POO

Manuel J. Molino Milla

IES V. del Carmen

Departamento de Informática

7 de enero de 2026

Logo

Figura: Logo Java

Contenido

¿Que es un objeto?

- Un objeto representa una entidad en el mundo real que se puede identificar claramente.
- Por ejemplo, un estudiante, un escritorio, un círculo, un botón, e incluso un préstamo todos pueden ser vistos como objetos.
- Un objeto tiene una identidad única definido por su estado y su comportamiento.

Estado También conocido como sus propiedades o atributos. Ejemplo para un círculo un atributo puede ser el radio. Un rectángulo su ancho y alto.

Comportamiento También conocido como sus acciones y es definida por *métodos*. Cuando invocamos un método le estamos pidiendo al objeto que haga algo. Ejemplo *getArea()* en un círculo nos daría el área.

Objetos

- Objetos de un mismo tipo son definidos usando una clase común.
- Una *clase* es una plantilla o modelo que nos dice que atributos y métodos tendrá un objeto.
- Un objeto es una *instancia* de una clase.
- Podemos crear muchas *instancias* de una clase.
- Los términos *objetos* e *instancias* son sinónimos.
- La relación entre las clases y los objetos es análoga a la que existe entre una receta de pastel de manzana y pastel de manzana.
- Podemos hacer tantas tartas de manzana como quieras de una sola receta

