MÓDULO PROFESIONAL PROGRAMACIÓN

UD 5 – Utilización de objetos

Filosofía de enseñanza

- No dudes en pedirme que vaya más despacio si hablo demasiado rápido.
- No dudes en interrumpirme cuando tengas preguntas.
- Las tareas y los ejercicios en clase te pedirán que apliques los conceptos y técnicas aprendidas hasta ahora.
- Empieza por lo básico, codifica desde abajo.

"El conocimiento no tiene valor si no lo pones en práctica"

Programación Orientada a objetos (POO)

La mayoría de los programas útiles no se limitan a manipular números y cadenas.

En su lugar, tratan con elementos de datos que son más complejos y que representan más estrechamente las entidades en el mundo real. Ejemplos de estos elementos de datos son, las cuentas bancarias, los registros de empleados y las formas gráficas.

Programación Orientada a objetos (POO)

El lenguaje Java es ideal para diseñar y manipular estas entidades llamadas **objetos**.

En Java, se implementan clases que describen el comportamiento de estos objetos.

Programación Orientada a objetos (POO)

En esta unidad didáctica, aprenderás cómo **utilizar objetos que pertenecen a clases ya implementadas.**

Este conocimiento de conocimiento te preparará para la unidad didáctica 5 en la que aprenderás a implementar tus propias clases.

Programación Orientada a objetos (POO)

Hace ya algunas décadas que el **paradigma de orientación a objetos (OO)** tomó su lugar en el desarrollo de software.

POO

Aparece a finales de los 60, pero es a principios de los 80 cuando con el lenguaje **Smalltalk** comienza un interés claro hacia este paradigma.

Finalmente se acaba convirtiendo en el **paradigma de programación** referente a finales de los 80 con C++ y definitivamente a mediados de los **90** con **JAVA**.

POO

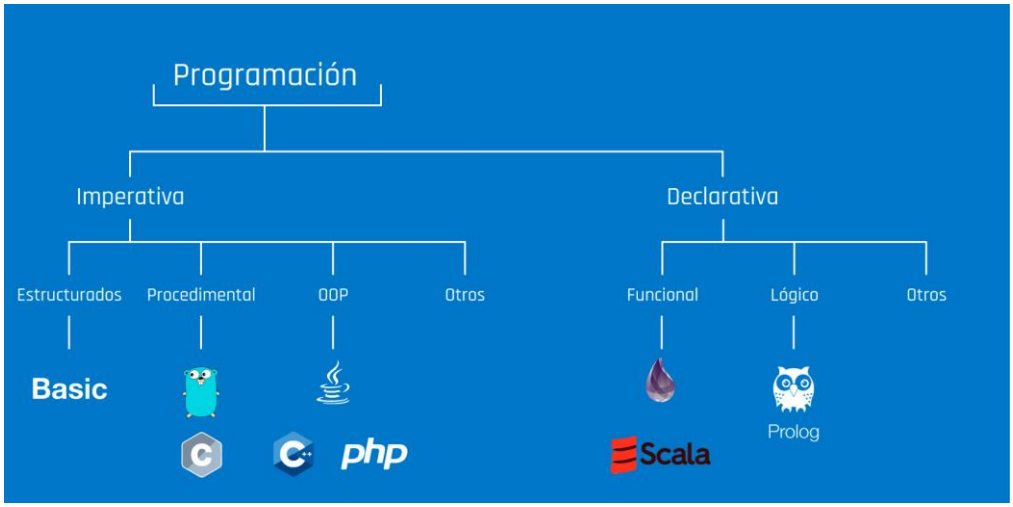
¿Qué es un **paradigma** de programación?

POO

Se considera un **paradigma de programación** como un estilo o forma de escribir programas.

Un paradigma de programación se caracteriza por los conceptos empleados y por la manera de abstraer los elementos utilizados en dar con la solución a un problema.

Paradigmas



POO

Cada nuevo paradigma responde a una necesidad real de nuevas formas de afrontar problemas, y a menudo un nuevo paradigma es creado como respuesta a las deficiencias de paradigmas anteriores.

La **programación orientada a objetos** (POO) surge en la historia como un intento para dominar la complejidad que, de forma innata, posee el software.

POO

Tradicionalmente, la forma de enfrentarse a esta complejidad ha sido empleando lo que llamamos **programación estructurada** que consiste en descomponer el problema objeto de resolución en subproblemas y más subproblemas hasta llegar a acciones muy simples y fáciles de codificar.

