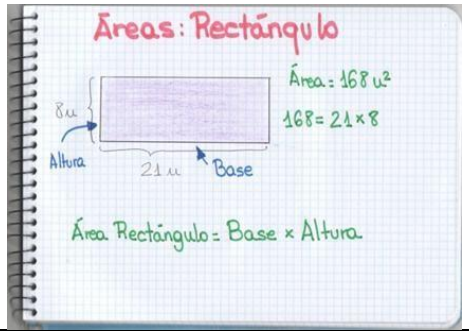
Las variables miembro

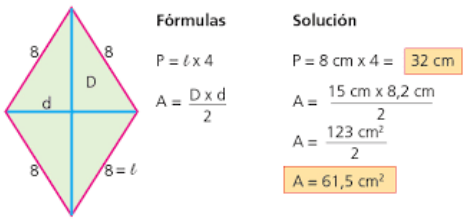
En Java es posible definir variables fuera del ámbito de un método. A estas variables se las denomina variables miembro (o también atributos como se verá más adelante, si cumplen ciertos requisitos de la Programación Orientada a Objetos). La característica principal de este tipo de variables (además de que se definen fuera de los métodos) es que son variables globales dentro de una clase. Por lo tanto, pueden ser utilizadas por todos los métodos de esa clase.
La forma en las que se las define es similar a la de una variable común y corriente, con la particularidad de que se les debe asignar un modificador de acceso. Entonces la nomenclatura para definir una variable miembro es

modificador_de_acceso tipo_de_dato nombre_variable [= asignación_valor];

En la siguiente sección se verán los modificadores de acceso, por ahora para el siguiente ejemplo usaremos un modificador de acceso conocido, debido a que se lo ha usado en los ejemplo de definición de métodos: el modificador de acceso `public`.

Ejemplo: Defina las clases denominadas `Rectangulo` y `Rombo` con las variables miembro necesarias que puedan utilizarse para calcular el perímetro y el área respectivamente, con los siguientes datos





A continuación, se define la clase Rectángulo

```
1  package ar.edu.unju.fi.modelo;
2
3  public class Rectangulo {
4      public  double base;
5      public double altura;
6
7      public double calcularArea() {
8          return base * altura;
9      }
10
11     public double calcularPerimetro() {
12         return 2*(base+altura);
13     }
14
15 }
16
```

A

Observe que en A se han definido 2 variables miembros. Cada una de ellas tiene modificador de acceso `public`. Observe además que son variables que se comportan como globales dentro de la clase. Por ese motivo pueden ser utilizadas por cualquiera de los métodos.

En este caso se han definido dos métodos donde en particular ambos utilizan ambas variables, por tanto, NO ES NECESARIO enviar los datos por medio de parámetros

Entonces, ¿cómo se les asigna valores para poder asignarles valores?

Para poder asignar y obtener los valores de una variable miembro no estática (luego veremos que significa este término) y publica, primero se debe instanciar un objeto de esa clase. Por ejemplo:

```
1  package ar.edu.unju.fi.principal;
2
3  import ar.edu.unju.fi.modelo.Rectangulo;
4
5  public class Principal {
6
7      public static void main(String[] args) {
8
9          // Se crea la instancia de Rectangulo
10         Rectangulo unRectangulo = new Rectangulo();
11
12         // se asignan valores a las variables miembro
13         // de unRectangulo
14         unRectangulo.altura = 8d;
15         unRectangulo.base = 21d;
16
17         // se visualizan los valores de las variables miembro
18         // de unRectangulo
19         System.out.println("Base:"+unRectangulo.base+"m");
20         System.out.println("Altura:"+unRectangulo.altura+"m");
21
22         // se calculan el área y perimetro de unRectangulo
23         double areaRectangulo = unRectangulo.calcularArea();
24         double perimetroRectangulo = unRectangulo.calcularPerimetro();
25
26         //se visualizan el área y perímetros calculados
27         System.out.println("La superficie vale: "+areaRectangulo+"m2");
28         System.out.println("El perimetro vale: "+perimetroRectangulo+"m");
29
30     }
31
32 }
```

En las líneas 14 y 15 se asignan valores a las variables miembro `altura` y `base`. Previamente se debió instanciar un objeto de la clase `Rectángulo`. La asignación de una variable miembro no estática y publica es similar a la asignación de cualquier variable, con la salvedad de que previamente se debe indicar el objeto a la que pertenece y luego mediante un "." se puede indicar cual variable miembro se está operando.

