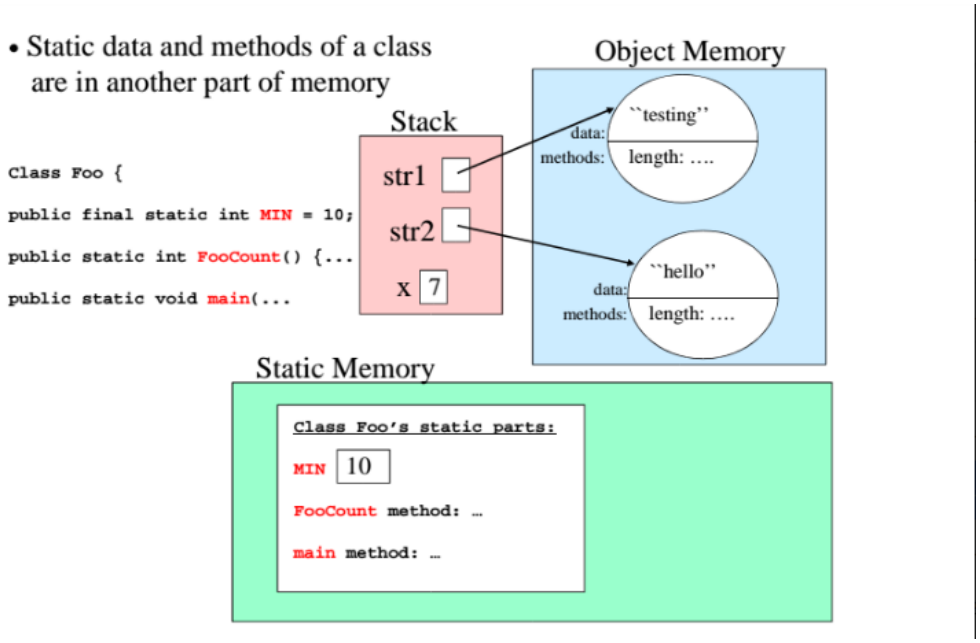
Recuerda gráfico que vimos en la primera evaluación al respecto de gestión de memoria en Java



ATRIBUTOS STATIC

Un atributo o variable static es una variable miembro que no se asocia a un objeto (instancia) de una clase, sino que se asocia a la clase misma; no hay una copia del dato para cada objeto sino una sola copia que es compartida por todos los objetos de la clase.

Todos los objetos de la misma clase pueden acceder a los datos static, pero incluso, aunque la clase no tenga objetos, estos datos existen y son utilizables a través del nombre de la clase. Como cabría esperar, los datos static no se almacenan con los objetos si no en una zona de memoria especial para dichos datos.

Desde este punto de vista, un atributo puede ser:

- Estático, *static* . También se les llama *variables de clase*, ya que no son de un objeto en concreto.
- no estáticos también se llaman *variables instancia*.

¿Una variable local se puede definir como static?

¡NO!. Recuerda que las variables locales son variables que sólo "viven en la pila" mientras se ejecuta un método y no tiene sentido que sean static, ya que el "alcance de vida" de una variable static es mientras la clase esté cargada en memoria. Si quieres una variable static tienes que declararla como atributo.

Ejemplo: observa el error del compilador si queremos calificar una variable local como static.

```
class Unidad5{
   public static void main(String args[]){
    int x =1; //o.k.
    static int y =2;//error
  }
}
```

Da error porque static indica al compilador que un atributo se cree en la zona static "verde", pero *y* no es un atributo, es una var local y se crea siempre en la pila

UN RESUMEN DE TERMINOLOGÍA DE "VARIABLES JAVA":
Una variable puede ser:

- local(se almacena en la pila)
- miembro (atributo)
  - de instancia(no static) (se almacenan dentro de un objeto)
  - de clase (static) (se almacenan en zona de datos static, fuera de todo objeto)

