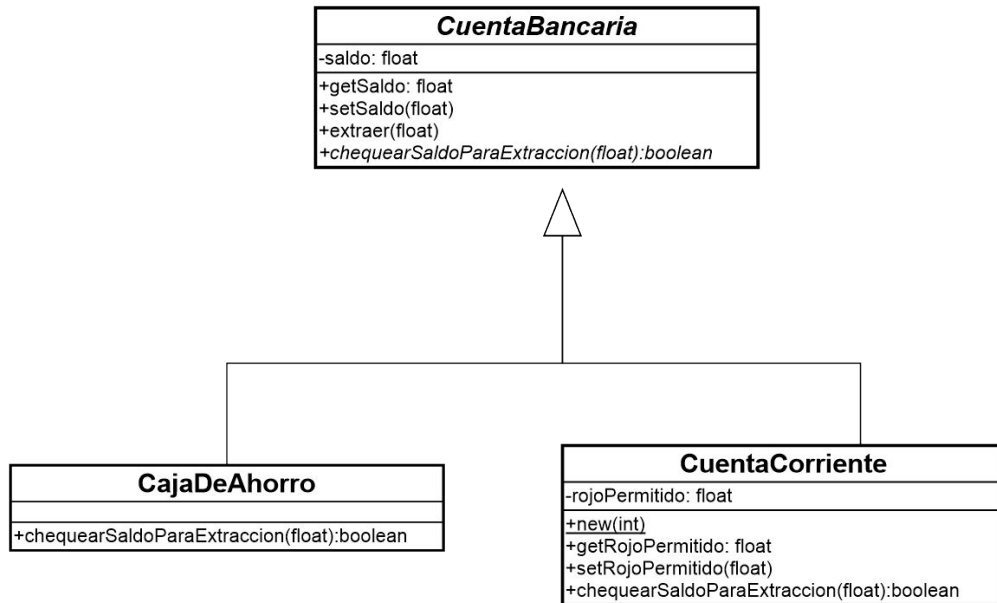
- ¿Diferencia entre sobrescribir y sobrecargar un método?

P00 en Java: Herencia

¿Los constructores se heredan?

P00 en Java: Herencia

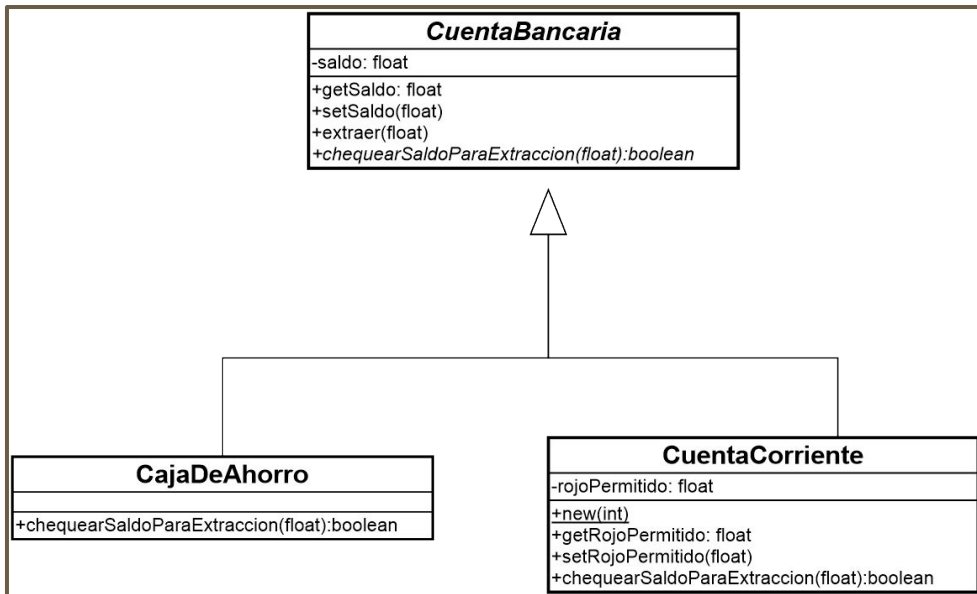
UML: tengo que volver a escribir los metodos en las subclases?



Repaso P00

Clases y métodos abstractos

P00 en Java: Herencia. Clases métodos abstractos.



```
package ar.edu.unq.po2.tp1.herencia;

public abstract class CuentaBancaria {

    private float saldo=0;

    public void extraer(float unMonto) {
        if (chequearSaldoParaExtraccion(unMonto)) {
            this.setSaldo(this.getSaldo()-unMonto);
        }
    }

    abstract boolean chequearSaldoParaExtraccion(float unMonto);

    public float getSaldo() {
        return saldo;
    }

    public void setSaldo(float saldo) {
        this.saldo = saldo;
    }

}
```

Clases y métodos abstractos

- Una clase abstracta puede tener métodos con comportamiento (no abstractos)?
- Una clase concreta (no abstracta) puede tener métodos abstractos?

