Resolución de Problemas y Algoritmos

Modularización

Facultad de Informática
Universidad Nacional del Comahue

2021

Temario

Temario

Diversidad de algoritmos de una misma clase

```
MÓDULO areaCuadrado(REAL lado) RETORNA REAL
        | (* el módulo calcula el area de un cuadrado
        | lado: lado del cuadrado*)
        | REAL area
        | area ← lado * lado
        | RETORNA area
FIN MÓDULO areaCuadrado
```

Diversidad de algoritmos de una misma clase

```
ALGORITMO calculaAreas() RETORNA ∅
        | (* el módulo calcula areas de tres figuras geométricas*)
        | REAL areaFigura
        | REAL lado1, lado2
        | ESCRIBIR(" Ingrese medida de un lado del cuadrado")
        | LEER(lado1)
        | areaFigura ← areaCuadrado(lado1)
        | ESCRIBIR(" El area es : ", areaFigura)
FIN ALGORITMO calculaAreas
```

Diversidad de algoritmos de una misma clase

```
MÓDULO areaRect(REAL ladoMayor, ladoMenor) RETORNA REAL
        | (* el módulo calcula el area de un rectangulo
        | ladoMayor: lado mayor del cuadrado
        | ladoMenor: lado menor del cuadrado*)
        | REAL area
        | area ← ladoMayor * ladoMenor
        | RETORNA area
FIN MÓDULO areaRect
```

Diversidad de algoritmos de una misma clase

```
MÓDULO areaCirculo(REAL radioCirculo) RETORNA REAL
        | (* el módulo calcula el area de un circulo
        | radioCirculo: radio del Circulo*)
        | REAL area
        | area ← PI * radioCirculo * radioCirculo
        | RETORNA area
FIN MÓDULO areaCirculo
```

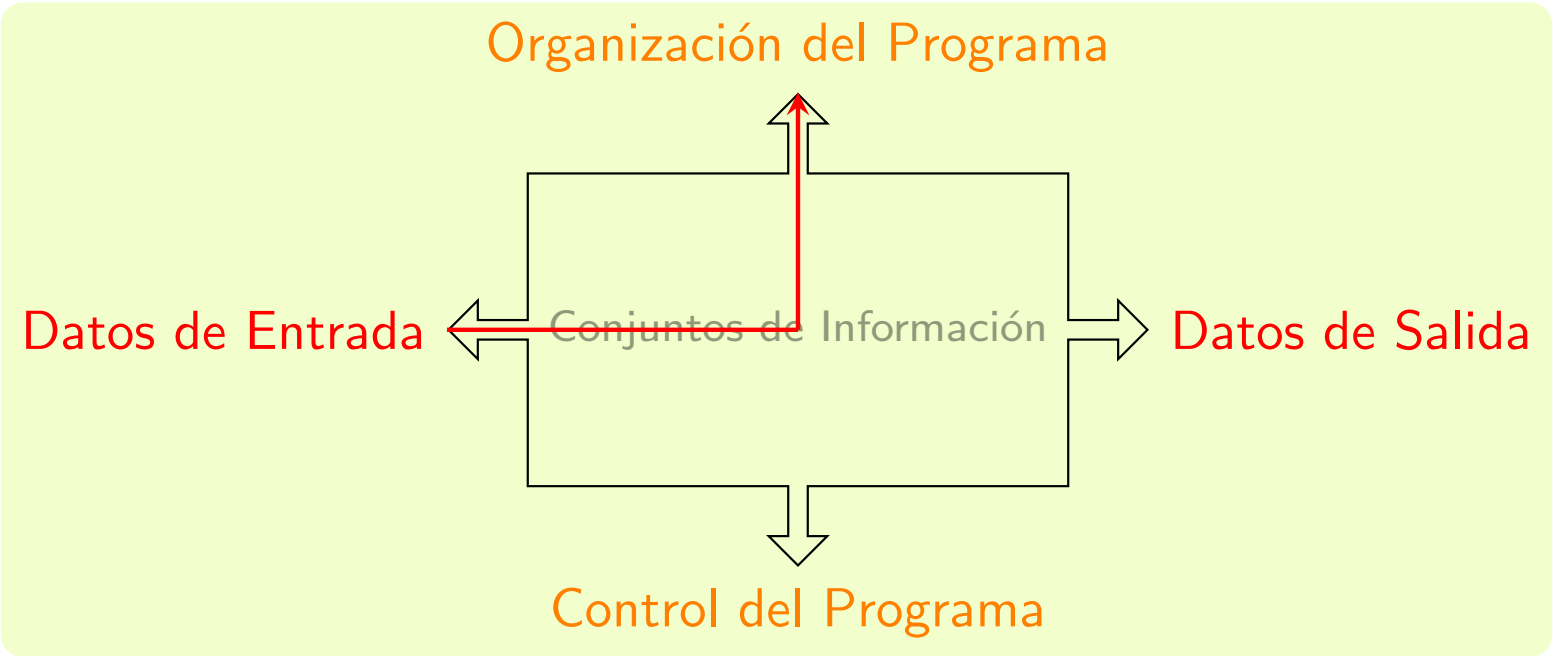
Diversidad de algoritmos de una misma clase

```
ALGORITMO calculaAreas() RETORNA ∅
        | (* el módulo calcula areas de tres figuras geométricas*)
        | REAL areaFigura
        | REAL lado1, lado2
        | ESCRIBIR(" Ingrese medida de un lado del cuadrado")
        | LEER(lado1)
        | areaFigura ← areaCuadrado(lado1)
        | ESCRIBIR(" El area del cuadrado es : ", areaFigura)
        | areaFigura ← areaRect(2.7, 5.3)
        | ESCRIBIR(" El area del rectangulo es : ", areaFigura)
        | areaFigura ← areaCirculo(4.5)
        | ESCRIBIR(" El area del Circulo es : ", areaFigura)
FIN ALGORITMO calculaAreas
```