Ejemplo: Una variable static para llevar la cuenta de los objetos creados de una clase.

```
class Cohete{
   static int numCohetes=14;
   String nombre;
   Cohete(String nombre){
      numCohetes++;
      this.nombre=nombre;
   }
 }
class Unidad5 {
   public static void main(String[] args) {
      System.out.println(" total de cohetes "+Cohete.numCohetes);
   }
}
```

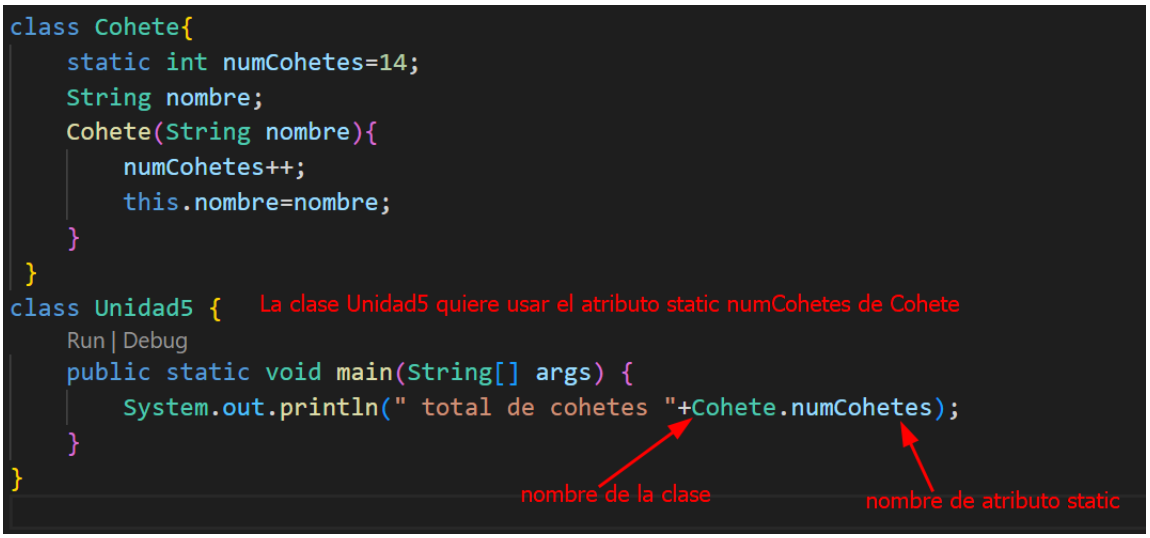
Observa que no hemos creado ningún objeto (no hay ningún new) y numCohetes vale 14, el valor inicial, por tanto:

- Al no haber hecho ningún new, nunca se ejecutó el constructor cohete pero en cambio la variable numCohetes tiene su valor inicial: 14.
- Demostramos que las variables static se inicializan al cargar la clase en memoria, y esto sucede antes de haber creado (new) algún objeto, y además no se tiene porque crear ningún objeto para trabajar con estas variables.

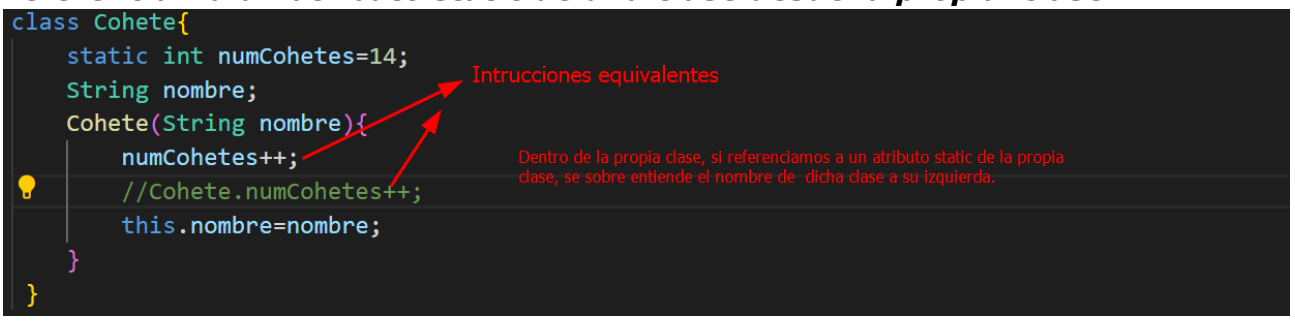
Referenciar a un atributo static de una clase desde otra clase.
Simplificando, porque existen realmente detalles adicionales:

