Republic of the Philippines
Aurora State College of Technology
DEPARTMENT OF INFORMATION TECHNOLOGY
Zabali, Baler, Aurora, 3200

PROGRAMMING II

JAVA METHODS

ROSE ANN L. BUENCONSEJO

OUTLINE

- Java Methods
- Java Method Parameter
- Java Method Overloading

Java Methods

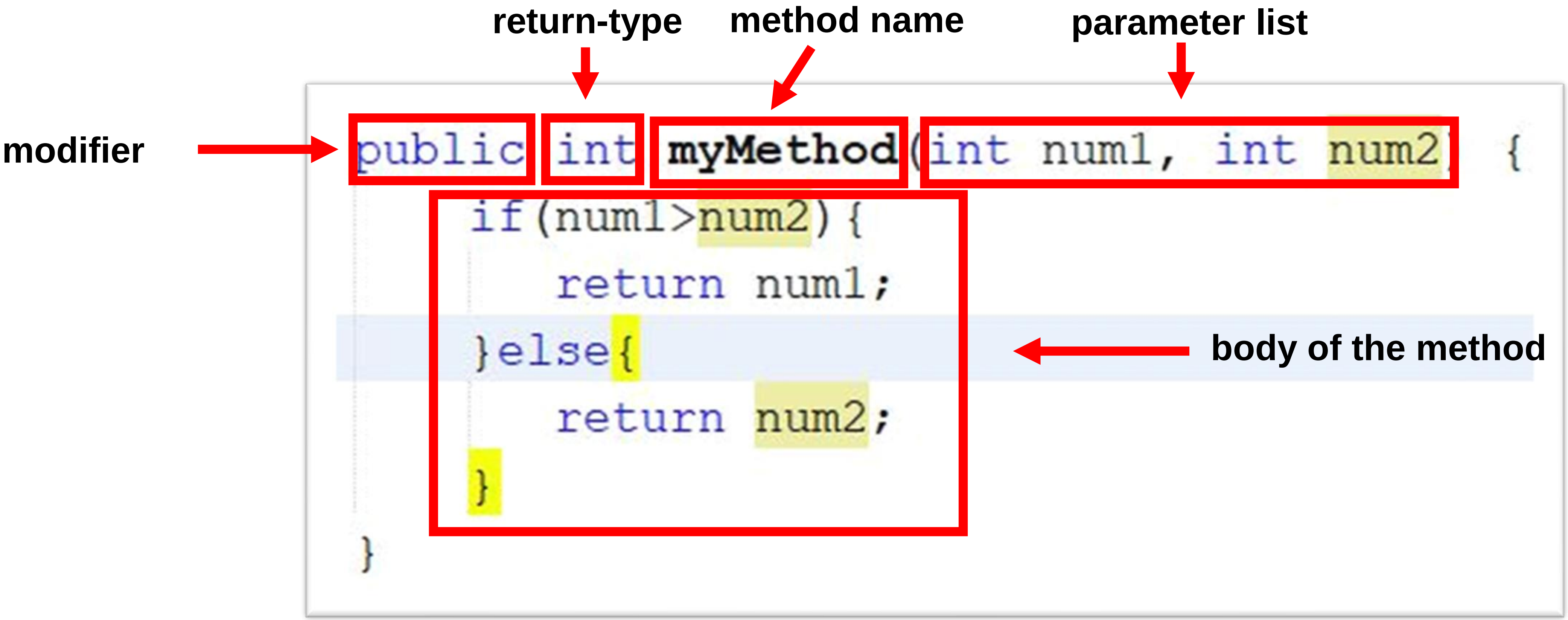
- **Method**
  - a block of code which only runs when it is called.
  - used to perform certain actions, and they are also known as **functions**.
  - Why use methods? **To reuse code**: define the code once, and use it many times.

Java Methods

- **Method**
  - must declare within a class.
  - it is defined with the name of the method, followed by parentheses ().

Java Methods

- Method



return-type

method name

parameter list

modifier

```
public int myMethod(int num1, int num2) {
    if(num1>num2){
        return num1;
    }else{
        return num2;
    }
}
```

body of the method

Java Methods

- **Method**
  - **Modifier** - Defines access type of the method i.e. from where it can be accessed in your application.
    - **public:** accessible in all class in your application.
    - **protected:** accessible within the class in which it is defined and in its subclass(es)
    - **private:** accessible only within the class in which it is defined.
    - **default (declared/defined without using any modifier)** : accessible within same class and package within which its class is defined.

