



ORACLE

Academy



Java Foundations

4-1

¿Qué es un método?

ORACLE
Academy



Objetivos

- En esta lección se abordan los siguientes objetivos:
 - Estructurar código en una clase
 - Instanciar un objeto
 - Comprender las ventajas de los métodos
 - Usar el operador de punto (.) para acceder a los métodos y campos del objeto
 - Proporcionar argumentos para un método
 - Devolver valores desde un método



Clases que encontrará

- En el desarrollo Java, encontrará muchas clases para numerosos tipos de objetos distintos, incluidas...
 - Clases propias que escribirá
 - Clases escritas por otra persona
 - Clases que pertenecen a Java

Clases que encontrará

- Estas clases describen los siguientes elementos de los objetos...
 - Propiedades (campos)
 - Comportamientos (métodos)
- El objetivo de esta lección es ofrecer una descripción de cómo trabajar con cualquier clase, sus campos y sus métodos
- En las siguientes lecciones de esta sección se explorarán las clases importantes que proporciona Java
- Comenzaremos explorando las clases y los métodos de un modo más detallado

Ejercicio 1, parte 1

- Consideremos un caso y cómo podemos modelar los componentes implicados:
 - Es el cumpleaños de Alex Ha reunido a un grupo de ocho amigos para celebrarlo en un restaurante local. Cuando llega la factura, nadie sabe exactamente lo que debe Solo se conoce el total antes de impuestos (5%) y la propina (15%) ¡Pero que no cunda el pánico! Se ha traído el portátil y se le pide que escriba un programa para calcular el total de todos



Ejercicio 1, parte 2

- Cree un nuevo proyecto y agréguele el archivo Tip01.java
- Esto es lo que debe cada uno antes de impuestos (5%) y la propina (15%):

Persona 1: 10 \$	Persona 5: 7 \$
Persona 2: 12 \$	Persona 6: 15 \$ (Alex)
Persona 3: 9 \$	Persona 7: 11 \$
Persona 4: 8 \$	Persona 8: 30 \$

Ejercicio 1, parte 3

- El programa debe generar la siguiente salida:

```
person1: $12.0  
person2: $14.4  
person3: $10.8  
person4: $9.6  
person5: $8.4  
person6: $18.0  
person7: $13.2  
person8: $36.0
```


Modelado de objetos

- Puede que haya sentido la tentación de modelar el total de cada persona escribiendo esto:

```
public class Tip01{  
    public static void main(String args[]){  
  
        double person1 = 10;  
        double total1 = person1*(1 +.05 +.15);  
        System.out.println(total1);  
    } //end method main  
} //end class Tip01
```

Modelado de más objetos

- Si tuviera que modelar dos invitados a cenar, podría tener la tentación de copiar, pegar y cambiar el nombre:

```
public class Tip01{
    public static void main(String args[]){

        double person1 = 10;
        double total1  = person1*(1 +.05 +.15);
        System.out.println(total1);

        double person2 = 12;
        double total2  = person2*(1 +.05 +.15);
        System.out.println(total2);
    }//end method main
}//end class Tip01
```

Modelado de varios objetos

- ¿Y si tuviera que hacer el cálculo para 1000 invitados?

```
//You might think ...  
//Do I really have to copy, paste, and rename 1,000  
//times?
```

- ¿Qué sucede si uno de sus amigos olvida la cartera?
¿Qué sucede si ha cometido un error en la fórmula?

```
//You might think ...  
//Do I need to make 1,000 edits?!  
//There has to be a better way!!!
```



Las variables ofrecen flexibilidad

- Si se tiene que cambiar el porcentaje de los impuestos o la propina...
 - No habría que hacer 1000 modificaciones
 - Solo se debe editar cada variable una vez

```
double tax = 0.05;  
double tip = 0.15;
```

```
double person1 = 10;  
double total1 = person1*(1 +tax +tip);  
System.out.println(total1);
```

```
double person2 = 12;  
double total2 = person2*(1 +tax +tip);  
System.out.println(total2);
```

Los métodos ofrecen una flexibilidad similar

- Se repiten los mismos cálculos y comportamientos de impresión
- Pero esta vez la lógica se puede escribir una sola vez en un método

```
double tax = 0.05;  
double tip = 0.15;
```

```
double person1 = 10;  
double total1 = person1*(1 +tax +tip);  
System.out.println(total1);
```

```
double person2 = 12;  
double total2 = person2*(1 +tax +tip);  
System.out.println(total2);
```

