1. ¿Qué es un Método en Java?

Los métodos son subrutinas que manipulan los datos definidos por la clase y, en muchos casos, brindan acceso a esos datos. En la mayoría de los casos, otras partes de tu programa interactuarán con una clase a través de sus métodos.

Un método contiene una o más declaraciones. En un código Java bien escrito, cada método **realiza solo una tarea**. Cada método tiene un nombre, y es este el que se usa para llamar al método. En general, puede dar el nombre que desee a un método cualquiera. Sin embargo, recuerde que *main()* está reservado para el método que comienza ejecución de su programa. Además, no use las palabras clave de Java para nombres de métodos.

2. ¿Cómo se escribe un método?

Un método tendrá paréntesis después de su nombre. Por ejemplo, si el nombre de un método es *getval*, se escribirá *getval()* cuando su nombre se usa en una sentencia. Esta notación lo ayudará a distinguir los nombres de las variables de los nombres de los métodos.

La forma general de un método se muestra a continuación:

```
tipo-retorno nombre(lista parámetros){
        //Cuerpo del método
 }
```

- Aquí, *tipo-retorno* especifica el tipo de datos devueltos por el método. Puede ser cualquier tipo válido, incluidos los tipos de clase que cree. Si el método no devuelve un valor, su tipo de devolución debe ser `void`.
- El nombre del método se especifica por *nombre*. Puede ser cualquier identificador legal que no sea el que ya utilizan otros elementos dentro del alcance actual.
- La *lista de parámetros* es una secuencia de tipos e identificadores separados por comas. Los parámetros son esencialmente variables que reciben el valor de los argumentos pasados al método cuando se llama. Si el método no tiene parámetros, la lista de parámetros estará vacía.

3. Ejemplos de Métodos

Agregar un método a la clase de Vehiculo

Anteriormente:

```
/* Un programa que usa la clase Vehiculo
 El archivo se llama DemoVehiculo.java
 */
class Vehiculo {
    int pasajeros;  //números de pasajeros
    int capacidad;  //capacidad del combustible en galones
    int mpg;        //combustible consumido en millas por galon
}
```

```
//Esta clase declara un objeto de tipo Vehiculo
class DemoVehiculo {
    public static void main(String[] args) {
        Vehiculo minivan = new Vehiculo();
        int rango;
        //asignando valores a los campos de minivan
        minivan.pasajeros = 9;
        minivan.capacidad = 15;
        minivan.mpg = 20;
        //Calcular el rango asumiendo un tanque lleno
        rango = minivan.capacidad * minivan.mpg;
        System.out.println("La Minivan puede llevar " + minivan.pasajeros + " pasajeros con un rango de " + rango + " millas");
    }
}
```

Como acabamos de explicar, los métodos de una clase suelen manipular y proporcionar acceso a los datos de la clase. Con esto en mente, recuerde que `main()` en los ejemplos anteriores calculó el rango de un vehículo al multiplicar su *tasa de consumo de combustible* por su *capacidad de combustible*. Si bien es técnicamente correcto, esta no es la mejor manera de manejar este cálculo.

El cálculo del alcance de un vehículo es algo que se maneja mejor con la clase de vehículo en sí. El motivo de esta conclusión es fácil de entender: el alcance de un vehículo depende de la capacidad del tanque de combustible y la tasa de consumo de combustible, y ambas cantidades están encapsuladas por el vehículo. Al agregar un método al Vehículo que calcula el rango, está mejorando su **estructura orientada a objetos**.