Introducción.

El uso apropiado de la modularización tiene importantes ventajas:

- Facilita las fases de programación posteriores a la codificación (en especial la fase de pruebas y de mantenimiento).
- Cada módulo debe hacer una única cosa (y de manera genérica).
- Intentar que cada módulo oculte algo (encapsulamiento).
- Su organización esta en relación con un principio fundamental de Warnier.



Temario

Modularización.

Favorece:

1. Construcción
   - No olvidemos el lema `'Divide y Vencerás'`.
   - Permite que los equipos de programadores trabajen en módulos independientes.
2. Depuración
   - La modularización `aisla los errores`.
3. Lectura
   - `Aumenta la legibilidad y comprensión` de un programa.
   - Un módulo debe ser inteligible a partir de su nombre, sus comentarios y los nombres de los módulos que los llaman.
4. Modificación
   - La modularización `aisla las modificaciones`, hace más manejable los cambios.
5. Elimina la redundancia de código
   - El código de una operación aparecerá una sola vez, i.e. `reuso`.

- Hay distintas maneras de definir métodos en Java.
- `Parámetros de entrada`: todos los parámetros de tipo primitivo entran por 'valor' (no cambian).
- `Salida de un método`: se puede devolver un sólo valor (de tipo primitivo o clase) o ninguno (void).

Modularización.

Entonces, un método incluye:

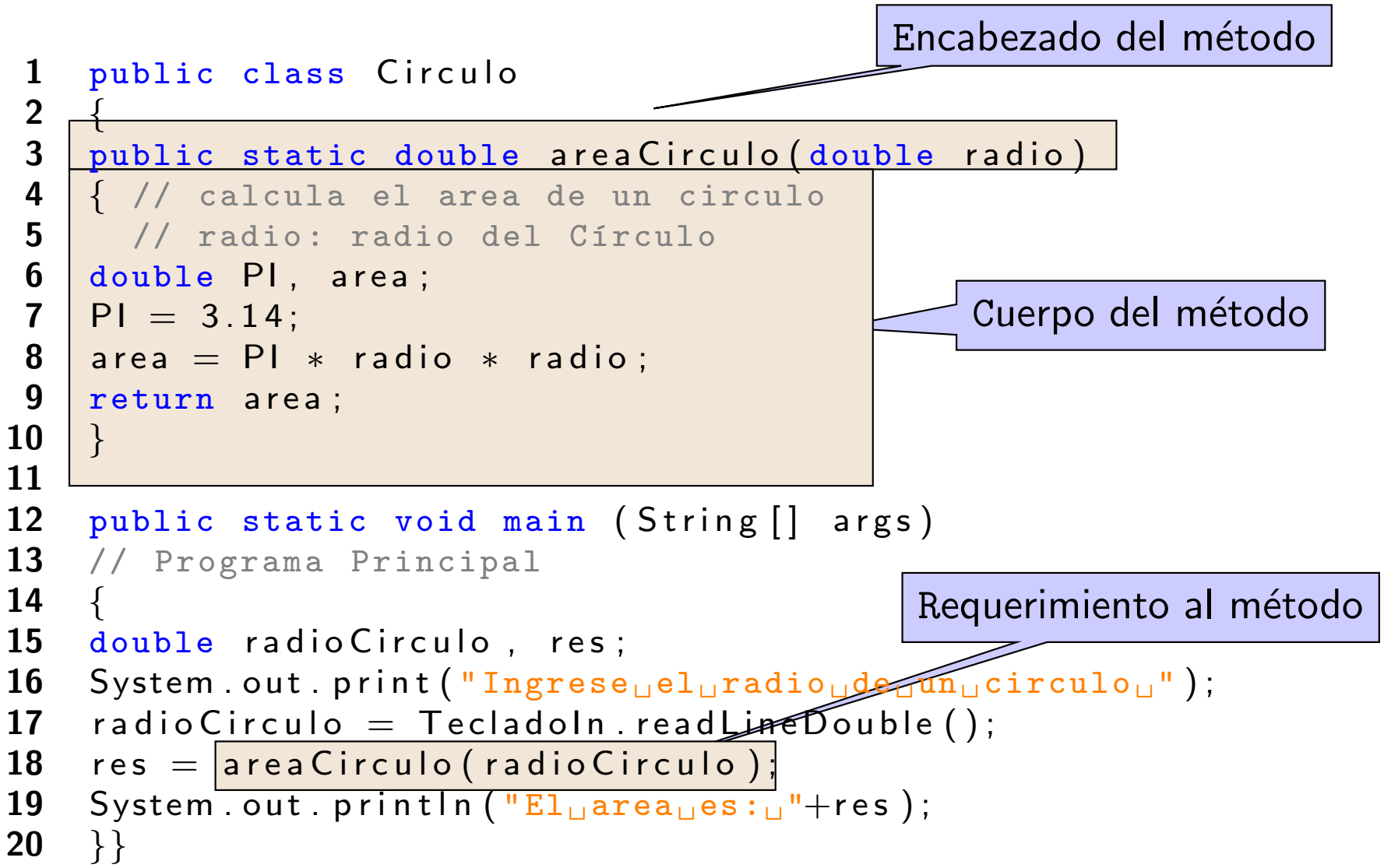
- Un bloque de código que tiene un nombre,
- recibe unos parámetros o argumentos (opcionalmente),
- contiene sentencias o instrucciones para realizar algo y
- devuelve un valor de algún Tipo conocido (opcionalmente).

Su encabezado se denomina signatura o firma.
La sintaxis global es:

```
Tipo_Valor_devuelto nombre_método ( lista_argumentos ) {
bloque_de_codigo;
}
```

y la lista de argumentos se expresa declarando el tipo y nombre de los mismos (como en las declaraciones de variables). Si hay más de uno se separan por comas.