Cuándo se deben utilizar los métodos

- Es recomendable escribir un método si...
- Repite líneas de código muy parecidas, incluidos los cálculos
- Necesita describir el comportamiento de un objeto

Cómo utilizar un método main

- El método main se denomina como controlador
 - Utilízelo para controlar los eventos de un programa
 - Utilízelo para acceder a campos y métodos, o bien a otras clases
- El método main no describe el comportamiento de un objeto concreto
 - Manténgalo separado de las clases de objeto
 - Utilice solo un método main para cada aplicación

¿Cuál es el aspecto de las clases de objeto?

- Cambiaremos el código para que se ajuste al siguiente formato
- Veamos cómo se puede conseguir que nuestro código tenga este aspecto:

```
1 public class Calculator{  
2  
3  
4     Properties  
5  
6  
7  
8     Behaviours  
9  
10  
11 }
```

Paso 1) Trasladar los campos desde el método main

```
public class Calculator{  
    //Fields  
    public double tax = 0.05;  
    public double tip = 0.15;  
    public double originalPrice = 10;  
  
    public static void main(String args[]){  
        //double tax = 0.05;  
        //double tip = 0.15;  
  
        //double person1 = 10;  
        double total1 = person1*(1 + tax + tip);  
        System.out.println(total1);  
    } //end method main  
} //end class Calculator
```


Las variables locales
se convierten en
campos



Paso 2) Trasladar los comportamientos repetidos desde el método main

```
public class Calculator{  
    //Fields  
    public double tax = 0.05;  
    public double tip = 0.15;  
    public double originalPrice = 10;  
  
    //Methods  
    public void findTotal(){  
        //Calculate total after tax and tip  
        //Print this value  
    } //end method findTotal  
  
    public static void main(String args[]){  
        //double total1 = person1*(1 + tax + tip);  
        //System.out.println(total1);  
    } //end method main  
} //end class Calculator
```

*Este método lo
escribirá en el
siguiente
ejercicio*



Paso 3) Eliminar el método main

```
public class Calculator{
    //Fields
    public double tax = 0.05;
    public double tip = 0.15;
    public double originalPrice = 10;

    //Methods
    public void findTotal(){
        //Calculate total after tax and tip
        //Print this value
    } //end method findTotal

    //public static void main(String args[]){
        //double total1 = person1*(1 + tax + tip);
        //System.out.println(total1);
    //} //end method main
} //end class Calculator
```

¡Correcto!

```
public class Calculator{  
    //Fields  
    public double tax = 0.05;  
    public double tip = 0.15;  
    public double originalPrice = 10;  
  
    //Methods  
    public void findTotal(){  
        //Calculate total after tax and tip  
        //Print this value  
    } //end method findTotal  
} //end class Calculator
```

¿Dónde incluyo el método main?

```
public class CalculatorTest {  
    public static void main(String args[]){  
        //Create Calculator object instance  
        Calculator calc = new Calculator();  
  
        calc.tip = 0.10;    //Altering a field  
        calc.findTotal();  //Calling a method  
    } //end method main  
} //end class CalculatorTest
```

- Incluya el método main en otra clase, como una clase de prueba
- El método main controla la acción del programa:
 - Crea instancias de los objetos
 - Llama a los campos y los métodos de una instancia con el operador de punto (.)

Variables para objetos

```
int      age  = 22;  
String   str  = "Happy Birthday!";  
Scanner  sc   = new Scanner();  
Calculator calc = new Calculator();
```

tipo *nombre* *valor*

- Los objetos, como los primitivos, se representan mediante variables
- La mayoría de los objetos requieren la palabra clave `new` cuando se inicializan para crear instancias nuevas
 - Esto se denomina instanciar un objeto
 - Hay algunas excepciones, como los objetos `String`, que no requieren la palabra clave `new`

Uso del operador de punto

- Coloque el operador de punto (.) después del nombre de una variable para acceder a sus campos o métodos

```
public class CalculatorTest {  
    public static void main(String args[]){  
        Calculator calc = new Calculator();  
        calc.printTip();           //prints 0.15  
        calc.tip = 0.10;  
        calc.printTip();           //prints 0.10  
    } //end method main  
} //end class CalculatorTest
```

```
public class Calculator{  
    public double tip = 0.15;      //initialized value 0.15  
    public void printTip(){  
        System.out.println(tip);  
    } //end method printTip  
} //end class Calculator
```

