UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CIENCIAS
PROGRAMACIÓN

Clases y objetos en Java

Gerardo Avilés Rosas
gar@ciencias.unam.mx

Creación y uso de Clases en Java

Creación de clases

- En toda clase se debe definir tanto la **estructura** como el **comportamiento** que tendrán sus objetos.
- La **parte es estructural** de los objetos se define mediante la **declaración de datos**, éstos pueden ser de **cualquier tip**o definido por el lenguaje.
- El **comportamiento** de los objetos se modela mediante **métodos**, y es sólo mediante éstos que se pueden ***asignar, alterar y conocer*** el **estado** de un objeto.

Diseño de clases

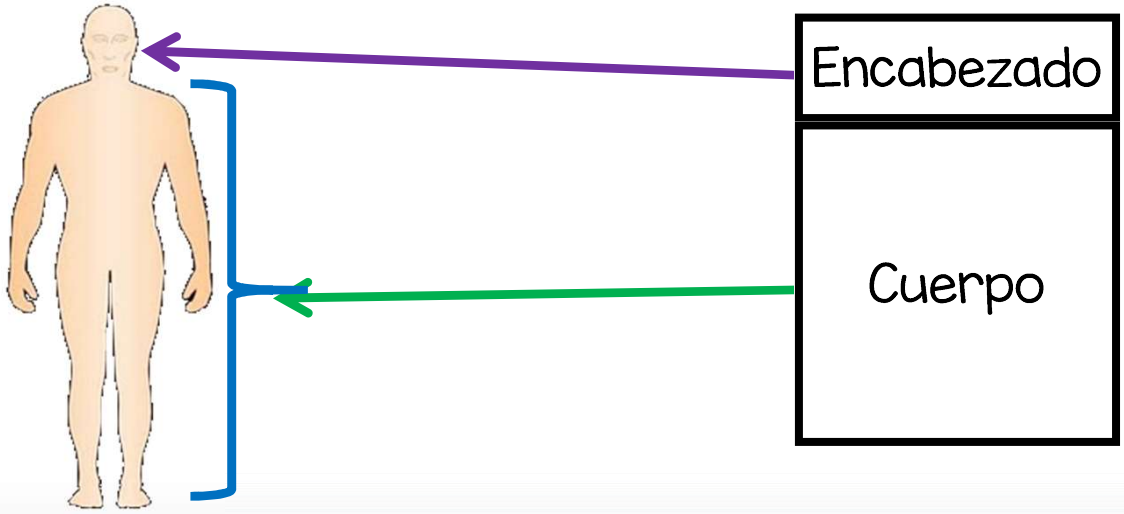
1. **Descubrir** las clases que se requieren en la solución del problema; es necesario asociar los **sustantivos** de la definición del problema con los objetos/clases.

2. **Asignar o determinar** responsabilidades a cada clase (*acciones que realiza la clase*), generalmente se usan los **verbos** de la descripción del problema y sólo considerar las responsabilidades principales.

3. La **colaboración** entre objetos se determina mediante el establecimiento de **escenarios** que ejemplifiquen la actividad del programa.

Definición de clases

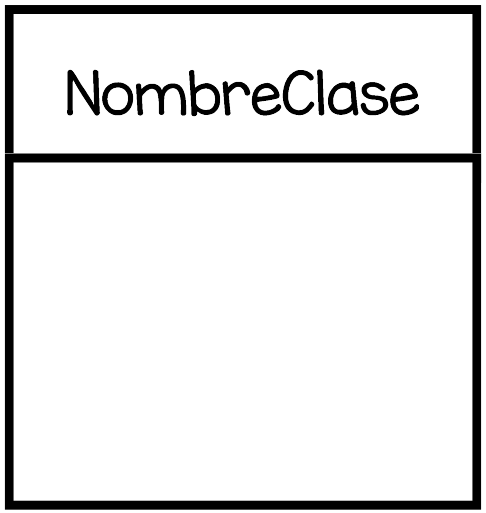
- En cada clase se define la estructura y el comportamiento que tendrán los objetos de la clase.
- Para programar una clase en Java se requiere de un **encabezado** y un **cuerpo**:
  - El **encabezado** incluye el nombre de la clase.
  - El **cuerpo** contiene atributos y métodos, estos pueden estar definidos en cualquier orden, sin embargo, por cuestión de orden y de estilo primero se definen los atributos y luego los métodos.



Encabezado de la clase

- Consta, como mínimo, de la **visibilidad de la clase:** pública (**public**) o privada (**private**). En caso de omitir este calificador se asume que es pública.
- Después aparece la palabra reservada **class** seguida del **nombre de la clase** (normalmente en singular)

```
public class NombreDeLaClase {...}
```



