VARIABLES REFERENCIA

Tienes que entender a la perfección este documento. Es un momento crítico en el avance del módulo.

DIFERENCIA ENTRE OBJETO Y VARIABLE REFERENCIA

Para crear un objeto golf utilizamos la sentencia:

```
Coche golf = new Coche();
```

En realidad estamos haciendo dos cosas ya que lo anterior es equivalente a:

```
Coche golf;
golf = new Coche();
```

Es decir:

1. Definimos una variable referencia *golf* para almacenar la dirección de memoria de un objeto de tipo Coche. De momento no existe ningún objeto.
2. Con el operador new creamos un objeto Coche y su dirección en memoria se almacena en la variable referencia golf

En java no hay variables que almacenan objetos, hay variables que almacenan referencias (direcciones) a objetos. A estas variables se le llaman variables referencia. En contraposición tenemos las variables de tipos primitivos que sí almacenan directamente un valor.

Hagamos una representación del efecto en memoria de estas tres instrucciones:

```
Coche golf;
golf = new Coche();
int a = 5;
golf.deposito=60;
```

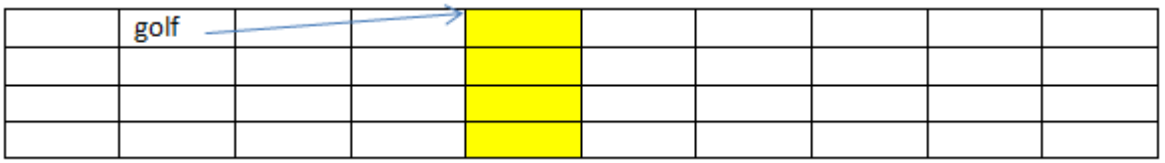
1. Coche golf;

Se crea en memoria una variable referencia golf. Por el momento, esta variable no almacena ninguna dirección. Para indicar que la variable no referencia a ningún objeto tras la declaración sin inicialización java le da automáticamente el valor especial NULL que significa "sin referencia".

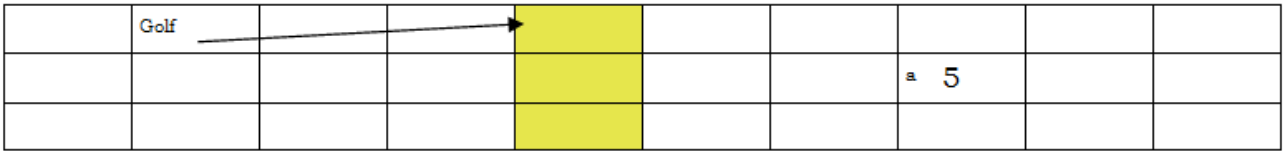
	Golf NULL								

2. golf = new Coche();

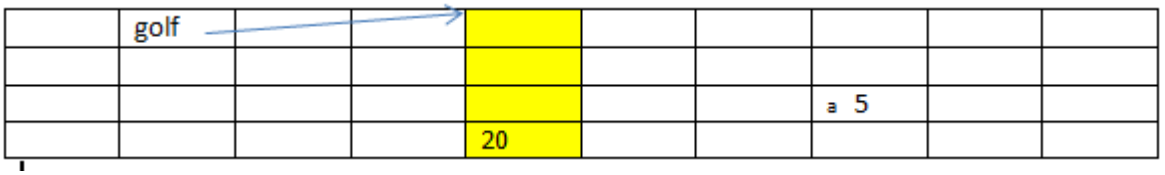
En amarillo representamos el espacio en memoria que new reserva para almacenar un coche. Además new devuelve la dirección de memoria correspondiente a donde se creó el objeto Coche y esta dirección se almacena en la variable referencia golf. Representamos con una "flechita apuntadora" el hecho de que la variable referencia contiene la dirección del objeto creado.



3. int a = 5;
a no es una variable referencia y almacena directamente el valor.



4. golf.kpl=20;
Cómo golf es de tipo coche utilizando la definición de clase correspondiente, sabe cómo desplazarse dentro del espacio de memoria reservado para el objeto



Evidentemente la gestión de objetos y en general la gestión de la memoria en la que se ejecuta el programa es una complicada interacción entre compilador y sistema operativo y la explicación anterior no es más que una aproximación intuitiva de dicha gestión para entender la relación y diferencia entre objeto y variable referencia.