Clase círculo

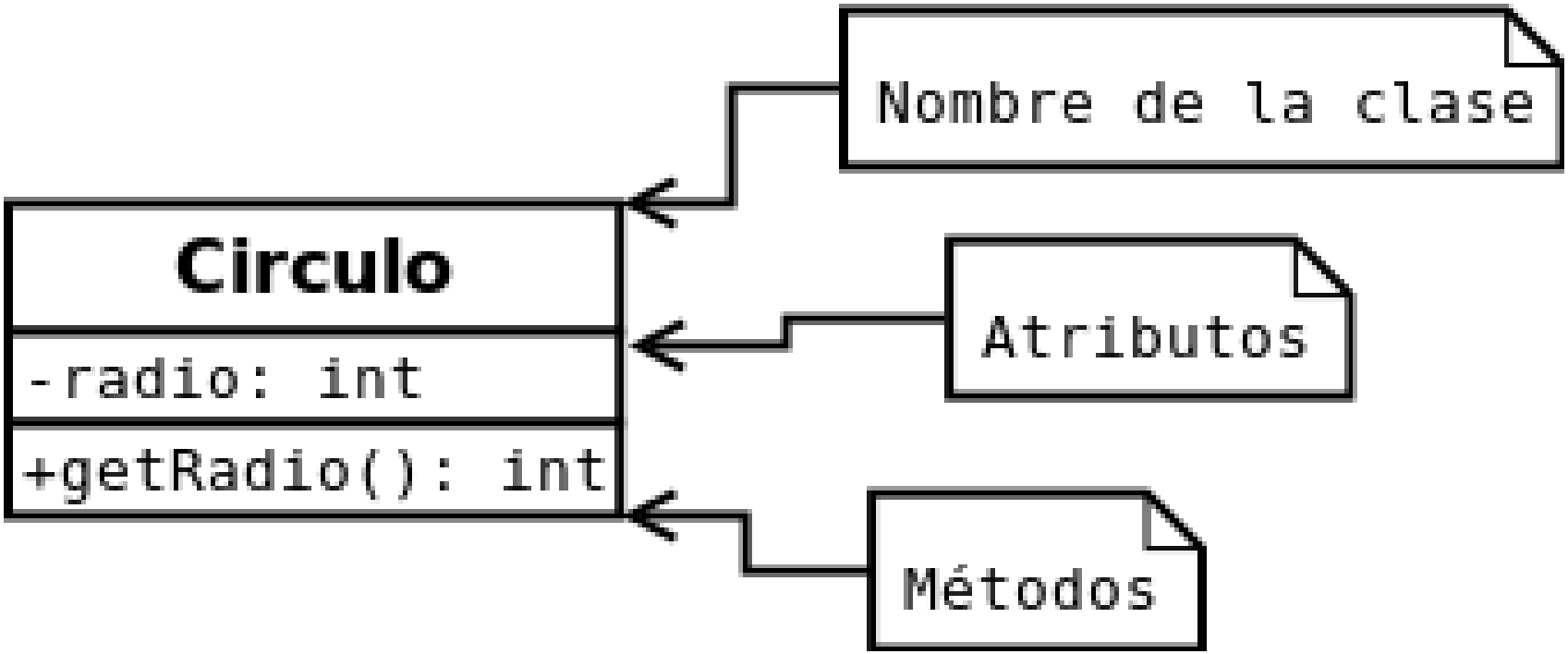


Figura: UML de la clase círculo

Clase rectángulo

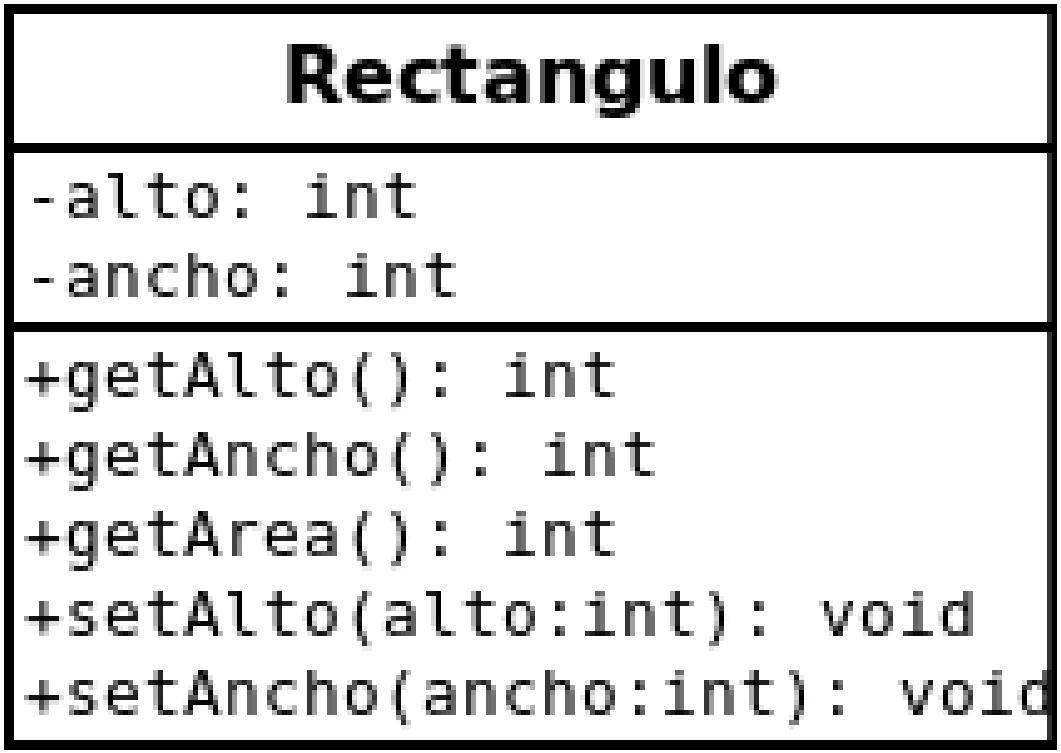


Figura: UML de la clase rectangulo

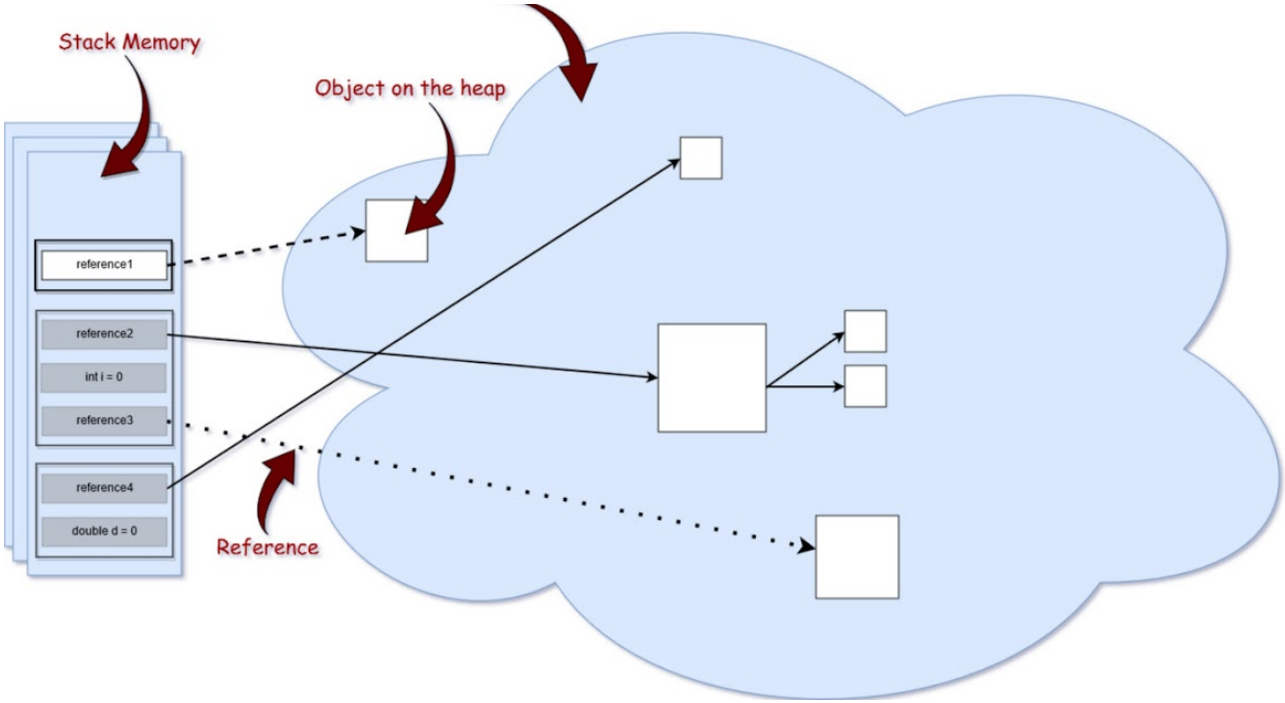
- El signo + indica que el atributo o método es de acceso público.
- El signo - indica que el atributo o método es de acceso privado.

Introduccion

- Java es un lenguaje orientado a objetos *puros*
- C++ y Java es un lenguaje hibrido.
- En Java se asume que se desea realizar *POO*
- Todo *objeto* se manipula mediante *referencias*.
- Símil: una television (objeto) se manipula con un mando a distancia (la referencia).
- Todos los objetos se suelen crear usando la palabra clave **new**
- Las referencias se almacenan en la pila.
- Los objetos se almacenan en el monticulo.
- Hemos visto que existen los tipos de datos *primitivos*
- Los objetos los podemos considerar nuevos tipos de datos.

Stack or Heap

Pila o montículo



Clases

En Java utilizamos clases

- Los lenguajes *OO* suelen usar la palabra **class**
- *class UnNuevoTipo* {código de la clase}
- Creamos los nuevos tipos de esta manera:
- *UnNuevoTipo t = new UnNuevoTipo();*
- Comparemos con los datos primitivos:
- int numeroEntero;
- double numeroReal = 2.23;
- No definimos ni *int* o *double* porque van implicitos en java.
- *POO* conlleva la creacion de objetos y el envio de mensajes entre dichos objetos.
- Las clases definen los atributos de los objetos (caracteristicas).
- Y tambien los metodos usados para intercambiar mensajes entre los objestos.

