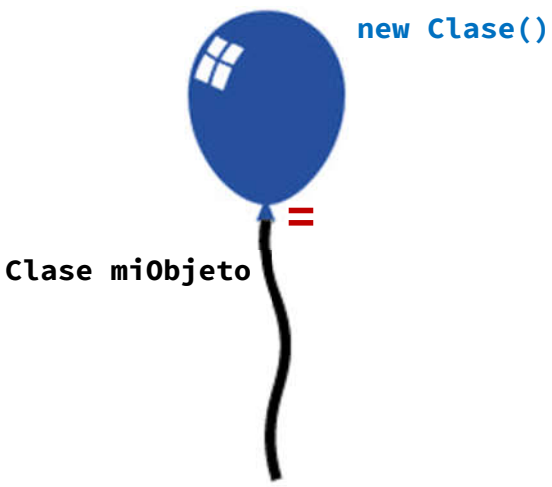
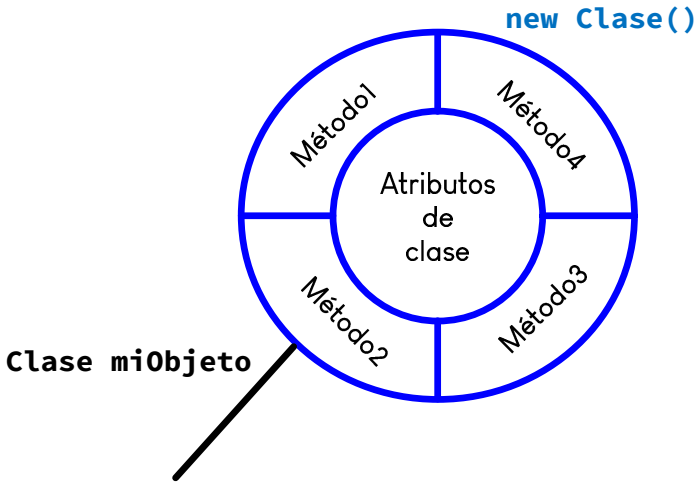
Clase Complejo. Utilizar objetos de clases que se están definiendo

Como se ha estudiado, la **creación de objetos**, requiere, algo de este estilo:

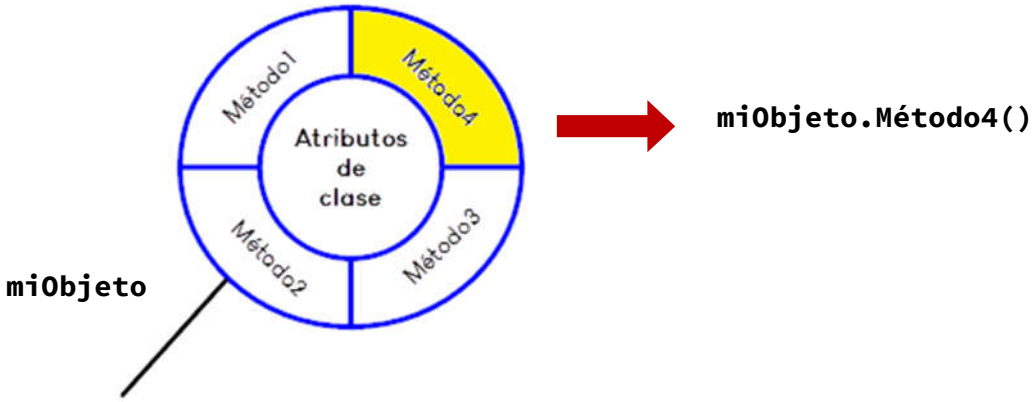
```
Clase miObjeto = new Clase();
```

Lo cual, crearía un objeto, bajo dos posibles abstracciones de fácil comprensión:





Para poder utilizar un objeto, se requiere forzosamente hacerlo desde la **referencia al objeto** y a través de la **notación punto**, se puede **invocar a cualquiera de los métodos** definidos en la clase. En lo sucesivo, utilizaré la segunda forma de representar a los objetos:



Dependiendo del tipo de método (si tiene **tipo de retorno** o si es **`void`**), será la forma en cómo se va a invocar el mismo:

- **Con tipo de retorno.** Se puede utilizar dentro de un método **`println`** o bien, **crear una variable** que pueda recibir la respuesta que devuelve el método (**receptora**), la variable debe ser **del tipo de retorno** del método:

```
System.out.println(miObjeto.nombreMétodo());

tipoDato variable = miObjeto.nombreMétodo();
```

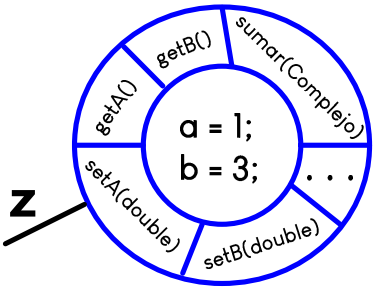
- **Método void.** Se hace la invocación en el punto del programa dónde se necesite (NO SE PUEDE UTILIZAR DENTRO DE UN MÉTODO **println** Y NO SE REQUIERE NINGUNA VARIABLE RECEPTORA).

```
miObjeto.nombreMétodo();
```

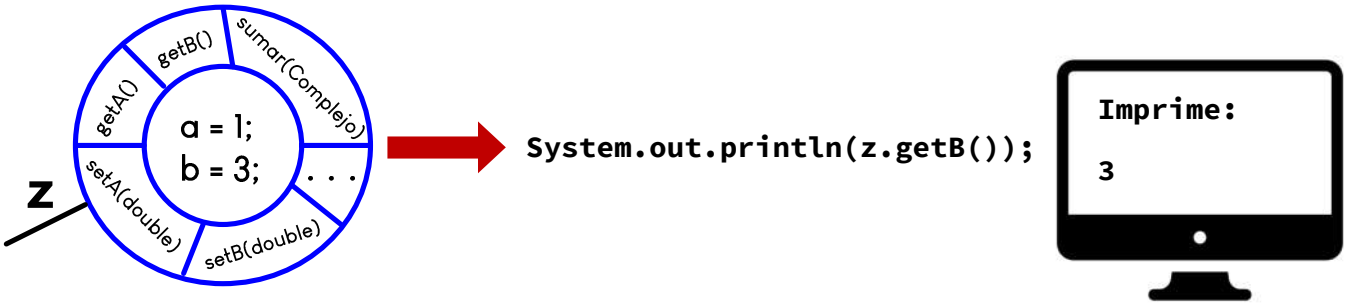
Regresando al ejemplo de los números Complejos, vamos a suponer que se requiere construir el objeto **Complejo z**, correspondiente al valor **1 + 3i**:

```
//La parte real del número Complejo es 1, la parte imaginaria es 3
```

```
Complejo z = new Complejo(1,3); // 1 + 3i
```



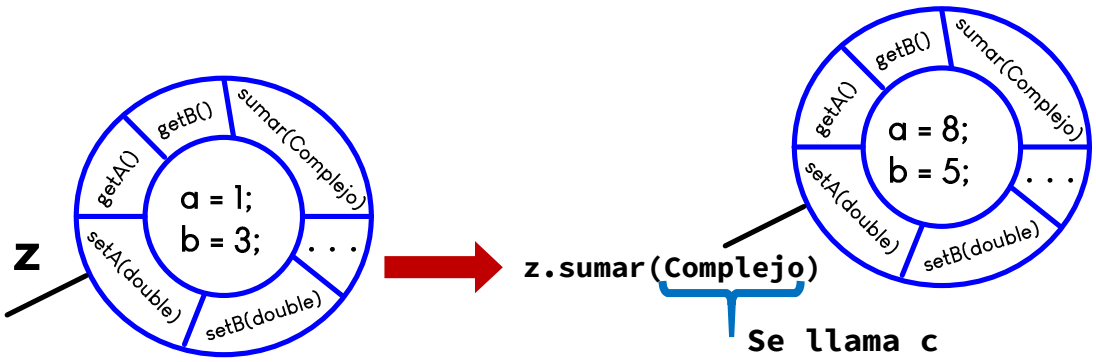
Y queremos, conocer el valor de la parte imaginaria:



Ahora bien, los métodos que tienen que ver con las operaciones de **suma, resta, multiplicación** y **división**, son **operaciones binarias**, es decir, se requieren tener dos números **Complejos** para poder realizar la operación, es decir: $z_1 + z_2 = (a+c)+(b+d)i$, sin embargo, en la documentación se especifica solo un número **Complejo** como parámetro, ¿por qué sucede esto?

Complejo
+ a: double + b: double
+ Complejo() + Complejo(double a, double b) + getA(): double + getB(): double + setA(double a): void + setB(double b): void + sumar(Complejo c): Complejo + restar(Complejo c): Complejo + multiplicar(Complejo c): Complejo + dividir(Complejo c): Complejo + conjugado(): Complejo + equals(Complejo c): boolean + toString(): String

Una de las características que tiene la programación **Orientada a Objetos** es que se pueden programar en cualquier lugar los métodos y se puede por ende, invocar en cualquier lugar, la única regla que se debe cumplir es que**, cuando se utilice el método, éste ya debe estar creado**, para que no tengamos ningún problema en ejecución. Si retomamos la abstracción que de objetos que hemos venido trabajando, vamos a suponer que al **número z** (que ya se tiene definido), queremos sumar el número **Complejo `8 + 5i`**:



Como se puede observar, para poder invocar cualquier método, se requiere tener una referencia, apuntando a un objeto creado y dado que este objeto tiene definida una parte real y una parte imaginaria, se puede aprovechar de este estado, para solo necesitar de un número complejo adicional para poder realizar la operación indicada.

De acuerdo a la definición de **suma** de números complejos ($z_1 + z_2 = (a+c)+(b+d)i$), para el **Complejo suma**, se requiere definir, una **parte real** y una **parte imaginaria** (el conjunto es cerrado):

