UD 3: EXTRAS

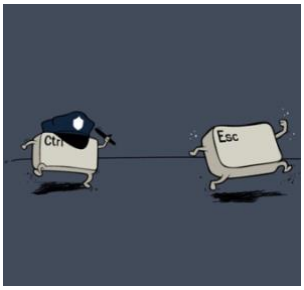
1. RECORDATORIO

Recordemos que la mayor parte del código que se escribe para una clase está contenido dentro de uno o varios métodos (todo excepto los atributos). Los métodos permiten dividir el trabajo que realiza el programa en diferentes tareas o "comportamientos".

Posteriormente, estos métodos son "llamados", bien desde la misma clase bien desde otras clases diferentes.

Veamos dos ejemplos sencillos:

a) Llamada a un método desde la misma clase




```
package ejemplollamadamismaclase;
public class Ascensor {
	private boolean puertaAbierta;
	private int pisoActual;
	private int PISOMAX;
	private int PISOMIN;
	public void subir (){
		System.out.println("Subiendo una planta");
		pisoActual ++;
		System.out.println("Piso: "+ pisoActual);
	}
	public void bajar (){
		System.out.println("Bajando una planta");
		pisoActual --;
		System.out.println("Piso: "+ pisoActual);
	}
	public void ir (int pisoElegido){
			while (pisoActual !=pisoElegido){
				if (pisoActual <pisoElegido){
					subir ();
				}else {
					bajar ();
				}
			}
	}
```

```
}
```

En este caso, el método ir ( ), llama a los métodos subir ( ) y bajar ( ) y los tres métodos están dentro de la misma clase Ascensor.

Para llamar a un método desde la misma clase, basta con incluir el nombre del método a llamar y sus argumentos, si los tiene.

A veces, pueden verse llamados con la palabra this antes, por ejemplo,

this.subir ( );

b) Llamada a un método desde una clase diferente

Primero la clase Camiseta que representa a los productos de este tipo

```
package ejemplos;

public class Camiseta {

	//Atributos

	private int ID;

	private String descripcion;

	private int codigoColor;

	public void mostrarInformación (){

		System.out.println("El ID del producto es: "+ID);

		System.out.println("Descripción de la camiseta: "+descripcion);

		System.out.println("Código de color: "+codigoColor);

	}

}
```

Ahora la clase diferente (en este caso la Principal. Podría ser otra no principal y en ese caso, se llamaría igual pero la llamada debería estar dentro de un método de dicha clase) desde la que llamamos al método mostrarInformación () mediante el operador punto (.)

```
package ejemplos;

public class TestCamiseta {

	public static void main(String[] args) {

		Camiseta miCamiseta;

		miCamiseta= new Camiseta();

		miCamiseta.mostrarInformación();

	}

}
```

Para llamar a un método desde otra clase, se usa el operador punto (.) con una variable de referencia del tipo de la clase a la que pertenezca el método, en nuestro caso, como el método mostrarInformación está en la clase Camiseta, un objeto de este tipo (miCamiseta) llama al método.

Las llamadas pueden estar en el main o dentro de otro método. Si están dentro de otro método necesitamos el objeto que va a llamar a dicho método (y debe ser del tipo de la clase donde está definido el método). En ese caso, se debe pasar el objeto como atributo o crearlo dentro del mismo método. Normalmente, se pasa como parámetro, por ejemplo, si estamos en una clase llamada Aula y Alumno a no es atributo de la misma, se le debe pasar al método que lo necesite:

```
Public void mostrarMedia (Alumno a){
        double notaAux=0.0;
        notaAux=a.calcularNotaMedia ( );//calcularNotaMedia debe estar en la clase Alumno
        System.out.printf ("Su nota es: %.2f ", notaAux);
}
```

c) En general

- No existen límites en cuanto al número de llamadas a métodos que puede realizar un método.
- El método desde el que se llama y el llamado, pueden estar en la misma clase o en clases diferentes.
- La forma de llamar al método que hace las operaciones o tareas, varía según se encuentre en la misma clase o en una clase distinta de la del método desde el que se llama.
- Las llamadas a métodos pueden hacerse en cualquier orden, su ejecución no necesariamente debe efectuarse en el orden en el que figuran dentro de la clase en la que se han declarado.

2. MÉTODOS Y VARIABLES STATIC

En determinadas ocasiones, puede que nos interese declarar un método o una variable como “static”. Algunas pistas para saber cuándo es adecuado serían, si:

- No es importante realizar la operación sobre un objeto independiente o asociar la variable a un tipo de objeto concreto.
- Es importante acceder a la variable o al método antes de instanciar un objeto.
- El método o la variable no pertenecen “**lógicamente**” a ningún objeto, pero posiblemente pertenece a una clase de utilidades como, por ejemplo, Math incluida en el API de java.

