CREACIÓN DE OBJETOS

CLASE Y OBJETO

Lee(sólo leer, no estudiar) el párrafo de "Qué é unha clase" de
http://manuais.iessanclemente.net/index.php/Conceptos_de_programaci%C3%B3n_orientada_a_obxectos

y como se define una clase

http://manuais.iessanclemente.net/index.php/Clases_e_obxectos#Declaraci.C3.B3n_de_clases

CREAR Y USAR OBJETOS

3 pasos:

1. Definir la clase con la que construir el objeto.
2. Crear un objeto de esa clase con el operador new()
3. Usar el objeto accediendo a sus datos y métodos con el operador " ."

Crear una clase

Para crear un objeto previamente tengo que definir su clase. Las clases se componen de atributos y/o métodos. Comenzamos con un ejemplo de clase muy sencilla que sólo contiene atributos.
Los atributos también se llaman:

- propiedades
- campos(fields)
- variables miembro

Ejemplo:

```
class Coche{
   String modelo;// por ejemplo "Renault Megane"
   int pasajeros; //número de pasajeros
   int deposito; //capacidad del depósitos en litros
   int kpl; //kilómetros que se pueden recorrer con un litro,
}
```

Crear objetos con el operador new y acceder a un objeto con el operador .

Ahora puedes crear objetos de esa clase con el operador new. ¿Dónde utilizo new?. En nuestros primeros ejemplos lo haremos desde el método main() de Unidad2. Por el momento piensa que la clase Unidad2 es una clase especial, que tiene un método especial (main) desde la que podemos manejar la clase Coche. Para entender la relación clase-objeto focaliza ahora tu pensamiento en la clase Coche.

Ejemplo : La clase Unidad2 , desde su método main, crea un objeto de la clase Coche.

```
class Coche{
   String modelo;
   int pasajeros;
   int deposito;
   int kpl;
}
class Unidad2{
   public static void main(String[] args) {
      Coche golf = new Coche();//se crea el objeto
      golf.pasajeros=5;//se usa el objeto
      System.out.println("pasajeros:" + golf.pasajeros); //se usa el objeto
   }
}
```

Por lo tanto, un objeto se crea con el operador new. El código anterior crea un objeto golf. **En realidad, golf no es realmente el objeto, sino una referencia a un objeto, pero por el momento damos por bueno decir que golf es un objeto**. Observa que para acceder a los atributos del objeto se utiliza el operador punto '.'

Estado de un objeto

Los valores de los atributos de un objeto definen en qué **estado** se encuentra el objeto. Los atributos de un objeto pueden ir cambiando a lo largo del tiempo de ejecución de un programa, por tanto, el objeto irá cambiando de **estado**.

Por lo tanto, el estado de un objeto se define como los valores de los atributos en un instante concreto.

Ejercicio U2_B1_E1 : A este código

```
class Coche{
    String modelo;
    int pasajeros;
    int deposito;
    int kpl;
}
class Unidad2{
    public static void main(String[] args) {
        Coche golf = new Coche();//se crea el objeto
        golf.pasajeros=5;//se usa el objeto
        System.out.println("pasajeros:" + golf.pasajeros); //se usa el objeto

    }
}
```

Añade instrucciones para que cree también un objeto megane para el que almacenamos que tiene un depósito 60 litros e imprimimos este atributo por pantalla.

Observa que un programa java es un conjunto de clases, como mínimo una. Una y sólo una clase debe contener el método main() que es el punto de entrada al programa y por tanto sólo puede haber un método main en todo el programa.

Ejercicio U2_B1_E2.

Crea un objeto peugeot308 con las siguientes características:

- Modelo "peugeot 308 sport"
- 5 pasajeros
- 60 litros de deposito
- 20 kpl

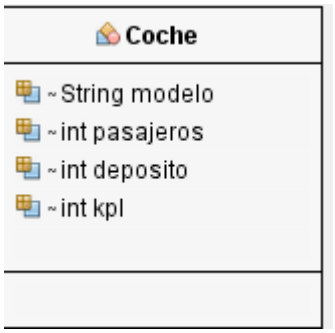
El programa da los datos anteriores a los atributos correspondientes, calcula la autonomía del coche (kpl*deposito) y finalmente imprime los atributos del coche y su autonomía.

LA CLASE PRINCIPAL

Un programa normalmente necesita escribir varias clases. La ejecución del programa va a comenzar por el método main(). A la clase que contiene el método main() le llamamos "Clase principal", no porque sea más importante, sino porque la ejecución del programa comienza en ella. En el ejemplo anterior Unidad2 sería la clase principal.