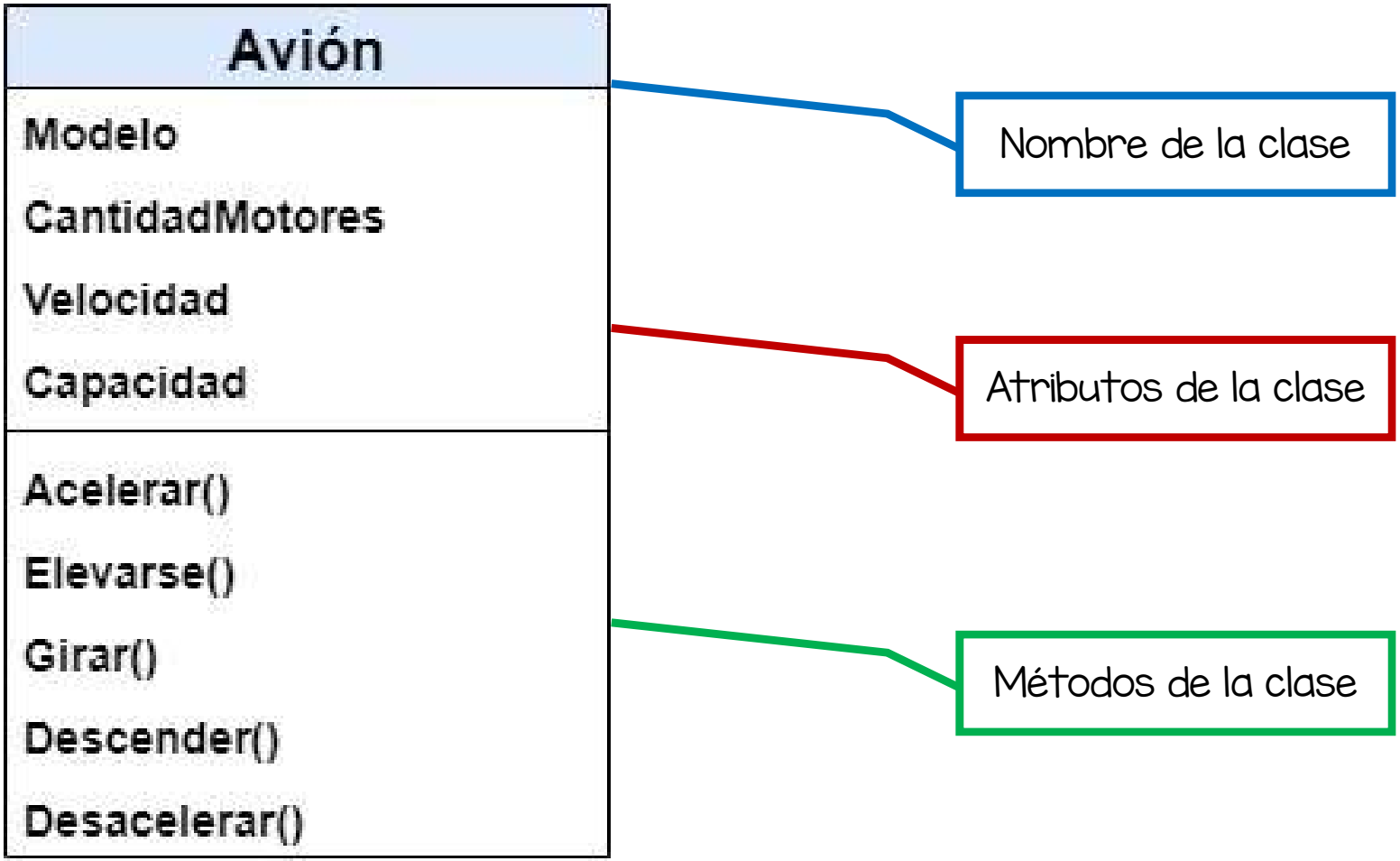
Atributos de clase

- La **estructura de los objetos** de la clase se define declarando sus atributos como datos de cualquiera de los tipos definidos por el lenguaje, incluyendo referencia a otros objetos.
- Cuando se define la estructura para los objetos de una clase, ésta no puede alterarse, pero el estado del objeto puede cambiar todas las veces que sea necesario.
- Las **variables de instancia** son las variables utilizadas para definir los atributos de una clase y tienen los siguientes componentes:
  - **Visibilidad:** especifica cuáles atributos pueden tener acceso desde le cliente que usará el objeto. Se tiene solo el acceso al estado del objeto pero sólo a través de los métodos proporcionados para ello. La visibilidad se expresa mediante las palabras reservadas: **public**, **private** o **protected**; y **todo** atributo debe tener visibilidad **privada.**

…Atributos de clase

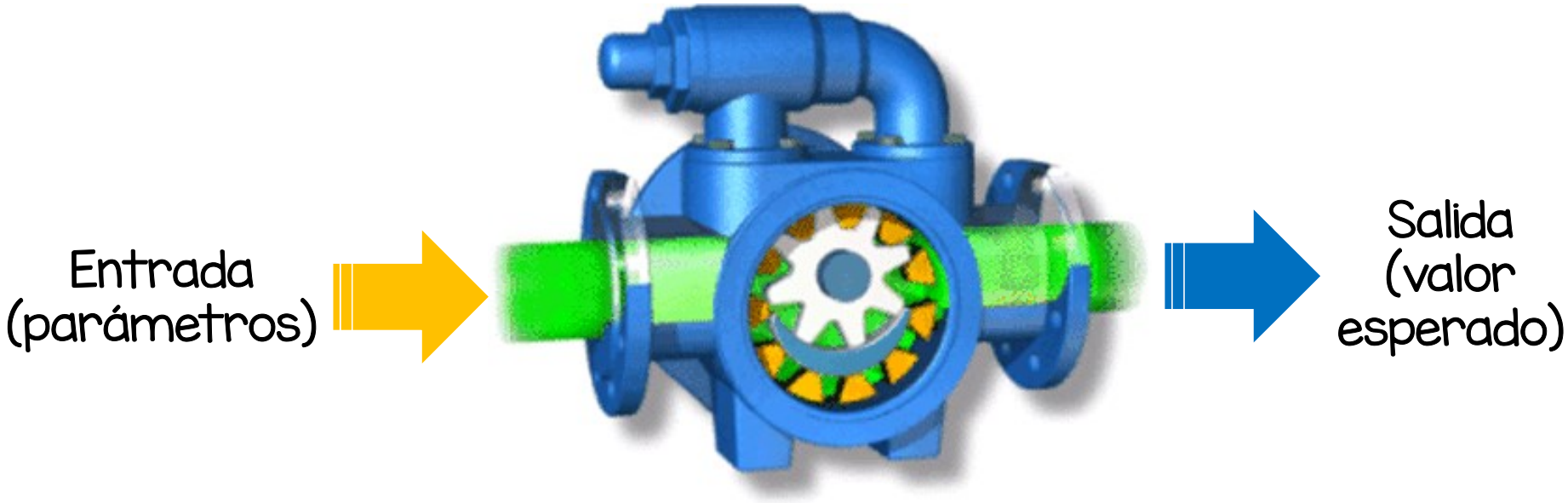
- **Calificador static (o ninguno):** si se utiliza, significa que es un atributo compartido por todos los objetos de la clase (atributo de clase). Si se modifica el valor de un objeto, éste se modifica para todos los objetos de la clase. Si no, se trata de un atributo que tendrá cada objeto de la clase.
- **Calificador final (o ninguno):** si se utiliza este calificador se está definiendo un atributo constante, sino, se trata de un atributo cuyo valor puede cambiar.
- **Tipo:** Algún tipo primitivo o nombre de clase.
- **Nombre:** identificador del atributo, por convención, el identificador debe empezar con minúscula.
- **Valor inicial (opcional)**