Campos

Son los atributos de los objetos

Ejemplo de una clase Fecha, con atributos privados:

```
1 public class Fecha{
2     private int dia;
3     private int mes;
4     private int anno;
5 }
```

Podemos intentar usar la clase de la siguiente forma:

```
1 public class TestFecha{
2     public static void main(String[] args){
3         Fecha f = new Fecha();
4         f.dia=23;
5         f.mes=12;
6         f.anno=2001;}
7     }
```

Genera errores de compilacion: *The field Fecha.dia is not visible...*

Métodos

Son lo que se conoce como funciones en programacion

A traves de ellos podemos acceder a los atributos y darles valores. *getters y setters*

```
1  public class Fecha{
2      private int dia;
3      private int mes;
4      private int anno;
5  }
6  public int getDia(){
7      return dia;
8  }
9
10 public void setDia(int valor){
11     dia=valor;
12 }
13 public int getMes(){
14     return mes;
15 }
16 public void setMes(int valor){
17     mes=valor;
18 }
```

Métodos

Métodos de acceso

- Podemos acceder a los atributos a pesar de la visibilidad *private*.
- Se llaman métodos de acceso ya que podemos obtener su valor con los *get*
- Los métodos para asignar valor a los atributos son los *set*. Se llaman mutantes porque cambian el valor de los atributos.
- Al ser atributos privados solo se pueden acceder desde la clase, sin usar los *getters*.
- Esto recibe el nombre de *encapsulamiento*.

Ejemplo de uso de objetos

```
1  public class TestFecha{
2      public static void main(String[] args){
3          Fecha f = new Fecha();
4          f.setDia(23);
5          f.setMes(12);
6          f.setAnno(2001);
7          System.out.println("Día: "+ f.getDia());
8          System.out.println("Mes: "+ f.getMes());
9          System.out.println("Año: "+ f.getAnno());
10       }
11 }
```

Crear y usar objetos

Pasos para crear y usar objetos

- Definimos el objeto, *creamos una referencia en la pila*
- Fecha f;
- Creamos (instanciamos) el objeto, *reservamos memoria en el monticulo*
- f = new Fecha();
- Las dos lineas anteriores se pueden crear en una sola:
- Fecha f = new Fecha();
- A partir de aqui usamos los *setters* y *getters*

Otros metodos que no son de acceso

```java
public class Numero{
    private int digito;
    public void setValor(int valor){
        digito = valor;
    }
    public int getValor(){
        return digito;
    }
    public int valorDoble(){
        return digito * 2;
    }
}
```

```java
public class TestNumero{
    public static void main(String[] args){
        Numero n = new Numero();
        n.setValor(23);
        System.out.println("Numero: " + n.getValor());
        System.out.println("Doble: " + n.valorDoble());
     }
```

Generalidades

- Las clases deben escribir en mayuscula.
- Si la clase se llama *Numero*
- Se guarda en un fichero llamado *Numero.java*
- Hay metodos que retornan valores.
- En este caso indicamos que tipo de valor:
- public *int* getValor()
- Indicamos lo que se devuelve con *return valor*
- Cuando no se retorna ningun valor implica:
- Se retorna el tipo *void*
- *public void setValor(int valor)*
- A veces hay que pasarle parámetros
- Entre parentesis indica el parametro/s que se le pasa al metodo.
- *public void serValor(int valor)*

Parametros y valores de retorno

Definicion de metodo

- Las partes fundamentales de un metodo son:
- *Nombre*
- *Sus parametros*
- *Los valores de retorno*

Metodo:

- visibilidad tipoRetorno nombreMetodo (/* Lista parametros*/) {
- /* Cuerpo del metodo */
- }

Ejemplo:

- public int sumaValores (int valorUno, int ValorDos) {
- return valorUno+valorDos;
- }

Métodos

- El tipo de retorno es el tipo de valor que surge después de invocarlo.
- *public int metodoNuevo(...*
- int numero = metodoNuevo(...
- La lista de parámetros indica los tipos y nombres de la informaciones que es necesario pasar a ese método.
- Cada método se identifica por el nombre del método y la lista de parámetros.
- *public int metodoUno(int Uno, int Dos) {...*
- *public int metodoUno(int Uno) {...*
- *public int metodoUno(double Uno) {...*
- *public int metodoDos(double Uno) {...*