Observa también que el nombre de una clase lo utilizo para dos cosas, relacionadas entre sí, pero diferentes:

- Para definir "el tipo" de una variable referencia. Ejemplo: Coche golf ;
- Para crear, con new, un objeto con la estructura que indica la clase. Ejemplo: new Coche();

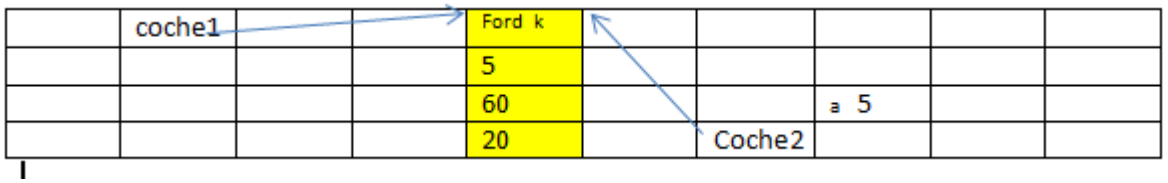
Observa también que entre variable referencia y objeto tiene que haber concordancia de clase:
Coche golf = new Coche(); //O.K.
Persona p = new Coche();//MAL
Coche golf = new Persona();//MAL

Algo parecido a lo que ocurría entre variables de tipo primitivo y literales de tipo primitivo
byte b =2.0 //MAL
double d = 2.0//OK

La concordancia entre variable referencia y objeto es necesaria para que cuando se use el operador punto se acceda a la parte correcta del objeto. Por ejemplo
golf.kpl
Con golf conozco el principio del objeto, luego como golf es de tipo Golf java conoce la estructura del objeto y se desplaza correctamente hasta kpl, 64 bits más allá del comienzo del objeto.

ASIGNACIÓN DE UNA VARIABLE REFERENCIA A OTRA

Un objeto puede ser referenciado por varias variables referencia, por ejemplo vamos a probar la siguiente situación que utiliza un objeto Coche.




```
class Coche{
  String modelo;
  int pasajeros;
  int deposito;
  int kpl;
 }
class Unidad2 {
  public static void main(String[] args) {
      int a=5;
      Coche coche1 = new Coche();
      Coche coche2;
      coche1.modelo="Ford K";
      coche1.pasajeros=5;
      coche1.deposito=60;
      coche1.kpl=20;
      //fijate bien en el efecto de asignar una referencia a otra
      coche2 = coche1;  //coche2 referencia al mismo objeto que coche1
      //demostramos ahora que referencian al mismo objeto
      System.out.println("datos del objeto que referencia coche1:");
      System.out.println("\tpasajeros:" + coche1.modelo);
      System.out.println("\tpasajeros:" + coche1.pasajeros);
      System.out.println("\tpasajeros:" + coche1.deposito);
      System.out.println("\tpasajeros:" + coche1.kpl);


      System.out.println("datos del objeto que referencia coche2:");
      System.out.println("\tpasajeros:" + coche2.modelo);
      System.out.println("\tpasajeros:" + coche2.pasajeros);
      System.out.println("\tpasajeros:" + coche2.deposito);
      System.out.println("\tpasajeros:" + coche2.kpl);

  }
}
```

Debes de entender a la perfección las respuestas de las siguientes preguntas

¿coche2=coche1 copia un objeto y por tanto tras esta sentencia pasa a haber dos objetos?.

Debes de tener claro que no. En el ejemplo anterior sólo hay un objeto. Y no se copia ningún objeto.

¿coche1 es un objeto?.

NO. Es una variable referencia.

¿Cómo se llama ese objeto que se creó con `Coche coche1 = new Coche();` en el código anterior?.

Un objeto no tiene un nombre manejable directamente desde java como las variables

de tipo primitivo.

Y si un objeto no tiene nombre, ¿cómo puedo acceder a él?
A través de un "intermediario" que se llama *variable referencia.* Los objetos son trozos de información a los que se accede a través de variables referencia que contienen una referencia o dirección de ese objeto en memoria.
Para acceder a una parte concreta del objeto se combina una variable referencia que referencia a un objeto con el operador "."

Puede pensarse una variable referencia como una suerte de "acceso directo al objeto". Así, si pensamos en un fichero de disco como en un objeto, resulta que puede tener uno o muchos accesos directos para acceder a él, por ejemplo, uno en el escritorio y otro en la barra de inicio.

¿QUÉ SIGNIFICA QUE UNA VARIABLE REFERENCIA CAMBIE DE VALOR?