...Atributos de clase



Métodos de la clase

- El comportamiento de los objetos de una clase se programa mediante la creación de métodos.
- Un método es el conjunto de instrucciones que se deben realizar al llamar o ejecutar tal método:



…Métodos de la clase

- **Visibilidad:** especifica a cuáles métodos se puede tener acceso desde un objeto cliente. La visibilidad de los métodos se especifica mediante las palabras reservadas: **public**, **private** o **protected**; si no se especifica, se asume que es pública. Un **método privado** es auxiliar en el trabajo de algún otro método de la clase y no puede ser usado por los clientes. La visibilidad de los atributos y métodos se relaciona directamente con la encapsulación.
- **Tipo de valor que devuelve:** Es el nombre de la clase, un **tipo primitivo** o bien **void** en el caso de que no devuelva nada.
- **Nombre:** se trata de un identificador que describa la función del método y por convención empieza con una letra minúscula.

...Métodos de la clase

	`public`	`private`
Atributo	Viola la encapsulación	Forza la encapsulación
Método	Proporciona servicio a clientes	Auxiliar a otro métodos de la clase

- **Información necesaria para realizar esta tarea:**
  - *Parámetros formales:* especifican el tipo de valor que requiere el método para trabajar como una lista separada por comas entre paréntesis, cada pareja incluye el tipo y el nombre. No se especifica la visibilidad y no se puede definir un valor inicial.
  - *Parámetro real (actual):* cuando se llama a ejecución un método, el valor del parámetro real se asigna como valor inicial del parámetro formal y termina la relación entre ambos parámetros.
  - *Paso de parámetros:* si en el cuerpo del método se modifica el valor del parámetro formal, no cambia el valor del parámetro real.

Métodos constructores

- Se trata de un método cuyo objetivo es asignar valor inicial a cada atributo de un objeto recién creado, con lo cual se garantiza que el objeto se cree con un estad válido.
- Generalmente, en el diseño de un programa no se especifican los constructores, pero es obligatorio tenerlos.
- Se distinguen de cualquier otro método porque:
  - Tienen el mismo nombre de la clase a la que pertenece
  - No tienen especificado el valor de regreso, ni siquiera **void**
  - La única forma de llamarlo es usando el operador **new**

...Métodos constructores

- Las clases comúnmente tienen más de un constructor, de manera que se tengan distintas posibilidades de creación de un objeto (**sobrecarga**).
- El **constructor por omisión** se denomina a aquel constructor que no tiene parámetro y se llama cuando se quiere asumir un estado inicial predeterminado.
- Una forma de definir los valores de un objeto en base a otro objeto de su misma clase es a través de un objeto que toma como parámetro un objeto de su misma clase que se conoce como **constructor copia**.

Métodos modificadores

- Tienen el propósito de **modificar** el valor de los atributos privados de los objetos, el estado del objeto debe ser diferente antes y después de llamar al método **modificador**.
- Reciben como parámetro el nuevo valor para el atributo correspondiente y el tipo del parámetro debe coincidir con el tipo del atributo o ser de tipo compatible.
- No devuelven ningún valor.
- Su objetivo es mantener la integridad de los objetos de la clase, pues son la única forma posible de modificación.
- Generalmente su nombre comienza con la palabra en inglés **set** o **asignar** en español.

Métodos de acceso

- Son el medio para conocer el valor de los atributos privados de los objetos.
- No reciben parámetros pues su objetivo es obtener el valor de un atributo y por la misma razón el valor que devuelve es del tipo definido en el atributo.
- Su nombre empieza con la palabra en inglés **get** u **obtener** en español.
- Dentro del cuerpo del método se especifica mediante la instrucción **return** que un método devolverá un valor.

Métodos calculadores

- Estos métodos se emplean para implementar cualquier comportamiento deseado de los objetos.
- Pueden recibir y/o devolver valores, estos métodos no modifican el estado del objeto ni devuelven el valor de algún atributo.
- Trabajan con el estado del objeto para calcular un valor.
- Cuando se escribe un método se debe de tener presente que los métodos se llaman con un objeto, sobre cuyo estado se está trabajando.

El método main

- En Java se tiene el método **main**, el cual es indispensable pues en este punto es donde comienza la ejecución del programa:

```
public static void main(String[] args){
        //Cuerpo del método
}
```

- Se trata de un método público no relacionado con objetos sino con la clase (**static**), que no devuelve ningún valor (**void**) y recibe un parámetro **args** y no se trata de una palabra reservada, así que se puede llamar de otra forma.
- Como se trata de un método estático, no requiere de un objeto para ser llamado y como no opera en el contexto de un objeto particular no puede referenciar variables de instancia, lo que si puede hacer es referencias variables estáticas pues éstas son independientes de los objetos.
- Sólo se puede acceder a este método con variables estáticas, locales y métodos estáticos.

El método equals

- En ocasiones es importante saber si dos objetos particulares son iguales, por ejemplo, supongamos que deseamos comparar si el objeto **o1** es igual al objeto **o2**.

