Sobrecarga de métodos

Módulo 7

Capítulo 6

Clase Printer

Vamos a diseñar una clase que nos permita imprimir cualquier arreglo o matriz que reciba como parámetro. Empecemos con los siguientes métodos:

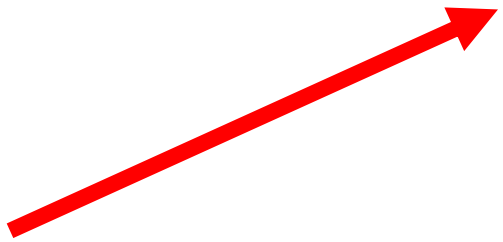
- `static printArrayOfInts(int[] array)`
- `static printMatrixOfInts(int[][] array)`

```java
public class Printer {

  public static void printArrayOfInts(int[] array) {
    for(int element: array) {
      System.out.print(element + "\t");
    }
  }

  public static void printMatrixOfInts(int[][] matrix) {
    for(int[] row: matrix) {
      Printer.printArrayOfInts(row);
      System.out.println(); //jump line after every row
    }
  }

  //main method
  public static void main(String[] args) {
    int[][] matrix = {{1,2},{3,4}};
    Printer.printMatrixOfInts(matrix);
  }
}
```



```
Output
1   2
3   4
```