Para un atributo instancia: varReferencia.atributo

Para un atributo static: NombreDeClase.atributo



Referenciar a un atributo static de una clase desde la propia clase.



Ejercicio U5_B1_E1

Añade a la clase Punto la capacidad de controlar el número de objetos Punto creados. Observa que es una propiedad del colectivo de Puntos no de un punto individual, por tanto requerimos una variable de clase.

```
class Punto {
   int x , y ;

   Punto ( int x, int y ) {
      this.x = x ;
      this.y = y;

   }
}
```

MÉTODOS STATIC

Idem atributos static.

Ejemplo:

```
class Cohete{
   static int numCohetes=14;
   static int totalInversionCohetes(int precioMedioCohete){
      return Cohete.numCohetes*precioMedioCohete;
   }
   String nombre;
   Cohete(String nombre){
      numCohetes++;
      this.nombre=nombre;
   }
}

class Unidad5 {
   public static void main(String[] args) {
      System.out.println("cantidad cohetes : "+Cohete.numCohetes);
      System.out.println(" inversion en cohetes: "+Cohete.totalInversionCohetes(100));
   }
}
```

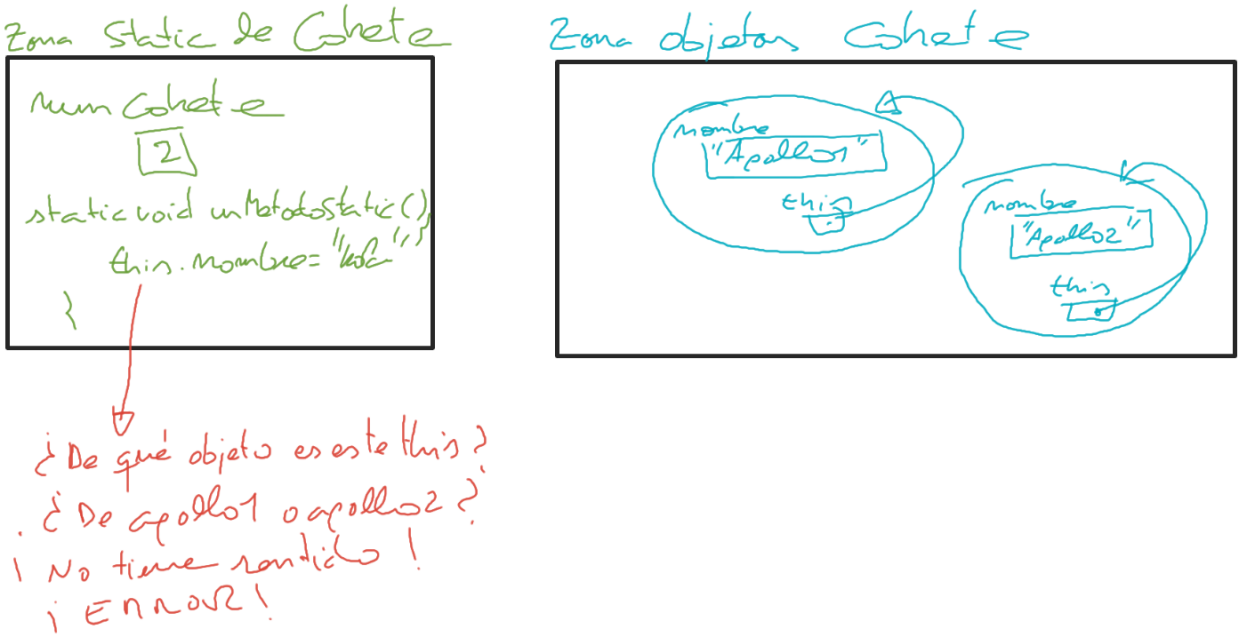
los métodos static no pueden usar la referencia this