Es muy importante destacar que cuando hacemos referencia a la **programación orientada a objetos** no estamos hablando de unas cuantas características nuevas añadidas a un lenguaje de programación. Estamos hablando de una **nueva forma de pensar** acerca del proceso de descomposición de problemas y de desarrollo de soluciones de software.

POO

La programación orientada a objetos basa su resolución de problemas, en la creación de los llamados **objetos**.

La idea es combinar en una **única unidad**, tanto los datos como las operaciones que actúan sobre estos datos, tales unidades son los objetos.

POO

La **programación orientada a objetos** es el paradigma de programación, análisis y diseño de aplicaciones más dominante en la actualidad.

Prácticamente todos los lenguajes de programación de mayor demanda actuales, soportan el paradigma de la POO: **C++, Java, C#, PHP, Python,** etc.

POO

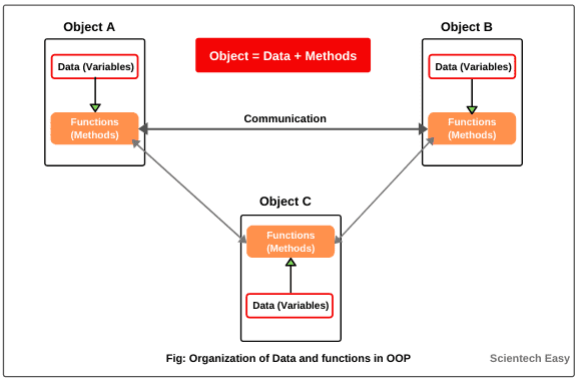
El mundo está lleno de objetos reales, los cuales se pueden representar como tales en una solución computarizada.

POO

Un programa realizado en Java está formado por un conjunto de clases las cuales en tiempo de ejecución del programa permiten construir objetos.

Por lo tanto, un programa Java, una vez instalado en la memoria RAM del ordenador, es un conjunto de objetos que interactúan entre si para desempeñar una tarea.

POO

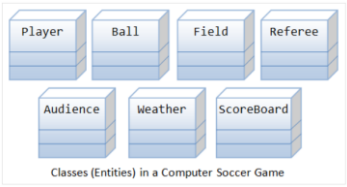


En este paradigma un programa se organiza como **colecciones cooperativas de objetos** que contienen datos y operaciones que operan sobre esos datos, y que se comunican entre sí mediante mensajes (solicitudes de ejecución de operaciones).

POO: ejemplo

Como ejemplo, supone que deseas escribir **juegos de fútbol de ordenador** (que considero una aplicación compleja). Es bastante difícil modelar el juego en lenguajes procedimentales. Pero al usar lenguajes OO, puede modelar fácilmente el programa de acuerdo con las "cosas reales" que aparecen en los juegos de fútbol:

- **Jugador**: los atributos incluyen nombre, número, ubicación en el campo y etc. y las operaciones incluyen correr, saltar, patear la pelota, etc.
- **Balón** (Ball)
- **Árbitro** (Referee)
- **Campo** (Field)
- **Espectadores** (Audience)
- **Clima** (Weather)



Classes (Entities) in a Computer Soccer Game

Lo más importante es que algunas de estas clases (como Balón y Espectadores) se pueden reutilizar en otra aplicación, por ejemplo, un juego de baloncesto de computadora, con poca o ninguna modificación.

POO: clase y objeto

Los objetos se crean a partir de una clase.

Para explicar el concepto de clase, se podría decir que una clase constituye el diseño de un objeto. Es una **plantilla** para crear objetos.

Una clase describe la **estructura y comportamiento** de los objetos que se crean a partir de ella.

POO: clase y objeto

Nos podemos imaginar a una clase como un molde para crear objetos.

Cada vez que **instanciemos una clase**, o lo que es lo mismo usemos el molde, estamos fabricando un objeto idéntico al original que determinó el molde, es decir, la clase.

POO: clase y objeto

Un objeto se puede definir desde un punto de **vista conceptual** como una entidad que tiene **estado, comportamiento e identidad.**

objeto = estado +comportamiento + identidad

- El **estado** son los valores que tienen los **atributos** en un instante determinado.
- El **comportamiento** es lo que puede hacer que está definido por los **métodos**.
- La **identidad** distingue a un objeto del resto y es independiente del estado. Es posible distinguir dos objetos en los cuáles todos sus atributos sean iguales. Cada objeto tiene su propia identidad.

