Clases

PROGRAMACIÓN

Contenidos:

1) Clases
2) Tipos de modificadores.
3) Atributos.
4) Módulos.
5) Sobrecarga.
6) Operador de referencia.

Clases

- Una clase es un *tipo de dato referenciado*, definido por el usuario, que describe una "plantilla".
- Una clase está formada por una *cabecera y un cuerpo*.
- La cabecera de una clase, viene dada por: modificador de acceso, palabra reservada class y nombre de la clase.
  - El nombre de la clase empezará por letra capital, ejemplo: Vehículo, Persona,….
- El cuerpo de la clase, pueden contener un conjunto de **atributos** que definen su estado y **métodos** que definen su comportamiento.
- Recuerda que al conjunto de atributos y métodos de una clase se le denomina **miembros de la clase**.
- Las clases tienen que contener uno o varios **constructores** que permitan su inicialización.

Ejemplo definición de una clase:

```
[modificadorAcceso] class NombreClase //Cabecera de la clase
{ /*Inicio del cuerpo de la clase
        //Definición de atributos.
        //Definición de métodos.
Fin del cuerpo de la clase*/}
```

Clases

Ejemplo:

```
public class Punto
{
        //Definición de atributos.
        private int abscisa;      //Eje X
        private int ordenada; //Eje Y
        //Definición de métodos.
        public Punto()
        {
                abscisa = 0;
                ordenada = 0;
        }
        int getAbscisa(){return abscisa;}
        void setAbscisa(int x){abscisa = x;}
}
```

Al implementar una clase es necesario tener en cuenta:

- Por convenio, los nombres de las clases empiezan por una letra mayúscula. Si el nombre de la clase está formado por varias palabras, cada una también comienza con mayúscula. Ejemplos: Recta, PrismaCuadrangular, etc.
- El archivo en el que se encuentra la clase debe tener el mismo nombre que esa clase para poder usarla desde otras clases que están fuera del archivo.
- Tanto la definición como la implementación de una clase se incluye en el mismo archivo (archivo ".java"). En otros lenguajes como por ejemplo C++, definición e implementación podrían ir en archivos separados (archivos .h y .cpp en C++).

Clases: modificadores

Los modificadores de clase que podemos añadir en la *cabecera de la clase* son: **[public] [final | abstract]**

- **Public**: Indica que la clase es visible (se pueden crear objetos de esa clase) desde cualquier otra clase (otra parte del programa). Si no se especifica "public" la clase sólo se puede usar desde clases en el mismo paquete.
- **Abstract**: Indica que la clase es abstracta. Una clase abstracta no es instanciable, es decir, no se pueden crear objetos de esa clase, solo de otras que hereden de ella.
- **Final**: Indica que no podrás crear clases que hereden de ella.

Los modificadores final y abstract son **excluyentes**, es decir, solo se puede utilizar uno de ellos a la vez.

Los modificadores de clase que podemos añadir al de la *cuerpo de la clase* son:

- Modificadores de acceso
- Modificadores de contenido
- Otros tipos.

Existen modificadores solamente aplicables a los atributos y otros a los métodos, por lo tanto, vamos a verlos de forma individual.

Clases: cuerpo

En el cuerpo de una clase, podemos encontrar:

- **Atributos**: Especifican los datos que podrá contener una clase.
- **Métodos**: Implementan las acciones que podrán realizar los objetos.

Una clase no puede estar **vacía**, es decir, tiene que contener algún atributo y/o método.

Para el ejemplo anterior, clase Punto, tenemos:

- **Atributos:** Almacena dos enteros abscisa y ordenada.
- **Métodos:** Es recomendable tener un método Getter y Setter por cada uno de los atributos get**A**bscisa y set**A**bscisa

Cada objeto que instanciemos, almacenará información en esos atributos y, podrá, llamar a esos métodos, ejemplo :

```
Punto p1 = new Punto();
p1.setAbscisa(20);
```