Si piensas en el gráfico inicial, this es una variable de la zona azul, cada objeto tiene su variable this. Dentro del código de un método static no tiene sentido la palabra *this*, ya que *this* almacena la dirección del objeto actual y un método static no

pertenece a ningún objeto, de hecho, recuerda que se puede usar un método static sin que exista ningún objeto de su clase.

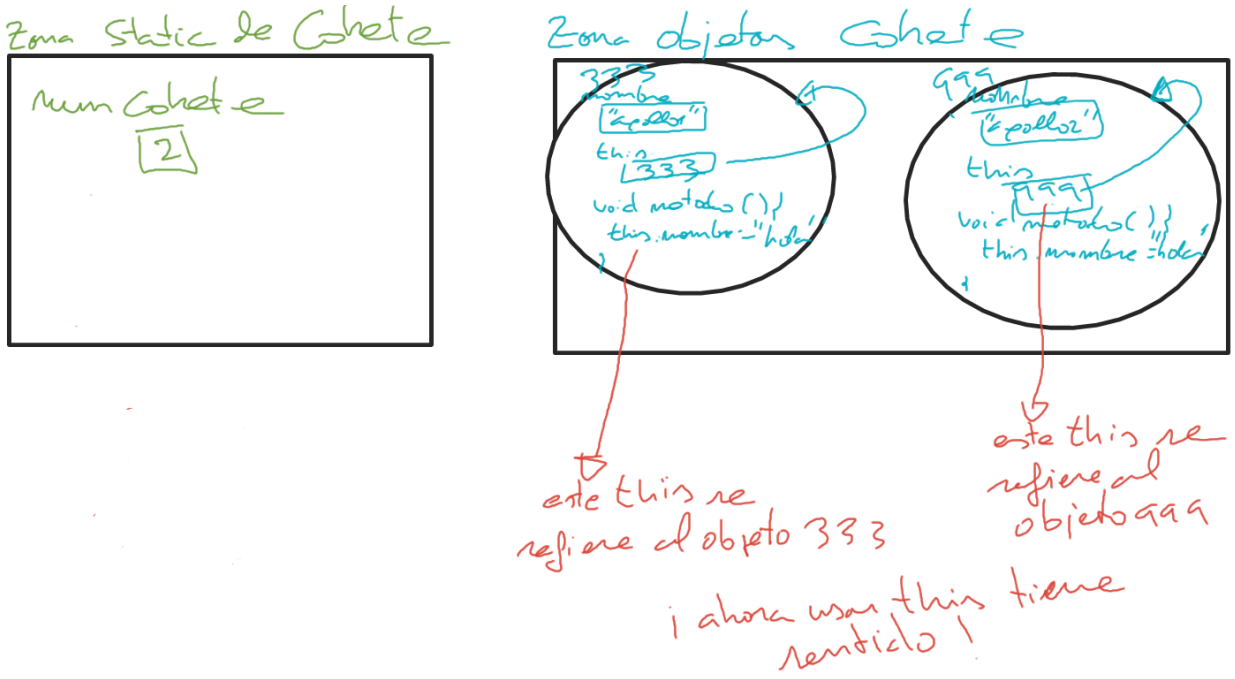
Ejemplo: Compramos que dan error los intentos de acceder desde unMetodoStatic() al atributo nombre que no es static

```
class Cohete{
    static int numCohetes=14;
    String nombre;//algo que pertenece a un objeto concreto
    static void unMetodoStatic(){//ieste método no pertenecerá a ningún objeto!
        this.nombre="hola";//falla por que este código está en la zona verde y no conoce ningún this
        nombre="adios"; //falla porque nombre no es static y esto realmente es this.nombre
    }
    Cohete(String nombre){
         numCohetes++;
        this.nombre=nombre;
    }
 }
class Unidad5 {
    public static void main(String[] args) {
        Cohete c1=new Cohete("Apollo 1");
        Cohete c2=new Cohete("Apollo 2");
    }
}
```