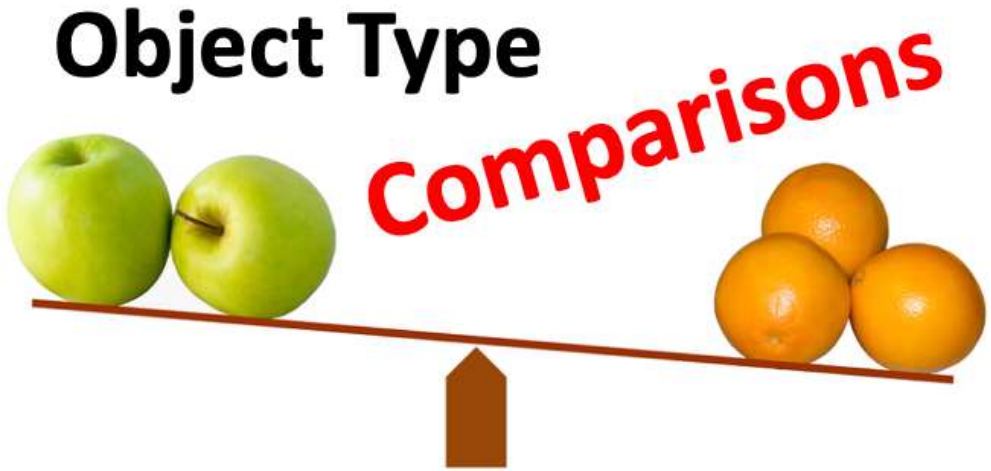
```
if(objeto1 == objeto2)
        System.out.println("Los objetos son iguales");
```

- Esta comparación que en apariencia es la solución al problema planteado, no es correcta debido a que se están comprobando las **referencias a los objetos**, no el estado de cada objeto.
- El **operador ==** verifica si el contenido de las variables de referencia es el mismo, es decir, verifica si ambas variables son referencia al mismo objeto, o dicho en términos más formales, si ambas referencias son **alias** del mismo objeto.

…El método equals

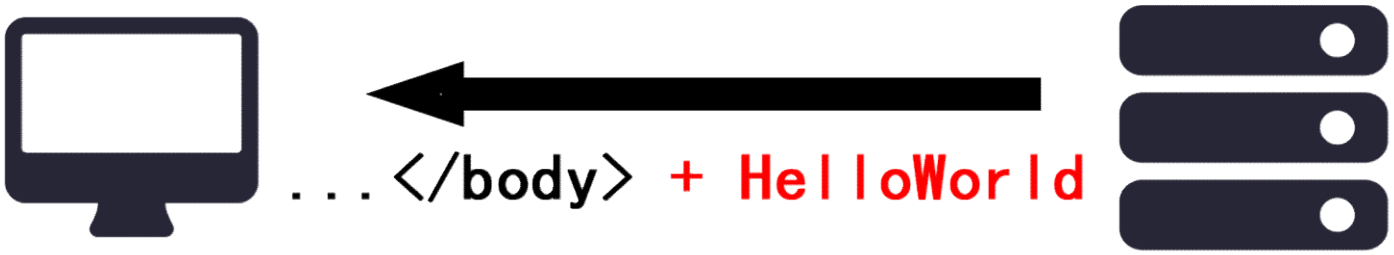
La forma de solucionar esto es comparando, no la referencia a la que apunta el objeto, sino que se debe comparar el contenido del objeto o bien utilizar el método **equals** de la clase **Object:**

```
if(objeto1.equals(objeto2))
        System.out.println("Los objetos son iguales");
```