POO: clase y objeto

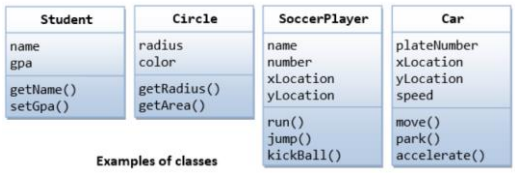
Veamos otro ejemplo, una aplicación para representar perros como Pluto, Milú, Snoopy, Brian…:

- Todos son perros que tienen unos **atributos**: raza, color, edad, peso, nombre…
- Todos realizan distintas acciones (**métodos**): ladran, comen, duermen…

Un objeto almacena sus estados en atributos y expone sus comportamientos a través de métodos

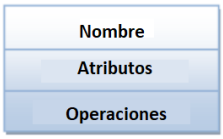
Si hubiese que realizar una aplicación orientada a objetos que los utilizara, ¿Cómo se realizaría? Crearíamos la **clase Perro** que tendría los **atributos**: raza, color, edad… y los **métodos** ladrar(), comer(), dormir()… Y Brian, Milú, Pluto y Snoopy serían **objetos de la clase Perro**.

POO: clase y objeto



Examples of classes

Una clase puede ser visualizada como una caja de 3 compartimientos que encapsula atributos y operaciones

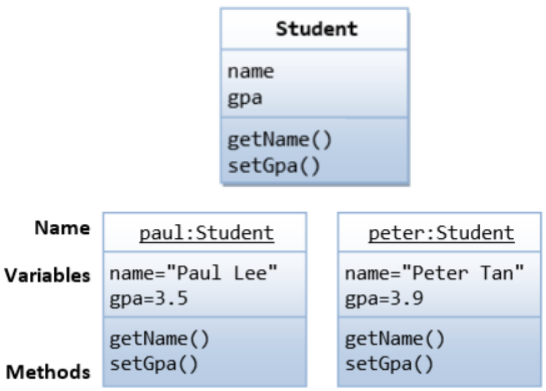


- **Nombre (o identidad):** identifica la clase.
- **Atributos:** caracteriza a los objetos de la clase.
- **Métodos (o comportamientos, funciones, operaciones):** contiene los comportamientos de la clase.

POO: clase y objeto

Durante la ejecución del programa se producirá la instanciación de la clase, e- decir la creación de los objetos.

La figura muestra 2 instancias de la clas Estudiante.



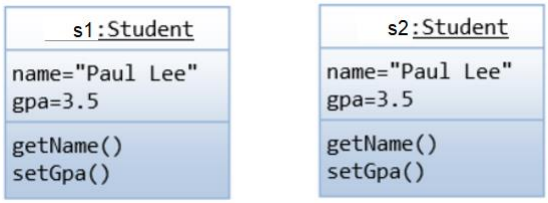
Two instances - paul and peter - of the class Student

GPA (**Grade Point Average**): es un término utilizado para asignar un valor numérico a las calificaciones acumuladas por un estudiante en el sistema estadounidense.

POO: clase y objeto

Aunque dos objetos tengan los mismos valores en sus atributos, son objetos diferentes.

- ¿Puede ocurrir esto?



POO: clase y objeto

Un objeto puede caracterizar una **entidad física** (un cliente, un coche, un piloto, una carta, un alumno, etc.) o una **entidad abstracta** (una fecha, una ecuación matemática, una conexión a una base de datos, un viaje, etc.)

POO: clase y objeto

¿Cómo se definen estas plantillas (clases) para fabricar (crear) objetos?

Cómo definir nuestras propias clases lo veremos en la unidad didáctica 5. Ahora nos encargaremos de aprender como **se crean y utilizan objetos** de clases ya definidas en Java.

POO: utilización de objetos

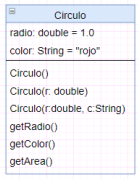
Puedes pensar en un calentador de agua como un objeto que puede llevar a cabo la funcionalidad de "obtener agua caliente".

Cuando requieres de esa funcionalidad (llamas a ese método) para disfrutar de una ducha caliente, no te importa si el calentador de agua utiliza gas o energía solar.

Por tanto aprenderemos a utilizar objetos en este apartado.