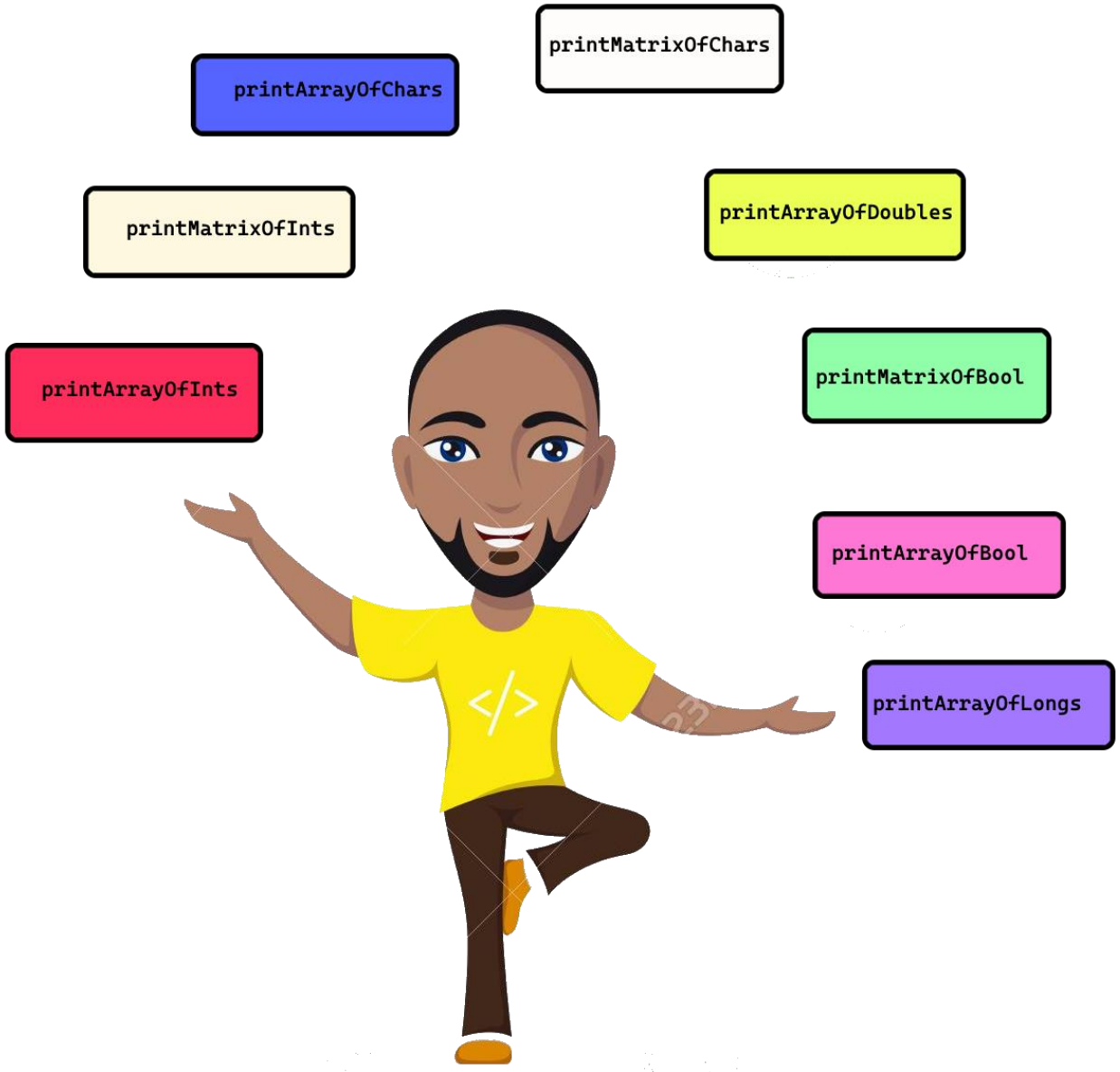
Clase Printer

Nuestra clase ahora imprime un arreglo o matriz de tipo `int[]`.

Pero si queremos imprimir `Strings, chars, double, etc?`

- `printArrayOfStrings`
- `printMatrixOfStrings`
- `printArrayOfChars`
- `printMatrixOfChars`
- `printArrayOfDoubles`
- `printMatrixOfDoubles`
- `… etc!`

Clase Printer



Ocuparíamos un programador que conozca los nombres de todos los métodos de la clase `Printer` para poder hacer uso de la clase.

¿Cuál es la solución?

Method Overloading

Method overloading es un concepto de la programación orientada a objetos que nos permite definir dos o más métodos con el mismo nombre en la misma clase, pero con diferente **firma**.

La firma de un método está compuesta por:

- Nombre del método
- Parámetros de entrada (orden y tipos de datos)

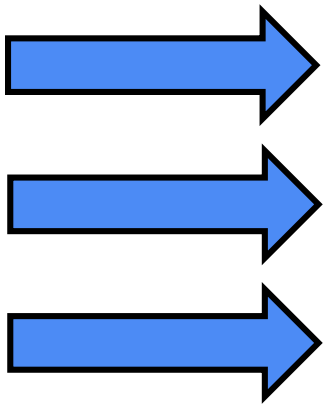
Sobrecarga de Métodos

```
public static void print(String[] array) {}
public static void print(String[][] matrix) {}
public static void print(int[] array) {}
public static void print(int[][] matrix) {}
public static void print(double[] array) {}
public static void print(double[][] matrix) {}
public static void print(char[] array) {}
public static void print(char[][] matrix) {}
```

Todos los métodos tienen el mismo nombre, pero están sobrecargados por sus parámetros de entrada!

Antes

```
public static void printArrayOfInts(int[] array) {
  for(int element: array) {
    System.out.print(element + "\t");
  }
}
```

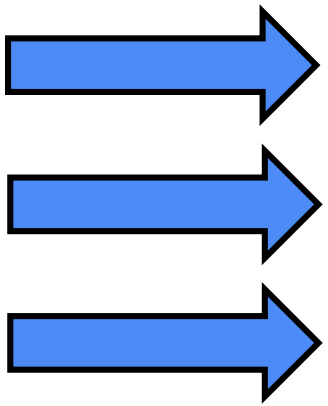


Después

```
public static void print(int[] array) {
  for(int element: array) {
    System.out.print(element + "\t");
  }
}
```

Antes

```
public static void printMatrixOfInts(int[][] matrix) {
  for(int[] row: matrix) {
    Printer.print(row);
    System.out.println();
  }
}
```