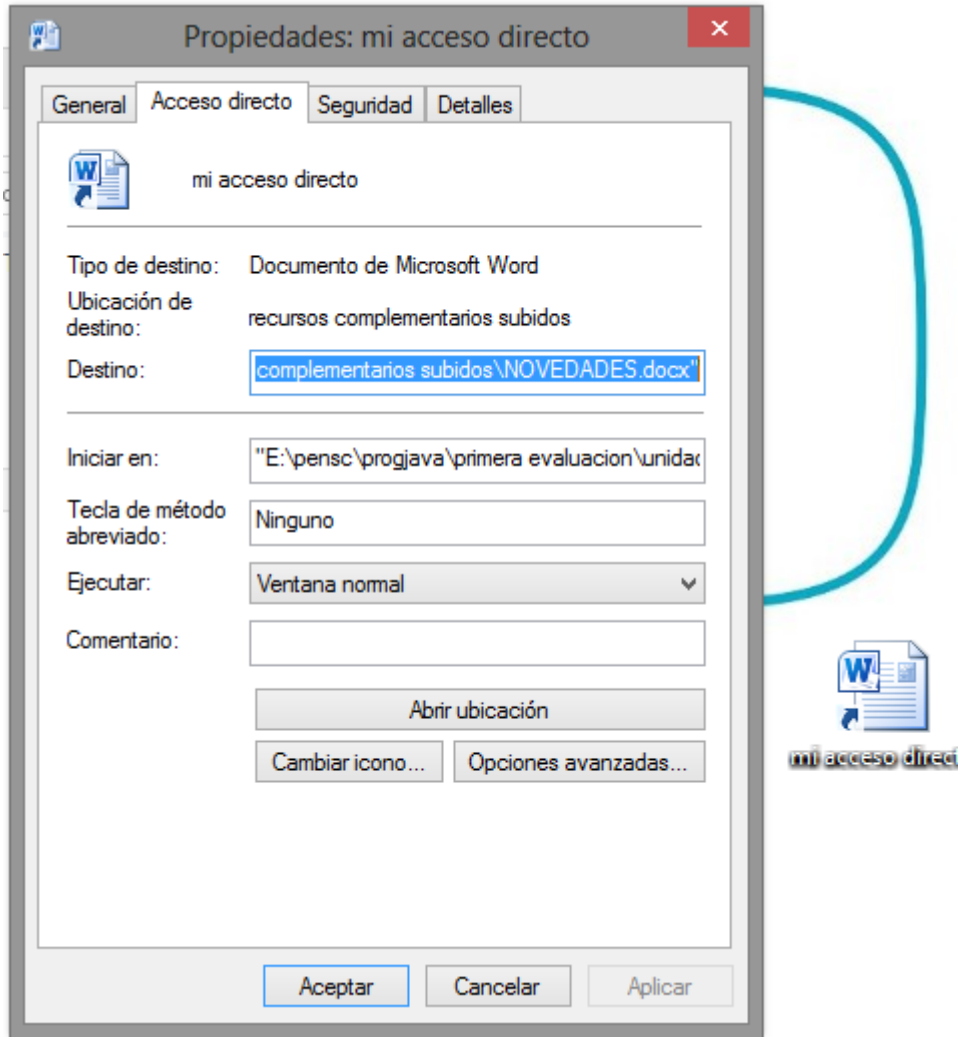
Una variable referencia es una variable, y por tanto puede cambiar de valor. En un momento dado, una variable referencia contiene la referencia de un objeto, pero en el transcurso de un programa puede pasar a almacenar una referencia a otro objeto distinto, eso sí, el tipo del objeto y de la referencia tienen que coincidir (una referencia de tipo Coche no puede referenciar a un objeto de tipo Fruta).