Creando objetos

Definición de la clase:



Para crear una **instancia de una clase (un objeto)** debes:

- **Declarar** una referencia a un objeto de una clase particular.
- **Construir** la instancia (es decir, asigne almacenamiento para la instancia e inicialice la instancia) utilizando el **operador "new"**.
- **Conectar** el objeto con la referencia mediante el operador de asignación

Por ejemplo, supongamos que ya tenemos definida una clase llamada Circulo, podemos crear instancias de Circulo de la siguiente manera:

```
//Declaro 3 instancias de la clase Circulo: c1, c2 y c3
Circle c1, c2, c3;

//Construyo las instancias(objetos) con el operador new y
//las vinculo con las referencias
c1 = new Circulo();            //1º constructor
c2 = new Circulo(2.0);         //2º constructor
c3 = new Circulo(3.0, "rojo"); //3º constructor

//Puedes declarar y construir en la misma sentencia
Circulo c4 = new Circulo();
```

Cuando una instancia se declara pero no se construye, tiene un valor especial llamado **null**.

Referencias a objetos

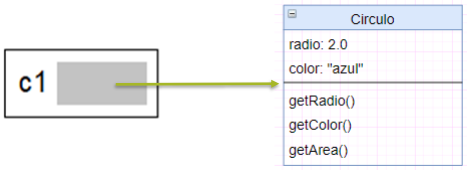
En Java, una **variable** cuyo tipo es una clase no contiene realmente un objeto. Simplemente contiene **la ubicación en memoria de un objeto**. El objeto en sí se almacena en otro lugar.

```
Circulo c1 = new Circulo(2.0, "azul");
```

La variable c1 se refiere al objeto Circulo que construyó el operador new. Técnicamente el operador new devuelve una referencia al nuevo objeto, y esa referencia se almacena en la variable c1.

Referencias a objetos

Es muy importante que recuerdes que la variable c1 no contiene el objeto. Se refiere al objeto.



Ejemplo

Person (Class)

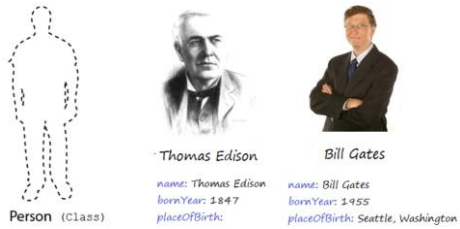
Thomas Edison

name: Thomas Edison
bornYear: 1847
placeOfBirth:

Bill Gates

name: Bill Gates
bornYear: 1955
placeOfBirth: Seattle, Washington

Ejemplo




```
Person billGates
        = new Person("Bill Gate", 1955, "Seattle, Washington");

public class Person {
    private String name;
    private Integer bornYear;
    private String placeOfBirth;

    public Person(String name, Integer bornYear, String placeOfBirth) {
        this.name = name;
        this.bornYear = bornYear;
        this.placeOfBirth = placeOfBirth;
    }
}
```

El operador punto

Definición de la clase:

Circulo
radio: double = 1.0 color: String = "rojo"
Circulo() Circulo(r: double) Circulo(r:double, c:String) getRadio() getColor() getArea()

Los atributos y métodos que pertenecen a un objeto se denominan formalmente **atributos (variables) miembro y métodos miembro. Significa que solo están disponibles cuando un objeto es creado.**

Para hacer referencia a un atributo o método miembro, debes:

- Primero, identificar el objeto que quieres usar y luego
- Usar el **operador punto (.)** para hacer referencia al atributo o método miembro deseado.

El operador punto

Volvamos con la clase **Circulo**, con dos atributos miembro: radio y color; y dos métodos miembros: getRadio () y getArea().

Hemos creado **dos instancias** de la clase Circulo, a saber, c1 y c2.

Para invocar el método getArea(), primero debes identificar la instancia de interés, por ejemplo c1, luego usar el operador de punto, en forma de c1.getArea(). Por ejemplo,

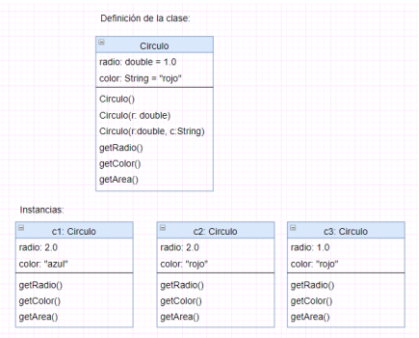
```
//Supongamos que la clase circulo tiene los atributos radio y color
//y los métodos getArea() y getRadio()
//Declaro y construyo las instancias c1 y c2 de la clase circulo

c1 = new Circulo();
c2 = new Circulo();

//Invoco los métodos miembros de la clase c1 con el operador punto
System.out.println(c1.getArea());
System.out.println(c1.getRadio());

//referencio los atributos miembros de la clase c1 con el operador punto
c2.radio=5.0;
c2.color="blue";
```

Atributos

Los atributos pueden ser de cualquiera de los tipos básicos de Java (boolean, char, byte, short, int, double, etc) o variables de referencia a otros objetos.