Java Methods

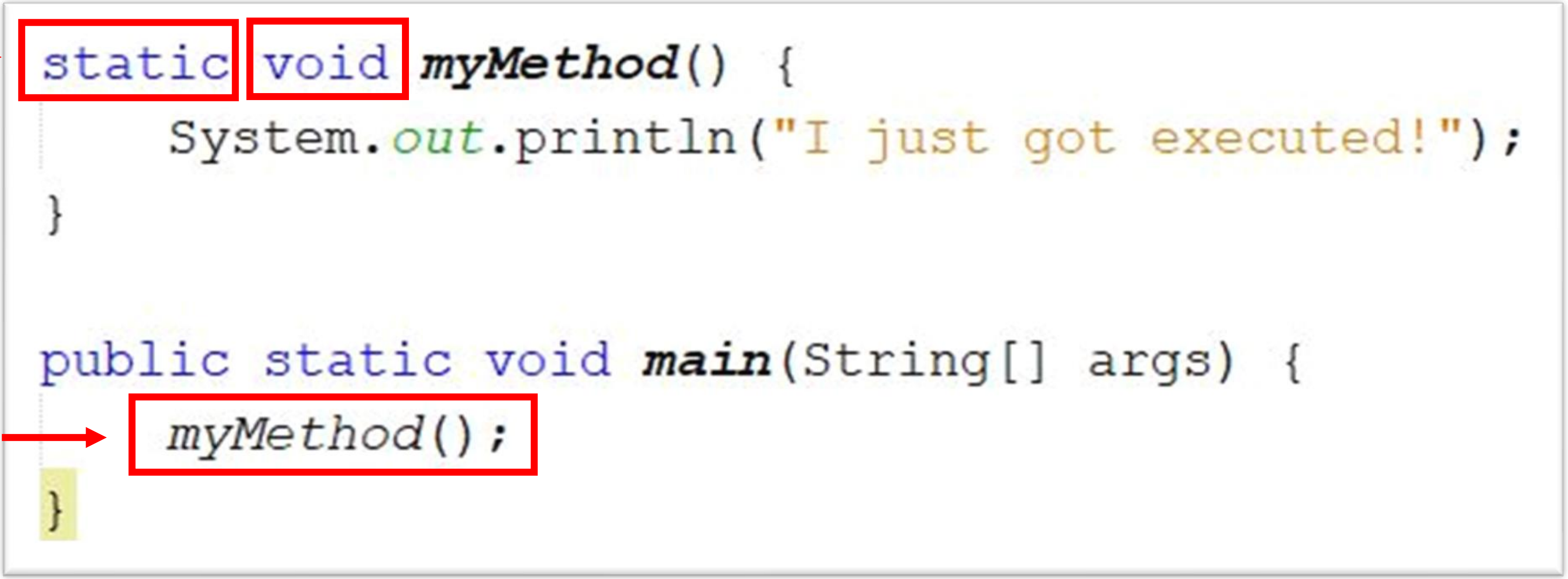
- **Method**
  - **The return type -** The data type of the value returned by the method or void if does not return a value.
  - **Method Name -** the rules for field names apply to method names as well, but the convention is a little different.
  - **Parameter list -** Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
  - **Method body -** it is enclosed between braces. The code you need to be executed to perform your intended operations.

Java Methods

- Method

void

static



```
static void myMethod() {
    System.out.println("I just got executed!");
}

public static void main(String[] args) {
    myMethod();
}
```