¿Cómo sería el mapa en memoria si borro la palabra static de la cabecera de la definición del método anterior?

El método pasa a ser un método instancia y se almacena dentro de los objetos. Ahora si es correcto usar this.



Lo que sí que puede hacer un método static es acceder a algo de un objeto de su clase u otra si tiene su referencia(excepto a la var this que sólo la conoce el propio objeto)

```
class Cohete{
    static int numCohetes=14;
    String nombre;//algo que pertenece a un objeto concreto
    static void unMetodoStatic(){
        //este método almacenado en la zona static con este código
        //tiene una referencia de un cohete y puede acceder a su nombre
        Cohete miCohete= new Cohete("fogete 3");
        miCohete.nombre="fogete 333";

    }
    Cohete(String nombre){
         numCohetes++;
        this.nombre=nombre;
    }
 }
class Unidad5 {
    public static void main(String[] args) {
        System.out.println(" total de cohetes "+Cohete.numCohetes);
    }
}
```

RESUMEN DE LO MÁS IMPORTANTE:

- Los métodos static no pueden usar la referencia this
- Si desde un método static quiero acceder a algo de un objeto, necesito conocer su referencia, y de nuevo, no vale this
- Usa la sintaxis NombreClase.miembroStatic para acceder a un miembro static

Ejercicio U5_B1_E2

Soluciona el error del siguiente código

```
class Unidad5 {
    int masmas(int x){
        return ++x; //ojo x++ no devuelve valor incrementado
    }
```

```
    public static void main(String[] args) {
        int x=8;
        System.out.println(masmas(x));
    }
}
```

Ejercicio U5_B1_E3

Observa el siguiente ejemplo:

```
class Racional{
    int numerador;
    int denominador;
    Racional(int numerador, int denominador){
        this.numerador=numerador;
        this.denominador=denominador;
    }
    void multiplicar(Racional r1, Racional r2){
        this.numerador=r1.numerador*r2.numerador;
        this.denominador=r1.denominador*r2.denominador;
    }
}
class Unidad5{
    public static void main(String[] args) {
        Racional r1=new Racional(3,4);
        Racional r2=new Racional(1,2);
        Racional r3=new Racional(1,1);
        r3.multiplicar(r1, r2);
        System.out.println("MUTIPLICACIÓN DE NÚMEROS RACIONALES");
        System.out.println("r1 vale: "+r1.numerador+"/"+r1.denominador);
        System.out.println("r2 vale: "+r2.numerador+"/"+r2.denominador);
        System.out.println("r3 vale: "+r3.numerador+"/"+r3.denominador);
    }
}
```

Modifica el ejemplo anterior , de forma que ahora la multiplicación se puedan usar de la siguiente forma:

r3=Racional.multiplicar(r1,r2);

Ejercicio U5_B1_E4: Observa el siguiente ejemplo ya hecho.

```
class Potencia{
   int elevar(int base,int exponente){
        int resultado=1;
        for(;exponente>0;exponente--){
            resultado=resultado*base;
        }
        return resultado;
   }
}
class Unidad5{
    public static void main(String[] args) {
        Potencia p = new Potencia();
        System.out.println(p.elevar(2,1));
        System.out.println(p.elevar(5,3));
        System.out.println(p.elevar(9,0));
    }
}
```

