Tema 5: Introducción a la programación orientada a objetos

Explicaciones del tema 5

Lección 1

1. Crear una clase *Main*, que sólo contendrá el método *main( )*.
2. Crear una clase *Persona*.
    1. Atributos: nombre, edad, dni.
    2. Ningún método, de momento.
3. Declarar una variable de tipo *Persona*.
4. Inicializarla con *new*.
5. Crear un objeto de la clase *Persona*.
6. Rellenar los atributos del objeto creado.
7. Mostrarlos con unos *print*.
8. Crear otro objeto de la clase *Persona*. Esta vez hacer la declaración y la inicialización en la misma línea.
9. Repetir el proceso de rellenar atributos y mostrarlos en pantalla.
10. Crear el método *mostrar( )* en la clase *Persona*.
11. Usarlo en la clase *Main* para mostrar los datos de cada clase.

Lección 2

1. Crear una clase *Main*, que sólo contendrá el método *main( )*.
2. Crear una clase *Punto*.
    1. Atributos: *x*, *y*, *color*.
    2. Ningún método, de momento.
3. Repetir el proceso del ejemplo anterior, creando un par de objetos y mostrando luego sus valores.
4. Añadir el método *mostrar( )*. Sustituir en la clase *Main*.
5. Añadir el método *desplazar(dx, dy)*.

Lección 3

1. Crear una clase *Main*, que sólo contendrá el método *main( )*.
2. Crear una clase *Circulo*.
    1. Atributos: *x*, *y*, *radio*.
    2. Ningún método, de momento.
3. Repetir el proceso del ejemplo anterior, creando un par de objetos y mostrando luego sus valores.
4. Añadir el método *mostrar( )*. Sustituir en la clase *Main*.
5. Añadir el método *area( )*.
6. Añadir el método *perimetro( )*.

Lección 4

1. Crear una clase *Main*, que sólo contendrá el método *main( )*.
2. Crear una clase *Rectángulo*.
    1. Atributos: *x*, *y*, *ancho*, *alto*.
    2. Ningún método, de momento.
3. Repetir el proceso del ejemplo anterior, creando un par de objetos y mostrando luego sus valores.
4. Añadir el método *mostrar( )*. Sustituir en la clase *Main*.
5. Añadir el método *area( )*.
6. Añadir el método *perimetro( )*.
7. Añadir el método *mover(dx, dy)*, que moverá el rectángulo *dx* píxeles horizontalmente y *dy* píxeles verticalmente.
8. Añadir el método *boolean estaDentro (x, y)*, que devuelve *true* si el punto proporcionado está dentro del rectángulo.

Lección 5 (Inicialización automática)

1. Los atributos de una clase se inicializan automáticamente a valores como 0, false, null, etc., si no les damos valores explícitamente.
2. Comprobar mediante la clase *Persona* los valores por defecto que toman los atributos.

Lección 6 (Constructores)

1. Explicar el concepto de constructor y constructor implícito.
2. Comentar que el constructor implícito desaparece cuando creamos un constructor explícitamente.

3. Añadir un constructor a la clase *Persona*, que tome como parámetros el nombre, la edad y el dni. Los parámetros tendrán nombres distintos a los altributos de la clase.
4. Hacer lo mismo del apartado anterior usando la palabra clave *this*. Los parámetros tendrán entonces los mismos nombres que los atributos.
5. Añadir un constructor sin parámetros.
6. Añadir un constructor a la clase *Punto* que tome como parámetros todos los atributos.
7. Añadir un constructor a la clase *Rectangulo* que tome como parámetros todos los atributos.
8. Añadir un constructor a la clase *Circulo* que tome como parámetros todos los atributos.

Lección 7 (Constructores llamando a otros constructores)

1. Añadir un constructor a la clase *Persona*, que tome como parámetros el nombre y la edad, y establezca el dni como "DESCONOCIDO".
2. Añadir un constructor a la clase *Rectangulo* que tome como parámetros las coordenadas y establezca el ancho y el alto en 100.
3. Añadir un constructor a la clase *Circulo* que tome como parámetros las coordenadas del centro y establezca el radio como 1.