Ejemplo: observa como la variable coche2 referencia al principio al mismo coche que coche1 (a "ford K") y como luego pasa a referenciar a otro coche (a "Renault Clio")

```
class Coche{
  String modelo;
  int pasajeros;
  int deposito;
  int kpl;
 }
 class Unidad2 {
   public static void main(String[] args) {
      Coche coche1 = new Coche();
      Coche coche2;
      coche1.modelo="Ford K";
      coche1.pasajeros=5;
      coche1.deposito=60;
      coche1.kpl=20;
      coche2 = coche1;  //coche2 referencia al mismo objeto que coche1
      //demostramos ahora que referencian al mismo objeto
      System.out.println("datos del objeto que referencia coche1:");
      System.out.println("\tmodelo:" + coche1.modelo);
      System.out.println("\tpasajeros:" + coche1.pasajeros);
      System.out.println("\tdeposito:" + coche1.deposito);
      System.out.println("\tkpl:" + coche1.kpl);


      System.out.println("datos del objeto que referencia coche2:");
      System.out.println("\tmodelo:" + coche2.modelo);
      System.out.println("\tpasajeros:" + coche2.pasajeros);
      System.out.println("\tdeposito:" + coche2.deposito);
      System.out.println("\tkpl:" + coche2.kpl);

      //creamos un nuevo objeto y lo referenciamos a través de coche2.
      coche2 = new Coche();
      coche2.modelo="Renault Clio";
      coche2.pasajeros=7;
      coche2.deposito=70;
      coche2.kpl=25;
```

```
        System.out.println("datos del objeto que referencia coche2:");
        System.out.println("\tmodelo:" + coche2.modelo);
        System.out.println("\tpasajeros:" + coche2.pasajeros);
        System.out.println("\tdeposito:" + coche2.deposito);
        System.out.println("\tkpl:" + coche2.kpl);


    }
}
```

Siguiendo la analogía con accesos directos, a un acceso directo, editando sus propiedades, puedo hacer que apunte a otro fichero distinto. En el gráfico siguiente tenemos un acceso directo en el escritorio de mi portátil llamado "mi acceso directo" que en el momento de hacer la captura de pantalla de abajo, tenía asignado en "Destino" el fichero novedades.docX. Editando el campo destino puedo hacer que el acceso directo apunte a otro fichero.



La conclusión es que acceso directo y fichero son cosas relacionadas entre sí, pero son cosas distintas. De la misma forma en POO(Programación Orientada a Objetos), objeto y variable referencia son cosas relacionadas pero distintas.

RESUMEN DE USO DE VARIABLES REFERENCIAS.

- Asignarle a una variable referencia un objeto, dos formas:
  - de paso que se crea el objeto `Coche coche1 = new Coche();`
  - si el objeto ya existe, a través de otra referencia `coche2 = coche1;`
- Acceder con una variable referencia al contenido del objeto con el operador "."
  - "Leer valores del objeto" `System.out.println("\tpasajeros:" + coche1.pasajeros);`
  - "Modificar valores del objeto" `coche1.pasajeros=7;`
  - Por lo tanto la variable referencia contiene la dirección de un objeto, sabiendo la dirección del objeto y en combinación con el operador "punto", puedo acceder realmente a los diversos componentes del objeto

LAS VARIABLES REFERENCIA Y LA DESTRUCCIÓN DE UN OBJETO.

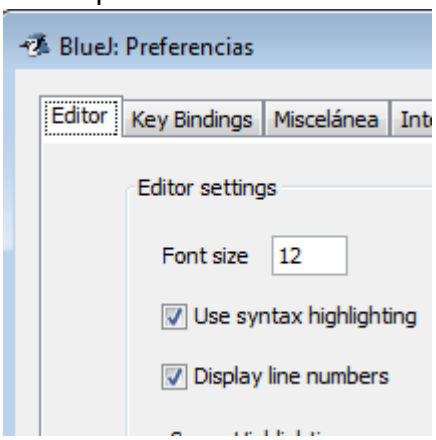
Un objeto puede ser referenciado por una o muchas variables de referencia. Si en un momento dado, un objeto deja de ser referenciado por al menos una variable de referencia ¡perdemos el acceso al objeto!. Los objetos sin referenciar son periódicamente eliminados por java en tiempo de ejecución para dejar espacio a los nuevos objetos. Esta gestión de memoria es automática, el programador no tiene que borrar objetos. A esta afirmación, le añadiremos una puntualización cuando veamos el funcionamiento de la pila de llamadas.

Y en este aspecto, no podemos establecer analogía con los acceso directos del sistema de ficheros ya que si borro todos los accesos directos de un fichero, el fichero no se borra.

ANALIZAR EL EJEMPLO ANTERIOR CON BLUEJ

Para comprobar el ejemplo anterior utilizaremos el debugger de bluej o tracer. Nos permite ejecutar el programa paso a paso (instrucción a instrucción) y ver como van evolucionando los objetos creados.