Para agregar un método al Vehículo, especifíquelo dentro de la declaración del Vehículo. Por ejemplo, la siguiente versión de *Vehiculo* contiene un método llamado *rango()* que muestra el rango o alcance del vehículo.

```
//Añadiendo rango a Vehiculo


class Vehiculo {
    int pasajeros;  //números de pasajeros
    int capacidad;  //capacidad del combustible en galones
    int mpg;        //combustible consumido en millas por galon


    //Mostrando el rango
    void rango (){
        System.out.println("Con rango de "+ capacidad*mpg);
    }
}


class MetodoAdicional {
```

```
    public static void main(String[] args) {
        Vehiculo minivan = new Vehiculo();
        Vehiculo sportscar = new Vehiculo();



        //Asigando valores a los campos en minivan
        minivan.pasajeros=9;
        minivan.capacidad=15;
        minivan.mpg=20;

        //Asigando valores a los campos en minivan
        sportscar.pasajeros=10;
        sportscar.capacidad=25;
        sportscar.mpg=30;

        System.out.print("La Minivan puede llevar " +minivan.pasajeros +". ");
        minivan.rango();

        System.out.print("El Sportscar puede llevar " +minivan.pasajeros +". ");
        sportscar.rango();
    }
}
```

Este programa genera la siguiente salida:

```
La Minivan puede llevar 9. Con rango de 300
El Sportscar puede llevar 9. Con rango de 750
```

4. Entendiendo el funcionamiento del Método

Veamos los elementos clave de este programa, comenzando con el método `rango()`:

- La primera línea de *rango()* es

```
void rango() {
//Esta línea declara un método llamado rango que no tiene parámetros.
//Su tipo de devolución es void, osea nulo.
//Por lo tanto, rango() no devuelve un valor al ser llamado.
//La línea termina con la llave de apertura del cuerpo del método.
```

- El cuerpo del *rango()* consiste únicamente en esta línea:

```
System.out.println("Con rango de "+ capacidad*mpg);

//Esta declaración muestra el rango del vehículo multiplicando la capacidad del combustible por mpg.

//Dado que cada objeto de tipo Vehiculo tiene su propia copia de capacidad y mpg, cuando se lla ma

//a rango(), el cálculo de rango utiliza las copias del objeto que realiza la llamada de esas v ariables.

//El método rango() finaliza cuando se encuentra su llave de cierre.

//Esto hace que el control del programa se transfiera nuevamente a quién lo llama.
```

- A continuación, mira de cerca esta línea de código desde adentro de *main()*:

```
minivan.rango();
```

Esta declaración invoca el método *rango()* en minivan. Es decir, llama a rango() relativo al objeto de minivan, utilizando el nombre del objeto seguido del operador de punto. **Cuando se llama a un método, el control del programa se transfiere al método**. Cuando el método finaliza, el control se transfiere de vuelta a quién lo llama, y la ejecución se reanuda con la línea de código que sigue a la llamada.

En este caso, la llamada a *minivan.rango()* muestra el rango del vehículo definido por *minivan*. De manera similar, la llamada a *sportscar.rango()* muestra el rango del vehículo definido por *sportscar*. Cada vez que *rango()* se invoca, muestra el rango para el objeto especificado.

Hay algo muy importante para notar dentro del método *rango()*: las variables de instancia *capacidad* y *mpg* se refieren directamente, sin precederlas con un nombre de objeto o el operador de punto. Cuando un método usa una variable de instancia que está definida por su clase, lo hace directamente, **sin referencia explícita a un objeto y sin el uso del operador de punto**. Esto es fácil de entender si lo piensas mejor.

Un método siempre se invoca en relación con algún objeto de su clase. Una vez que se ha producido esta invocación, se conoce el objeto. Por lo tanto, dentro de un método, no hay necesidad de especificar el objeto por segunda vez. Esto significa que *capacidad* y *mpg* dentro de *rango()* se refieren implícitamente a las copias de esas variables encontradas en el objeto que invoca *rango()*.

5. Retornando de un Método

En general, hay dos condiciones que hacen que un método retorne:

- Primero, como lo muestra el método *rango()* en el ejemplo anterior, cuando se encuentra la llave de cierre del método.
- El segundo es cuando se ejecuta una declaración de retorno (**return**).

Hay dos formas de devolución:

- Una cuando usa métodos con la palabra ***void*** (aquellos que no devuelven un valor) y otra para devolver valores. Veremos la primera forma y luego pasaremos a retornar valores.