- Existen tres tipos de atributos:
  - Atributos **miembro** o de instancia
  - Atributos **estáticos** o de clase
  - Atributos **finales**

Generalmente, cuando hablamos de atributos nos referimos a atributos miembro.



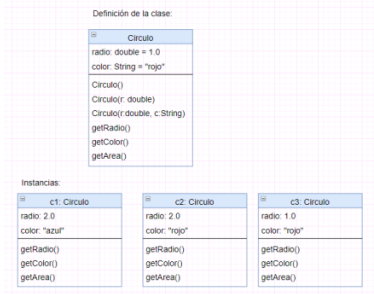
Métodos

Los métodos son funciones definidas dentro de una clase. Definen el comportamiento de los objetos instancia de la clase.

Pueden ser de cuatro tipos:

- Métodos **miembro** o de instancia.
- Métodos **estáticos** o de clase.
- Métodos **finales**. (los estudiaremos más adelante).
- Métodos **abstractos**. (los estudiaremos más adelante).

Generalmente, cuando hablamos de métodos nos referimos a **métodos de miembro.**



Métodos

Un método:

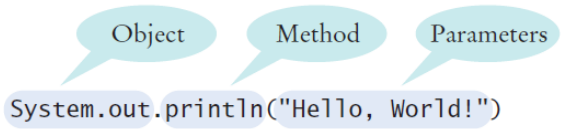
- Recibe parámetros de la parte del programa que lo llama,
- Realiza las operaciones definidas en el cuerpo del método, y
- Devuelve un valor (o nulo) a la parte del programa que lo llama.

Invocación a métodos

Otro ejemplo,

Para **utilizar un objeto**, como **System.out**, se especifica lo que se quiere hacer con él. En este caso, quieres imprimir una línea de texto. El método println realiza esta tarea.

No es necesario implementar este método -los programadores que escribieron la biblioteca biblioteca de Java ya lo hicieron por nosotros, pero sí es necesario llamar al método-



Invocación a métodos

Cada vez que se llama a un método en Java, es necesario especificar tres elementos

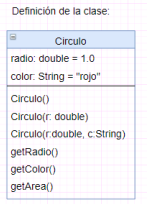
1. El **objeto** que se quiere utilizar (en este caso, System.out).
2. El **nombre** del método que se quiere utilizar (en este caso, println).
3. Un **par de paréntesis**, que contienen cualquier otra información que el método necesita (en este caso, "¡Hola, mundo!"). El término técnico para esta información es un **parámetro** para el método.

Nota: Ten en cuenta que los dos puntos en System.out.println tienen diferentes significados. El primer punto significa "localizar el objeto out en la clase System". El segundo punto significa "invocar el método println de ese objeto".

Inicialización de objetos mediante constructores

Un **constructor** se diferencia de un método ordinario en los siguientes aspectos:

- Un constructor se parece a un método especial que tiene el mismo nombre que el nombre de la clase.
- Los constructores son un tipo particular de métodos cuya ÚNICA función es inicializar los atributos de un objeto
- Los constructores **nunca devuelven un valor**



Inicialización de objetos mediante constructores

Los constructores solo pueden ser invocados por el operador **new**. Solo se puede utilizar una vez para inicializar el objeto construido. Una vez que se construye el objeto, ya no se puede llamar al constructor.

```
//Construyo las instancias(objetos) con el operador new y
//las vinculo con las referencias
c1 = new Circulo();             //1º constructor
c2 = new Circulo(2.0);          //2º constructor
c3 = new Circulo(3.0, "rojo");  //3º constructor
```