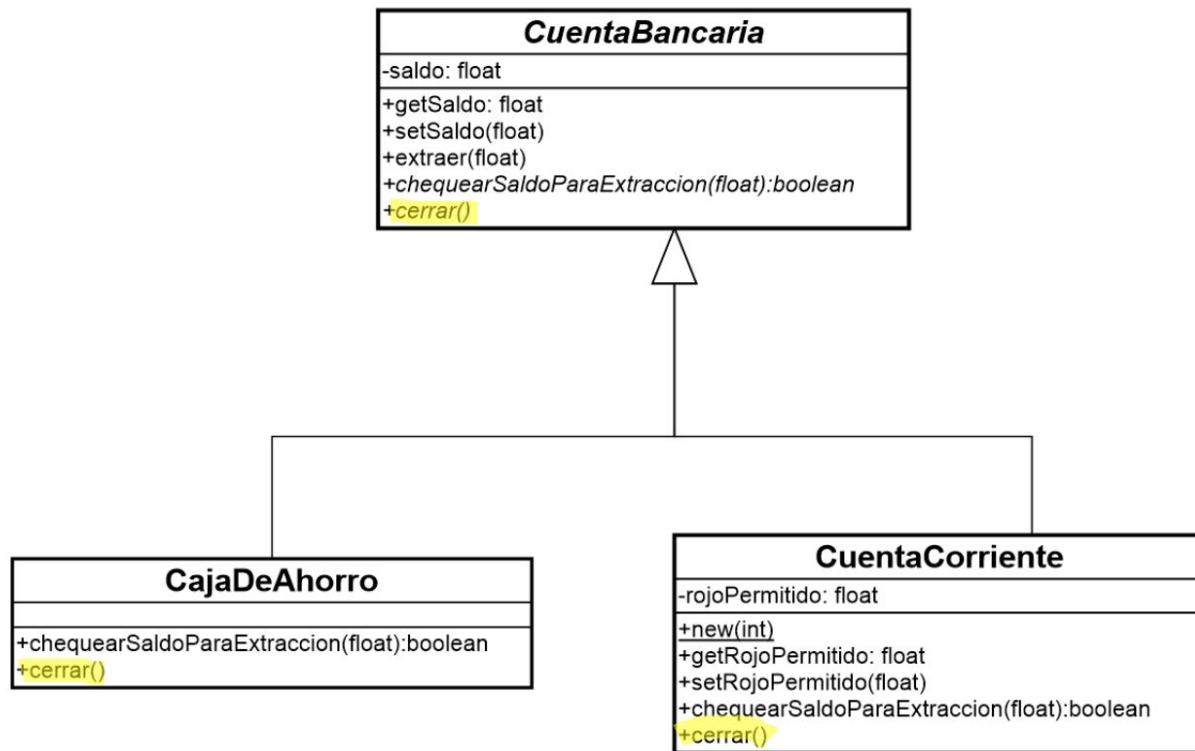
Repaso P00

Polimorfismo

Temas visto la clase pasada

- Lenguajes de programación
 - Estáticamente tipados vs dinámicamente tipados
 - Sintaxis y semántica
- Repaso POO:
 - encapsulamiento,
 - herencia,
 - clase abstractas,
 - métodos abstractos
 - mensaje y método no son lo mismo
- Java
 - Clases
 - Instanciación
 - Test (básico)
 - Paquetes
 - Niveles de visibilidad
 - JDS y JRE
 - Ambiente de desarrollo

P00 en Java: polimorfismo



P00 en Java: Polimorfismo

```
public abstract class CuentaBancaria {  
  
    public abstract void cerrar();  
}
```

```
public class CuentaCorriente extends CuentaBancaria{  
  
    @Override  
    public void cerrar() {  
        //La Cuenta corriente tiene una lógica propia para su cierre  
        //Asumimos que es totalmente diferente a la de la CajaDeAhorro  
    }  
}
```

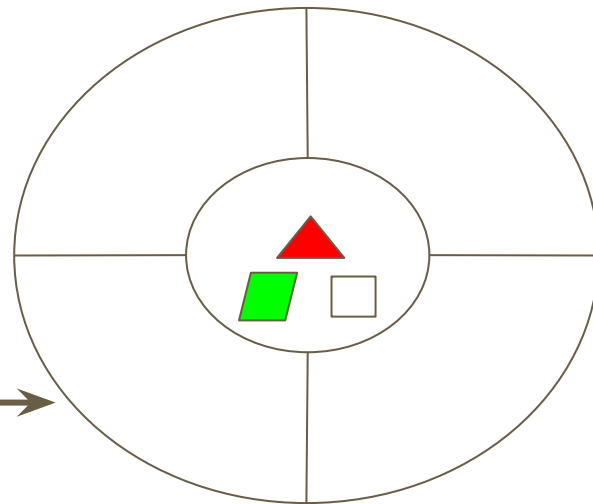
```
public class CajaDeAhorro extends CuentaBancaria{  
  
    @Override  
    public void cerrar() {  
        //La Caja de Ahorro tiene una lógica propia para su cierre  
        //Asumimos que es totalmente diferente a la de la CuentaCorriente  
    }  
}
```

```
public class Banco {  
  
    public void cerrarCuenta(int numeroCuenta) {  
        CuentaBancaria cuenta = this.buscarCuenta(numeroCuenta);  
        cuenta.cerrar();  
    }  
}
```

BANCO

mensaje
cerrar()

CUENTA BANCARIA



```
public class Banco {  
  
    public void cerrarCuenta(int numeroCuenta) {  
        CuentaBancaria cuenta = this.buscarCuenta(numeroCuenta);  
        cuenta.cerrar();  
    }  
}
```

CUENTA BANCARIA

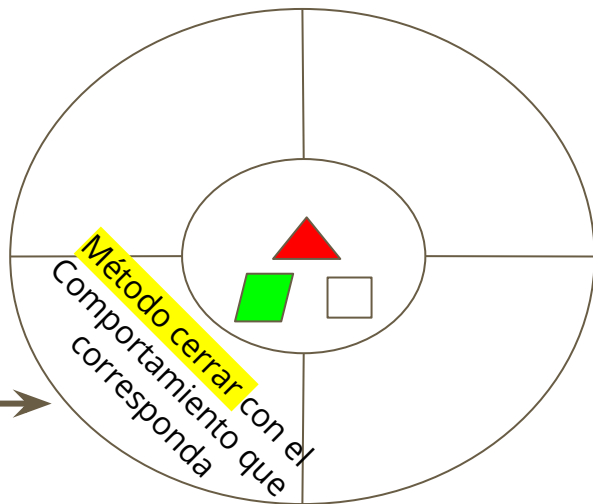
BANCO

mensaje

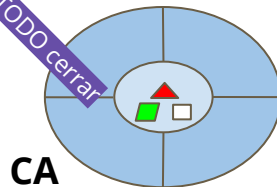
cerrar()

mensaje

cerrar()

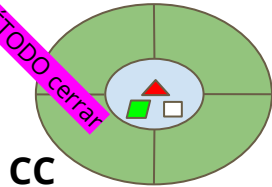


MÉTODO cerrar
CA



CA

MÉTODO cerrar
CC



CC

P00 en Java: Polimorfismo

```
public class Banco {  
  
    public void cerrarCuenta(int numeroCuenta) {  
        CuentaBancaria cuenta = this.buscarCuenta(numeroCuenta);  
        cuenta.cerrar();  
    }  
}
```

MENSAJE

```
public class CajaDeAhorro extends CuentaBancaria{
```

```
@Override
```

```
public void cerrar() {
```

METODO

```
//La Caja de Ahorro tiene una lógica propia para su cierre
```

```
//Asumimos que es totalmente diferente a la de la CuentaCorriente
```

```
}
```

```
public class CuentaCorriente extends CuentaBancaria{
```

```
@Override
```

```
public void cerrar() {
```

METODO

```
//La Cuenta corriente tiene una lógica propia para su cierre
```

```
//Asumimos que es totalmente diferente a la de la CajaDeAhorro
```

```
}
```

Opción 1)

```
CuentaBancaria>>extraer:unMonto  
|rojo|  
self class = CuentaCorriente if True:[rojo:= self rojoPermitido.]  
if False:[rojo := 0.]  
(self saldo + rojo >= unMonto)  
if True: [self saldo: self saldo - unMonto].
```

```
class CuentaBancaria  
{  
  method extraer(unMonto)  
  {  
    var rojo = 0  
    if (self.class() == CuentaCorriente) rojo = self.rojoPermitido()  
    if ((self.getSaldo() + rojo) >= unMonto) self.setSaldo(self.getSaldo() - unMonto)  
  }  
}
```