El valor de una variable miembro se obtiene de manera similar (líneas 19 y 20)

Para el Rombo quedaría de la siguiente manera

```
1  package ar.edu.unju.fi.modelo;
2
3  public class Rombo {
4      public double lado;
5      public double diagonalMayor;
6      public double diagonalMenor;
7
8
9      public double calcularPerimetro() {
10         return lado*4;
11     }
12
13     public double calcularArea() {
14         return (diagonalMayor*diagonalMenor/2d);
15     }
16
17 }
```

Observe que en este caso las variables miembro no se usan en ambos métodos, pero de todas formas estarían "*disponibles*" de ser necesario.

Como se verá más adelante, esta característica de que se puedan definir variables miembro responde a una característica fundamental del Paradigma Orientado a Objetos, las clases son contenedores de datos y métodos, a diferencia del Paradigma Estructurado donde se separan los programas de los datos. Esto de alguna manera define las variables miembros permitidas que tendrá un objeto.

Cada objeto, es decir cada instancia concreta de una clase, tiene su propia copia de las variables miembro. Las variables miembros de una clase (también denominadas atributos) pueden ser de tipos primitivos (boolean, int, long, double, ...) o referencias a objetos de otra clase.

Un aspecto muy importante para el correcto funcionamiento de los programas es que no haya datos sin inicializar. Por eso las variables miembros de tipos primitivos se inicializan siempre de modo automático, incluso antes de llamar al constructor (false para `boolean`, el carácter nulo para `char` (codigo Unicode '\u0000') y cero para los tipos numéricos). De todas formas, lo más adecuado es inicializarlas en el constructor (esto se verá más adelante).

También pueden inicializarse explícitamente en la declaración, como las variables locales, por medio de constantes o llamadas a métodos. Por ejemplo

```
public class Alumno
{
    int edad = 18;
}
```

Las variables miembros se inicializan en el mismo orden en que aparecen en el código de la clase. Esto es importante porque unas variables pueden apoyarse en otras previamente definidas. Cada *objeto* que se crea de una clase tiene *su propia copia* de las variables miembro.

Modificadores de acceso

Los modificadores de acceso se aplican principalmente a las variables miembro y métodos. Permiten definir la visibilidad y acceso a los mismos. Los modificadores de acceso determinan desde qué clases se puede acceder a un determinado elemento (variable miembro o método). En JAVA existen 4 tipos: `public`, `private`, `protected` y el tipo por defecto, que no tiene ninguna palabra clave asociada, pero se suele conocer como `default` o ***package-private***.
Si no especificamos ningún modificador de acceso se utiliza el nivel de acceso por defecto

- Nivel de acceso por defecto: consiste en que el elemento puede ser accedido sólo desde las clases que pertenezcan al mismo paquete.
- Nivel de acceso `public`: permite a acceder al elemento desde cualquier clase, independientemente de que esta pertenezca o no al paquete en que se encuentra el elemento.
- Nivel de acceso `private`: es el modificador más restrictivo y establece que los elementos que lo utilizan sólo pueden ser accedidos desde la clase en la que se encuentran. Este modificador sólo puede utilizarse sobre los miembros (atributos o métodos) de una clase y sobre interfaces y clases internas, no sobre clases o interfaces de primer nivel, dado que esto no tendría sentido. El modificador `private` convierte los elementos en privados para otras clases, no para otras instancias de la clase. Es decir, un objeto de una determinada clase puede acceder a los miembros privados de otro objeto de la misma clase.
- Nivel de acceso `protected`: indica que los elementos sólo pueden ser accedidos desde su mismo paquete (como el acceso por defecto) y desde cualquier clase que extienda la clase en que se encuentra, independientemente de si esta se encuentra en el mismo paquete o no. Este modificador, como `private`, no tiene sentido a nivel de clases o interfaces no internas.