Las llamadas a variable o métodos static, se realizan poniendo el nombre de la clase, el operador punto y el nombre de la variable o método static, por ejemplo:

```
NombreClase.nombreMetodo ( );
NombreCalse.nombreVariable;
```

Además, debemos recordar que:

- Los métodos static no pueden referirse a this (o super que ya lo veremos en el tema siguiente) de ninguna manera.

- Un método **static** no puede acceder (llamar) a miembros (atributos o métodos) que no sean static.
- Un método **no static** puede acceder a miembros static y no static.

a) Ejemplo de método static

En el ejemplo anterior de las camisetas, podríamos añadir un método que convierte el tamaño de las camisetas en números a tallas expresadas en letras, como pequeña, mediana o grande. En este caso, el método sería **static** porque:

- No utiliza directamente ningún atributo de la clase Camiseta.
- Puede que sea conveniente llamar al método sin que se disponga de ningún objeto camiseta, porque se use para otros objetos de otro tipo o porque se configuren las tallas sin tener los productos.

```
//El ejemplo usa números para no tener que declarar tantas variables y podría hacerse con un switch
public static String convertirTamanyo (int tallaNumerica){
	int small=10,medium=14, large=18;
	if (tallaNumerica<small){
		return "S";
	}
	else if (tallaNumerica<medium){
		return "M";
	}
	else if (tallaNumerica<large){
		return "L";
	}
	else {
		return "XL";
	}
}
```

La llamada al método, desde el main, sería, por ejemplo:

String tamanyo;

tamanyo=Camiseta.*convertirTamanyo* (16);

b) Ejemplo de variable static

En este caso, que una variable sea static, significa que solo puede haber en la memoria una copia de la variable asociada a una clase, no una copia por cada objeto de ese tipo.

```
static double impuesto=8.25;
```

Para acceder a ella se usaría:

```
Camiseta.impuesto=6.25;
```

Un ejemplo concreto:


```
package ejemploestaticos;
public class Cohete {
	private static int numCohetes;
	public Cohete (){
		numCohetes++;
	}
	public int getCohetes (){
		return numCohetes;
	}
}
```

```
package ejemploestaticos;
public class TestCohetes {

	public static void main(String[] args) {
		Cohete c1 = new Cohete ();
		Cohete c2 = new Cohete ();
		Cohete c3 = new Cohete ();

		System.out.println(c1.getCohetes());
		System.out.println(c3.getCohetes());
	}
}
```

¿Qué se mostrará por pantalla? **Pues 3 en cada llamada**. La variable numCohetes se inicializa a cero la primera vez, cuando se crea el objeto c1. Cuando se crean los objetos c2 y c3, no se vuelve a inicializar pues ya existe y es estática, solo se incrementa en uno. Las variables static existen incluso cuando no se han instanciado todavía objetos de la clase.

3. PASO DE PARÁMETROS POR VALOR Y POR REFERENCIA

Qué significa "por valor" y "por referencia":

- **Por valor**: cuando los argumentos son pasados por valor a los métodos, significa que se realiza una copia de la variable y esta es enviada al método y no la original, entonces todos los cambios realizados dentro del método solo afectan a la copia actual.
- **Por referencia**: cuando los argumentos son pasados por referencia, significa que la referencia o el puntero a la variable original son pasadas a los métodos y no el dato original.

En Java, los argumentos son siempre pasados por valor independientemente del tipo de variable. Cada vez que el método es invocado, pasa lo siguiente:

- Una copia del argumento es pasada en la pila de memoria y la copia es pasada en el método:
  - Si la variable original es primitiva, es simple, una copia de la variable es creada en la pila de memoria y esta se pasa al método.
  - Si la variable original no es primitiva, entonces una nueva referencia es creada dentro de la pila de memoria que apunta al objeto actual y la nueva referencia es pasada al método, (en esta etapa, 2 referencias apuntan al mismo objeto).

a) Pasando un tipo básico

Ocurre cuando los argumentos son pasados por valor a los métodos. Significa que se realiza una copia de la variable y esta es enviada al método (y no la original) entonces todos los cambios realizados dentro del método solo afectan a la copia actual.

```
public class Ppal {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        int x = 1;
        int y = 2;
```

```
            System.out.print("Valores de x e y antes de la modificación: ");
            System.out.println(" x = " + x + "; y = " + y);
            modifyPrimitiveTypes(x, y);
            System.out.print("Valores de x e y después de la modificación: ");
            System.out.println(" x = " + x + "; y = " + y);
        }

        private static void modifyPrimitiveTypes(int x, int y) {
            x = 5;
            y = 10;
        }

}
```

La salida es:

Valores de x e y antes de la modificación: x = 1; y = 2

Valores de x e y después de la modificación: x = 1; y = 2

b) Pasando una referencia a un objeto ("un objeto")

Significa que lo que se pasa es la dirección de memoria y no es una copia sino la original.