Call method

Java Methods

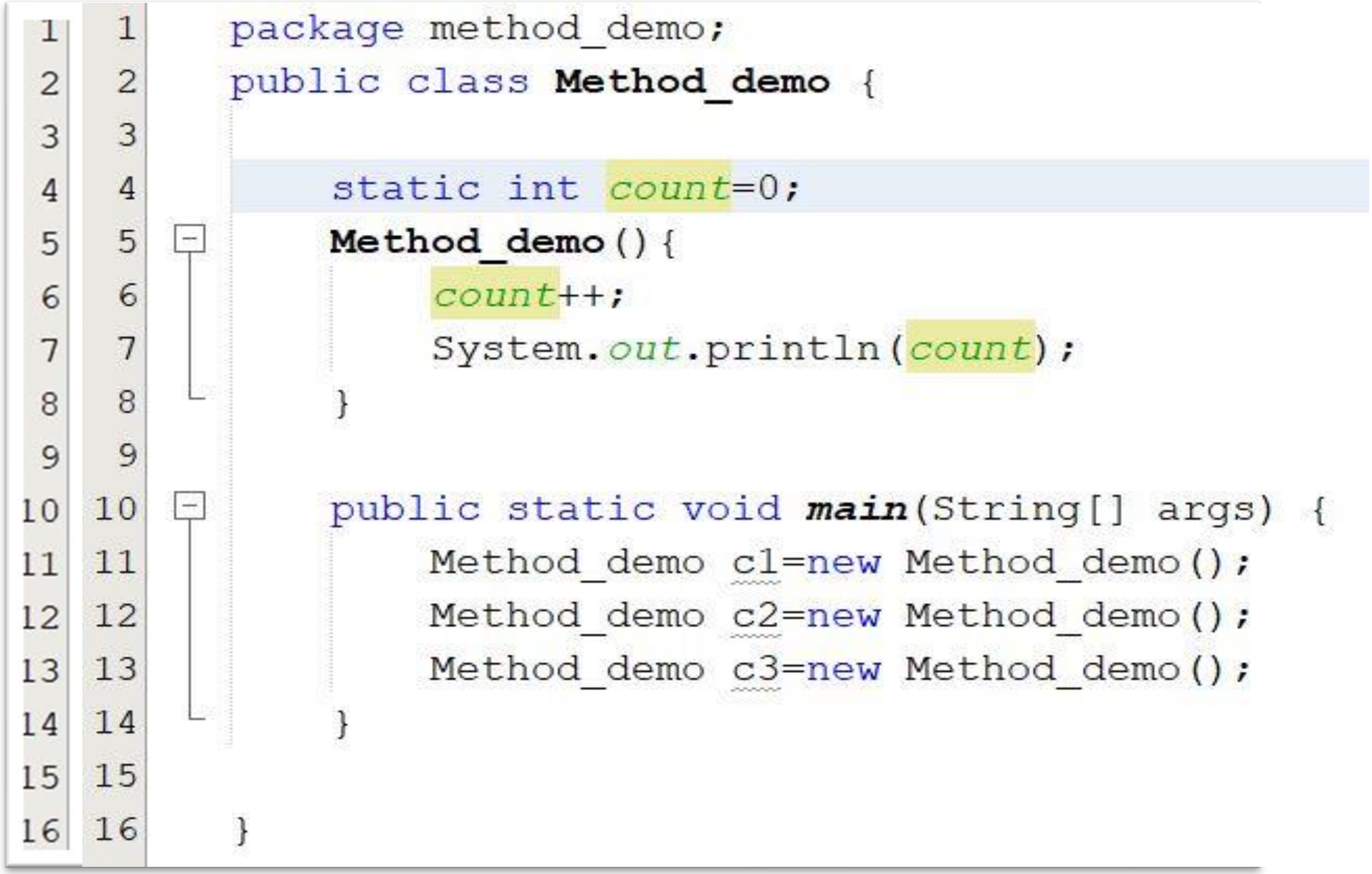
- **Method**
  - **void -** means that this method does not have a return value.
  - **static -** means that the method belongs to the **MyClass** class and not an object of the **MyClass** class.
  - **to call a method** in Java, write the method's name followed by two parentheses () and a semicolon ;

Java Methods

- **Method**
  - **To call a method (cont.)**
    - write the method's name followed by two parentheses () and a semicolon ;
    - The method needs to be called for using its **functionality**. There can be three situations when a method is called:
    - A method returns to the code that invoked it when:
      - It completes all the statements in the method
      - It reaches a return statement
      - Throws an exception.