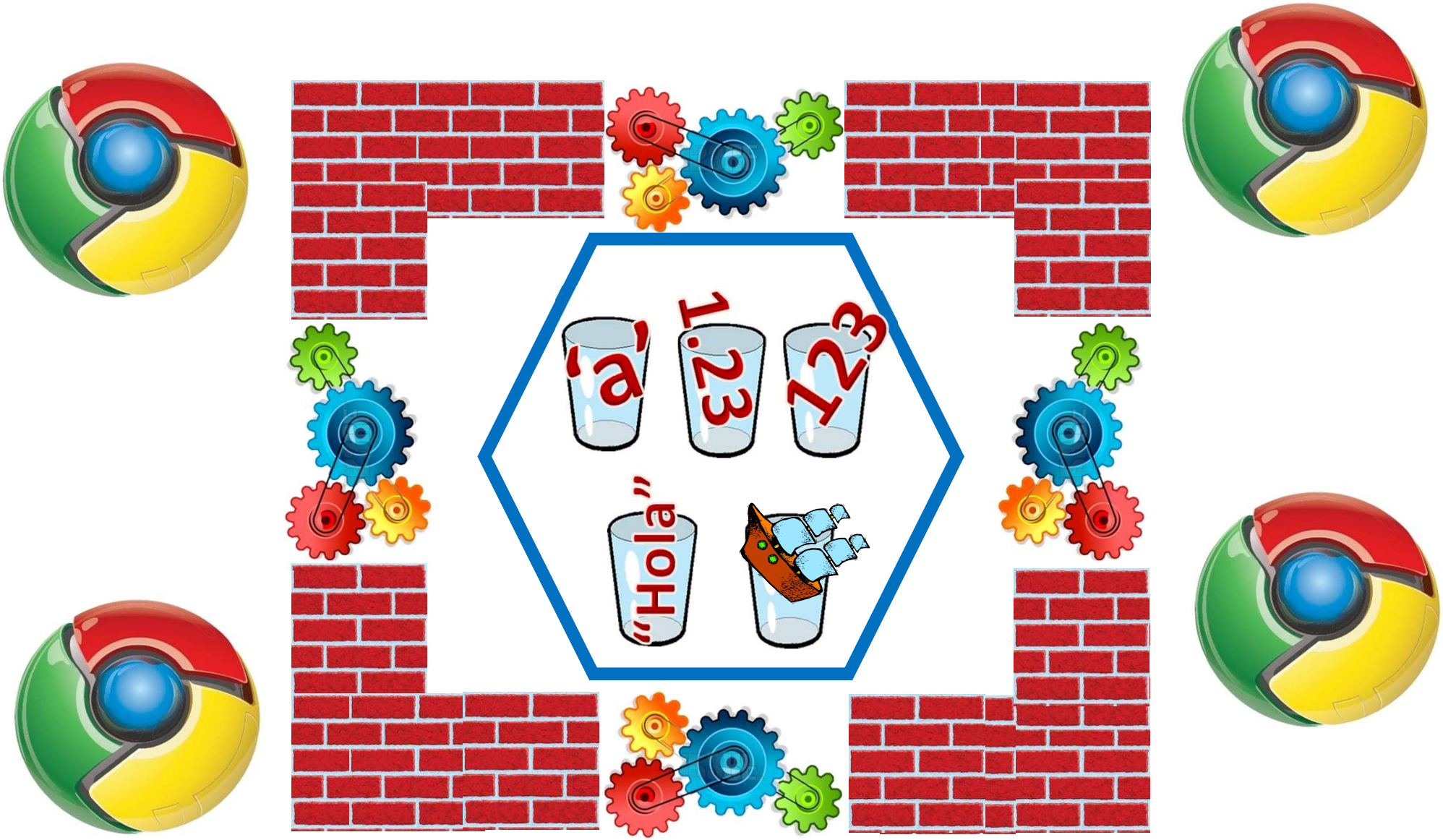


El método toString

- El método **toString** nos permite obtener de una forma sencilla la representación en forma de cadena de caracteres.
- No es necesario que cada vez que queramos imprimir el contenido de un objeto invoquemos el método **obtener** dentro del método **System.out.println()**; lo cual resulta incómodo y propenso a errores.



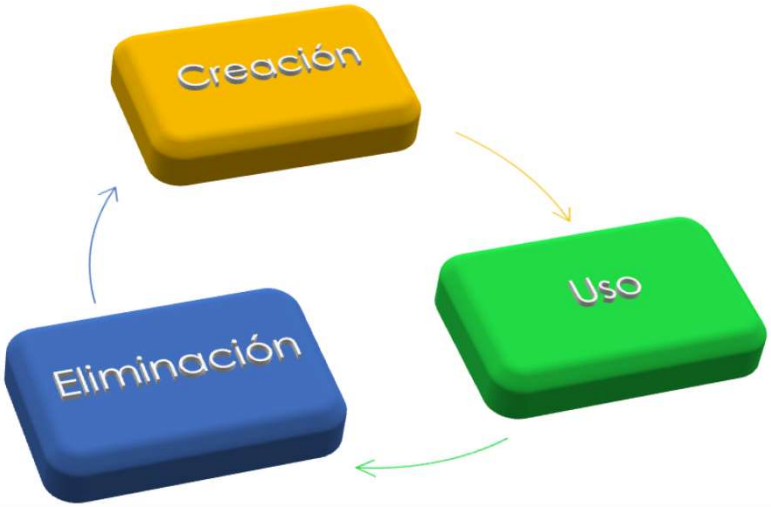
Clase y métodos



Creación y uso de Objetos en Java

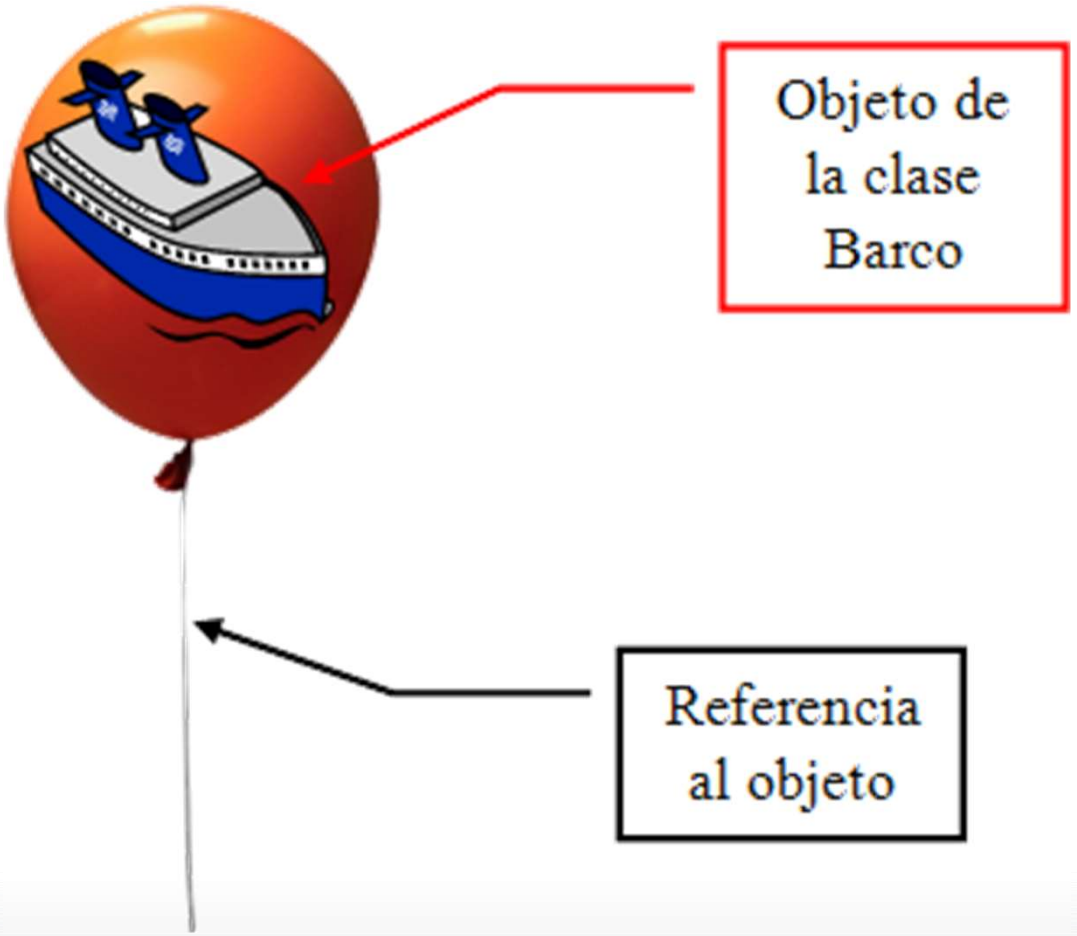
Creación de objetos

- Al escribir un programa en Java, es común que se tengan la necesidad de **crear varios objetos**.
- Éstos interactuarán entre sí mediante el envío y recepción de **mensajes**.
- Una vez que un objeto termina el trabajo para el que fue creado, se recicla la memoria utilizada por él y de esta manera puede ser usada por otros objetos.
- La creación, uso y eliminación de un objeto se conoce como **ciclo de vida de un objeto**.