Ejercicio 2, parte 1

- Cree un nuevo proyecto y agréguele los archivos CalculatorTest2.java y Calculator2.java
- Complete el método findTotal(), que debe:
 - Calcular un total en función de los campos tax, tip y originalPrice
 - Imprimir el total de una persona



Ejercicio 2, parte 2

- En el método main:
 - Instancie un objeto Calculator llamado calc
 - Observe su IDE después de escribir "calc"
 - Acceda a los campos y métodos de este objeto para imprimir el total de cada persona en la fiesta de cumpleaños
- Si prefiere usar otros valores, cambie tip y tax



Lo que puede haber escrito

- Es posible que haya escrito un programa como el siguiente:
 - Se necesitan dos líneas para cada persona
 - Y más si decide imprimir los nombres o cambiar los valores de impuestos o propina

```
public class CalculatorTest{  
    public static void main(String args[]){  
        Calculator calc = new Calculator();  
        calc.originalPrice = 10;  
        calc.findTotal();  
        calc.originalPrice = 12;  
        calc.findTotal();  
    } //end method main  
} //end class CalculatorTest
```

Aumento de la flexibilidad

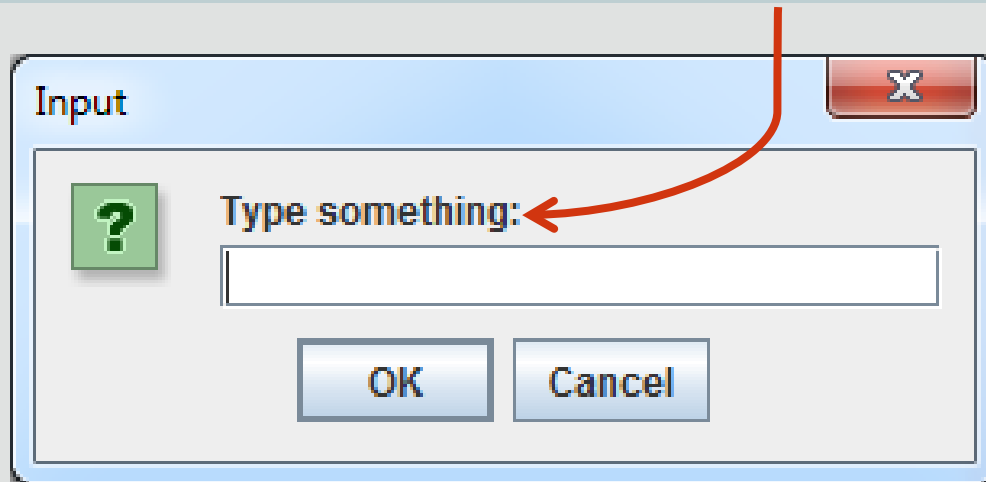
- Pero es posible hacer lo mismo en una sola línea
- También es peligroso escribir programas que accedan directamente a los campos
 - Lo explicaremos más adelante
 - El objetivo de esta lección es prepararle para trabajar con clases importantes proporcionadas por Java

```
calc.originalPrice = 10;           //Dangerous  
calc.findTotal();
```

Recuerdo de la clase JOptionPane

- Cuando agregamos el literal String "type something:" a la llamada de método, proporcionamos argumentos al método
- Este argumento modifica la clase JOptionPane resultante

```
JOptionPane.showInputDialog("Type something:");
```



¿Cuándo pueden aceptar argumentos los métodos?