Parámetros

Conexión del objeto con su referencia

Creación del objeto

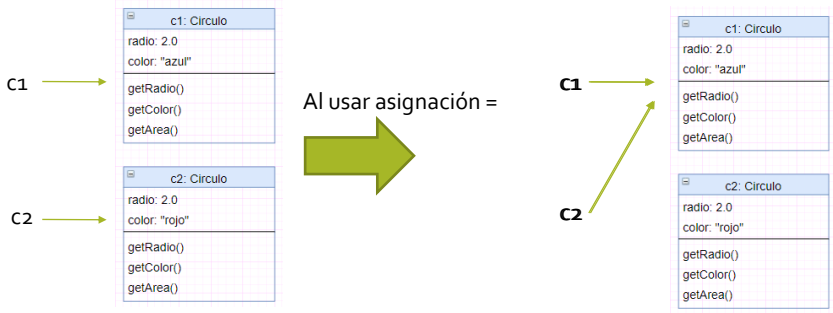
Asignación

La asignación de objetos no es trivial y no basta con el operador de asignación (=). Si se crean dos objetos y se asignan con el operador = se están convirtiendo en el mismo, no copiando sus valores. Es decir:

```
Circulo c1 = new Circulo(2.0, "azul");
Circulo c2 = new Circulo(2.0, "rojo");


c2=c1;

System.out.println(c1.color);   //azul
System.out.println(c2.color);   //azul

//parece que funciona pero...
c1.setColor("verde");

System.out.println(c1.color);   //verde
System.out.println(c2.color);   //verde
//¿Qué ha ocurrido?
```

Asignación

Al utilizar el operador de asignación lo que hemos hecho no es copiar los valores de un objeto en el otro, sino convertirlos en el mismo objeto:



Asignación

Por tanto si lo que se pretende es copiar sólo los valores de los atributos del objeto, lo que hay que hacer es copiar el valor de los atributos uno a uno.

```
Circulo c1 = new Circulo(2.0, "azul");
Circulo c2 = new Circulo(2.0, "rojo");

c2.radio=c1.radio;
c2.color=c1.color;
```

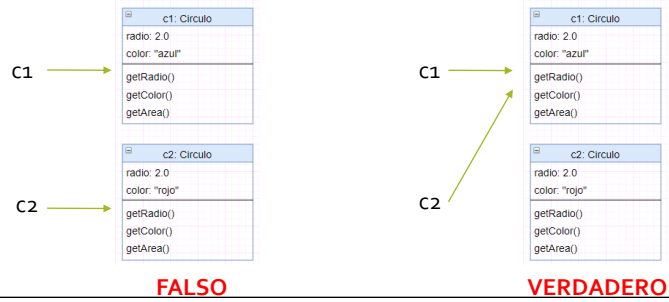
Importante esto sólo funciona cuando los atributos del objeto son tipos primitivos, si los atributo son otros objetos habrá que copiar también todos sus atributos uno a uno

Comparación

Al igual que la asignación, la comparación tampoco es trivial y no es válido utilizar el operador ==

```
if (c1==c2)
```

Esta comparación devuelve verdadero si son el mismo objeto y falso en otro caso



API Java

Las clases y los métodos de la biblioteca de Java aparecen en la documentación de la API.

La **API** es la **"interfaz de programación de aplicaciones".** Un programador que utiliza las clases Java para crear un programa informático (o aplicación) es un programador de aplicaciones.

Puedes encontrar la documentación de la API en la web.

https://docs.oracle.com/en/java/javase/11/docs/api/

La documentación de la API documenta todas las clases de la biblioteca de Java hay miles de ellas. La mayoría de las clases son bastante especializadas, y sólo unas pocas son de interés para el programador principiante.

API Java

Busquemos la clase **Random**

La clase **Random** proporciona un generador de números aleatorios.

API Java

La **documentación de la API** para cada clase comienza con una sección que describe el propósito de la clase. Luego vienen las tablas resumen de los constructores y métodos. Haz clic en el enlace de un método para obtener una descripción detallada.

La **descripción detallada de un método** muestra

- La **acción** que realiza el método.
- Los **parámetros** que recibe el método.
- El **valor que devuelve** (o la palabra reservada void si el método no devuelve ningún valor).

Como puedes ver, la clase Random tiene bastantes métodos. Aunque en ocasiones intimidante para el programador principiante, esto es un punto fuerte de la biblioteca estándar. Si alguna vez necesitas hacer un procesamiento con números aleatorios, es probable que haya un método que haga todo el trabajo por ti.

API Java

La documentación de la API contiene otra información importante sobre cada clase. **Las clases están organizadas en paquetes.** Un paquete es una colección de clases con un propósito relacionado.

La clase **Random** pertenece al paquete **java.util** que contiene muchas clases que proveen distinta utilidad.

Uso de la clase Random

Para utilizar la clase Random del paquete java.util, debes **importar el paquete.** Simplemente coloca la siguiente línea en la parte superior de su programa:

```
import java.util.Random;
```

Uso de la clase Random

La clase Random posee dos constructores.

```
Random rand = new Random();
```

Este constructor crea un generador de números aleatorios cuya semilla* es inicializada automáticamente, en base al tiempo actual. Esto conlleva que en cada ejecución la semilla cambie, es decir, que la secuencia de números aleatorios que se genera en cada ejecución siempre será diferente.