Reescribe el ejemplo anterior para que funcione el siguiente main:

```
public static void main(String[] args) {
	System.out.println(Unidad5.pot(2,1));
	System.out.println(pot(5,3));
	System.out.println(new Unidad5().pot(9,0));//
}
```

REFLEXIONES SOBRE STATIC

Una reflexión muy importante: Muchos programadores no están de acuerdo con el uso de los métodos estáticos, ya que dicen que se alejan de la programación orientada a objetos. Sin embargo, hoy en día en muchos contextos, ya no se considera "pecado" usar static si se hace correctamente. Sobre esto puedes leer en la web por ejemplo:

- http://blog.koalite.com/2016/11/mitos-y-leyendas-sobre-metodos-estaticos/
- O este enlace también está bien

https://www.arquitecturajava.com/static-method-vs-instance-method-y-su-uso-correcto/

Una conclusión después de leer los artículos anteriores es que que usando static perdemos "el poder de la POO", como por ejemplo la capacidad de polimorfismo. Otra cosa es que para lo que estás haciendo en un momento dado no te interese para nada ese poder.

Y tras la conclusión anterior, seguro que opinas que la gran mayoría de las soluciones de acepta el reto no precisan realmente instanciar ningún objeto, ya que con un main() y quizá con la ayuda de 1 o 2 métodos static es suficiente. Hay situaciones más ambiguas, por ejemplo la clase InterfaceConsola del SieteYMedia e incluso la propia clase SieteYMedia ¿podrían basarse todas ellas en métodos static?. Bueno … depende del contexto, si quieres que evolucione y por ejemplo se ejecute dentro de un hilo, o si quieres que sea parte de una jerarquía y tenga métodos sobreescritos es necesario crear instancia pero para algo sencillo no necesariamente … Todo depende del contexto. En cualquier caso, nosotros para aprender POO nos obligamos a menudo a evitar static.

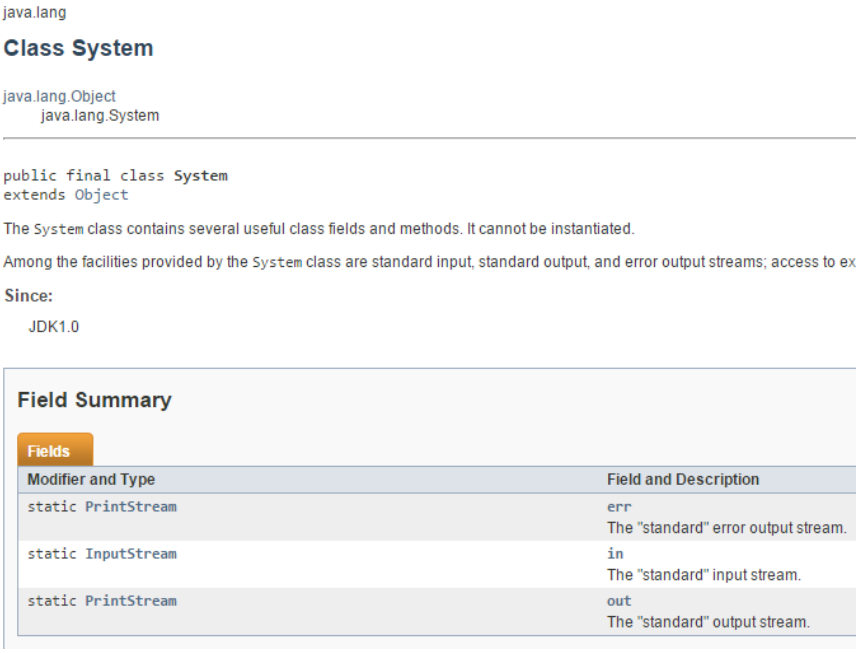
Una reflexión sobre la clase System

Ahora estamos en disposición de entender algo mejor
System.out.println();

- Está claro que *out* es "algo" static sea atributo o método ya que lo invocamos a través del nombre de su clase, no de un objeto.
- Además "ese algo", si observo la sintaxis
  out.println()

no va a ser un atributo de tipo primitivo ni un método, sino una variable referencia ya que observo el operador punto, lo que delata que se trata de:
variableobjeto.método.