- Verá que hay muchos métodos que se ven afectados por los argumentos
 - Pero los métodos se deben escribir de forma que acepten argumentos
 - De lo contrario, el compilador se queja
 - El método calculate se escribe de modo que no acepte argumentos

```
Calculator calc = new Calculator();  
calc.calculate();           //Good  
calc.calculate(3, 2.0);     //Fail
```


```
public void calculate(){  
    //How do I calculate?  
} //end method calculate
```


Argumento del método

- Pero el método calculate se ha escrito para aceptar dos argumentos:
 - El primer argumento debe ser un valor de tipo int
 - El segundo argumento debe ser un valor de tipo double

```
Calculator calc = new Calculator();  
calc.calculate(3, 2.0);
```

```
public void calculate(int x, double y){  
    System.out.println(x/y);           //prints 1.5  
} //end method calculate
```



- A la variable int x se le asigna el valor 3
- A la variable double y se le asigna el valor 2.0

El orden de los argumentos es importante

- ¿Qué sucede si invertimos el orden de los argumentos?

```
Calculator calc = new Calculator();  
calc.calculate(2.0, 3);
```

- Se obtiene un error del compilador:
 - A int x no se le puede asignar un valor double
 - El primer argumento debe ser un valor int

```
public void calculate(int x, double y){  
    System.out.println(x/y);  
}//end method calculate
```

2,0

3

Ejercicio 3, parte 1

- Cree un nuevo proyecto y agréguele los archivos CalculatorTest3.java y Calculator3.java
- En el método main:
 - Utilice una instancia de objeto Calculator y proporcione argumentos a findTotal() para imprimir el total de cada persona
 - **Indicación:** Observe el método findTotal() de la clase Calculator para averiguar cuántos argumentos acepta este método



Ejercicio 3, parte 2

- ¿A quién pertenece cada total?
- Modifique el método findTotal() para aceptar un argumento name de tipo String adicional
- Concatene la sentencia print para incluir name
- Observe la queja de su IDE en el método main y revise sus llamadas al método findTotal()



Argumentos y parámetros de método

- Un argumento es un valor que se transfiere durante una llamada a un método:

```
Calculator calc = new Calculator();  
calc.calculate(3, 2.0);    //should print 1.5
```

Argumentos

- Un parámetro es una variable que se define en la declaración de método:

```
public void calculate(int x, double y){  
    System.out.println(x/y);  
} //end method calculate
```

Parámetros



Parámetros de método: Ejemplos

- Los métodos pueden aceptar cualquier número o tipo de parámetros:

```
public void calculate0(){  
    System.out.println("No parameters");  
}//end method calculate0
```

```
public void calculate1(int x){  
    System.out.println(x/2.0);  
}//end method calculate1
```

```
public void calculate2(int x, double y){  
    System.out.println(x/y);  
}//end method calculate2
```

```
public void calculate3(int x, double y, int z){  
    System.out.println(x/y +z);  
}//end method calculate3
```

Ámbito de los parámetros

- A los métodos se les debe indicar lo que tienen que hacer con los argumentos que reciban
- Para ello, se utilizan parámetros de método
 - Los parámetros de método son variables que existen en todo el ámbito de un método
 - Se crean en la declaración de método
 - Ámbito hace referencia al **{bloque de código}** que pertenece a un método después de su declaración

```
public void calculate(int x, double y){  
    System.out.println(x/y);  
}//end method calculate
```


Referencia a parámetros de método

- Se puede hacer referencia a una variable en cualquier parte de su bloque actual después de declararse
- No se puede hacer referencia a una variable fuera del bloque donde se ha declarado o antes de que se declare

```
public void calculate(int x, double y){  
    System.out.println(x/y);    Ámbito de x  
} //end method calculate
```



```
public void calculate2(){  
    System.out.println(2*x);    No es ámbito de x //What is x?  
} //end method calculate2
```

Búsqueda de la suma total: Caso

- Sus amigos están impresionados con lo que ha aprendido en el curso de Fundamentos de Java Alex pregunta: “¿Cuál es el total de toda la mesa?” Saber la respuesta a esta pregunta permitirá asegurarse de que todos contribuyan y que el camarero reciba el importe correcto
- ¿Cómo se puede incluir en el código?



Suma de totales

- Otra forma de plantearlo:
 - He calculado un valor en un método...
 - Pero está almacenado como una variable que no puede existir fuera del ámbito de su bloque de método...
 - ¿Cómo puedo obtener este valor desde fuera?

```
public void findTotal(double price, String name){  
    double total = price * (1 + tax + tip);  
    System.out.println(name + ": $ " + total);  
} //end method findTotal
```

Suma de totales

```
public class CalculatorTest
```

```
    public static void  
    main(String[] args)
```

```
public class Calculator
```

```
    public void findTotal()
```

```
        double total
```

¡Ajá! ¡Intenta obtenerme!