Opción 2)

CuentaBancaria>>extraer:unMonto

```
|rojo|
self tipo = 'cuentacorriente'
if True:[rojo:= self rojoPermitido.]
if False:[rojo := 0.]
(self saldo + rojo >= unMonto)
if True: [self saldo: self saldo - unMonto].
```

Opción 2)

```
class CuentaBancaria {
  method extraer(unMonto) {
    var rojo = 0
    if (self.getTipo() == 'cuentacorriente') rojo = self.rojoPermitido()
    if ((self.getSaldo() + rojo) >= unMonto) self.setSaldo(self.getSaldo() - unMonto)
  }
}
```

Opción 3)

```
CuentaBancaria>>extraer:unMonto  
^self subclassResponsibility
```

```
CajaDeAhorro>>extraer:unMonto  
(self saldo >= unMonto)  
if True: [self saldo: self saldo - unMonto].
```

```
CuentaCorriente>>extraer:unMonto  
(self saldo + self rojoPermitido >= unMonto )  
if True: [self saldo: self saldo - unMonto].
```

Opción 3)

```
class CuentaBancaria {  
    method extraer(unMonto)  
}  
  
class CajaDeAhorro inherits CuentaBancaria {  
    override method extraer(unMonto) {  
        if (self.getSaldo() >= unMonto) self.setSaldo(self.getSaldo() - unMonto)  
    }  
}  
  
class CuentaCorriente inherits CuentaBancaria {  
    override method extraer(unMonto) {  
        if ((self.getSaldo() + self.rojoPermitido()) >= unMonto)  
            self.setSaldo(self.getSaldo() - unMonto)  
    }  
}
```

Opción 4)

```
CuentaBancaria>>extraer:unMonto  
  (self chequearSaldoParaExtraccion:unMonto)  
  ifTrue: [self saldo: self saldo - unMonto].
```

```
CajaDeAhorro>>chequearSaldoParaExtraccion:unMonto  
^self saldo >= unMonto
```

```
CuentaCorreinte>>chequearSaldoParaExtraccion:unMonto  
^self saldo + self rojoPermitido >= unMonto
```

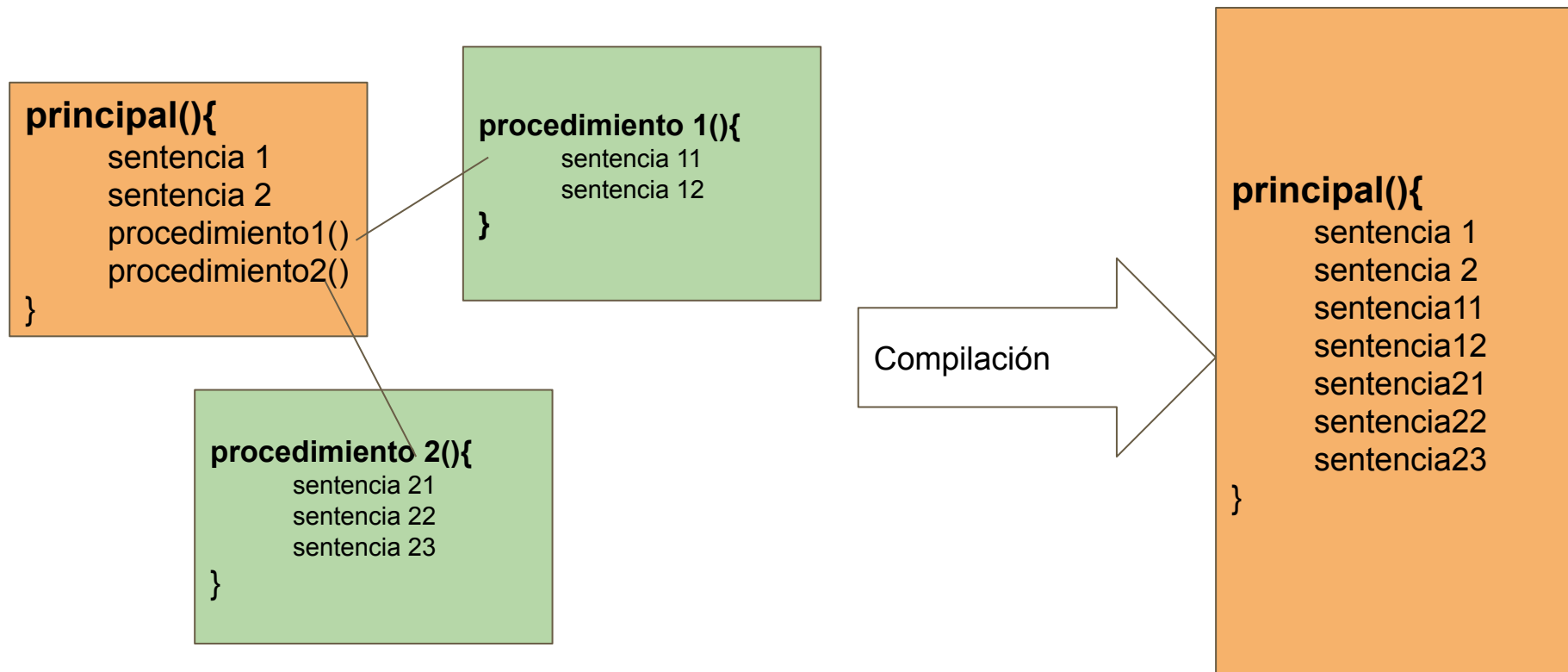
Opción 4)

```
class CuentaBancaria {  
  method extraer(unMonto) {  
    if(self.chequearSaldoParaExtraccion(unMonto)) self.setSaldo(self.getSaldo() - unMonto)  
  }  
}  
  
class CajaDeAhorro inherits CuentaBancaria {  
  override method chequearSaldoParaExtraccion(unMonto) = self.getSaldo() >= unMonto  
}  
  
class CuentaCorriente inherits CuentaBancaria {  
  override method chequearSaldoParaExtraccion(unMonto) =  
    (self.getSaldo() + self.rojoPermitido()) >= unMonto  
}
```