Y efectivamente, si consulto la API java observo



java.lang

Class System

java.lang.Object
    java.lang.System

```
public final class System
extends Object
```

The System class contains several useful class fields and methods. It cannot be instantiated.

Among the facilities provided by the System class are standard input, standard output, and error output streams; access to ex

Since:

JDK1.0

Field Summary

Fields

Modifier and Type	Field and Description
static PrintStream	err The "standard" error output stream.
static InputStream	in The "standard" input stream.
static PrintStream	out The "standard" output stream.

La clase System tiene tres atributos (Field) uno de ellos se llama *out* y observo que es static y de tipo PrintStream es decir una referencia a un objeto de tipo PrintStream
Si consulto los métodos de la clase PrintStream veo que entre otros, hay "muchos" println() ya que es un método altamente sobrecargado.

Otra reflexión: ¡el gran main() es static!
Una explicación como dios manda en
http://javarevisited.blogspot.com.es/2011/12/main-public-static-java-void-method-why.html

Un resumen: es la forma más simple de que la máquina java comience a ejecutar un programa. La alternativa sería evidentemente que en lugar de ejecutar un main() creará una instancia de la clase por la que queremos comenzar el programa, para ello debería utilizar un constructor de la clase y dado que un constructor puede tener varias versiones, esto lo complica todo.

EL API Y LOS MÉTODOS STATIC

A pesar de que es saltarse a la torera el fundamento más importante de la POO (un programa es un conjunto de objetos que se envían mensajes), si se usan comedidamente y en situaciones concretas son útiles y pueden mejorar el código, y si esto es cierto, sin duda encontraremos ejemplo de esto en el API.

La clase Math tiene métodos static

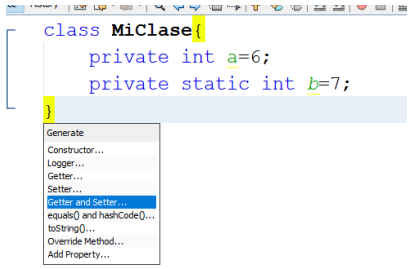
Muchas de las clases predefinidas de java incluyen a menudo métodos static ya que hay muchas situaciones en las que crear un objeto de una clase es una complicación innecesaria. Por ejemplo, la clase Math tiene asociados un montón de métodos y todos ellos son static, por ejemplo :

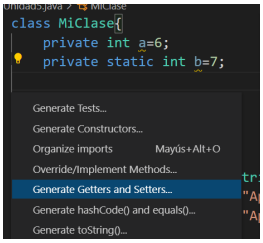
- static double sqrt(double a) ó
- static double pow(double a, double b)

Ejercicio U5_B1_E5: Utilizando los métodos anteriores de la clase Math, imprime la raiz cuadrada de 25 y el valor de la expresión $2^{8.}$

En el programa anterior piensa que ventajas te habría aportado crear un hipotético objeto Math.

Ejercicio U5_B1_E6:
Lógicamente si un atributo es static su get/set correspondiente también debe serlo. Investiga cómo genera el IDE un set/get para un atributo static. Por ejemplo para la siguiente clase
netbeans



VSC


Otros ejemplos de métodos static en API
Recuerdas Integer.parseInt(), Double.parseDouble(), ….