*semilla= es un número utilizado para inicializar un generador de números pseudoaleatorios. Un generador pseudoaleatorio de números es un algoritmo que produce una sucesión de números que es una muy buena aproximación a un conjunto aleatorio de números.

Uso de la clase Random

Un ejemplo de uso del segundo constructor

```
Random rand = new Random(1234);
```

Este constructor nos permite inicializar la semilla manualmente con un número entero cualquiera. Si este número es el mismo en cada ejecución, la secuencia de números aleatorios que se genera en cada ejecución será igual.

Uso de la clase Random

Un ejemplo de uso de la clase Random es el siguiente

```
Random rand = new Random();
int num = rand.nextInt(10); // que devuelve un valor entre 0 y 10, excluido 10
```

Un aspecto importante a tener en cuenta, es que el valor devuelto por el método miembro nextInt(10) es **mayor o igual a cero y menor a 10.** Es decir, el número devuelto puede ser 0, 1, 2, 3, 4, 5, 6, 7, 8 o 9.

Actividad

Copia el siguiente código y ejecútalo

```java
public class ClaseRandom {

    public static void main(String[] args) {

        Random rnd=new Random();

        int a1=rnd.nextInt();
        System.out.println(a1);

        a1=rnd.nextInt(100);
        System.out.println(a1);

        float a2=rnd.nextFloat();
        System.out.println(a2);

        double a3=rnd.nextDouble();
        System.out.println(a3);

        long a4=rnd.nextLong();
        System.out.println(a4);

        boolean a5=rnd.nextBoolean();
        System.out.println(a5);
    }

}
```

Uso de la clase Random

¿Si quiero generar un número aleatorio entre 1 y 10?

```java
Random rnd=new Random();
int num=rnd.nextInt(10)+1;
```

Actividad

1. Escribe un programa **SimuladorDado** que utilice la clase **Random** para simular el lanzamiento de un dado, imprimiendo un número aleatorio entre 1 y 6 cada vez que se ejecute el programa.

1. Si quisieras simular el lanzamiento de una moneda, ¿qué **método** de la clase Random usarías?

Actividad

3. Escribe un programa que muestre 50 números enteros aleatorios entre 100 y 199 (ambos incluidos) separados por espacios. Muestra además, el **máximo**, el **mínimo** y la **media** de esos números.
4. Escribe un programa que genere **2 enteros aleatorios** y pida al usuario que responda a la **expresión matemática** repetidamente hasta que la respuesta sea correcta. Por ejemplo el programa muestra 2 + 5 =? Si el usuario responde 7 el programa muestra "Correcto". De lo contrario el programa muestra "Respuesta incorrecta. ¿Cómo extenderías el programa para incluir expresiones con -, * y /?

Atributos y métodos estáticos

Un atributo o método miembro está disponible para su uso solo cuando el correspondiente objeto es creado. También se les llama **atributo/método de instancia.**

Por otro lado, un atributo o método estático pertenece a la clase y es compartido por todas las instancias de la misma. No es necesario crear un objeto para poder usarlos. También se les llama **atributo/método de clase.**

Atributos y métodos estáticos

Hasta ahora vimos que para hacer referencia a una atributo/método miembro, debe **identificar la instancia** y hacer referencia a ella mediante

- unObjeto.unAtributo o
- unObjeto.unMetodo().

Ejemplo:

```
Empleado unEmpleado = new Empleado("Pedro",1500);
unEmpleado.nombre;
unEmpleado.getSalario();
```

Atributos y métodos estáticos

Una atributo/método estático tiene una única ubicación de memoria común mantenida en la clase y compartida por todas las instancias. La JVM asigna una estática durante la carga de la clase. La variable estática existe incluso si no se crea ninguna instancia e independientemente del número de instancias creadas.

Se puede hacer **referencia a un atributo/método estático** a través de

- NombreClase.unAtributo o
- NombreClase.unMetodo()

Ejemplo:

```
Empleado.id;
Empleado.getId();
```

También se puede hacer referencia desde cualquiera de sus instancias **(pero no se recomienda)**, por ejemplo, instancia1.aVaraibleName. Ej: unEmpleado.id;

Atributos y métodos estáticos

Los **atributo/métodos miembro** pertenecen a las instancias(objetos). Para usar un atributo/método no estático, primero se debe construir una instancia (objeto).

Por otro lado, los **atributos/métodos estáticos** pertenecen a la clase, son de **naturaleza global.** No necesita construir ninguna instancia (objeto) antes de usarlos.