Después

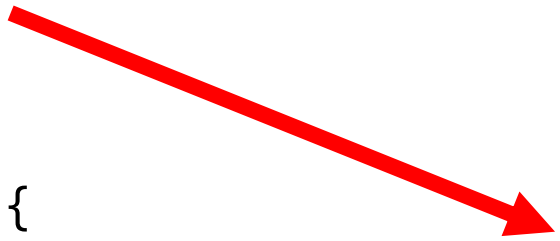
```
public static void print(int[][] matrix) {
  for(int[] row: matrix) {
    Printer.print(row);
    System.out.println();
  }
}
```

```java
public class Printer {

  //main method
  public static void main(String[] args) {
    int[][] matrix = {{1,2},{3,4}};
    Printer.print(matrix);
    System.out.println("---------");
    Printer.print(matrix[0]);
  }

  public static void print(int[] array) {
    for(int element: array) {
      System.out.print(element + "\t");
    }
  }

  public static void print(int[][] matrix) {
    for(int[] row: matrix) {
      Printer.printArrayOfInts(row);
      System.out.println();
    }
  }
}
```



Output

1 2

3 4

---

1 2

¿Cuáles son las ventajas?

- Manejar un solo nombre para todos los métodos `print()`.
- Simplificar la lectura y escritura del código.
- Simplificar la interfaz de los métodos públicos de la clase.

#1 Sobrecarga por parámetro de entrada

```java
public static void main(String[] args) {
  int out1 = add(1,1);   //out1 = 2
  int out2 = add(1,1,1); //out2 = 3
  int out3 = add(1.1, 1.9); //out3 = 3
}

//Sobrecargar por tipo de dato
public static int add(int num1, int num2) {
  return num1 + num2;
}
//Sobrecargar por cantidad de parámetros
public static int add(int num1, int num2, int num3) {
  return num1 + num2 + num3;
}
//Sobrecargar por tipos de dato
public static int add(double num1, double num2) {
  return (int)(num1 + num2);
}
```

#2 Sobrecarga por orden de los parámetros

```java
public static void main(String[] args) {
  String o1 = secretMethod("Hola", 1); //o1 = Hola1
  String o2 = secretMethod(1,"Hola");  //o2 = 1Hola
}

private static String secretMethod(String s1, int num1) {
  return s1 + num1;
}

private static String secretMethod(int num1, String s1) {
  return num1 + s1;
}
```

#3 Sobrecarga de Constructores

```
public class Pet{
  public Pet() {}
  public Pet(String initialName, int initialAge, double initialWeight) {}
  public Pet(String initialName) {}
  public Pet(int initialAge) {}
  public Pet(double initialWeight) {}

  public static void main(String[] args) {
    Pet aPet = new Pet();
    Pet myCat = new Pet("Fluffy", 2, 4.5);
    Pet myDog = new Pet("Spot");
    Pet myTurtle = new Pet(20);
    Pet myHorse = new Pet(750.5);
  }
}
```

NO se puede sobrecargar por nombre

```
private static int secretMethod(int num1) {
  return num1*10;
}

public static int secretMethod(int num2) {
  return "Hola" + num2;
}
```

Error!

El nombre de los parámetros de entrada no es suficiente para sobrecargar un método.

NO se puede sobrecargar por tipo de método

```java
private static double secretMethod(int num1) {
    return num1*10;
}

public static String secretMethod(int num1) {
    return "Hola" + num1;
}

public String secretMethod(int num1) {
    return "Hola" + num1;
}
```

Error!

El modificador de acceso, ser estático vs de instancia o el valor de retorno no son suficientes para sobrecargar un método!

NO se puede sobrecargar por valor de retorno

```
private static double secretMethod(int num1) {
  return num1*10;
}

private static String secretMethod(int num1) {
  return "Hola" + num1;
}
```

Dos métodos con la misma firma

Error!