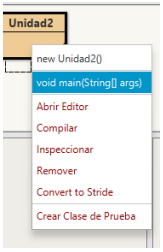
1. pon números de línea en el editor de código



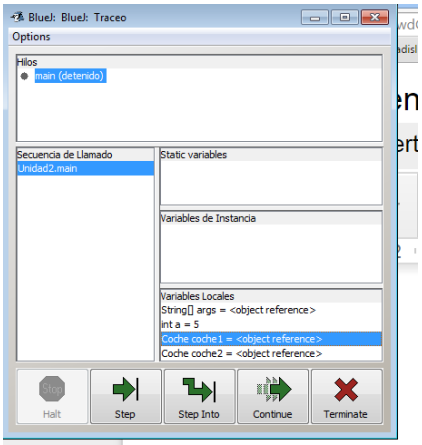
2. Indicar un punto de interrupción(clic en nº de línea). A partir de este punto podemos ejecutar el programa paso a paso.

```
1  class Unidad2 {
2      public static void main(String[] args) {
3          Coche coche1 = new Coche();
4          Coche coche2;
5          coche1.modelo="Ford K";
6          coche1.pasajeros=5;
7          coche1.deposito=60;
```

3. Ejecutar el main



4. Se abre una ventana para avanzar paso a paso similar a la siguiente



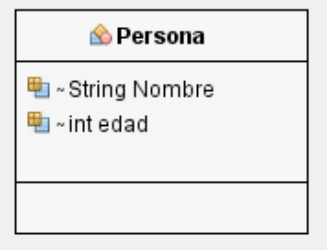
5. Para inspeccionar un objeto clic en la variable referencia. Observa que al hacer clic en la variable referencia bluej me enseña el objeto referenciado por eso en este ejemplo tanto al hacer clic en coche1 o coche 2 vemos el mismo objeto

Ejemplo: comprueba con Bluej(ejecución paso a paso) y con el siguiente código que efectivamente perdí irremediablemente el acceso al objeto creado al llegar al último println()

```
class Unidad2 {
  public  static void main() {
      Coche coche1 = new Coche();
      Coche coche2;
      coche1.modelo="Ford K";
      coche1.pasajeros=5;
      coche1.deposito=60;
      coche1.kpl=20;
      coche2 = coche1;  //coche2 referencia al mismo objeto que coche1
      coche1=null; //desconecto  a coche1 del objeto creado, ahora no referencia a ningún objeto
      coche2=null;//idem para coche2
      System.out.println("El objeto coche creado en la primera línea se perdió para siempre");
```

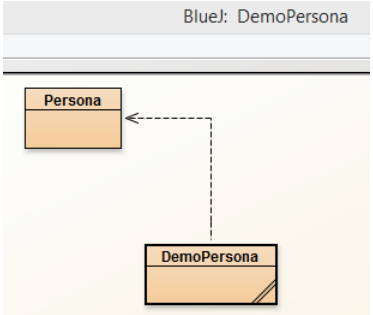
```
 }
}
```

Ejercicio U2__B2_E1
Con la clase Persona



Crea en blueJ un proyecto **DemoPersona**, que contiene dos clases:

- la clase persona
- Una clase DemoPersona que contiene un main() que usa la clase Persona.



- DemoPersona hace lo siguiente:
    - crea un objeto persona con nombre "Ana" y edad 3 y asigna este objeto a la variable referencia p1.
    - Crea un objeto persona con nombre "David" y edad 5 y asigna este objeto a la variable referencia p2.
    - Imprime los nombres de los objetos Persona asociados a p1 y p2 para comprobar que se hizo correctamente.

Ejercicio U2__B2_E2
Igual que sabes intercambiar el valor de variables tipo primitivo

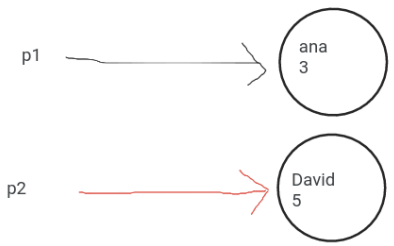
```
int x=3,y=9;
int temp;
temp=x;
x=y;
y=temp;
```

Debes saber intercambiar el valor de la variables referencia.
Se pide completar el siguiente código para generar la salida ejemplo:

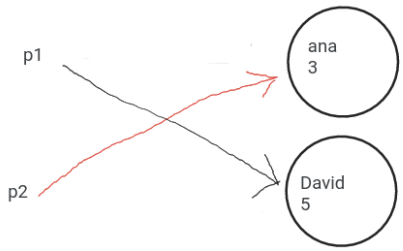
```
class Persona{
   String nombre;
   int edad;
}

class DemoPersona {
   public static void main(String[] args) {
      Persona p1= new Persona();
      Persona p2= new Persona();
      p1.nombre="Ana";
      p1.edad=3;
      p2.nombre="David";
      p2.edad=5;
      System.out.println("ANTES DE INTERCAMBIO etc.
     ETC…..

  }
}
```

```
L:\Programacion>java DemoPersona
ANTES DE INTERCAMBIO, Nombre asociado a variable p1: Ana
ANTES DE INTERCAMBIO, Nombre asociado a variable p2: David
DESPUES DE INTERCAMBIO, Nombre asociado a variable p1: David
DESPUES DE INTERCAMBIO, Nombre asociado a variable p2: Ana

L:\Programacion>
```

La idea es que antes del intercambio …



Y después del intercambio ….



El operador == y referencias

El operador de igualdad == se puede aplicar:

- A tipos primitivos. Por ejemplo 3==5 que devuelve false.
- A variables referencias. Para comparar si el contenido de dos variables referencias es el mismo, es decir, si se referencia al mismo objeto.

Por tanto, la expresión

```
coche1==coche2
```

No está comparando dos objetos, está comparando el valor de las variables coche1 y coche2. Ambas contienen una referencia. Si la igualdad es True quiere decir que ambas variables referencian al mismo objeto, **no** que hay dos coches iguales

Ejercicio U2__B2_E3

Observa este código.

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

class Unidad2 {

  public  static void main(String[] args) {
       Coche coche1 = new Coche();
       coche1.modelo="Ford K";
       coche1.pasajeros=5;
       coche1.deposito=60;
       coche1.kpl=20;

       Coche coche2 = new Coche();
       coche2.modelo="Ford K";
       coche2.pasajeros=5;
       coche2.deposito=60;
       coche2.kpl=20;

       Coche coche3=coche1;

  }
}
```

coche1 y coche2 apuntan realmente a objetos distintos, cada uno de estos objetos tiene los mismos valores pero son objetos distintos. Por el contrario coche3 y coche1 referencia al mismo objeto.

SE PIDE: Hacer un dibujo que relacione los dos objetos con las 3 variables y Demostrar lo anterior utilizando el operador == obteniendo una salida similar a la siguiente

```
¿Es cierto que coche1 == coche2?false
¿Es cierto que coche1 == coche3?true
```

¿ Qué imprime *System.out.println("coche1 es: "+coche1)*;?

Imprime un valor Coche@numeroHexadecimal

No es en absoluto exacto pero puedes por el momento pensar que el numeroHexadecimal es la posición de memoria del objeto que referencia coche1, por lo tanto, en el siguiente código coche3 y coche1 tienen el mismo numeroHexadecimal.

```
class Unidad2 {

  public  static void main() {
      Coche coche1 = new Coche();
      coche1.modelo="Ford K";
      coche1.pasajeros=5;
      coche1.deposito=60;
      coche1.kpl=20;

      Coche coche2 = new Coche();
      coche2.modelo="Ford K";
      coche2.pasajeros=5;
      coche2.deposito=60;
      coche2.kpl=20;

      Coche coche3=coche1;
      System.out.println("coche1 es: "+coche1);
      System.out.println("coche2 es:"+coche2);
      System.out.println("coche3 es:"+coche3);


  }
}
```

```
coche1 es: Coche@178905a
coche2 es:Coche@113be31
coche3 es:Coche@178905a
```

En cada ejecución del programa, los objetos se crearán en parte diferentes de memoria y por tanto obtendremos números distintos pero con el código anterior el número de coche1 y coche3 siempre coinciden ya que referencian al mismo objeto.

Realmente, la auténtica dirección de memoria que almacena una referencia no se puede imprimir en java y esto está hecho así de forma deliberada. Uno de los principios de diseño de java era precisamente ocultar el manejo directo de direcciones de memoria.

CONCLUSIONES IMPORTANTES DE LO QUE LLEVAMOS VISTO HASTA AHORA

1. Por el momento, podemos ver un objeto como un super dato que empaqueta o contiene muchos datos dentro de él. Realmente un objeto es mucho más que esto pero con lo que llevamos visto hasta ahora esta visión no está mal para

empezar
2. La clase, es el libro de instrucciones o molde con el que fabricamos objetos.
3. Para acceder a los objetos utilizo variables referencia.
4. Una variable referencia es una variable que contiene un número entero que llamamos referencia. Dicho número entero no es un objeto si no una posición de memoria en la que se almacena un objeto, por tanto, a través de dicha referencia tengo acceso al objeto.