Lección 8 (ejercicio)

1. Ejercicio: Crear una clase CuentaBancaria:
    1. Atributos: balance;
    2. Métodos:
        1. Constructor (int balance)
        2. void depositar(int cantidad)
        3. boolean retirar(int cantidad)
            1. Sólo si la cantidad es menor que el balance.
            2. Devuelve true si se ha podido retirar y false si no.
        4. int obtenerBalance( )

```
public class CuentaBancaria {
	private int balance;

	CuentaBancaria(int balance) {
		this.balance = balance;
	}

	public void depositar(int cantidad) {
		balance += cantidad;
	}
```

```
	public boolean retirar(int cantidad) {
		if (balance >= cantidad) {
			balance -= cantidad;
			return true;
		}
		return false;
	}

	public int obtenerBalance() {
		return balance;
	}
}
```

Lección 9 (métodos de acceso, *getters*, *setters*)

1. Concepto de variables privadas, públicas, protegidas y por defecto.

Modificadores de acceso

- **Public**: Da acceso a cualquiera.
- **Private**: Da acceso sólo a los métodos de la propia clase.
- **Protected**: Da acceso a las clases de su propio paquete y a sus derivadas.
- **Ninguno**: Da acceso a las clases del propio paquete.

2. Colocar todos los atributos como *private* y los métodos como *public*.
3. Añadir a la clase *Persona* los métodos de acceso necesarios (cambiar los atributos public a private).
4. Añadir a la clase *Punto* los métodos de acceso necesarios (cambiar los atributos public a private).
5. Añadir a la clase *Circulo* los métodos de acceso necesarios (cambiar los atributos public a private).
6. Añadir a la clase *Rectangulo* los métodos de acceso necesarios (cambiar los atributos public a private).

Lección 10 (atributos estáticos de una clase)

1. Concepto de atributo **estático** en una clase (palabra clave ***static***): Atributo cuyo valor será el mismo en cualquier instancia de la clase. Entonces se puede (y es recomendable) acceder a este atributo usando el nombre de la clase en lugar del nombre de un objeto.
2. Ejemplo: En un concesionario de coches se hace siempre el mismo descuento para todos los vehículos. Entonces la variable *descuento* de la clase *Coche* debería ser estática. (Después desarrollaremos una nueva clase para este ejemplo).

3. Ejemplo: La variable Pi de la clase Math es estática.
4. **Observación**: Si una variable estática va a ser constante, lo lógico es declararla pública.
5. Añadir a la clase *Persona* un contador estático para indicar cuántos objetos *Persona* se van creando.
6. Añadir a la clase *Punto* un contador estático para indicar cuántos objetos *Punto* se van creando.
7. Añadir a la clase *Circulo* un contador estático para indicar cuántos objetos *Circulo* se van creando.
8. Añadir a la clase *Rectangulo* un contador estático para indicar cuántos objetos *Rectangulo* se van creando.

Lección 11 (ejercicio)

1. Desarrollar una clase Coche (y una clase Main para utilizarla).
    1. Atributos: marca, modelo, color, matricula, precio y descuento.
    2. Métodos: arrancar, detenerse, acelerar, frenar, verPrecio (devuelve un valor a partir del precio y el descuento).
2. Desde la clase Main crear un par de objetos de la clase Coche. Asignar un valor a la variable *descuento* de uno de ellos y comprobar que ese valor se ha cambiado en todos los demás.
3. Comprobar que se puede acceder a la variable descuento usando el nombre de la clase en lugar del nombre de una instancia: *Coche.descuento*.

Lección 12 (métodos estáticos de una clase)

1. Un método estático (**static**) es aquel que es referenciado siempre a partir del nombre de la clase en la que está definido, y no a partir de los objetos particulares.
2. Ejemplo: El método *pow* de la clase *Math* es estático.
3. **Observación**: En el código de un método estático sólo se puede acceder a los atributos y métodos estáticos de su clase. Tampoco se puede usar la palabra clave *this*, ya que no se ejecuta sobre un objeto concreto.

4. Añadir a la clase *Persona* un método estático para indicar cuántos objetos *Persona* se van creando.
5. Añadir a la clase *Punto* un método estático para indicar cuántos objetos *Punto* se van creando.
6. Añadir a la clase *Circulo* un método estático para indicar cuántos objetos *Circulo* se van creando.
7. Añadir a la clase *Rectangulo* un método estático para indicar cuántos objetos *Rectangulo* se van creando.