Java Methods

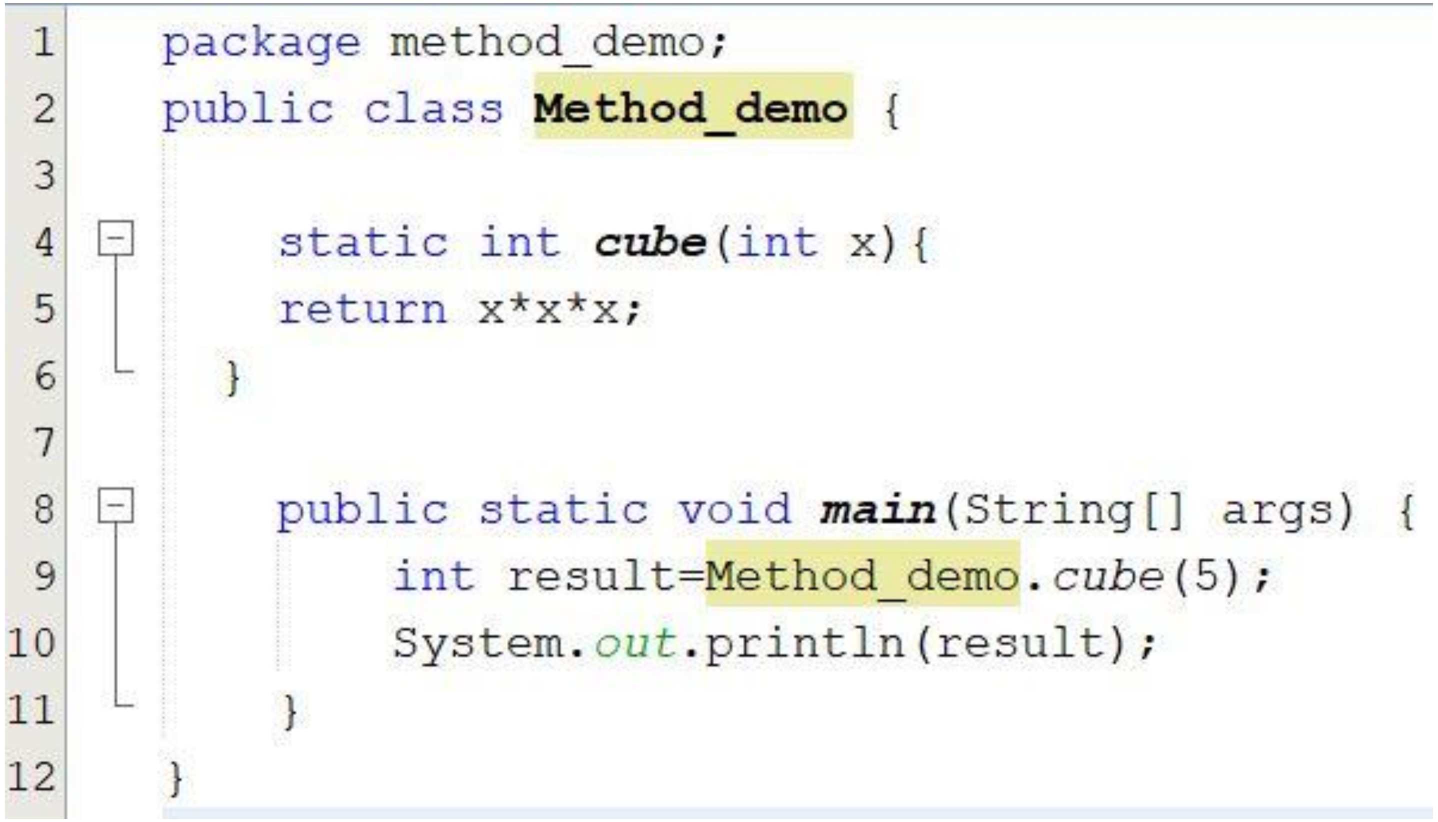
- **Method**
  - **Example:**



```java
package method_demo;
public class Method_demo {

    static int count=0;
    Method_demo(){
        count++;
        System.out.println(count);
    }

    public static void main(String[] args) {
        Method_demo c1=new Method_demo();
        Method_demo c2=new Method_demo();
        Method_demo c3=new Method_demo();
    }

}
```

Java Methods

- Method
  - Example:
    - example1



```
package method_demo;
public class Method_demo {

    static int cube(int x){
    return x*x*x;
  }

    public static void main(String[] args) {
        int result=Method_demo.cube(5);
        System.out.println(result);
    }
}
```

Java Methods

- **Method**
- **Q) Why is the Java main method static?**
- Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.
- **Q) Can we execute a program without main() method?**
- Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the main method.