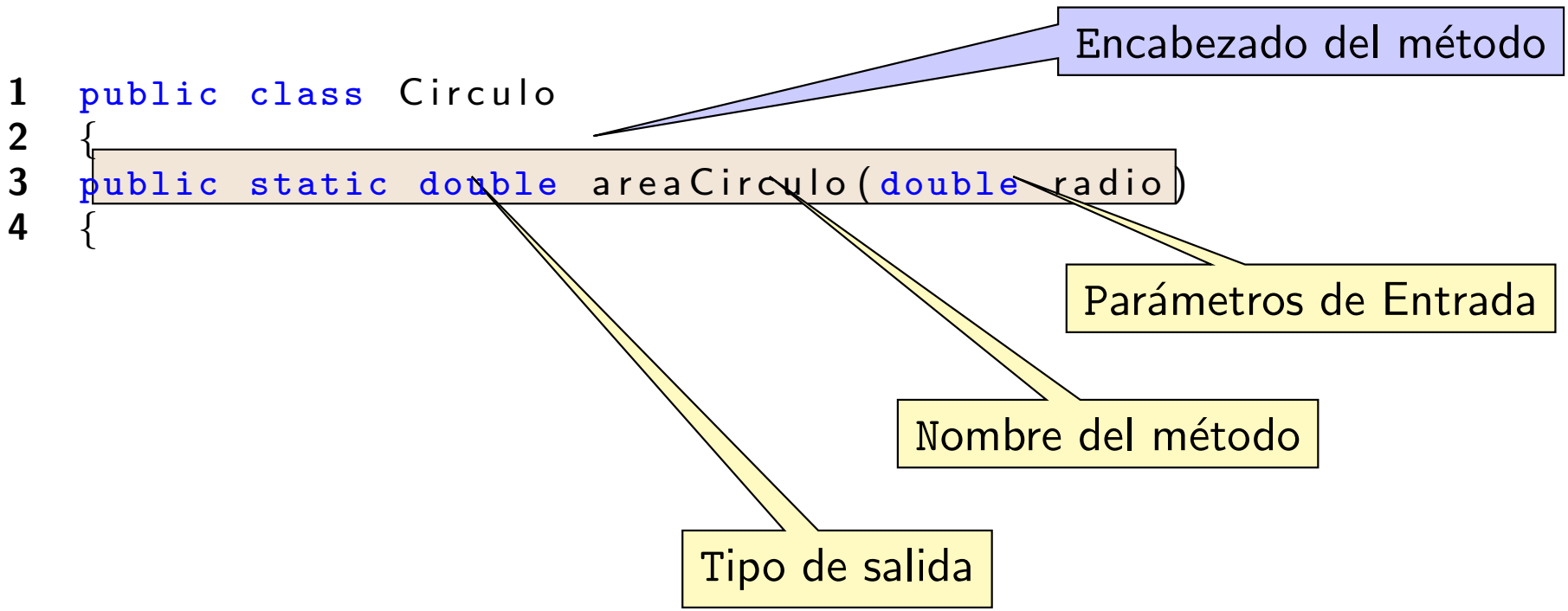
Modularización.




```
1  public class Circulo
2  {
3  public static double areaCirculo(double radio)
4  { // calcula el area de un circulo
5    // radio: radio del Círculo
6  double PI, area;
7  PI = 3.14;
8  area = PI * radio * radio;
9  return area;
10 }
11
12 public static void main (String[] args)
13 // Programa Principal
14 {
15 double radioCirculo, res;
16 System.out.print("Ingrese el radio de un circulo ");
17 radioCirculo = TecladoIn.readLineDouble();
18 res = areaCirculo(radioCirculo);
19 System.out.println("El area es: "+res);
20 }}
```

Modularización.

Componentes de la Signatura de un Método



Modularización.

Recomendación de Calidad

- Siempre hay que describir el propósito del metodo con un comentario.
- El comentario describe tambien el propósito de cada parámetro.

```
1  public class Circulo
2  {
3  public static double areaCirc(double radio)
4  // Este método calcula el area de un circulo
5  // radio: parametro que representa el radio de un circulo
6  {
7      // calcula el area de un circulo
8  double PI, area;
9  PI = 3.14;
10 area = PI * radio * radio;
11 return area;
12 }
```

La variables de los parámetros actuales no tienen que tener el mismo nombre que los parámetros formales

```
 1  public class Circulo
 2  {
 3  public static double areaCirculo(double radio)
 4  { // calcula el area de un circulo
 5  // radio: radio del Circulo
 6  double PI, area;
 7  PI = 3.14;
 8  area = PI * radio * radio;
 9  return area;
10  }
11  public static void main (String[] args)
12  // Programa Principal
13  {
14  double radioCirculo, res;
15  System.out.print("Ingrese el radio de un circulo ");
16  radioCirculo = TecladoIn.readLineDouble();
17  res = areaCirculo(radioCirculo);
18  System.out.println("El area es: "+res);
19  }}
```

Sentencias que modifican el flujo de control

Sentencia RETURN

- La Sentencia de salto `sentencia return` se puede utilizar para salir del 'sub-programa' en curso y retornar a la sentencia dentro de la cual se realizó la llamada.
- Para retornar un valor se debe poner el valor (o una expresión) a continuación de la palabra return.
- El valor devuelto por return debe coincidir con el tipo declarado como valor de retorno del sub-programa.