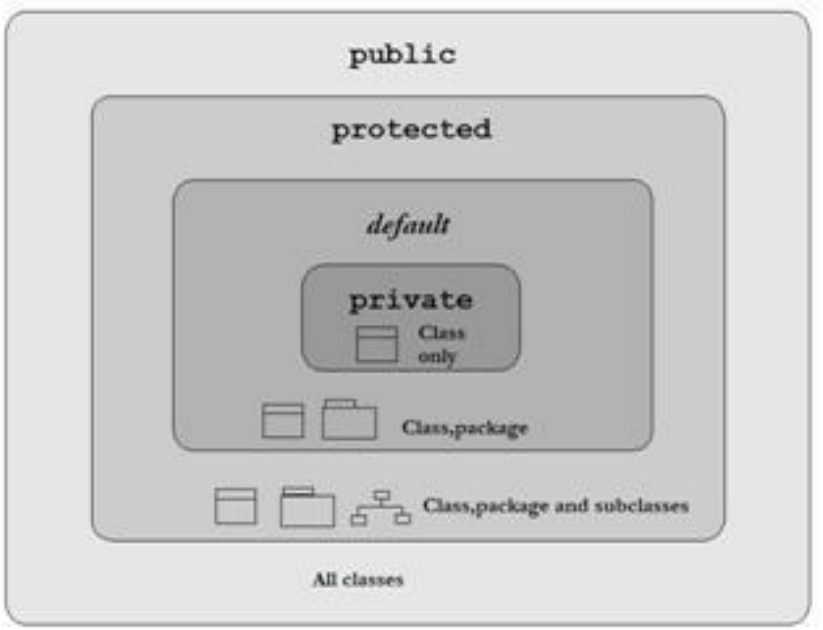
Clases: atributos del cuerpo.

- Son el conjunto de datos que los objetos de una determinada clase almacenan cuando son creados.
- Podemos definirlos como variables (**variables miembro o objeto**) cuyo ámbito de existencia es el objeto dentro del cual han sido creadas. Esto lógico, ya que sin la existencia del objeto, no son necesarias.
- Pueden ser de diversos **tipos**: primitivos, referenciados u objetos.
- Pueden contener distintos **modificadores de acceso:** default, public, private, protected.
  - **default (**por omisión o de paquete): Se utiliza cuando no se indica ningún modificador de acceso en la declaración del atributo. Este podrá ser accededido desde todas las clases que estén dentro del mismo paquete (package) que esta clase.
  - **public**: Indica que cualquier clase tiene acceso a ese atributo.
  - **private**: Indica que solo se puede acceder al atributo desde dentro de la propia clase y mediante métodos desde fuera de la clase. Es el modificador más utilizado en los atributos.
  - **protected**: Permitirá acceder al atributo desde cualquier subclase de la clase en la que se encuentre declarado el atributo y desde las clases del mismo paquete.

Clases: atributos del cuerpo.

Podemos representar de forma resumida el uso de los modificadores de acceso de la siguiente forma:

	Misma clase	Subclase	Mismo paquete	Otro paquete
Default				
Public				
Private				
protected				



Clases: atributos del cuerpo.

EJERCICIO:
Se pide desarollar una clase que represente un rectángulo en el plano. Para ello tendrá que contener:

1)Atributos x1, y1, que representan la coordenadas del vértice inferior izquierdo del rectángulo. Ambos de tipo double (números reales).
2)Atributos x2, y2, que representan las coordenadas del vértice superior derecho del rectángulo. También de tipo double (números reales).

Escribe una clase que contenga todos esos atributos teniendo en cuenta que queremos que sea una clase visible desde cualquier parte del programa y que sus atributos sean también accesibles desde cualquier parte del código.

Clases: atributos del cuerpo.

- Ademas de los modificadores de acceso, los atributos pueden contener **modificadores de contenido:**
  - **static:** Hace que el atributo sea común a los objetos de una misma clase. Toda la clase compartirá el atributo con igual valor.
  - **final:** Indica que el atributo es una costante, es decir, su valor no se verá modificado (tras su inicialización) a lo largo de la vida del objeto. Utilizaremos letras mayúsculas para nombrarlos.
- Los modificadores de contenido no son excluyentes entre ellos.

Ejemplo:

```
public class Punto
{
	private int abscisa;	//Eje X
	private int ordenada; //Eje Y
	public static int cantidadPuntos;
	public final double PI = 3.1416f;
	Punto()
	{cantidadPuntos++;}
}
```

Clases: atributos del cuerpo.

EJERCICIO:

Ampliar el ejercicio anterior del rectángulo incluyendo los siguientes atributos:

1) *numRectangulos, que almacena el número de objetos de tipo rectángulo creados hasta el momento.*
2) *nombre, que almacena el nombre de cada rectángulo.*
3) *nombreFigura, que almacena el nombre de la clase, "Rectángulo".*
4) *PI, que contiene el nombre de la constante PI con una precisión de cuatro cifras decimales.*
5) *Los* atributos nombre y numRectangulos no pueden ser visibles desde fuera de la clase.
6) Se desea que la clase sea accesible solamente desde su propio paquete.

Clases: atributos del cuerpo.

SOLUCIÓN:

```
class Rectangulo
{
	// Sin modificador “public” para que solo sea
	// accesible desde el paquete
	// Atributos de clase
	private static int numRectangulos; // Número de rectángulos creados
	public static final String nombreFigura= “Rectángulo”; // Nombre clase
	public static final double PI= 3.1416; // Constante PI
	// Atributos de objeto
	private String nombre; // Nombre del rectángulo
	public double x1, y1; // Vértice inferior izquierdo
	public double x2, y2; // Vértice superior derecho
}
```

Clases: atributos del cuerpo.