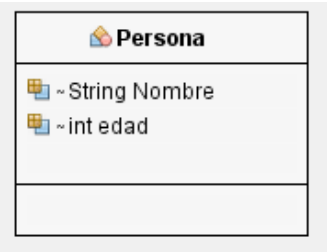
SOBRE UML

El estudio del estándar UML pertenece a otro módulo, nosotros simplemente lo usaremos discretamente para aprovecharnos de que UML nos permite una notación para describir gráficamente una clase y esto nos evita tener que hacer "pesadas" descripciones verbales de los atributos y otros componentes de las clases. UML es mucho más que esto pero a nosotros con la explicación anterior nos basta. En UML las clases se describen con un rectángulo que incluye la definición de los atributos. Por ejemplo la clase Coche anterior se representa:



El rectángulo se divide en tres, en la primera división se indica el nombre de la clase, en la siguiente los atributos y en la siguiente los métodos. Observa que la de los métodos está vacía, ya que por el momento no hemos definido ningún método para la clase Coche.

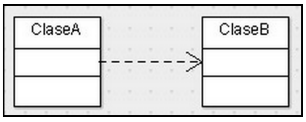
Ejercicio U2_B1_E3.
Con la siguiente clase Persona



Escribe el código necesario para:

1. Crear dos personas p1 y p2.
2. La persona p1 tiene como nombre "yo" y edad 45 años.
3. La persona p2 tiene como nombre "tu" y edad 37 años.
4. Tras crear los objetos suma las edades de p1 y p2 e imprimelas por pantalla.

Relaciones entre clases. La relación de dependencia.
La relación entre clases es un tema amplio, hay varios tipos y subtipos de relaciones. A nosotros ahora nos interesa saber únicamente que hay un tipo de relación entre clases denominado relación de dependencia o relación de uso. En esta relación una de las clases es cliente de la otra, de forma que una clase cliente A *usa* el servicio de otra clase proveedora B. Gráficamente esta relación se representa por una línea discontínua y siguiendo el sentido de la flecha leemos: *La clase A usa la clase B*



Una de las formas más habituales en las que ocurre esta relación es cuando una clase cliente A *crea una instancia* de una clase proveedora B, es decir cuando dentro de A se hace un new B(), y entonces también podemos llamar a esta relación *relación de instanciación*. Ejemplo en java:

```
class A{
        …..
        public static void main(String[] args){
                …….
                new B();
        }
}

class B{
        ……..
}
```

Puede existir entre dos clases una relación de uso sin que haya *new* por el medio, pero esto queda para más adelante.

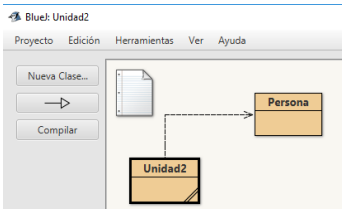
BLUEJ

Con el profesor:

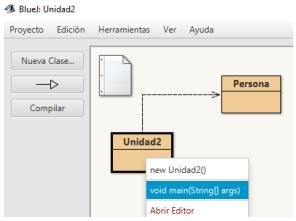
- crea un proyecto Unidad2
- Vamos a escribir en bluej el siguiente código

```
class Persona{
   String nombre;
   int edad;
}

class Unidad2{
   public static void main(String[] args) {
        Persona p1= new Persona();
        Persona p2= new Persona();

        p1.nombre="yo";
        p1.edad=45;
        p2.nombre="tu";
        p2.edad=37;

        int sumaEdad=p1.edad+p2.edad;
        System.out.println("suma de edades de p1 y p2: "+ sumaEdad);

   }
}
```

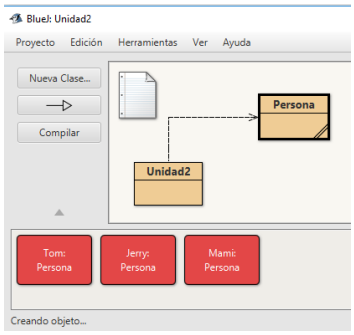
para conseguir.