Suma de totales

- Si piensa escribir un programa como este:

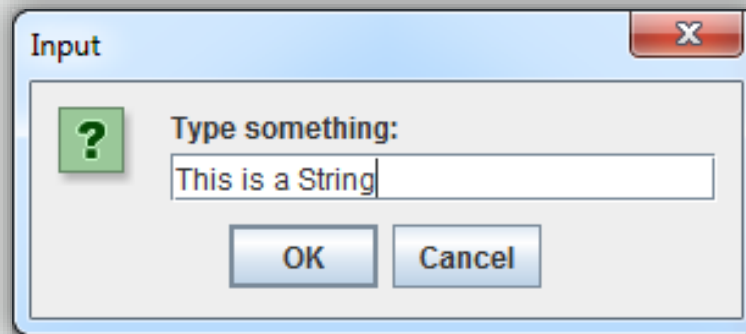
```
public class CalculatorTest{  
    public static void main(String args[]){  
        Calculator calc = new Calculator();  
        calc.findTotal(10);  
        calc.findTotal(12);  
        System.out.println(calc.findTotal(10) +  
            calc.findTotal(12));  
    } //end method main  
} //end class CalculatorTest
```

- Ha acertado a medias
- Sin embargo, su IDE muestra el siguiente error:
 - 'void' type not allowed here

¿Qué es un tipo void?

- `showInputDialog()` es un método de tipo `String`
 - Devuelve un valor que se puede almacenar como `String`

```
String input = JOptionPane.showInputDialog("Type something:");
```



- Los métodos de tipo `void` no devuelven ningún valor
 - No hay valores para devolver después de llamar a un método de tipo `void`

```
System.out.println("println is a void type method");
```

Tipos de retorno de método

- Las variables pueden tener valores de muchos tipos distintos:

int *double* *long* *char* *float* *byte*
short *String* *boolean* *Calculator*

- Las llamadas de método también devuelven muchos tipos distintos:

int *double* *long* *char* *float* *byte*
short *String* *boolean* *Calculator*

- Cómo conseguir que un método devuelva un valor:
 - Declare el método como un tipo de devolución que no sea void
 - Utilice la palabra clave return en un método, seguida de un valor



Tipos de retorno de método: Ejemplos

- Los métodos deben devolver datos que coincidan con su tipo de retorno:

```
public void printString(){  
    System.out.println("Hello");  
} //end method printString
```

```
public String returnString(){  
    return("Hello");  
} //end method returnString
```

```
public int sum(int x, int y){  
    return(x + y);  
} //end method sum
```

```
public boolean isGreater(int x, int y){  
    return(x > y);  
} //end method isGreater
```