El método join de la clase String

```
public class Unidad5{
   public static void main (String[]args){
       String union = String.join(":", "margarita ", "azalea","rosa","alstroemeria");
       System.out.println(union);
    }
}
```

El método showMessageDialog() de JOptionPane

```
import javax.swing.JOptionPane;
public class Unidad5{
   public static void main (String[]args){
      JOptionPane. showMessageDialog(null,"hola mundo");
    }
}
```

Y un largo etc.

la combinación static final
supongamos que una variable de instancia constante

```
class MiClase{
final int MI_CONSTANTE=3;
…
}
```

final indica que MI_CONSTANTE se inicializa a 3 y luego jamás podrá ser modificada
Si creo objetos de la clase anterior
x= new Miclase(); y= new Miclase(); z= new Miclase();
Ocurre que para los objetos referenciados por x, y, z MI_CONSTANTE siempre vale 3 y no es posible modificarla. Es decir, siempre ocurre lo mismo para todos los objetos, por tanto, sacando factor común … ¡la convierto en variable de clase añadiéndole static !

conclusión: los atributos con final, no es obligatorio, pero suelen ser también static

CARGA DE UNA CLASE

Ahora que vimos el concepto de static podemos ver de forma más completa que pasa cuando se invoca a una clase por primera vez:

- Al crear el primer objeto de la clase o bien al utilizar al primer método o variable static se localiza la clase en el disco (fichero *.class) y se carga en memoria.
- Se ejecutan los inicializadores static (sólo una vez).
- Para crear un nuevo objeto:
    - se comienza reservando memoria
    - se da valor por defecto a las variables miembro de los tipos primitivos
    - se ejecutan los inicializadores explícitos o de objeto
    - se ejecutan los constructores

INICIALIZADORES STATIC (no los usamos, leer simplemente por curiosidad)

Son bloques de código de la forma

```
static{
instrucciones;
}
```

Sus instrucciones por sentido común deben dedicarse a inicializar atributos static aunque pueden hacer otras cosas como en el ejemplo de abajo que usamos println() por motivos didácticos

```
class B {
   static int variableStatic=5;
   static {
      System.out.println("valor de variableStatic al ppio de inicializador static "+variableStatic);
      variableStatic=99;
      System.out.println("valor de variableStatic al final de inicializador static "+variableStatic);
   }

}

class Unidad5 {
   public static void main(String[] args) {
      //new B();
      System.out.println("Valor de variable Static con clase ya cargada "+B.variableStatic);
      B.variableStatic=115;
      System.out.println("Valor de variable Static al final de main "+B.variableStatic);

   }
```

```
}
```

Cuando en Unidad5 se invoca por primera vez a *B.variableStatic* se produce el proceso de carga de la clase B en memoria

Observa que el inicializador static también se invoca al crear un objeto, prueba ahora el siguiente main

```
class Unidad5 {
   public static void main(String[] args) {
      new B();
    }
}
```

INICIALIZACIÓN DE UN OBJETO

Hay tres mecanismos y que se ejecutan en el siguiente orden:

1. En la declaración del atributo
2. En bloque de inicialización **(no los usamos, leer por curiosidad)**
3. En constructor.

El método 2 no es muy usado. Observa en el ejemplo como el bloque inicializador se ejecuta antes que el constructor ya que el constructor imprime el valor de c esperado.

```
class B {

   int a = 15;
   int b = 10;
   int c;

   public B() {
      System.out.println("Valor de c:" + c);
   }

   { //bloque de inicialización
      c = a + b;
   }
}

class Unidad5 {
   public static void main(String[] args) {
      new B();
   }
}
```

Un ejemplo con ambos tipos de inicializadores(static y de instancia)

Descomenta en el main una u otra instrucción para razonar el proceso

```
class B {
   static int variableStatic=5;
   static {
      variableStatic=99;
      System.out.println(variableStatic);
   }
   public int a = 15;
   public int b = 10;
   public int c;

   public B() {
      System.out.println("Valor de c:" + c);
   }
```

```
   { //bloque de inicialización
      c = a + b;
   }
}

class Unidad5 {
   public static void main(String[] args) {
        //la carga de la clase en memoria se dispara cuando aparece una instruccion que crea un objeto o invoca a algo
static
      //new B();
      //B.variableStatic=34;
   }
}
```