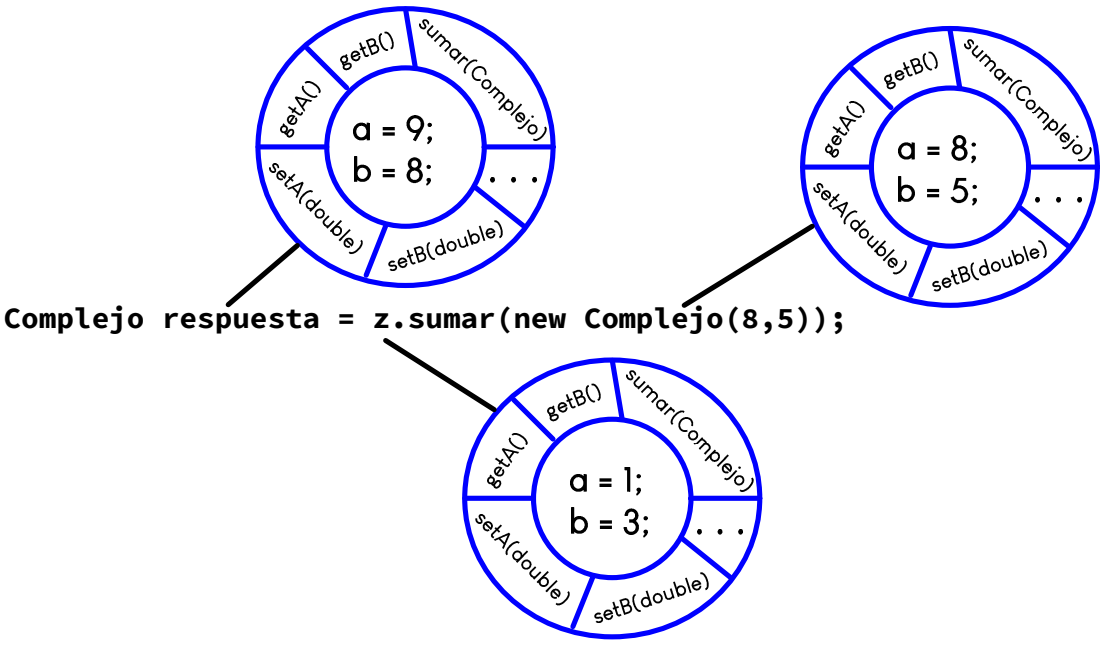
- **Parte real: parte real** de **z_1** + **parte real** de **z_2** y en nuestro ejemplo sería, **parte real** del número **Complejo que invoca** más la **parte real** del número **Complejo** que está llegando como **parámetro**, es decir, **`a + c.a`**. Como se observa, la **invocación** de la parte real del número **Complejo** que está invocando no requiere pasar por el método **`getA()`**, ya que el objeto está encapsulado y la manipulación es desde el interior del objeto, así que la invocación es transparente y está protegida. Ojo: **un usuario externo no puede hacer dicha invocación, para poder invocar la parte real de cualquier objeto es obligatorio hacerlo a partir del método get correspondiente**.
- **Parte imaginaria: parte imaginaria** de **z_1** + **parte imaginaria** de **z_2** y en nuestro ejemplo sería, **parte imaginaria** del número **Complejo** que invoca más la **parte imaginaria** del número **Complejo** que está llegando como **parámetro**, es decir, **`b + c.b`**. Como se observa, la invocación de la parte imaginaria del número Complejo que está invocando no requiere pasar por el método **`getB()`**, ya que el objeto está encapsulado y la manipulación es desde el interior del objeto, así que la invocación es transparente y está protegida. Ojo: **un usuario externo no puede hacer dicha invocación, para poder invocar la parte real de cualquier objeto es obligatorio hacerlo a partir del método get correspondiente**.

A partir de estos comentarios, la programación del **método suma** podría quedar de la siguiente forma:

```
public Complejo sumar(Complejo c){
   double r,i;
   r = a + c.a;
   i = b + c.b;
   return new Complejo(r,i);
}
```

Una vez que se han definido por separado la **parte real** e **imaginaria** del objeto **Complejo**, resultante de la suma, se requiere **devolver la respuesta** de la operación, sin embargo, el método sumar no tiene la responsabilidad de manejar la respuesta, sino de solo devolverla. Por esta razón, la respuesta del método **no requiere construir el objeto completo** (referencia + objeto), sino **solo el objeto** en si (sin la referencia). Por esta razón, en el **`return`** solo se tiene la creación del objeto, sin ninguna referencia.

Aquí surge la pregunta, si solo se devuelve el objeto, este no se va a poder utilizar (de acuerdo a lo que se comentó líneas arriba), entonces, **¿cómo se va a utilizar el objeto?** La respuesta es muy simple: será responsabilidad del programa en dónde se haga uso del método sumar, quien deberá crea la referencia requerida para poder utilizar el objeto resultante de la operación. En este caso, el responsable será en este caso, la clase que modelará la **Calculadora**, por ejemplo:



Es a partir de esta idea es que se definen las demás operaciones, no se debe olvidar que el número **Complejo** que invoca, utiliza los atributos **a** y **b** (directamente), mientras que el segundo número **Complejo** (el parámetro) toma sus atributos como **`c.a`** y **`c.b`**:

- **Resta.** $z_{invoca} - z_{parámetro} = (a - c.a) + (b - c.b)i$
- **Multiplicación.** $z_{invoca} \bullet z_{parámetro} = (a * c.a - b * c.b) + (a * c.b + b * c.a)i$
- **División.** $\frac{z_{invoca}}{z_{parámetro}} = \frac{(ac.a + bc.b)}{c.a^2 + c.b^2} + \frac{(bc.a - ac.b)}{c.a^2 + c.b^2}i$