Atributos y métodos estáticos

- Un método estático no puede acceder a un método miembro directamente.
- (lo veremos en detalle cuando implementemos nuestros propios métodos)

Atributos y métodos estáticos

Un uso de atributos/métodos estáticos es el de proporcionar una variable "global", que es aplicable a todas las instancias de esa clase en particular (para fines tales como contar el número de instancias, bloqueo de recursos entre instancias, etc.).

Otro uso es proporcionar atributos y métodos globales, a los que pueden acceder otras clases, sin la necesidad de crear una instancia de esa clase proveedora.

Por ejemplo, la clase **java.lang.Math** se compone atributos y métodos estáticos puramente públicos.

La clase Math

Para usar **atributos estáticos** en la clase Math (como PI y E) o **métodos estáticos** (como random () o sqrt ()), no tienes que crear una instancia de la clase Math. Puedes invocarlos directamente a través del nombre de la clase, por ejemplo,

```
Math.PI;         //constante número PI
Math.E;          //constante número e
Math.random()    //número aleatorio de tipo double mayor o igual a 0.0 y menor que 1.0
Math.sqrt(x);    //raiz cuadrada de un número
Math.min(x,y);   //menor de dos números
Math.max(x,y);   //mayor de dos números
```

La clase Math

Ejemplo de uso de la clase Math para generar números aleatorios

```
int x = (int) (Math.random() * 10); // enteros del 0 al 9
```

La clase Math

La clase Math se encuentra en el paquete java.lang que no es necesario importar ya que Java lo importa automáticamente.

Mira el API de la clase Math y descubre que otros métodos presenta

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Math.html

Actividad

Copia el siguiente código y ejecútalo

```
public class ClaseMath {

    public static void main(String[] args) {
            int x;
            double rand,y,z;
            float max;

            rand = Math.random();
            x = Math.abs(-123);
            y = Math.round(123.567);
            z = Math.pow(2,4);
            max = Math.max(4,-1);

            System.out.println(rand);
            System.out.println(x);
            System.out.println(y);
            System.out.println(z);
            System.out.println(max);
    }
}
```

Actividad

1. Reescribe la clase **SimuladorDado** en la que utilizaste la clase Random para generar números aleatorios pero utilizando **Math.random()**
2. Escribe un programa que determine el **máximo de 3 números aleatorios enteros** en el rango de 1 a 100.
3. Piensa en un programa que determine si un número es un **número primo:**
   a. Sea x un número natural cualquiera. El enfoque más ingenuo es dividir x entre todos los números naturales menores que x
   b. Un mejor enfoque es dividir x por todos los números naturales menores que √x. (¿Por qué?)
   c. Escribe un programa que determine si un número es primo utilizando Math.sqrt()

Actividad

Decimos que un número n>1 es primo si sus únicos divisores son 1 y n.

Los números que no son primos se llaman número compuestos. El número 1 no se considera ni primo ni compuesto.

Si queremos saber si un número es primo, podemos probar de **dividir** entre 2,3,4…. hasta n-1, si **ninguno** de ellos lo divide, el número será **primo**.

El problema es que tenemos que probar tantos números como el número sea de grande, por lo que podemos **mejorar la estrategia**.

Actividad

Pero… ¿Es necesario mirar todos los números hasta el número que queremos saber si es primo?

Test de primalidad

Asimismo, existen otros tests de primalidad, uno de ellos establece que para saber si un número es primo basta con probar si el número no es divisible por los **primos** hasta su raíz cuadrada.

CASO PRÁCTICO

¿Es 149 primo?

La **raiz cuadrada de 149** es 12,2… por lo que solo es necesario **probar los números primos hasta 12,2..**

probamos con **2,3,5,7,11**

149 : 2 = 74,5.. (no es divisible por 2)
149 : 3 = 49,6.. (no es divisible por 3)
149 : 5 = 29,8.. (no es divisible por 5)
149 : 7 = 21,2.. (no es divisible por 7)
149 : 11 = 13,5.. (no es divisible por 11)

149 no puede ser dividido por estos primos por tanto **149 es un número primo**.

Test de primalidad

En nuestro caso lo vamos a simplificar el algoritmo y para determinar si un número es primo basta con probar con los números más pequeños que su raíz cuadrada.

Recursos

- Curso youtube : Java desde 0
- Libro Java 9. Manual imprescindible. F. Javier Moldes Teo. Editorial Anaya
- App SoloLearn: Aprende a Programar. Curso Java