- Por ultimo, existen otro tipo de modificadores**:**
  - **transient:** Indica que ese atributo no es de tránsito, es decir, su valor no será serializado. Por serializar se entiende al proceso de traducir un conjunto de bytes, información en un fichero, a un objeto.
  - **volatile:** Indica que el valor de la variable puede ser modificado desde diferentes hilos.

Clases: módulos.

- Son las funciones y procedimientos ya vistas anteriormente.
- Por lo tanto, tendrán una cabecera y un cuerpo con el siguiente contenido:
  - Modificadores: de acceso, de contenido y otros.
  - Tipo devuelto por el método.
  - Nombre del método.
  - Los paréntesis, entre ellos irá la lista de parámetros o, simplemente, irá vacia.
    - ✓ Estos no pueden tener el mismo nombre, ambigüedad.
    - ✓ En caso que el nombre del atributo coincida con el del parámetro, utilizaremos el operador de autorreferencia **this**.
  - El cuerpo del método irá entre llaves: {/*Conjunto de instrucciones*/ }

Ejemplo:

```
public class Punto
{       private int abscisa; private int ordenada;
        //Definición de métodos.
        public Punto()
        {       abscisa = 0;ordenada = 0;    }
        public int getAbscisa(){return abscisa;}
        public void setAbscisa(int x){abscisa = x;}
}
```

Clases: módulos.

- Los **modificadores de acceso**, son los mismos y con el mismo significado que los ya vistos en los atributos.
- Sin embargo, los **modificadores de contenido**, no tienen el mismo significado:
  - **static** (ya visto): Se implementa de la misma forma para todos los objetos de la clase, es decir, pueden ser llamados sin necesidad de tener un objeto de la clase instanciado. Ejemplo: Math.abs(), Character.isDigit(),…
  - **final**: Método que no permite ser sobreescrito por las clases descencientes, relativo a la herencia.
  - **Native:** Es utilizado para señalar que un método ha sido implementado en código native (lenguaje compilado a máquina).
  - **abstract**: No require implementación, esta será realizada por sus clases descendientes. Está relacionado con las clases abstractas.
  - **Synchronized**: Relativo a la concurrencia (llamado multiples veces a la vez), el entorno de ejecución obligará a que, cuando un proceso esté ejecutando ese método, el resto de procesos que tengan que llamar a ese mismo método tengan que esperar a que el proceso termine.

Clases: módulos.

- Resumen de los modificadores:

	Clase	Atributo	Método
Default			
Public			
Private			
protected			
static			
final			
syncrhonized			
native			
abstract			

Clases: módulos del cuerpo.

EJERCICIO:

Seguimos ampliando la clase anterior, clase Rectangulo. Para ello, añadiremos los siguientes métodos públicos:

1)obtenerNombre y **establecerNombre**, que permiten el acceso y modificación del atributo nombre del rectángulo.

2)calcularSuperfice, que calcula el área encerrada por el rectángulo.

3)calcularPerímetro, que calcula el perímetro del rectángulo.

4)desplazar, que mueve la ubicación del rectángulo en el plano en una cantidad X (para el ejeX) y otra cantidad Y (para el eje Y). Se trata simplemente de sumar el desplazamiento X a las coordenadas x1 y x2, y el desplazamiento Y a las coordenadas y1 e y2. Los parámetros de entrada de este método serán por tanto X e Y, de tipo double.

5)obtenerNumRectangulos, que devuelve el número de rectángulos creados hasta el momento.

Realiza la implementación de cada uno de esos métodos en la clase Rectangulo.

Clases: módulos del cuerpo.

EJERCICIO SOLUCIÓN:

```
public String obtenerNombre () { return nombre;}
public void establecerNombre (String nom) {nombre= nom;}
public double calcularSuperficie ()
{
	double area, base, altura; // Variables locales
	// Cálculo de la base
	base= x2-x1;
	// Cálculo de la altura
	altura= y2-y1;
	// Cálculo del área
	area= base * altura;
	// Devolución del valor de retorno
	return area;
}
```

Clases: módulos del cuerpo.

EJERCICIO SOLUCIÓN:

```
public double calcularPerimetro ()
{
	double perimetro, base, altura; // Variables locales
	// Cálculo de la base
	base= x2-x1;
	// Cálculo de la altura
	altura= y2-y1;
	// Cálculo del perímetro
	perimetro= 2*base + 2*altura;
	// Devolución del valor de retorno
	return perimetro;
}
```

Clases: módulos del cuerpo.