Repaso P00

Binding dinámico

P00 en Java: Binding estático vs. Binding dinámico



P00 en Java: Binding dinámico

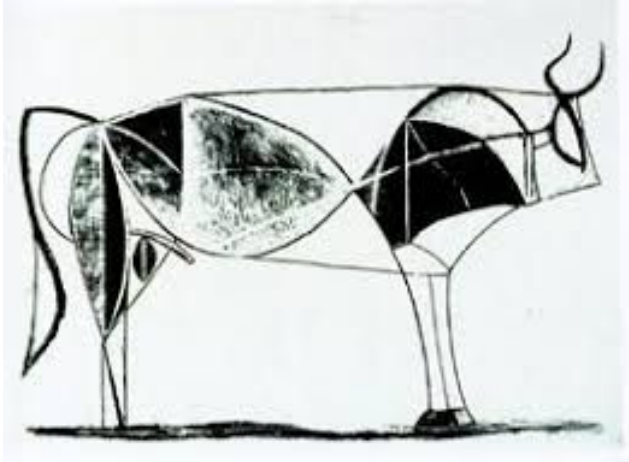
```
public class Banco {  
  
    List<CuentaBancaria> cuentas = new ArrayList<CuentaBancaria>();  
  
    public void cerrarCuentas() {  
        //cuentas.forEach((CuentaBancaria cuenta) -> cuenta.cerrar());  
        for (CuentaBancaria cuentaBancaria : cuentas) {  
            cuentaBancaria.cerrar();  
        }  
    }  
}
```

MENSAJE

```
public class CajaDeAhorro extends CuentaBancaria{  
  
    @Override  
    public void cerrar() { METODO  
        //La Caja de Ahorro tiene una lógica propia para su cierre  
        //Asumimos que es totalmente diferente a la de la CuentaCorriente  
    }  
}
```

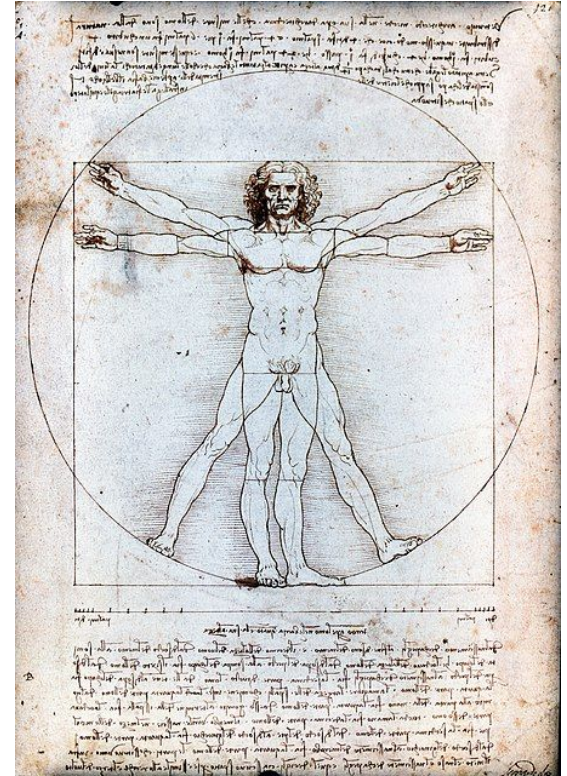
```
public class CuentaCorriente extends CuentaBancaria{  
  
    @Override  
    public void cerrar() { METODO  
        //La Cuenta corriente tiene una lógica propia para su cierre  
        //Asumimos que es totalmente diferente a la de la CajaDeAhorro  
    }  
}
```

Repaso POO - Abstracción



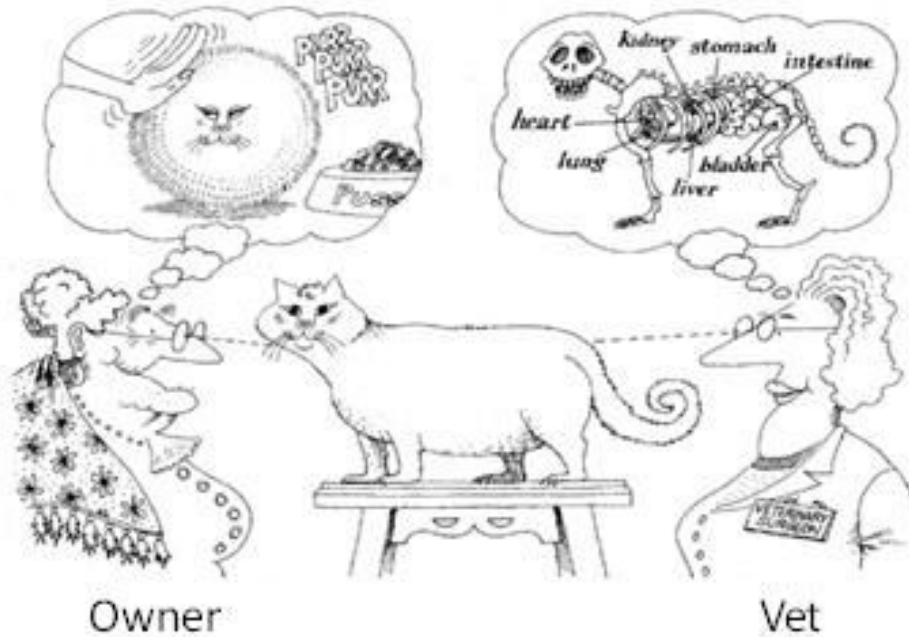
Pablo Picasso, Bull - plate December 1945

**Recomendación: Chapter 2 (Abstraction) del libro
“Budd - Intro to OOP”**



Hombre Vitruvio, da Vinci 1490

Repaso P00 - Abstracción



Conceptos vistos

- Clases
- Constructores
- Mensaje vs. método
- Encapsulamiento
- Abstracción
- Herencia
- Clases abstractas y métodos abstractos
- Polimorfismo

Objetivos originales de Java

- Orientado a objetos
- Simple
- Con soporte para concurrencia
- Con soporte para distribución
- Garbage collection
- Portabilidad
- Estáticamente tipado