8. Añadir a la clase *Coche* un método estático para indicar el valor actual del descuento de los coches.

Lección 13 (sobrecarga de métodos)

1. Ya trabajado al crear diferentes constructores en la misma clase.
2. Podemos crear tantos métodos con el mismo nombre siempre que tengan distintas firmas. La firma de un método es su lista de argumentos.
3. Agregar a la clase Punto un segundo método *desplazar(int dx)* que desplace horizontalmente el punto. ¿Podría añadirse además otro método *desplazar(int dy)*?
4. Agregar a la clase CuentaBancaria un segundo método *depositar(int cantidad, int comision)*. Éste método depositará en la cuenta la cantidad quitándole un % de comisión.
5. Agregar a la clase CuentaBancaria un segundo método *retirar(int cantidad, int comision)*. Éste método retirará de la cuenta la cantidad más un % de comisión. Devolverá *true* o *false* también en función de si ha sido posible hacer la operación o no.

Lección 14 (paquetes)

1. Los paquetes permiten agrupar clases. En el disco duro, un paquete se corresponde con una carpeta en cuyo interior se guardan las clases que forman parte del mismo.
2. Si no se indica, se entiende que la clase pertenece al paquete por defecto.
3. La sentencia *package* debe ser la primera del archivo fuente.
4. Los paquetes pueden incluir puntos y se corresponden con carpetas y subcarpetas dentro del proyecto. Se recomienda que estén en minúsculas.
5. Una convención utilizada por los programadores es nombrar los paquetes que crean empezando con un dominio de su propiedad escrito al revés, para separar sus nombres de clase del resto de programadores. Por ejemplo: *net.infocarlos.graficos*.

6. Para usar una clase de un paquete podemos:
   - Escribir *nombrePaquete.nombreClase* cada vez que queramos hacer referencia a la misma.
   - Escribir al inicio del programa: import *nombrePaquete.nombreClase*;
   - Si queremos poder acceder a todas las clases del paquete: import *nombrePaquete.**.

7. Ejercicio: Para cada uno de los proyectos crea un paquete llamado *inicio* que contendrá la clase Main, y otro llamado *clases* donde colocarás la otra clase.

***** A partir de aquí entra en la tercera evaluación *****

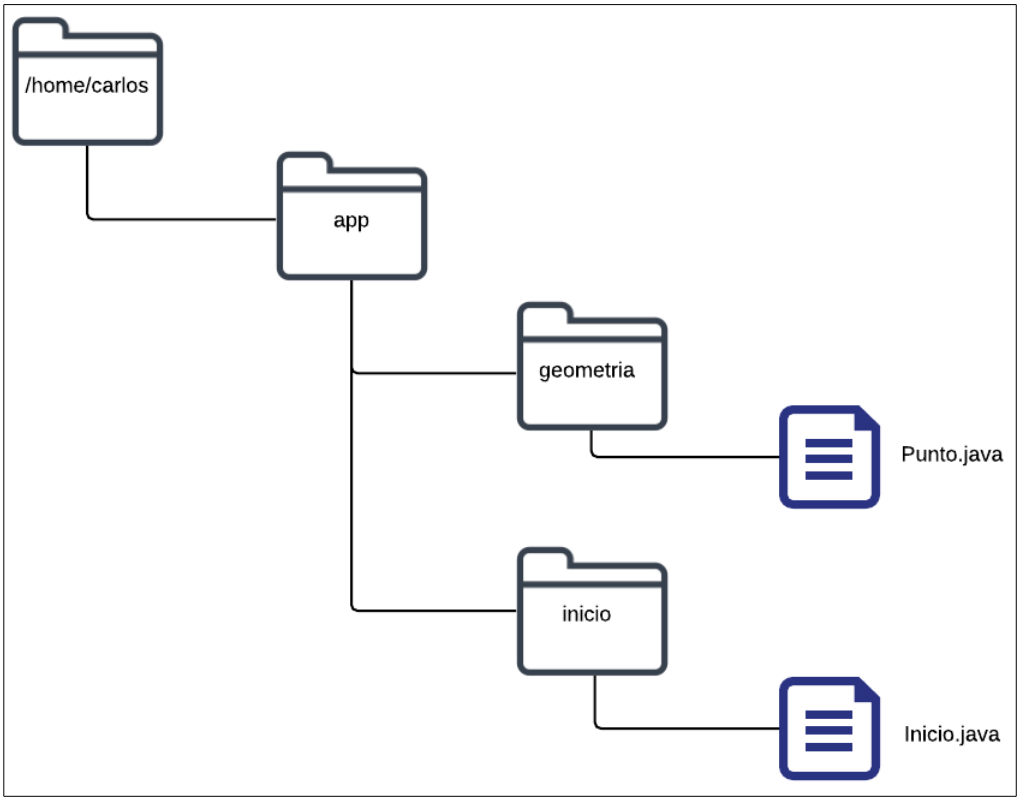
Lección 15 (archivos jar)