En un método `void`, se puede causar la terminación inmediata de un método utilizando esta forma de devolución:

```
return;
```

Cuando se ejecuta esta instrucción, el control del programa regresa a quién lo llama, omitiendo cualquier código restante en el cuerpo del método. Por ejemplo, considere este método:

```
void miMetodo(){
    int i;
    for (i=0; i<10; i++){
        if (i == 5) return; //Se detiene antes de llegar a 5
        System.out.println(i);
    }
  }
```

Aquí, el bucle for solo se ejecutará de 0 a 5, porque una vez que sea igual a 5, el método *retornará* (Ojo, no devuelve sino que finaliza).. Es permisible tener múltiples instrucciones *return* en un método, especialmente cuando hay dos o más rutas fuera de él. Por ejemplo:

```
void miMetodo(){
 // ...
if (hecho) return;
 // ...
if (error) return;
 // ...
}
```

Aquí, el método retorna si está *hecho* o si ocurre un *error*. Tenga cuidado, sin embargo, porque tener demasiados puntos de salida en un método puede desestructurar tu código; así que evita usarlos casualmente. Un método bien diseñado tiene puntos de salida bien definidos.

✕**Recuerde**: un método *void* puede retornar (regresa a quién lo llama) de una de las dos maneras: cuando alcanza su llave de cierre, o cuando se ejecuta una declaración return.

6. Devolviendo un valor

Aunque los métodos con un tipo de retorno *void* no son raros, la mayoría de los métodos devolverán un valor. De hecho, la capacidad de devolver un valor es una de las características más útiles de un método.

Los valores de retorno se utilizan para una variedad de propósitos en la programación. En algunos casos, el valor de retorno contiene el resultado de algún cálculo. En otros casos, el valor de retorno puede simplemente indicar éxito o falla. Y en otros, puede contener un código de estado. Cualquiera que sea el propósito, usar valores de retorno de método es una parte integral de la programación de Java.

Los métodos devuelven un valor a la rutina de llamada usando esta forma de devolución:

```
return valor;
```

Aquí, *valor* es el valor devuelto. Esta forma de devolución se puede usar solo con métodos que tienen un tipo de devolución **no-void**. Además, un método **no-void** debe devolver un valor utilizando esta forma de **return**.

6.1. Uso de return en un método

Puede usar un valor de retorno para mejorar la implementación de *rango()*. En lugar de mostrar el rango, un mejor enfoque es hacer que *rango()* calcule el rango y devuelva este valor. Una de las ventajas de este enfoque es que puede usar el valor para otros cálculos.

Ejemplo:

El siguiente ejemplo modifica *rango()* para devolver el rango en lugar de mostrarlo.

```
//Usando return

class Vehiculo {
    int pasajeros;  //números de pasajeros
    int capacidad;  //capacidad del combustible en galones
    int mpg;        //combustible consumido en millas por galon

    //Retornando o devolviendo el rango
    int rango (){
        return capacidad*mpg;
    }

}

class RetornandoMetodo {

    public static void main(String[] args) {
        Vehiculo minivan = new Vehiculo();
        Vehiculo sportscar = new Vehiculo();

        int rango1, rango2;

        //Asigando valores a los campos en minivan
        minivan.pasajeros=9;
        minivan.capacidad=15;
        minivan.mpg=20;

        //Asigando valores a los campos en minivan
        sportscar.pasajeros=10;
        sportscar.capacidad=25;
        sportscar.mpg=30;

        //Obteniendo los rangos
        rango1=minivan.rango();
```

```
        rango2=sportscar.rango();


        System.out.println("La Minivan puede llevar " +minivan.pasajeros +". Con rango de: "+ra
ngo1);


        System.out.println("El Sportscar puede llevar " +minivan.pasajeros +". Con rango de: "+
rango2);
        sportscar.rango();
    }
}
```

Salida:

```
La Minivan puede llevar 9. Con rango de: 300
El Sportscar puede llevar 9. Con rango de: 750
```

En el programa, observe que cuando se llama a *rango()*, se coloca en el lado derecho de una instrucción de asignación. A la izquierda hay una variable que recibirá el valor devuelto por rango(). Por lo tanto, después de que `rango1 = minivan.rango();` se ejecuta, el *rango* del objeto *minivan* se almacena en el *rango1*.