James Gosling

Objetivos originales del lenguaje

...simple



Objetivos originales del lenguaje

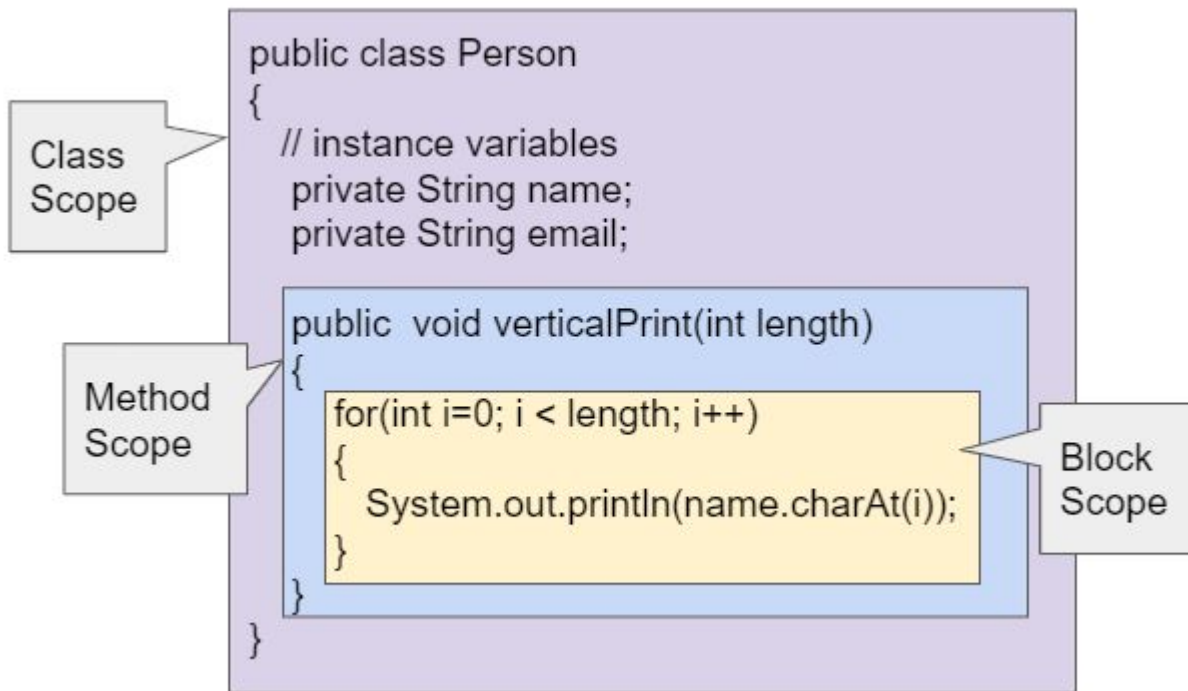
...garbage collection



Objetivos originales del lenguaje

...garbage collection y scopes

```
public class Ciclista {  
    public float calcularVelocidad()  
    {  
        float velocidad=1;  
        //calculo de velocidad  
        return velocidad;  
    }  
}
```



Objetivos originales del lenguaje

...con soporte para concurrencia

Si no la controlo:
posibles Inconsistencias de datos

Si la controlo mal: deadlock



Objetivos originales de Java

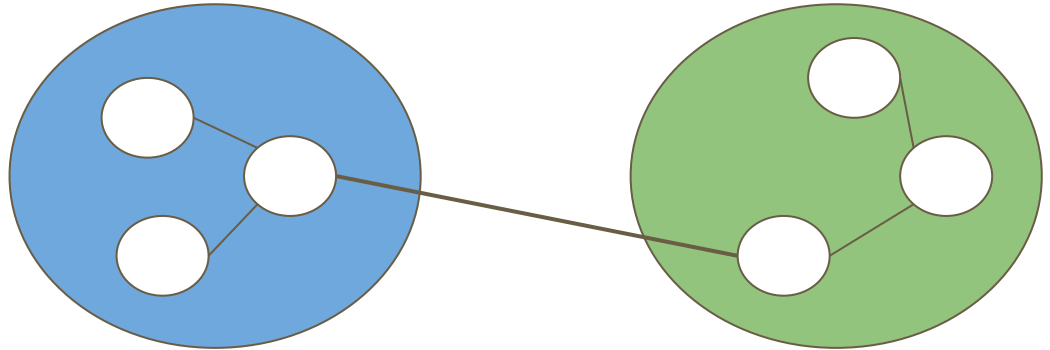
- Orientado a objetos
- Simple
- Garbage collection
- Con soporte para concurrencia
- Con soporte para distribución
- Portabilidad
- Estáticamente tipado



James Gosling

Objetivos originales del lenguaje

...con soporte para distribución



Objetivos originales de Java

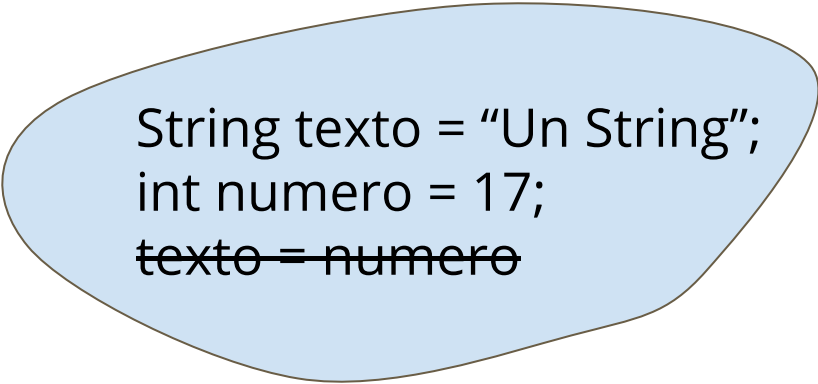
- Orientado a objetos
- Simple
- Garbage collection
- Con soporte para concurrencia
- Con soporte para distribución
- Estáticamente tipado
- Portabilidad



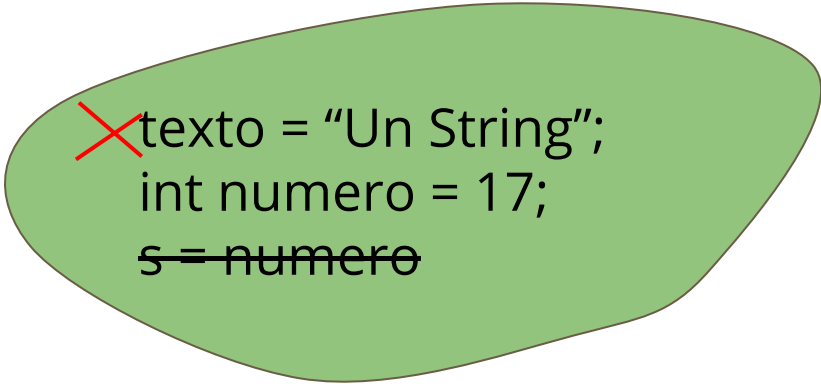
James Gosling

Objetivos originales del lenguaje

...estáticamente tipado



```
String texto = "Un String";  
int numero = 17;  
texto = numero
```