Un archivo jar no es más que un archivo comprimido con el algoritmo de compresión ZIP que puede contener:

- Los archivos *.class que se generan a partir de compilar los archivos *.java que componen nuestra aplicación.
- Los archivos de recursos que necesita nuestra aplicación (por ejemplo los archivo de sonido *.wav).
- Opcionalmente se puede incluir los archivos de código fuente *.java
- Opcionalmente puede existir un archivo de configuración "META-INF/MANIFEST.MF".

Lección 16 (crear un paquete jar)

Tenemos una aplicación con dos clases repartidas en dos paquetes:



Para las próximas prácticas supondremos que estamos dentro de la carpeta *app*. Los pasos para empaquetar una aplicación en un archivo jar no ejecutable son:

- Compilar el código fuente a bytecode:
  - ~/app$ javac geometria/*.java
  - ~/app$ javac inicio/*.java
- Ejecutar la aplicación para comprobar que funciona. Indicamos la clase con el método *main*:
  - ~/app$ java inicio.Inicio
- Crear archivo jar. Indicamos las rutas relativas de las carpetas que contienen los archivos *.class*:
  - ~/app$ jar -cf miapp.jar inicio geometria
    - Las siglas significan:
      - c: Crear
      - f: A continuación indicaremos el nombre del archivo (file) jar que se creará.

Podemos ejecutar el programa recién empaquetado con la instrucción:

  - ~/app$ java -cp miapp.jar inicio.Inicio
    - Las siglas significan:
      - cp: Classpath (ruta donde se encuentran las clases que componen el programa).

Lección 17 (crear un paquete jar ejecutable)

Una vez tenemos los archivos *.class* podemos crear un archivo jar ejecutable:

- Creamos un archivo de manifiesto:
  - ~/app$ nano manifiesto
- Escribimos en este archivo una línea con el nombre de la clase principal del programa (la que tiene el método main):
  - Main-class: inicio.Inicio
- Creamos el paquete indicando el nombre del archivo de manifiesto, el nombre del paquete y las carpetas con los archivos *.class* de la aplicación:
  - ~/app$ jar -cmf manifiesto miapp.jar geometria inicio

Podemos ejecutar la aplicación ahora con la instrucción:

- ~/app$ java -jar miapp.jar

Si se tratara de una aplicación con interfaz gráfica podríamos ejecutarla simplemente haciendo doble clic sobre el archivo *miapp.jar*.

Lección 18 (archivos jar desde Eclipse)

1. Seleccionamos el proyecto a exportar.
2. En el menú contextual seleccionamos E*xport*.
3. Seleccionamos *Java* > *JAR file* o *Java* > *Runnable JAR file*.

Lección 19 (ciclo de vida de los objetos)

1. Los objetos son borrados de la memoria del sistema por el recolector de basura (garbage collector). El borrado ocurre cuando un objeto no tiene ninguna referencia en el programa.

2. Las referencias a objetos se pierden en los casos siguientes:
   a) Cuando la variable que contiene la referencia deja de existir porque el flujo de ejecución del programa abandona definitivamente el ámbito en que había sido creada.
   b) Cuando la variable que contiene la referencia pasa a contener la referencia de otro objeto o pasa a valer null.

3. No se puede saber cuándo actuará el recolector de basura. Se puede pedir que pase ahora (aunque no hay garantías) con la orden *System.gc( )*.

4. Al destruir un objeto se permite que ejecute unas últimas voluntades, que van recogidas en el método *finalize( )*. Esto puede ser necesario en diversas situaciones:
   a) Cuando haya que liberar recursos del sistema gestionados por el objeto que está a punto de desaparecer (archivos abiertos, conexiones a bases de datos, ...).
   b) Cuando haya que liberar referencias a otros objetos para hacerlos candidatos a ser tratados por el recuperador de memoria.

5. De nuevo, el momento en que se ejecute el método *finalize( )* no puede saberse de antemano, ya que depende de la actividad del recolector de basura.

6. Puede comprobarse la acción del recolector de basura creando un método *finalize( )* que muestre un texto. A continuación, creamos un montón de objetos de ese tipo y los anulamos (=null). Si creamos y anulamos suficientes objetos, empezarán a salir los mensajes de *finalize( )* en la consola.