Referencias

- Una **referencia** es un *tipo de dato* que se usa para trabajar con objetos.
- El valor que se almacena es una **variable** o **constante** de **tipo referencia** y no el objeto en sí (dirección o referencia de un objeto).



...Referencias

- Antes de crear un objeto es necesario **declarar** una variable o constante para almacenar la referencia a él.
- La forma de la declaración es similar a cualquier tipo de dato primitivo:

```
NombreDeClase referencia = new NombreDeClase();

Barco titanic = new Barco();

Moneda laSuertuda;
laSuertuda = new Moneda;
```

- Los objetos se crean con el estado inicial definido en la clase y en ocasiones es necesario que se proporcionen los valores para este estado inicial a través de los paréntesis que se encuentran después del nombre de clase:

```
Barco titanic = new Barco(30000);

Moneda laSuertuda = new Moneda(20);
```

...Referencias

Declaración de referencia

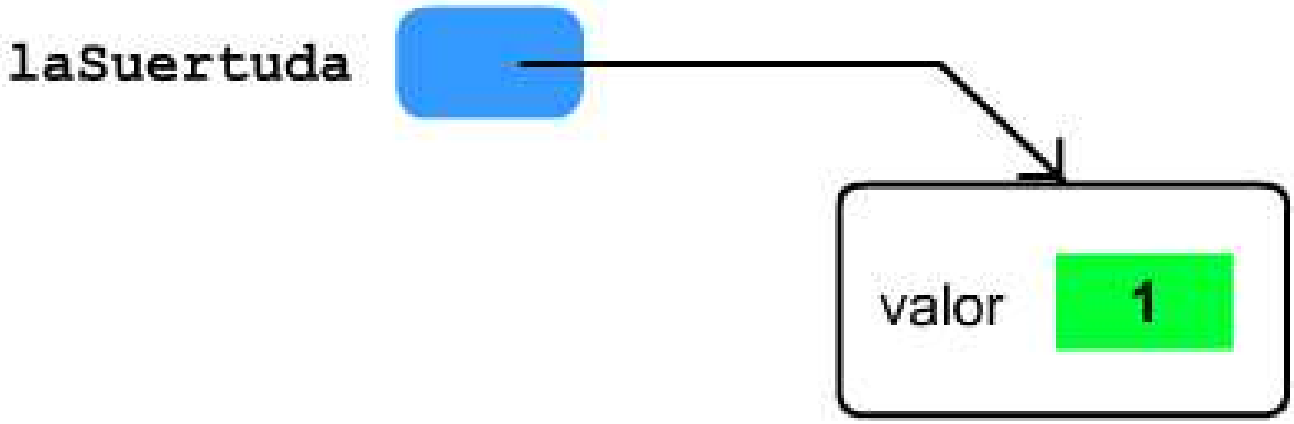
```
Moneda laSuertuda;
```

Creación del objeto

```
Moneda laSuertuda;
new Moneda();
```

Asignación de referencia

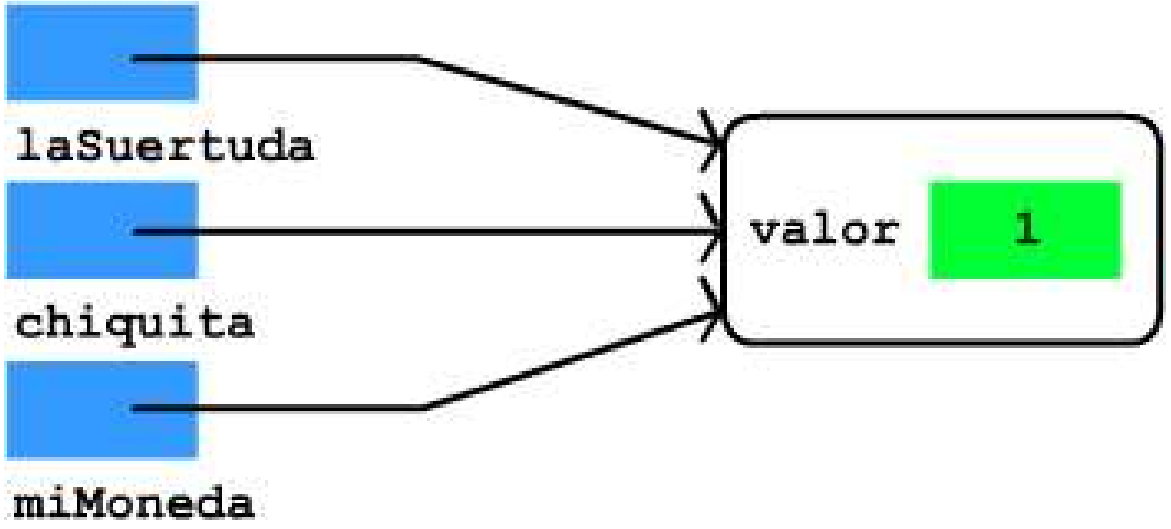
```
Moneda laSuertuda = new Moneda();
```



…Referencias

- Una **referencia** solo tiene un valor a la vez, pero un objeto puede tener varias **referencias**:

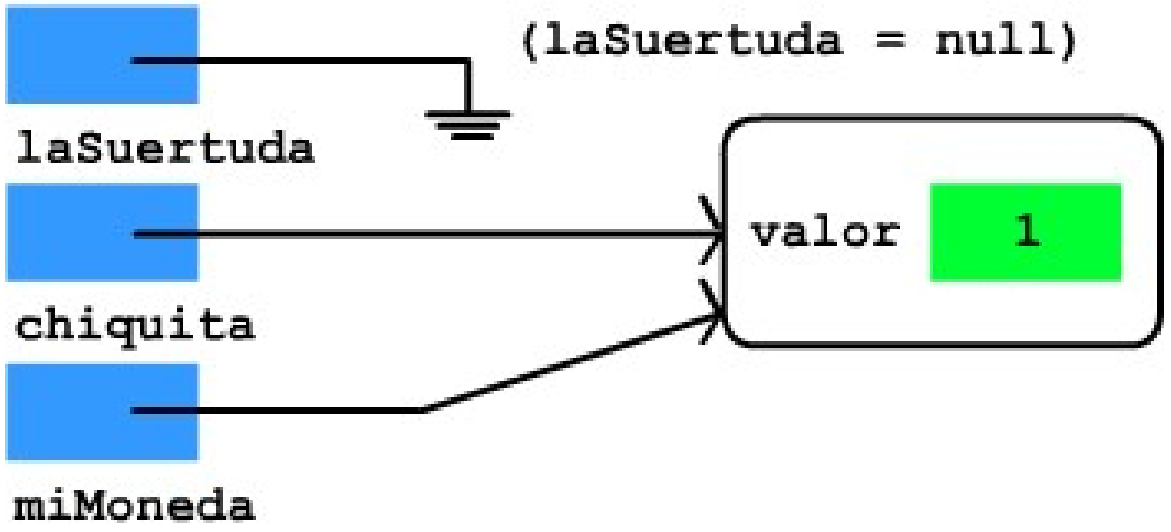
```
Moneda laSuertuda = new Moneda();
Moneda chiquita = laSuertuda;
Moneda miMoneda = laSuertuda;
```



…Referencias

- Cuando se termina de trabajar con un objeto se puede asignar el valor **null** a la referencia, de manera que ya no sea accesible desde tal referencia.
- La palabra **null** es una palabra reservada en Java que sirve para especificar que una referencia no está ligada a ningún objeto:

```
laSuertuda = null
```