No olvidar!

Si la sentencia return está presente en un método ésta debe aparecer como última sentencia del método.

Sentencias que modifican el flujo de control

Sentencia RETURN

```
1  public static boolean metodoPrueba()
2  {
3  boolean aux;
4  // otras sentencias previas
5  if ( condicion booleana )
6          aux = true;
7  else
8          aux = false;
9
10 return aux;
11 }
```

No olvidar!

Si la sentencia return está presente en un método ésta debe aparecer como última sentencia del método.

Nosotros hemos estado utilizando métodos definidos en otras clases

1. Métodos de TecladoIn
2. Métodos de Math

```
1  int numero, x;
2  .....
3  System.out.print("Ingrese un nro. entero ");
4  numero = TecladoIn.readLineInt();
5       .....
6
7       x= Math.abs(-15);
8       System.out.println("el valor absoluto de -15 es: "+x);
```

Temario

Modularización.

Métodos de la clase MATH

Método	**Devuelve**
static int abs (int num)	valor absoluto de num
static double acos (double num) static double asin (double num) static double atan (double num)	arco coseno de num arco seno de num arco tangente de num
static double cos (double $angulo$) static double sin (double $angulo$) static double tan (double $angulo$)	coseno de $angulo$ seno de $angulo$ tangente de $angulo$
static double ceil (double num)	techo de num, por ej. el entero más pequeño mayor o igual a num
static double exp (double pot)	valor e a la pot
static double floor (double num)	piso de num, por ej, el entero más grande menor o igual a num
static double pow (double num, double $power$)	num elevado a la potencia $power$
static double razon ()	número aleatorio entre 0 (inclusive) y 1 (inclusive)
static double sqrt (double num)	la raíz de num que debe ser positivo

Modularización.

Métodos de la clase MATH

Por ejemplo para calcular el valor absoluto de un número la llamada a la función abs será:

```
1  int x= Math.abs(-15);
2  System.out.println("el valor absoluto de -15 es: "+x);
```

Lo cual mostrará en pantalla el siguiente mensaje:

el valor absoluto de -15 es: 15.

Por ejemplo para calcular la potencia de un número elevado a otro, ambos leídos por teclado, el código será: double x, y;

```
1  x = TecladoIn.readDouble();
2  y = TecladoIn.readDouble();
3  double pot = Math.pow(x,y);
4  System.out.println
5        ("La potencia de "+x+" elevado : "+y+" es "+pot);
```

Lo cual dará, para una entrada donde $x = 2$ e $y = 8$, el siguiente mensaje en pantalla:

La potencia de 2.0 elevado 8.0 es 256.0

Temario

Actividad

Definir un método que permita transformar grados Fahrenheit en grados Celsius. Es decir su encabezado define un parámetro de entrada que representa una temperatura en grados Fahrenheit. El método retorna la temperatura en grados Celsius, equivalente a los grados Fahrenheit.

De	**a**	**Formula**
Fahrenheit	Celsius	C = 5 * (F-32)/9
Celsius	Fahrenheit	F = (1,8) C + 32

Modularización.

Ejemplo A.

Método que dada la temperatura en grados Fahrenheit retorne la temperatura en grados Celsius.

```
 1  public class farACelsius
 2  {
 3  public static void main (String[] args)
 4  {
 5  int gradosFahrenheit;
 6  double gradosCelsius;
 7  System.out.println("Ingrese la temp. en gr.Fahrenheit");
 8  gradosFahrenheit = TecladoIn.readLineInt();
 9  gradosCelsius = pasarFahrCel(gradosFahrenheit);
10  System.out.println(gradosFahrenheit+" grados Fahrenheit = "
11    +gradosCelsius+" grados Celsius ");
12  }
13
14  public static double pasarFahrCel (int grados)
15  {
16  double celsius = 5 * (grados -32)/9;
17  return celsius;
18  }
19  }
```

Modularización.