Supongamos la clase Fecha (no se trabaja así con las fechas en Java, es solo un ejemplo académico):

```
package ejemplo;
public class Fecha {
	private int dia; // 1-31 con base en el mes
	private int mes; // 1-12
	private int anio; // cualquier año
	public Fecha(int dia, int mes, int anio) {
		this.mes = mes;
		this.dia = dia;
		this.anio = anio;
	}
	public Fecha(int dia, int mes) {
		this.dia = dia;
		this.mes = mes;
```

```
	}
	public int getMes() {
		return mes;
	}
	public void setMes(int mes) {
		this.mes = mes;
	}
	public int getDia() {
		return dia;
	}
	public void setDia(int dia) {
		this.dia = dia;
	}
	public int getAnio() {
		return anio;
	}
	public void setAnio(int anio) {
		this.anio = anio;
	}
	@Override
	public String toString() {
		return "día: " + dia + " del " + mes + " de " + anio;
	}
}
```

Ahora la clase Empleado, con dos atributos Fecha:

```
package ejemplo;
public class Empleado {

	private String primerNombre;
	private String apellidoPaterno;
	private Fecha fechaNacimiento;
	private Fecha fechaContratacion;
```

```
	public Empleado(String primerNombre, String apellidoPaterno, Fecha fechaNacimiento, Fecha fechaContratacion) {
		this.primerNombre = primerNombre;
		this.apellidoPaterno = apellidoPaterno;
		this.fechaNacimiento = fechaNacimiento;
		this.fechaContratacion = fechaContratacion;
	}
	public String getPrimerNombre() {
		return primerNombre;
	}
	public void setPrimerNombre(String primerNombre) {
		this.primerNombre = primerNombre;
	}
	public String getApellidoPaterno() {
		return apellidoPaterno;
	}
	public void setApellidoPaterno(String apellidoPaterno) {
		this.apellidoPaterno = apellidoPaterno;
	}
	public Fecha getFechaNacimiento() {
		return fechaNacimiento;
	}
	public void setFechaNacimiento(Fecha fechaNacimiento) {
		this.fechaNacimiento = fechaNacimiento;
	}
	public Fecha getFechaContratacion() {
		return fechaContratacion;
	}
	public void setFechaContratacion(Fecha fechaContratacion) {
		this.fechaContratacion = fechaContratacion;
	}

	@Override
	public String toString() {
		return "Empleado \nNombre: " + primerNombre + "\nApellido Paterno: " + apellidoPaterno + "\nFecha Nacimiento: "
```

```
                        + fechaNacimiento + "\nFecha Contratación: " +
fechaContratacion;
        }

//Se le pasa como parámetro un objeto de la clase fecha y comprueba si el día y
mes pasados en la fecha como parámetro, coinciden con los del objeto que llama,
si es así, felicita
        public void felicitarCumple(Fecha cump){

            if (cump.getDia()==fechaNacimiento.getDia()&&cump.getMes()==fechaNacimiento.getMes()){
               System.out.println("Felicidades campeón!!!");
            }
        }

        public Fecha rejuvenecer (Fecha cum) {

               int num=5;
               cum.setAnio(cum.getAnio()-num);

               return cum;
        }

}
```

Podemos probar en la clase principal:

```
package ejemplo;
public class PruebaEmpleado {
      public static void main(String[] args) {
            int diaAct, mesAct;
            Fecha nacimiento = new Fecha(8, 07, 1976 );
            Fecha contratacion = new Fecha(17, 10, 2004 );
            Empleado empleado = new Empleado("Ángel", "Naranjo", nacimiento,
                  contratacion);
            System.out.println(empleado);
            System.out.println("Qué día es hoy");
            diaAct=Leer.datoInt();
            System.out.println("¿Qué mes?");
            mesAct=Leer.datoInt();
```

```
            Fecha actual= new Fecha (diaAct, mesAct);
            empleado.felicitarCumple(actual);

            System.out.println(empleado);
            System.out.println("*****************");
            empleado.rejuvenecer(nacimiento);
            System.out.println(empleado);

            //Compara con el paso por copia
            /*
            int a=2;
            Numero miNumero= new Numero ();

            System.out.println(miNumero.modificar(a));//Cambia la copia, vale 6

            miNumero.mostrar(a);*///El a del main no ha cambiado, vale 2

        }
}
```