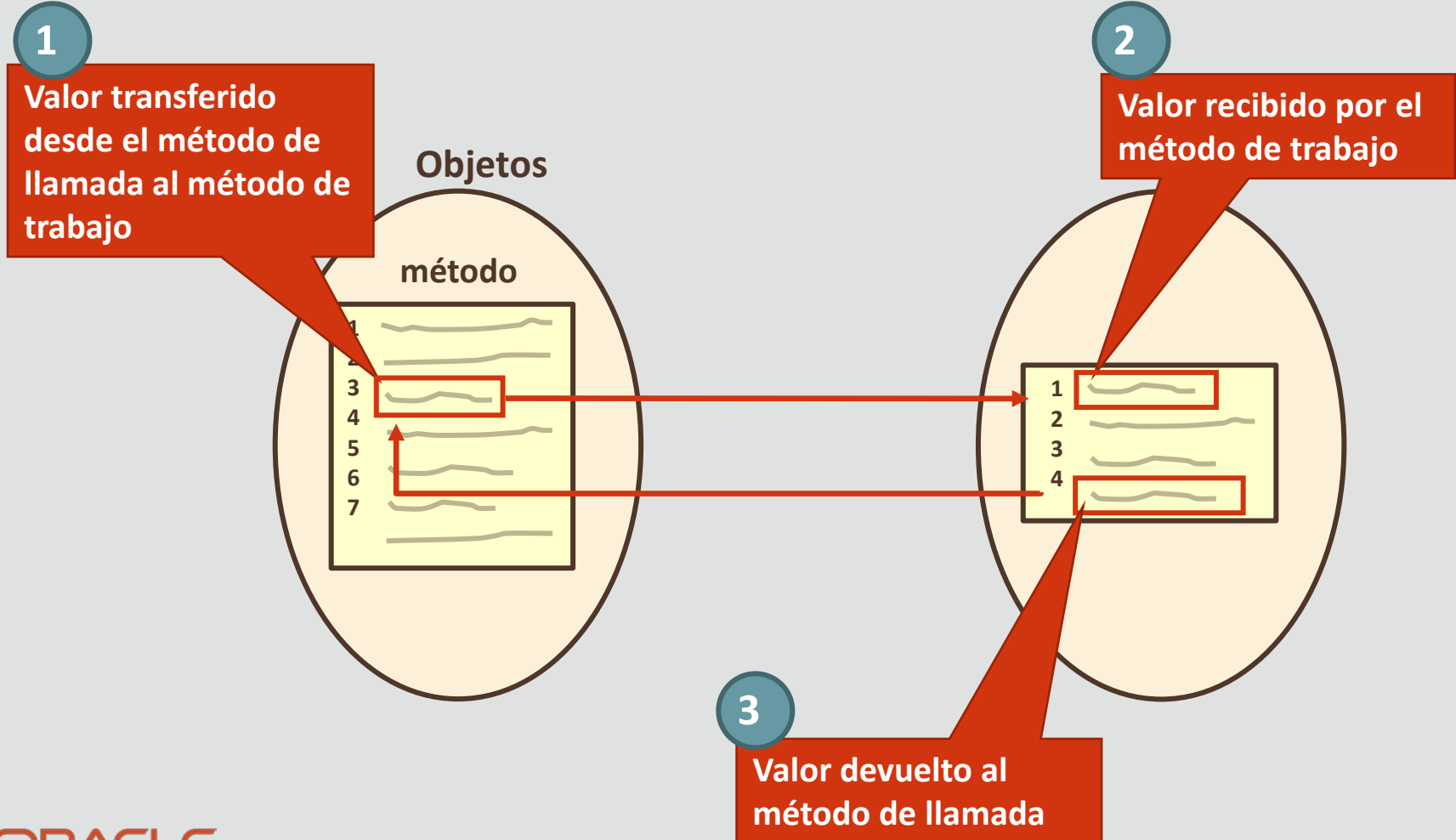

Retorno del método

- Los ejemplos de código siguientes producen resultados equivalentes:

```
public static void main(String[] args){  
    int num1 = 1, num2 = 2;  
    int result = num1 + num2;  
    System.out.println(result);  
} //end method main
```

```
public static void main(String[] args){  
    int num1 = 1, num2 = 2;  
    int result = sum(num1, num2);  
    System.out.println(result);  
} //end method main  
public static int sum(int x, int y){  
    return(x + y);  
} //end method main
```

Transferencia de argumentos y devolución de valores



Ejercicio 4, parte 1

- Edite la solución del ejercicio 3
 - O bien, cree un nuevo proyecto y agréguele los archivos CalculatorTest4.java y Calculator4.java
- Busque e imprima el total de toda la tabla, sin incluir los impuestos ni la propina
 - Deberá editar findTotal() para que devuelva su valor calculado



Ejercicio 4, parte 2

- Persona8 ha olvidado su cartera
- Y la comida de Alex pretende ser un regalo de cumpleaños
- Modifique findTotal() para que el costo de sus comidas se reparta por igual con el resto del grupo
- Vuelva a calcular el total de toda la tabla
- Este número no debe haber cambiado



Resumen de la sintaxis de método

The diagram illustrates the syntax of a Java method with the following components labeled:

- Tipo de retorno de método**: Points to the `public double` part of the method signature.
- Nombre del método**: Points to the `calculate` part of the method signature.
- Parámetros**: Points to the `(int x, double y)` part of the method signature.
- Implantación**: Points to the body of the method, which includes `double quotient = x/y;`, `return quotient;`, and the closing brace `}`.

```
public double calculate(int x, double y){  
    double quotient = x/y;  
    return quotient;  
} //end method calculate
```

Resumen

- En esta lección, debe haber aprendido lo siguiente:
 - Estructurar código en una clase
 - Instanciar un objeto
 - Comprender las ventajas de los métodos
 - Utilizar el operador de punto para acceder a los campos y los métodos de un objeto
 - Proporcionar argumentos para un método
 - Devolver valores desde un método





ORACLE

Academy