EJERCICIO SOLUCIÓN:

```
public void desplazar (double X, double Y)
{
	// Desplazamiento en el eje X
	x1= x1 + X;
	x2= x2 + X;
	// Desplazamiento en el eje X
	y1= y1 + Y;
	y2= y2 + Y;
}

public static int obtenerNumRectangulos () { return numRectangulos;}
```

Clases: sobrecarga de métodos

Es posible tener varias versiones de un mismo método (varios métodos con el mismo nombre) gracias a la **sobrecarga de métodos**.
Java soporta esta característica, la cual permite declarar en una misma clase varias versiones de un mismo método con el mismo nombre.
El compilador será capaz de distinguir entre ellos en función de la lista de parámetros del método, es decir, si el método tiene una lista de parámetros diferente, será considerado como un método diferente (aunque tenga elmismo nombre) y el analizador léxico no producirá un error de compilación al encontrar dos nombres de método iguales en la misma clase.

Clases: sobrecarga de métodos

Ejemplo:

Queremos desarrollar una clase para escribir sobre un lienzo con diferentes tipografías según el tipo de información que se va a escribir. Es probable que necesitemos un método diferente según se vaya a pintar un número entero (int), un número real (double) o una cadena de caracteres(String). Podemos definir un método diferente dependiendo de lo que se vaya a escribir en el lienzo. Por ejemplo:

Método pintarEntero (int entero).
Método pintarReal (double real).
Método pintarCadena (double String).
Método pintarEnteroCadena (int entero, String cadena).
...

Clases: sobrecarga de métodos

Ejemplo:

Si utilizamos la sobrecarga, podemos un mismo nombre para todos esos métodos (todos ellos realizan la misma funcionalidad). Todos los métodos serán diferentes para el compilador ya que se diferencian entre ellos en las listas de parámetros (bien en el número de parámetros, bien en el tipo de los parámetros). Por lo que los métodos resultantes de los anteriores quedarían:
Método pintar (int entero)
Método pintar (double real)
Método pintar (double String)
Método pintar (int entero, String cadena)

Clases: sobrecarga de métodos

Tenemos que tener en cuenta que el tipo devuelto por el método no es considerado a la hora de identificar un método, así que un tipo devuelto diferente no es suficiente para distinguir un método de otro. Es decir, no se pueden definir dos métodos exactamente iguales en nombre y lista de parámetros e intentar distinguirlos indicando un tipo devuelto diferente. El compilador producirá un error de duplicidad en el nombre del método y no lo permitirá.
Es conveniente *no abusar de sobrecarga de métodos* y utilizarla con cierta moderación (cuando realmente puede beneficiar su uso), ya que dificulta la legibilidad del código.

Clases: sobrecarga operadores

Al igual que ocurre con la sobrecarga de métodos, es posible con los operadores.
Del mismo modo que hemos visto la posibilidad de sobrecargar métodos, podría plantearse también la opción de sobrecargar operadores del lenguaje tales como +, -, *, ( ), <, >, etc. para darles otro significado dependiendo del tipo de objetos con los que vaya a operar.
En algunos casos puede resultar útil para ayudar a mejorar la legibilidad del código, pues los operadores resultan muy intuitivos y pueden dar una idea rápida de cuál es su funcionamiento.
En algunos lenguajes de programación como, por ejemplo, C++ o C se permite la sobrecarga, pero no es soportado en todos los lenguajes. De hecho, Java no soporta la sobrecarga de operadores.

```
class Ejemplo
{
        int valor;
        Ejemplo& Ejemplo::operator + (const Ejemplo &e)
        {
            this->valor += e.valo;
            return *this;
        }
}
```

Clases: this

La palabra reservada ***this*** consiste en una referencia al objeto actual.
Entre sus usos, es realmente útil para evitar la ambigüedad entre el nombre de un parámetro de un método y el nombre de un atributo cuando ambos tienen el mismo identificador (mismo nombre).
En tales casos, el parámetro *"oculta"* al atributo y no tendríamos acceso directo a él (haz la prueba en tu IDE.
Dado que this es una referencia a la propia clase en la que te encuentras en ese momento, puedes acceder a sus atributos mediante el operador punto (.) como sucede con cualquier otra clase u objeto; ejemplo: this.nombreAtributo.

Suponiendo que tenemos una clase con un atributo int x, dado el siguiente ejemplo:

```
void establecerX (int x)
{
        this.x= x;
}
```

Clases: this.

EJERCICIO:

Modificar aquellos métodos de la clase Rectangulo en los que se pueda utilizer correctamente el operador this.

EJERCICIO SOLUCIÓN:

```
public void establecerNombre (String nombre)
{
this.nombre= nombre;
}
```