4. Paso de Arrays como argumentos y devolución de Arrays

Veamos esto con un ejemplo sencillo:

```
public class Matriz {

    public void multiplicarPorDos (int tabla [][]){
        for (int i=0; i<tabla.length;i++){
            for (int j=0; j<tabla[i].length;j++){
                tabla [i][j]=tabla[i][j]*2;
            }
        }
    }
    public void mostrar (int [][] miArray){
        for (int i=0; i<miArray.length;i++){
            for (int j=0; j<miArray[i].length;j++){
                System.out.print(" "+miArray[i][j]);
```

```
                }
                System.out.println();
            }
        }
        //Rellena el array de enteros aleatorios

        public int [][] cargarArray(int desde, int hasta){

            int [][] miArray2D= new int [2][3];
            Random r= new Random (System.nanoTime());

            for (int i=0; i<miArray2D.length;i++){
                for (int j=0; j<miArray2D[i].length;j++){
                    miArray2D [i][j]=r.nextInt(hasta-desde+1)+desde;
                }
            }
            return miArray2D;
        }
}
```

Y las llamadas en la clase principal, quedarían:

```
public class PruebaMatriz {
        public static void main(String[] args) {

            int desde, hasta;
            //Objeto de la clase Matriz que usaremos para llamar a métodos
            Matriz paraLlamar= new Matriz ();

            //Primer array de prueba
            int [][]tabla={{1,2,3},{4, 3, 2}};

            //Este se cargará leyendo por teclado
            int [][]tabla2= new int [2][3];

            //Pasando un array como parámetro
            paraLlamar.multiplicarPorDos(tabla);
            paraLlamar.mostrar(tabla);
```

```
                System.out.println("Diga menor número con el que quiere rellenar elarray");
                desde=Leer.datoInt();
                System.out.println("Diga el mayor");
                hasta=Leer.datoInt();

                //Devolviendo un array
                tabla2=paraLlamar.cargarArray(desde,hasta);
                paraLlamar.mostrar(tabla2);

        }
}
```

5. Arrays de objetos

Lo veremos directamente con un ejemplo. Supongamos la clase Vehículo siguiente.

```
package ejemploarrayobjetos;
public class Vehiculo{
        //Atributos
        private String modelo;
        private double potencia;
        //Constructores
        public Vehiculo(){
        }
        public Vehiculo(String modelo, double potencia){
                this.modelo =modelo;
                this.potencia=potencia;
        }
        public String getModelo() {
                return modelo;
        }
        public void setModelo(String modelo) {
                this.modelo = modelo;
        }
```

```
	public double getPotencia() {
		return potencia;
	}
	public void setPotencia(double potencia) {
		this.potencia = potencia;
	}
	@Override
	public String toString() {
		return "Vehiculo [modelo=" + modelo + ", potencia=" + potencia + "]";
	}
}
```

Ahora la clase principal:

```
package ejemploarrayobjetos;

public class PruebaVehiculo {


	public static void main(String args[]){
		int nVs=0, seguir, tam=100;//Contador de objetos Vehículo, variable para salir del bucle y tamaño
		String modelo;
		boolean cRuedas;
		//Creamos un array de objetos Vehiculo para almacenar 100 Vehículos
		Vehiculo lista[] = new Vehiculo[tam];
		String aux;
		do {
			System.out.print("Introduce el modelo del vehículo: ");
			modelo = Leer.dato();
			System.out.print("Introduce la potencia: ");
			double potencia=Leer.datoDouble();
			lista[nVs]=new Vehiculo (modelo, potencia);
			nVs++;//Aumentamos en uno el número de vehículos
			System.out.println("Si desea terminar pulse 0, cualquier número para seguir");

			seguir=Leer.datoInt();

		} while(seguir !=0 && nVs<lista.length);

//Imprimimos los vehículos introducidos, ojo no hasta el tamaño del array, sino hasta donde hay vehículos con datos
		for (int i=0; i<nVs; i++){
```

```
                System.out.println (lista[i]);
            }
            System.out.println("Fin del programa");
        }
}
```

Los arrays de objetos, igual que los arrays de tipos básicos, se pueden pasar como parámetros y devolver en llamadas a métodos. Dejo para vosotros, cómo se trabaja con ellos, aunque básicamente será igual que hemos hecho en otras ocasiones, accediendo al objeto concreto del array y usando sus métodos getters o setters.

Por ejemplo, en el último syso, lista[op-1] es un objeto de tipo Vehículo, por lo que puede usar los métodos de esa clase, por ejemplo, un getModelo ( ).

```
System.out.println("¿El modelo de qué vehículo quiere ver?");
op=Leer.datoInt();
System.out.println(lista[op-1].getModelo());
```