Observe que *rango()* ahora tiene un tipo de retorno *int*. Esto significa que devolverá un valor entero a quién lo llama. El tipo de devolución de un método es importante porque el tipo de datos devueltos por un método debe ser compatible con el tipo de devolución especificado por el método.

✕Por lo tanto, si desea que un método devuelva datos de tipo *double*, su tipo de *return* debe ser de tipo *double*.

Aunque el programa anterior es correcto, no está escrito de la manera más eficiente posible. Específicamente, no hay necesidad de las variables *rango1* o *rango2*. Una llamada a *rango()* se puede usar en la instrucción *println()* directamente, como se muestra aquí:

```
System.out.println("La Minivan puede llevar " +minivan.pasajeros +". Con rango de: "+ minivan.r
ango());
```

7. Métodos con Parámetros

Es posible pasar uno o más valores a un método cuando se llama al método. Recuerde que un valor pasado a un método se llama **argumento**. Dentro del método, la variable que recibe el argumento se llama **parámetro**. Los parámetros se declaran dentro de los paréntesis que siguen al nombre del método. La sintaxis de declaración de parámetro es la misma que la utilizada para las variables.

Un parámetro está dentro del alcance de su método, y aparte de su tarea especial de recibir un argumento, actúa como cualquier otra variable local.

Aquí hay un ejemplo simple del uso de parámetros. Dentro de la clase *ComprobarNumero*, el método *esPar()* devuelve *true* si el valor que se pasa es par. Devuelve *false* de lo contrario. Por lo tanto, esPar() tiene un **tipo de retorno booleano**.

```
//Un ejemplo siemple del uso de parámetros
```

```
class ComprobarNumero {

    //Retorna true si x es par
    boolean esPar(int x){
        if ((x%2)==0) return true;
        else return false;
    }
}

class ParametroDemo {

    public static void main(String[] args) {
        ComprobarNumero e=new ComprobarNumero();

        if (e.esPar(10)) System.out.println("10 es par.");
        if (e.esPar(9)) System.out.println("9 es par.");
        if (e.esPar(8)) System.out.println("8 es par.");
    }
}
```

Salida:

```
10 es par.
8 es par.
```

En el programa, *esPar()* se llama tres veces y cada vez se pasa un valor diferente. Miremos este proceso de cerca. Primero, observe cómo se llama *esPar()*. El argumento se especifica entre paréntesis.

- Cuando se usa *esPar()* la primera vez, se pasa al valor 10. Por lo tanto, cuando *esPar()* comienza a ejecutarse, el parámetro *x* recibe el valor de 10.
- En la segunda llamada, 9 es el argumento, y *x*, entonces, tiene el valor de 9.
- En la tercera llamada, el argumento es *8,* que es el valor que recibe *x*.

✕El punto es que el valor pasado como argumento cuando se llama a *esPar()* es el valor recibido por su parámetro, *x*.

Un método puede tener más de un parámetro. Simplemente declare cada parámetro, separando uno del siguiente con una *coma*. Por ejemplo, la clase *Divisor* define un método llamado *esDivisor()* que determina si el primer parámetro es divisor del segundo.

```
//Un ejemplo siemple del uso de parámetros

class Divisor {

    //Retorna true si x es par
```

```
    boolean esDivisor(int a, int b){
        if ((b%a)==0) return true;
        else return false;
    }
}

class DivisorDemo {

    public static void main(String[] args) {
        Divisor x =new Divisor();

        if (x.esDivisor(2,20)) System.out.println("2 es Divisor de 20");
        if (x.esDivisor(3,20)) System.out.println("3 es Divisor de 20");
    }
}
```

Salida:

```
2 es Divisor de 20
```

Tenga en cuenta que cuando se llama a *esDivisor(),* los argumentos también están separados por comas. Al usar múltiples parámetros, **cada parámetro especifica su propio tipo**, que puede diferir de los demás.

Por ejemplo, esto es perfectamente válido:

```
int miMetodo(int a, double b, float c) {
// ...
```