Los distintos modificadores de acceso quedan resumidos en la siguiente tabla:

Tabla 1. Visibilidad y acceso de las variables miembro y los métodos según el modificador de acceso

Visibilidad	Public	Protected	Default	Private
Desde la misma Clase	SI	SI	SI	SI
Desde cualquier Clase del mismo Paquete	SI	SI	SI	NO
Desde una SubClase del mismo Paquete	SI	SI	SI	NO
Desde una SubClase fuera del mismo Paquete	SI	SI, a través de la herencia	NO	NO
Desde cualquier Clase fuera del Paquete	SI	NO	NO	NO

Las cadenas de Caracteres

Las variables de tipo `String` permiten manipular cadenas de caracteres. Vamos a ver cómo realizar las operaciones más corrientes en las cadenas de caracteres.
El método más sencillo para crear una cadena de caracteres consiste en considerar el tipo `String` como un tipo primitivo del lenguaje y no como un objeto. En este caso, la asignación de un valor a la variable va a provocar la creación de una instancia de la clase `String`. También es posible crear una cadena de caracteres como un objeto utilizando el operador `new` y alguno de los constructores disponibles en la clase `String`. El ejemplo de código siguiente presenta las dos soluciones:

```
String cadena1="eni";
String cadena2=new String("eni");
```

Después de su creación, no se puede modificar una cadena de caracteres. La asignación de otro valor a la variable provoca la creación de una nueva instancia de la clase `String`. La clase `String` contiene numerosos métodos que permiten manipular cadenas de caracteres. Al utilizarlos, se puede tener la sensación de que el método modifica el contenido de la cadena inicial, pero, en realidad, el método devuelve una nueva instancia que contiene el resultado.

Asignación de valores a una cadena

Para asignar un valor a una cadena es necesario especificarla entre los caracteres " y ". Se plantea un problema si se quiere que el carácter " forme parte de la cadena. Para que no se interprete como carácter de principio o de fin de cadena, hay que protegerlo por una secuencia de escape como en el ejemplo siguiente:

```
String Cadena;
Cadena=" Dijo: \"¡Basta ya!\"";
System.out.println(Cadena);
```

Los modificadores de acceso se aplican principalmente a las variables miembro y métodos. Permiten definir la visibilidad y acceso a los mismos. Los modificadores de acceso determinan desde qué clases se puede acceder a un determinado elemento (variable miembro o método).

Principales métodos de la clase `String`

Para explicar y ejemplificar los siguientes métodos se utilizarán dos cadenas de caracteres:

Método	Descripción	Ejemplo

charAt()	Para obtener el carácter situado en una posición determinada de una cadena de caracteres, se debe utilizar el método *charAt*() proporcionando como argumento el índice del carácter que se desea obtener. El primer carácter de un String tiene el índice cero.	`cadena1 = "el invierno será lluvioso";`
		`cadena2 = "el invierno será frío";`
		
		`System.out.println`
		`("el tercer carácter de la cadena1 es " +`
		
		`cadena1.charAt(2));`

`length`	No es un método sino la variable miembro
	
	Para determinar la longitud de una cadena se dispone de del atributo `length` de la clase `String`
		`System.out.println`
		`("el tercer carácter de la cadena1 es " +`
		
		`cadena1.charAt(2));`

substring()	Devuelve una sección de una cadena en función de las posiciones inicial y final que se le proporcionan como parámetros. La cadena obtenida empieza por el carácter ubicado en la posición inicial (principio) y termina en el carácter que precede a la posición final.	`System.out.println("un trozo de la cadena1: "`
		
		`+ cadena1.substring(3,11));`
		
		Generará la siguiente salida:
		
		`un trozo de la cadena1: invierno`

equals()	Las cadenas de caracteres son de tipo objeto. Hay que utilizar los métodos de la clase `String` para efectuar comparaciones de cadenas de caracteres. El método *equals*() compara la cadena con la que se pasa como parámetro.
	Devuelve un *boolean* igual a `true` si las dos cadenas son idénticas y por supuesto un *boolean* igual a `false` en caso contrario. El método distingue entre minúsculas y mayúsculas durante la comparación. El método *equalsIgnoreCase*() realiza la misma acción pero sin tener en cuenta esta distinción.
		`if (cadena1.equals(cadena2))`
		`{`
		`  System.out.println("las dos cadenas son idénticas");`
		`}`
		`else`
		`{`
		`  System.out.println("las dos cadenas son diferentes");`
		`}`

compareTo()	Permite realizar una clasificación entre cadenas. Hay que pasar como parámetros la cadena que se debe comparar. Se devuelve el resultado de la comparación bajo la forma de un entero	`if (cadena1.compareTo(cadena2)>0)`
		`{`
		`  System.out.println("cadena1 es superior a cadena2");`
		`}`
		`else`

	inferior a cero si la cadena es inferior a la recibida como parámetro, igual a cero si las dos cadenas son idénticas, y superior a cero si la cadena es superior a la recibida como parámetro.	`if (cadena1.compareTo(cadena2)<0)`
		`{`
		`  System.out.println("cadena1 es inferior a cadena2");`
		`}`
		`else`
		`{`
		`  System.out.println("las dos cadenas son idénticas");`
		`}`

startsWith() y *endsWith*()	Permiten comprobar si la cadena empieza con la cadena recibida como parámetro o si la cadena termina con la cadena recibida como parámetro.	`String nombre="Código.java";`
		