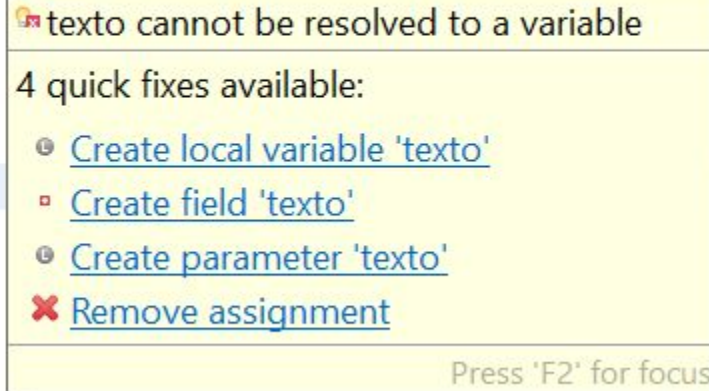


```
× texto = "Un String";  
int numero = 17;  
s = numero
```

Objetivos originales del lenguaje

...estáticamente tipado. Los tipos se chequean en tiempo de compilación

```
public void estaticamenteTipado() {  
    texto="String";  
}
```



A screenshot of an IDE error message. The message box is yellow with a black border. It contains the text "texto cannot be resolved to a variable" with a small icon of a lightbulb and a red 'x' to the left. Below this, it says "4 quick fixes available:". There are four options listed, each with a small icon to its left: a blue circle with a white 'e' for "Create local variable 'texto'", a red square for "Create field 'texto'", a blue circle with a white 'e' for "Create parameter 'texto'", and a red 'x' for "Remove assignment". At the bottom right of the box, it says "Press 'F2' for focus".

texto cannot be resolved to a variable

4 quick fixes available:

- [Create local variable 'texto'](#)
- [Create field 'texto'](#)
- [Create parameter 'texto'](#)
- ✗ [Remove assignment](#)

Press 'F2' for focus

Objetivos originales de Java

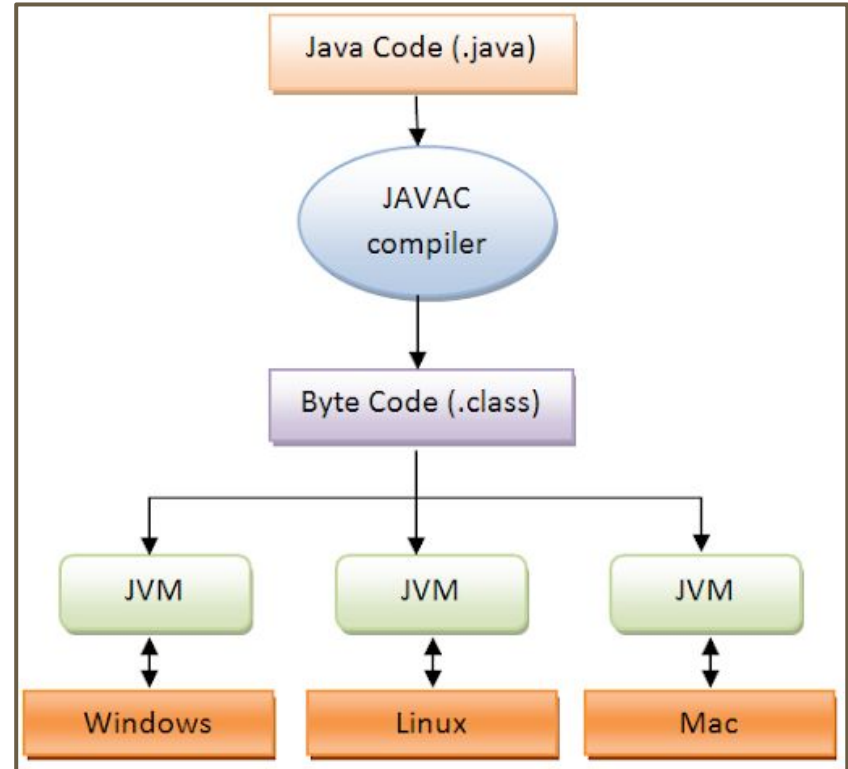
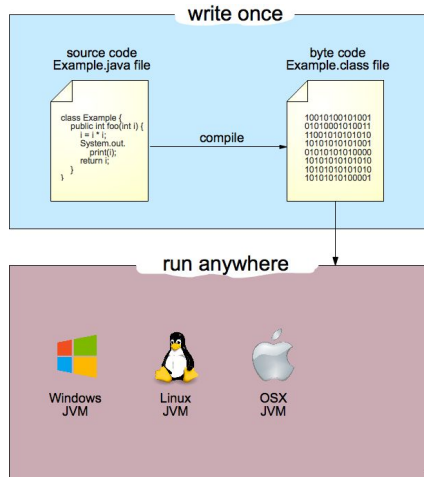
- Orientado a objetos
- Simple
- Garbage collection
- Con soporte para concurrencia
- Con soporte para distribución
- Estáticamente tipado
- Portabilidad