Ejemplo B.

Método para calcular el valor absoluto de un número.

```
1  package Metodos;
2  public class Punto2 {
3      public static void main(String[] args)
4      {
5          double a, respuesta;
6          System.out.println("Ingrese un numero entero: ");
7          a = TecladoIn.readLineDouble();
8
9          respuesta = valorAbsoluto(a);
10
11         System.out.println();
12         System.out.println("|"+a+"| = "+respuesta);
13      }
14     public static double valorAbsoluto(double numero)
15     {
16         double resultado;
17         resultado = Math.abs(numero);
18         return resultado;
19     }
20 }
```

Modularización.

Ejemplo B. Slide 2.

```
1  package Metodos;
2  public class Punto2 {
3      public static void main(String[]args)
4      {
5          double a, respuesta;
6          System.out.println("Ingrese un numero entero: ");
7          a = TecladoIn.readLineDouble();
8
9          respuesta = valorAbsoluto(a);
10
11         System.out.println();
12         System.out.println("|"+a+"| = "+respuesta);
13      }
14     public static double valorAbsoluto(double numero)
15     {
16         return Math.abs(numero);
17     }
18 }
```

Modularización.

Ejemplo C.

Notar diferencia, RETURN retorna el resultado de una invocación a otro método.

```
1  package Metodos;
2  public class Punto2 {
3      public static void main(String[]args)
4      {
5      double a, respuesta;
6      System.out.println("Ingrese un numero entero: ");
7      a = TecladoIn.readLineDouble();
8      respuesta = valorAbsoluto(a);
9      System.out.println();
10     System.out.println("|"+a+"| = "+respuesta);
11     }
12     public static double valorAbsoluto(double numero)
13     {
14         return Math.abs(numero);
15     }
16 }
```

Modularización.

Ejemplo D. Hablemos sobre el REUSO de código

Notar diferencia, utilizamos una requerimiento a un método como componente de un mensaje de salida.

```
 1  package Metodos;
 2  public class Punto2 {
 3      public static void main(String[]args)
 4      {
 5      double a, respuesta;
 6      System.out.println("Ingrese un numero entero: ");
 7      a = TecladoIn.readLineDouble();
 8      System.out.println("|"+a+"| = "+valorAbsoluto(a));
 9      }
10      public static double valorAbsoluto(double numero)
11      {
12          return Math.abs(numero);
13      }
14  }
```

- El acoplamiento se define como una relación entre una parte de software particular y su sistema de software asociado.
- El nivel de acoplamiento es indicador estable de la mantenibilidad y la propención a fallos.
- Cada invocación a métodos esta determinando un vinculo de acoplamiento.

Acoplamiento Preciso

- Es importante minimizar el acoplamiento. Principio de bajo acoplamiento.
- Es importante ser cuidadosos en la correspondencia de parámetros formales y actuales: Cantidad, tipo de parámetros, valor retornado.

Interfaces de metodos. Acoplamiento.

```
1  public static void main (String[] args)
2  {
3   ....
4   System.out.println("Ingrese la base de la potencia:");
5   x = TecladoIn.readLineInt();
6   System.out.println("Ingrese el índice de la potencia:");
7   n = TecladoIn.readLineInt();
8
9   respuesta = potencia(x,n);
10  System.out.println();
11  System.out.println(x+" elevado a la "+n+" = "+respuesta);
12 }
13 public static int potencia(int base,int indice)
14 {
15    ....
16 }
```

Lectura Adicional

Jean-Dominique Warnier.
Síntesis de Programación Lógica. Los Tratamientos y sus Datos.
Editores Técnicos Asociados, S.A.
Barcelona

Catedra de Resolución de Problemas y Algoritmos.
Apunte sobre Modularización
Facultad de Informática, Universidad del Comahue, 2021.

Sun
Code Convention for the JAVA Programming Language
Sun Microsystems, Inc. 1997

James Gosling, Bill Joy, Guy Steele, Gilad Bracha.
The Java Language Specification Third Edition.
Addison-Wesley, 1996-2005.
http://java.sun.com/docs/books/jls/