Lista de parámetros y return

- La lista de parámetros indica la informacion que se le pasa al metodo.
- Se puede pasar tanto tipos primitivos como objetos o *nada*
- *int metodo (int valor, Numero numero){...*
- *double metodoPI(){ return 3.1416;}*
- La palabra *return* devuelve el valor y obliga a abandonar el metodo.
- *boolean indicador(){ return true;}*
- *float naturalLogBase(){ return 2.718f;}*
- *void nada(){ return;}*
- *void nada2(){ }*
- En el caso de que el valor de retorno es *void* es innecesario la palabra *return*

Lista de parámetros dinamica

Podemos hacer llamadas del mismo metodo con distinto numero de parametros, siempre que sean del mismo tipo:

```
1  public class Arg {
2
3    public static void main(String[] args) {
4      System.out.println("Suma: " + suma(1));
5      System.out.println("Suma: " + suma(1,2));
6      System.out.println("Suma: " + suma(1,2,3));
7    }
8    public static int suma (int... args) {
9      int suma = 0;
10     for (int i = 0; i < args.length; i++)
11       suma += args[i];
12     return suma;
13   }
14 }
```

Javadoc: param y return

```
1  /** This method always returns immediately, whether or not
       the image exists. When this applet attempts to draw the
       image on the screen, the data will be loaded. The
       graphics primitives that draw the image will
       incrementally paint on the screen.
2  @param  url absolute URL giving the base location of the
       image
3  @param  name the location of the image, relative to the url
       argument
4  @return the image at the specified URL
5  @see    Image
6  */
7   public Image getImage(URL url, String name) {
8           try {
9               return getImage(new URL(url, name));
10          } catch (MalformedURLException e) {
11              return null;
12          }
13  }
```

Constructor

- Permite inicializar objetos.
- Si una clase tiene un constructor, *Java* llama automaticamente a este metodo cuando se crea un objeto.
- El nombre del constructor coindice con el nombre de la clase.
- Ejemplo:

```
1  class Numero {
2      private int valor;
3      public Numero(int valor) {
4          this.valor = valor;
5      }
6  }
7  public class Ejemplo {
8      public static void main(String[] args){
9          Numero numero = new Numero(3);
10     }
11 }
```

- ¿Cómo se debe llamar al fichero *Numero.java* o *Ejemplo.java*?

Constructores

- Cuando se ejecuta *new Numero()*:
- Se asigna almacenamiento y se invoca al constructor.
- A diferencia de otros metodos, suele escribirse en mayuscula.
- Los constructores pueden tener parámetros.
- Ejemplo:

```
1  class Numero {
2    private int valor = 5;
3      public Numerosss() {
4          System.out.println("Creando un nuevo numero");
5      }
6      public void setValor(int valor) {
7         this.valor = valor;
8      }
9  }
10 public class Ejemplo1 {
11     public static void main(String[] args){
12         Numero numero = new Numero();
13         numero.setValor(8);
14     }
15 }
```

Sobrecarga de métodos

Permite crear un objeto de mas una manera

```
1  class Numero {
2      private int valor;
3          public Numero() {
4              this.valor = 5;
5          }
6          public Numero(int valor) {
7              this.valor = valor;
8          }
9          .....
10 }
11 public class Ejemplo2 {
12     public static void main(String[] args){
13         Numero numero1 = new Numero1();
14         Numero numero2 = new Numero1(1);
15     }
16 }
```

¿Hay errores de compilación?

Constructores

- ¿Como distingue *Java* el constructor a invocar?
- Por los parámetros.
- public Numero() public Numero(int valor)
- ¿Que ocurre en el siguiente código?

```
1  class Numero{
2      public int valor;
3      ................
4  }
5
6  public class Ejemplo3{
7      public static void main(String[] args){
8      Numero n = new Numero();
9      }
10 }
```

- ¿Cómo se debe llamar el programa Numero.java o Ejemplo.java?
- No tiene constructor. ¿Funciona?
- El compilador de *Java* crea uno automáticamente.