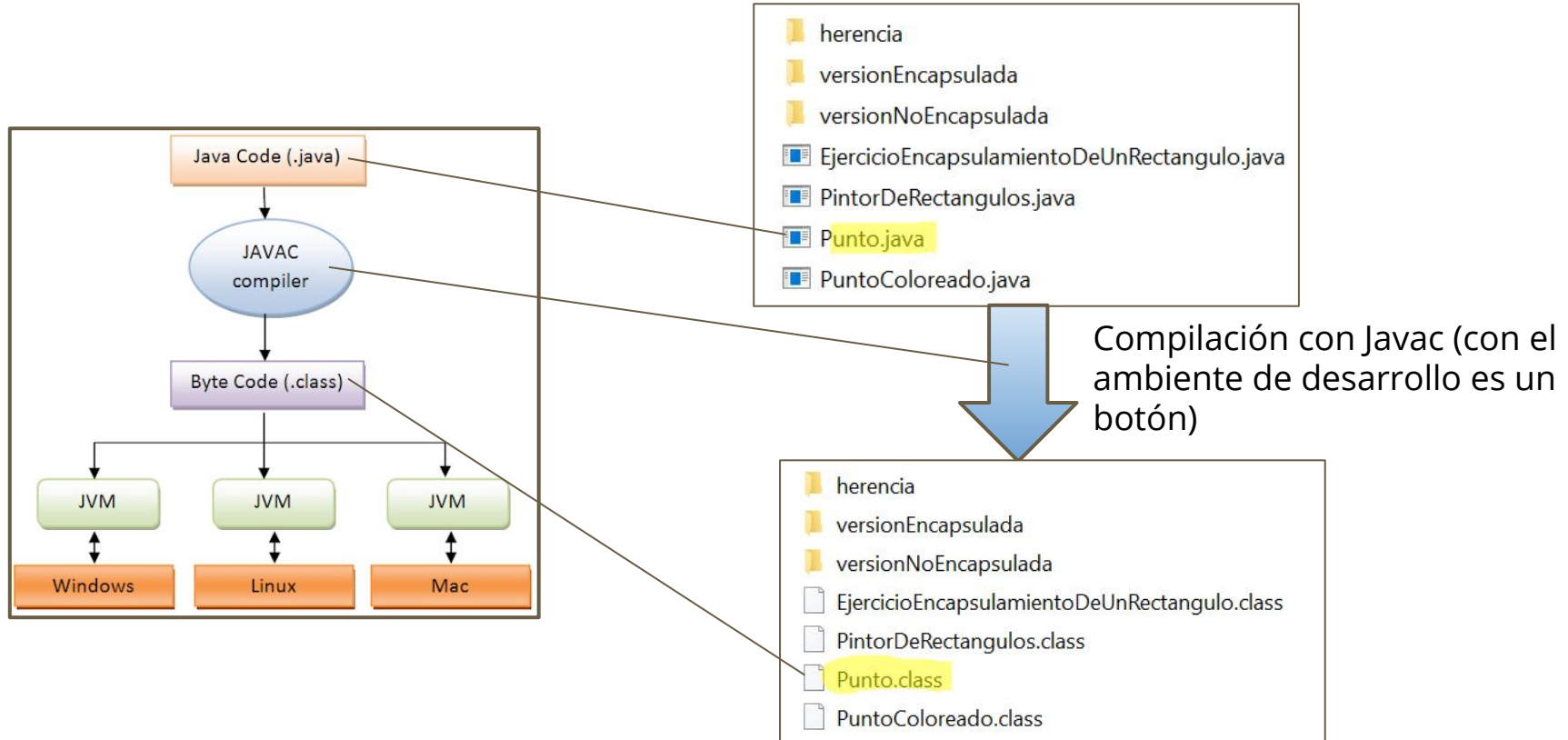
James Gosling

Objetivos originales del lenguaje

...portabilidad



Objetivos originales del lenguaje: Portabilidad



PARTE 2 - COMPONENTES DE LA PLATAFORMA

- **1-Lenguaje**
- **2-JDK: Java development toolkit**
- **3-JRE: Java Runtime Environment**

¿De qué se compone Java?: 1-Lenguaje



The Java® Language Specification *Java SE 16 Edition*

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley
Daniel Smith
Gavin Bierman

¿De qué se compone Java?: 1-Lenguaje

8.1. Class Declarations

A class declaration specifies a new named reference type.

There are two kinds of class declarations: *normal class declarations* and *enum declarations*.

ClassDeclaration:

[*NormalClassDeclaration*](#)

[*EnumDeclaration*](#)

NormalClassDeclaration:

[*{ClassModifier}*](#) class [*TypeIdentifier*](#) [*\[TypeParameters\]*](#) [*\[Superclass\]*](#) [*\[Superinterfaces\]*](#) [*ClassBody*](#)

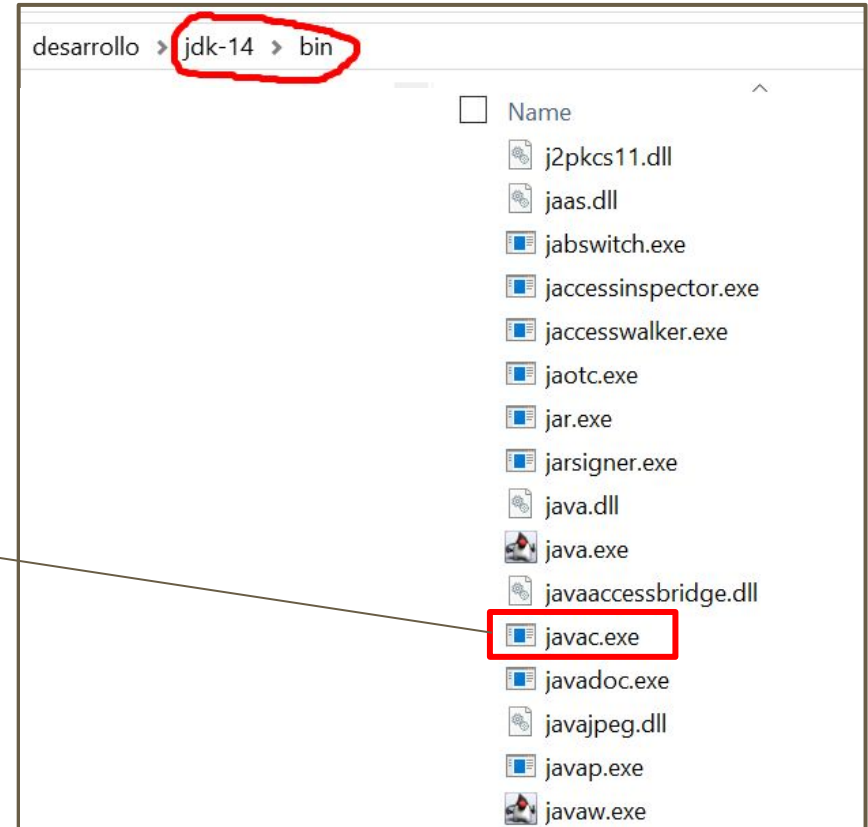
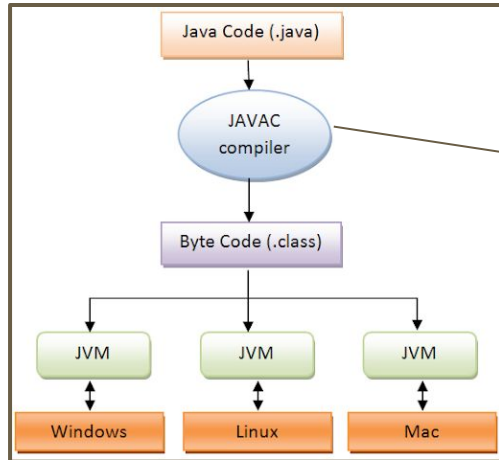
The rules in this section apply to all class declarations, including enum declarations. However, special rules apply to enum declarations with regard to class modifiers, inner classes, and superclasses; these rules are stated in [§8.9](#).

The *TypeIdentifier* in a class declaration specifies the name of the class.