		`if (nombre.endsWith(".java"))`
		`{`
		`  System.out.println("es un archivo fuente java");`
		`}`

trim()	Permite suprimir los espacios en blanco situados delante del primer carácter significativo y después del último carácter significativo de una cadena.	`String cadena="  eni  ";`
		`System.out.println("longitud de la cadena: " + cadena.length());`
		`System.out.println("longitud de la cadena sin espacios: " +`
		`cadena.trim()`
		`.length());`

toUpperCase() y to LowerCase()	Generar cadenas donde todos los caracteres están en mayúscula o minúscula respectivamente.	Todo a mayúsculas:
		
		`System.out.println(cadena1.toUpperCase());`
		
		Todo a minúsculas:
		
		`System.out.println(cadena1.toLowerCase());`

Index Of()	Permite buscar una cadena en el interior de otra. El parámetro corresponde a la cadena buscada. El método devuelve un entero que indica la posición donde se encuentra la cadena o -1 si no se encuentra. Por defecto la búsqueda empieza al principio de la cadena, salvo si utilizamos otra versión del método que recibe dos parámetros, el primer parámetro es, para esta versión, la cadena buscada, y el segundo la posición inicial de la búsqueda.	`String búsqueda;`
		`  int posición;`
		`  búsqueda = "e";`
		`  posición = cadena1.indexOf(búsqueda);`
		`  while (posición > 0)`
		`  {`
		`  System.out.println("cadena encontrada en la posición " + posición);`
		`  posición = cadena1.indexOf(búsqueda, posición+1);`
		`  }`
		`  System.out.println("fin de la búsqueda");`
		
		Que genera el siguiente resultado
		
		`cadena encontrada en la posición 0`
		`cadena encontrada en la posición 7`
		`cadena encontrada en la posición 13`
		`fin de la búsqueda`

Replace()	Busca porciones de cadenas y genera una nueva cadena donde se reemplaza la porción buscada. *replace*() permite especificar una cadena de substitución para la cadena buscada. Recibe dos parámetros: la cadena buscada, la cadena sustituta	`String cadena3;`
		`cadena3= cadena1.replace("invierno", "verano");`
		`System.out.println(cadena3);`
		
		Que genera lo siguiente:
		
		`el verano será lluvioso`

Dar formato a un `String`

El método *format*() de la clase `String` permite evitar largas y complicadas operaciones de conversión y de concatenación. El primer parámetro que recibe es una cadena de caracteres

que indica en qué forma se desea obtener el resultado. Esta cadena contiene uno o varios motivos de formateo representados por el carácter % seguido de un carácter específico que indica en qué forma debe presentarse la información.

Debe recibir, a continuación, tantos parámetros como motivos de formateo. La cadena devuelta se construye remplazando cada uno de los motivos de formateo por el valor del parámetro correspondiente, y el remplazo se realiza en función del orden de aparición de los motivos. La siguiente tabla presenta los principales motivos de formateo disponibles.

Motivo	Descripción
%b	Insertar un booleano
%s	Insertar una cadena de caracteres
%d	Insertar un número entero
%o	Insertar un número entero en representación octal
%x	Insertar un número entero en representación hexadecimal
%f	Insertar un número decimal
%e	Insertar un número decimal representado en formato científico
%n	Insertar un salto de línea

El siguiente es un código ejemplificador del formato de cadenas usando el método *format*()

```
boolean b=true;
   int i=56;
   double d=19.6;
   String s="cadena";
   System.out.println(String.format("booleano: %b %n" +
                    "cadena de caracteres: %s %n" +
                    "entero: %d %n" +
                    "entero en hexadecimal: %x %n" +
                    "entero en octal: %o %n" +
                    "decimal: %f %n" +
                    "decimal en formato científico: %e%n",
                    b,s,i,i,i,d,d));
```

El cual genera la siguiente salida

```
booleano: true
cadena de caracteres: cadena
entero: 56
entero en hexadecimal: 38
entero en octal: 70
decimal: 19,600000
decimal en formato científico: 1,960000e+01
```