- Ejecuta haciendo clic en el main() de Unidad2



- Prueba que también para cada clase se puede crear directamente objetos con bluej (sin necesidad crearlos en el main de Unidad2). Esto puede ser útil para probar rápidamente el funcionamiento de un objeto. Veremos su utilidad más adelante. Ahora consigue esto:



Ejemplo: También se pueden crear objetos en el JShell sin necesidad de escribir un main() pero no suele ser muy útil.

```
jshell> class Animal{
   ...> int numeroDePatas;
   ...> }
|  created class Animal

jshell> Animal perro= new Animal();
perro ==> Animal@1e397ed7

jshell> perro.numeroDePatas=4;
$3 ==> 4

jshell>
```

OBSERVAR LA RELACIÓN ENTRE .JAVA Y .CLASS

Desde que se escribe un programa en java hasta que se ejecuta hay el siguiente flujo básico

Código fuente en .java => **Javac** => byte code en .class => **Java** (MVJ) => código máquina para sistema operativo

Comprobamos la relación entre .java y .class. Mejor en una carpeta vacía para hacer comprobaciones. Creamos Coche.java dentro de un directorio(carpeta) *unaprueba*

```
PS C:\Users\donlo\programacion> mkdir unaprueba


    Directorio: C:\Users\donlo\programacion


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        03/10/2021      9:21                unaprueba


PS C:\Users\donlo\programacion> cd  unaprueba
PS C:\Users\donlo\programacion\unaprueba> notepad Coche.java
```

Coche: Bloc de notas

Archivo Edición Formato

```
class Coche{
    String modelo;
    int pasajeros;
    int deposito;
    int kpl;
}
```

Javac genera un Coche.class

```
PS C:\Users\donlo\programacion\unaprueba> ls


    Directorio: C:\Users\donlo\programacion\unaprueba


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        03/10/2021      9:27             93 Coche.java


PS C:\Users\donlo\programacion\unaprueba> javac Coche.java
PS C:\Users\donlo\programacion\unaprueba> ls


    Directorio: C:\Users\donlo\programacion\unaprueba


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        03/10/2021      9:27            279 Coche.class
-a----        03/10/2021      9:27             93 Coche.java
```

El .class ya no tiene instrucciones java de alto nivel, si no un código de bajo nivel que entiende la máquina virtual java. Si editamos ahora con bloc de notas el .class veo que efectivamente está escrito en un lenguaje raro (byte codes)

```
PS C:\Users\donlo\programacion\unaprueba> notepad Coche.class
PS C:\Users\donlo\programacion\unaprueba>
```

Coche.class: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
◆◆◆◆    <
   java/lang/Object <init> ()V Coche modelo Ljava/lang/String;  pasajeros I deposito kpl Code LineNumberTable
SourceFile
Coche.java
```

Si el fichero .java contiene varias clases, se genera un .class para cada clase

Para probar esta relación, primero borro todo lo que había en la carpeta *unaprueba*, compruebo que no queda nada con ls y creo un nuevo .java

```
PS C:\Users\donlo\programacion\unaprueba> rm Coche.*
PS C:\Users\donlo\programacion\unaprueba> ls
PS C:\Users\donlo\programacion\unaprueba> notepad Unidad2.java
PS C:\Users\donlo\programacion\unaprueba>
```

Unidad2: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
class Coche{
    String modelo;
    int pasajeros;
    int deposito;
    int kpl;
}
class Unidad2{
    public static void main(String[] args) {
        Coche golf = new Coche();//se crea el objeto
        golf.pasajeros=5;//se usa el objeto
        System.out.println("pasajeros:" + golf.pasajeros); //se usa el objeto
    }
}
```

Al compilar, observo que se generó un .class por cada clase de .java

```
PS C:\Users\donlo\programacion\unaprueba> javac Unidad2.java
PS C:\Users\donlo\programacion\unaprueba> ls


    Directorio: C:\Users\donlo\programacion\unaprueba


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         03/10/2021      9:43            281 Coche.class
-a----         03/10/2021      9:43            912 Unidad2.class
-a----         03/10/2021      9:38            343 Unidad2.java


PS C:\Users\donlo\programacion\unaprueba>
```

La MVJ a través del comando *java* sólo utiliza los .class, lo comprobamos borrando Unidad2.java y ejecutando Unidad2. Class que a su vez usa Coche.class

```
PS C:\Users\donlo\programacion\unaprueba> rm Unidad2.java
PS C:\Users\donlo\programacion\unaprueba> ls


    Directorio: C:\Users\donlo\programacion\unaprueba


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         03/10/2021      9:43            281 Coche.class
-a----         03/10/2021      9:43            912 Unidad2.class


PS C:\Users\donlo\programacion\unaprueba> java Unidad2
pasajeros:5
PS C:\Users\donlo\programacion\unaprueba>
```

¿Es apropiado escribir varias clases en el mismo fichero .java?

En una aplicación real o grande por varias razones se suele escribir cada clase en su .java. No obstante, nosotros, para hacer rápidamente copiar y pegar en estos primeros ejemplos de consola usamos mucho el escribir en el mismo .java 2 o 3 clases.