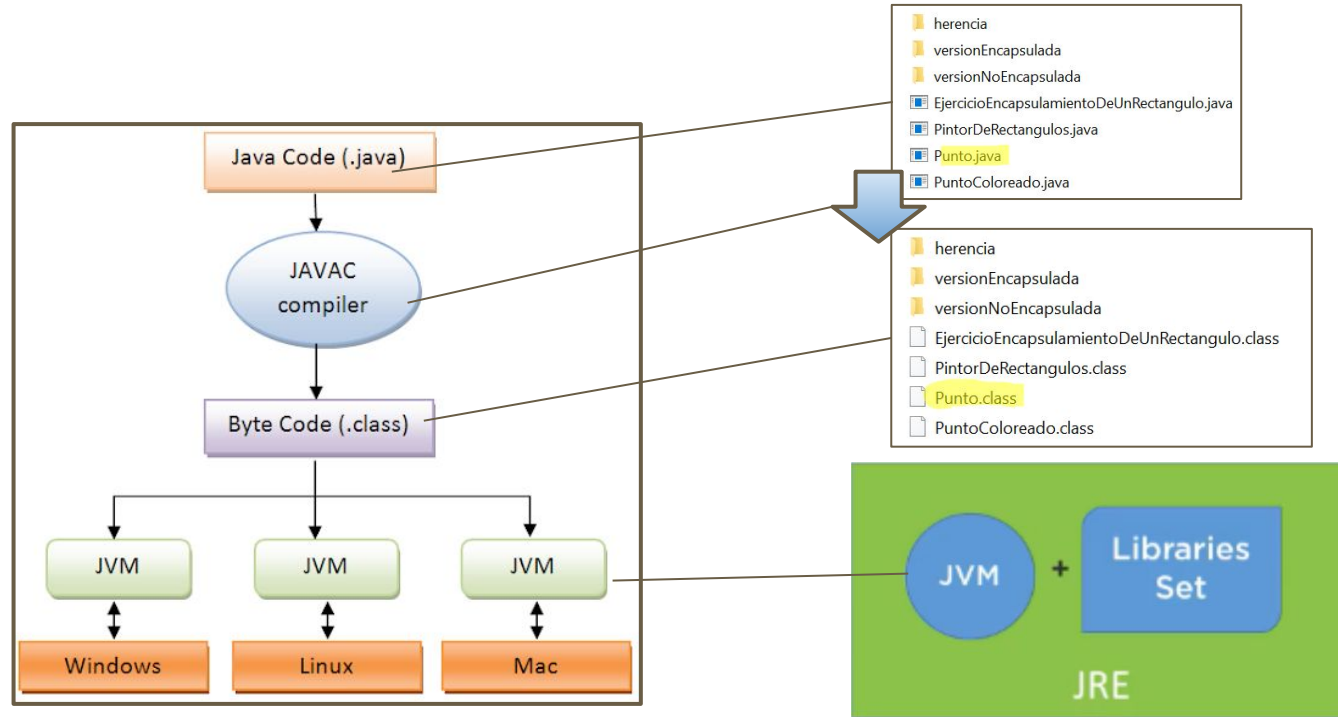
It is a compile-time error if a class has the same simple name as any of its enclosing classes or interfaces.

The scope and shadowing of a class declaration is specified in [§6.3](#) and [§6.4](#).

¿De qué se compone Java?: 2-JDK (Toolkit de desarrollo)



¿De qué se compone Java?: 3-JRE (PLATAFORMA DE EJECUCIÓN DE PROGRAMAS JAVA)



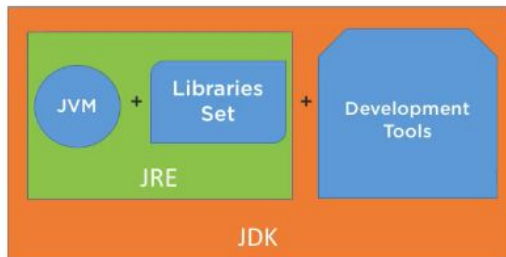
Varias implementaciones de JVM:
http://en.wikipedia.org/wiki/List_of_Java_virtual_machines

¿Entonces qué necesitamos para programar en Java?

Aprender y aplicar buenas prácticas y criterio para diseñar y desarrollar software de calidad



JDK (Ya incluye la JRE)



Java Development Kit

Ambiente de desarrollo



```
bioplat2 - BioPlatModeloClientefico/src/test/java/edu/unlp/medicine/packages/bioplatR/ImportExperimentsFromTCGATest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
1 package edu.unlp.medicine.rpackages.bioplatR;
2
3 import java.util.ArrayList;
4
5
6
7
8
9
10 /**
11  * It is necessary to install in R: Rserve and BioplatR. Then startup the Rserve on port 6336 thr
12  *
13  *
14  */
15
16 public class ImportExperimentsFromTCGATest extends RPackagesTestCase {
17
18     String cancer_study_id_for_testing = "brca_tcga";
19     String subset_id = "brca_tcga_mrna";
20     List<String> p53_pathway_genes=new ArrayList<String>();
21     List<String> clinical_attribute_names=new ArrayList<String>();
22     String a_mrna_profile_id="brca_tcga_mrna_median_zscores";
23
24     @Override
25     protected void setUp() {
26         //super.setUp();
27         p53_pathway_genes.add("TP53");
28         p53_pathway_genes.add("MDM2");
29         p53_pathway_genes.add("MDM4");
30         p53_pathway_genes.add("CDKN2A");
31         p53_pathway_genes.add("CDKN2B");
32         p53_pathway_genes.add("TP53BP1");
33
34         clinical_attribute_names.add("DFS_MONTHS");
35         clinical_attribute_names.add("DFS_STATUS");
36
37     }
38 }
```

Otros: IntelliJ IDEA, NetBeans,
BlueJ

Ambiente de desarrollo (Eclipse)

□ Qué aprovechar de un ambiente de desarrollo?

- Administración de workspaces
- Syntax highlighting.
- Auto Syntax Checking.
- Quick fix
- Content assistant o IntelliSense (Ctrl Space)
- Code generation
- Herramientas de debug.
- Herramientas de Testing
- Integración con controladores de versiones.
- Vista de problemas

Instalación de Java

<https://jdk.java.net/14/>

JDK 14 General-Availability Release

This page provides production-ready open-source builds of the Java Development Kit, version 14, an implementation of the Java SE 14 Platform under the GNU General Public License, version 2, with the Classpath Exception.

Commercial builds of JDK 14 from Oracle, under a non-open-source license, can be found at the Oracle Technology Network.

Documentation

- Features
- Release notes
- API Javadoc
- Tool Specifications

Builds

Linux/x64	tar.gz (sha256)	198578061 bytes
macOS/x64	tar.gz (sha256)	193317458
Windows/x64	zip (sha256)	198745139

Notes

- The Alpine Linux build previously available on this page was removed as of the first JDK 14 release candidate. It's not production-ready because it hasn't been tested thoroughly enough to be considered a GA build. Please use the [early-access JDK 15 Alpine Linux build](#) in its place.
- If you have difficulty downloading any of these files please contact jdk-download-help_ww@oracle.com.

<https://www.eclipse.org/downloads/>