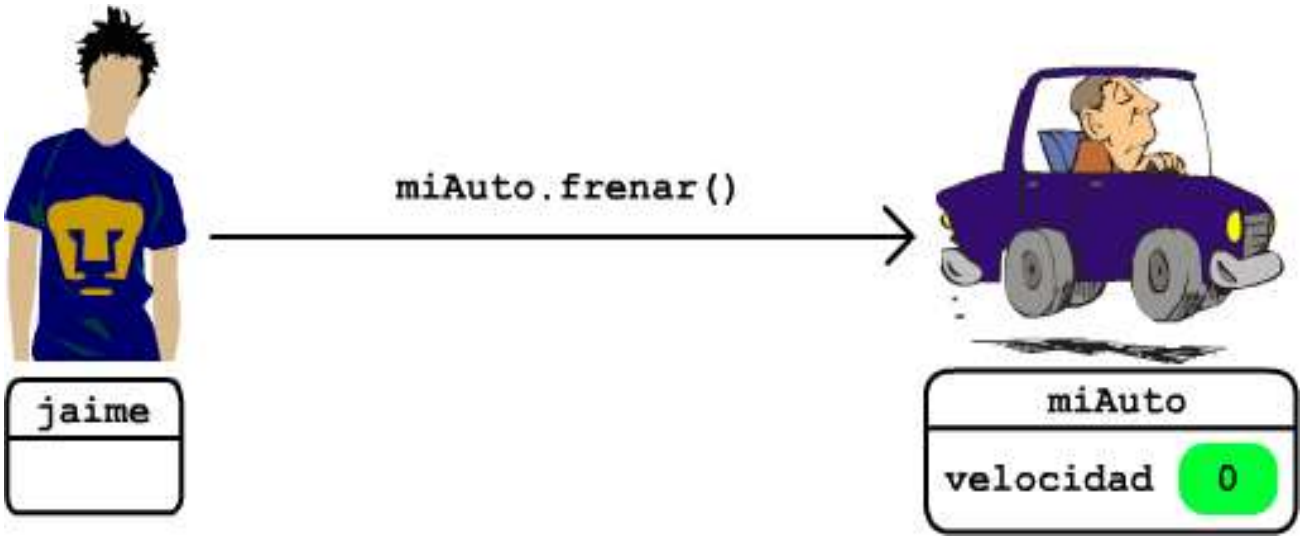
Uso de objetos

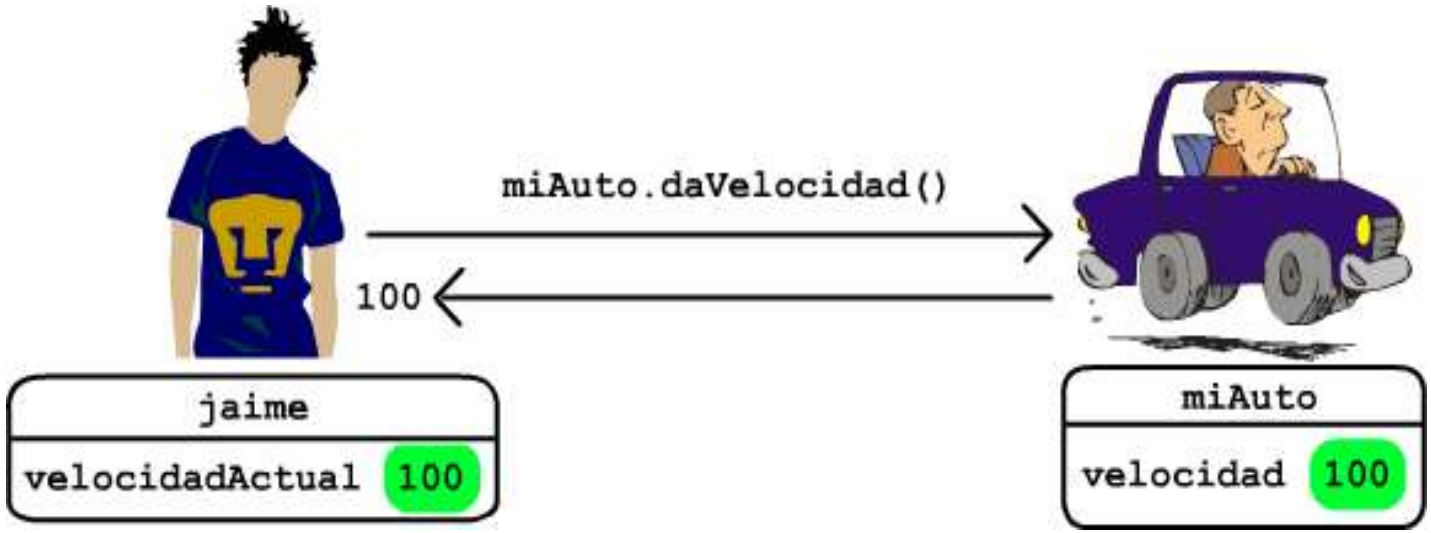
- Una vez que se ha creado el objeto se le pueden enviar **mensajes** para *conocer, modificar su estado* o bien *realizar un cálculo*.
- Cada mensaje se implementa por una llamada a un método usando la notación punto:

```
referenciaObjeto.nombreMétodo(lista_de_parámetros);
```

- La respuesta a un mensaje depende de la interacción entre los métodos definidos en la clase del objeto que recibió el mensaje y del valor asignado en el mensaje a cada uno de sus parámetros.
- En algunos casos, la respuesta a un mensaje implica cambiarle el estado a un objeto.
- Hay mensajes que requieren información adicional y en ocasiones se espera recibir el valor de un atributo como respuesta.

…Uso de objetos





…Uso de objetos

- Una vez que se envía un mensaje a un objeto, el objeto emisor debe esperar a que termine de ejecutarse el método que responde a dicho mensaje antes de realizar cualquier otra tarea (**ejecución secuencial**)
- Para llamar a un método se debe d conocer su **signatura o firma**, que consta del **nombre del método** y entre paréntesis una lista con los **parámetros requeridos** (tipo y nombre de cada parámetro, el orden del éstos está implícito).
- El tipo que devuelve el método no forma parte de la firma, pero es conveniente definirlo para almacenar el resultado de un método en una variable del tipo de valor que se especifica para poder trabajar con el valor más adelante:

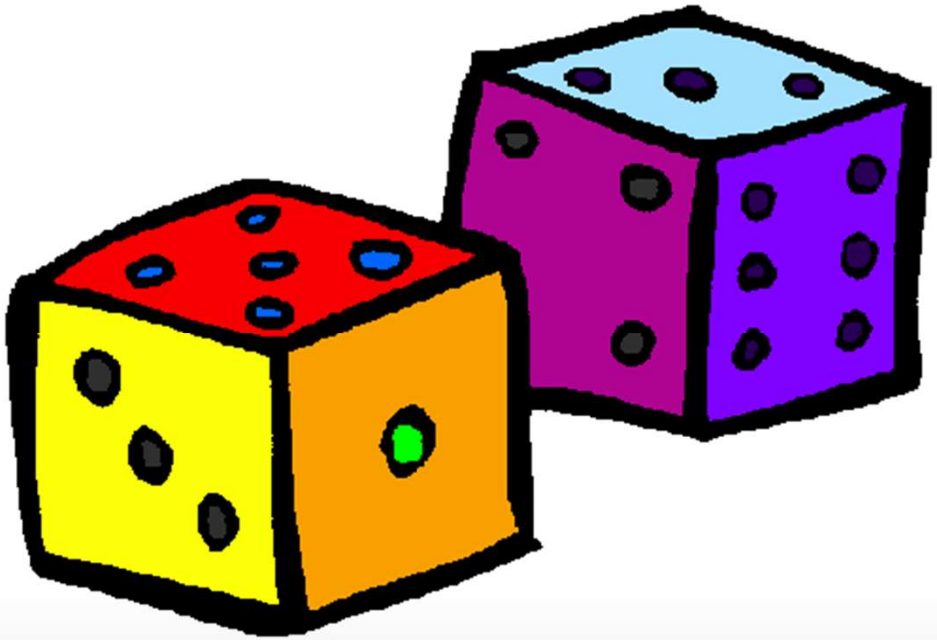
```
public char lanzar(){...}

public String getNombre(){...}

public void setNombre(String n){...}

public double distancia(Punto p){...}
```

Eliminación de objetos

- Java permite al programador crear todos los objetos que necesite si preocuparse si el sistema tiene suficientes recursos o no, la máquina virtual los destruye cuando ya no se necesitan (cuando ya no hay más referencias a él).
- Esta operación se hace a través de un **recolector de basura** que se llama automáticamente.
- Con el método **gc** de la clase **System** es posible liberar todos los recursos del sistema en un momento determinado.

Ejemplo

Escribir un programa que juegue dados con las siguientes reglas: se tiran dos dados y se suma el valor de la cara superior de cada uno. Si la suma es **7** u **11** el jugador gana. Si la suma es **2, 3 ó 12** el jugador pierde. Si la suma es **4, 5, 6, 8, 9 ó 10**, esta se convierte en los puntos del jugador quien, para ganar, debe seguir tirando los dados hasta que en su suma de la tirada más lo acumulado obtenga un total de 11 puntos para ganar. Si en esas tiradas extra la suma es 7, o bien en los acumulados excede 11 puntos, el jugador pierde.



¡Gracias!


No estés muy orgulloso de haber comprendido estas notas.
La habilidad para manejar las **Clases** y **Objetos** en Java es insignificante comparado con el poder de la **Fuerza**.