Records en Java

- Aparecen a partir de la JDK 14.
- La intención es evitar el *boilerplate*, el código repetitivo.
- Las clases típicas con atributos privados, constructores, getters y setters mas los métodos hashCode(), equals() y toString(), se denominan *POJO* (Plain Old Java Object)
- Esto implica muchas líneas de código, aunque todas las clases tengan la misma estructura.
- Se puede simplificar usando los *Record*
- Además crea objetos inmutables (no tiene *setters*), no puede cambiar su estado.

Ejemplo sencillo

```java
//Creando una clase POJO
record Person(int id, String name){}
//Creando un objetos y usándolos
Person p1 = new PersonRecord(1,"Peter Parker");
Person p2 = new PersonRecord(2,"Spiderman");
System.out.println(p1.toString());
System.out.println(p1.equals(p2));
System.out.println(p1.name());
```

Ejemplo más complejos

```java
record Person(int age, int height) {
  public Person(int age) {
    this(age, 180);
  }
  public Person() {
    this(18);
  }
  @Override
  public String toString() {
    return "example override";
  }
  public boolean isAdult() {
    return age >= 18;
  }
  public static Person newborn(int height) {
    return new Person(0, height);
  }
}
```

Tipo de datos y Wrappers

Tipo de datos:

- **Primitivos**: son los únicos elementos de todo el lenguaje que no son considerados como objetos, se ubican en la pila.
- **No primitivos**: al no ser tipos primitivos, son considerados como objetos, se ubican en le montículo.

Wrappers:

- Para todos los tipos de datos primitivos, existen unas clases llamadas *Wrapper*, también conocidas como envoltorio, ya que proveen una serie de mecanismos que nos permiten envolver a un tipo de dato primitivo permitiéndonos con ello el tratarlos como si fueran objetos.
- Podemos trabajar con Wrappers a partir de *JDK 5*,

Métodos interesantes

static int parseInt(String s) Convierte la cadena *String s* en un número entero. Ejemplo de uso: *int i = Integer.parseInt("1");*

static double parseDouble(String s) Convierte la cadena *String s* en un número double. Ejemplo de uso: *double i = Double.parseInt("1");*

boolean isInfinite() nos dice si el número es *infinito*. Ejemplo: double d = (2.0 / 0); System.outlprintln("2.0/0 es infinito? "+ d.isInfinite());

boolean isNaN() nos dice si el número es *NaN*. Ejemplo: double d = (0.0 / 0); System.outlprintln("0.0/0 es NaN? "+ d.isNaN());

Ejemplo

```
1
2  public class Test{
3
4      public static void main(String args[]){
5          int x =Integer.parseInt("9");
6          double c = Double.parseDouble("5");
7
8          System.out.println(x);
9          System.out.println(c);
10     }
11 }
```

Wrappers

1. Byte para byte
2. Short para short
3. Integer para int
4. Long para long
5. Boolean para boolean
6. Float para float
7. Double para double
8. Character para char

Autoboxing and Unboxing

Unboxing

- Conversión de un objeto de un tipo *Wrapper* a su correspondiente valor primitivo.
- Ejemplo la conversión de *Integer* a *int*

Ejemplo:

```
1 // Creating an Integer Object with custom value say it be 10
2 Integer i = new Integer(10);
3
4 // Unboxing the Object
5 int i1 = i;
```

Autoboxing

Unboxing

- Conversión de un objeto de un tipo primitivo a su correspondiente valor objeto.
- Ejemplo la conversión de *int* a *Integer*

Ejemplo:

```
1 int i1 = i;
2 // Autoboxing of character
3 Character gfg = 'a';
4
5 // Auto-unboxing of Character
6 char ch = gfg
```

Preguntas