Java Methods

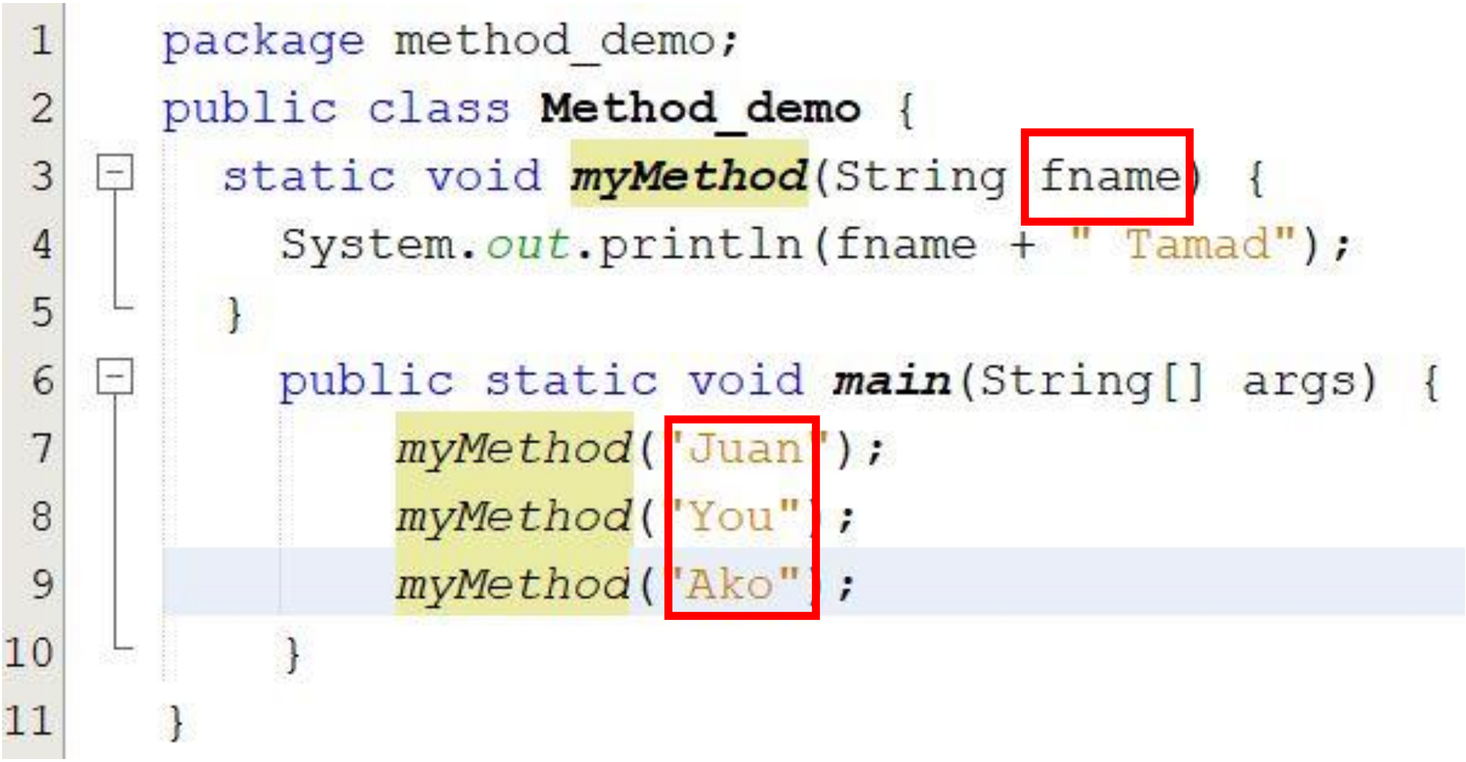
- **Java Method Parameters**
  - Information can be passed to methods as **parameter**.
  - Parameters act as variables **inside the method**.
  - Parameters are specified **after the method name**, inside the parentheses.
  - You can **add as many parameters as you want**, just separate them with a comma.

Java Methods

Java Method Parameters

- **Example:**

When a parameter is passed to the **method**, it is called an **argument**. So, from the example above: **fname** is a parameter, while **Juan**, **You** and **Ako** are arguments.



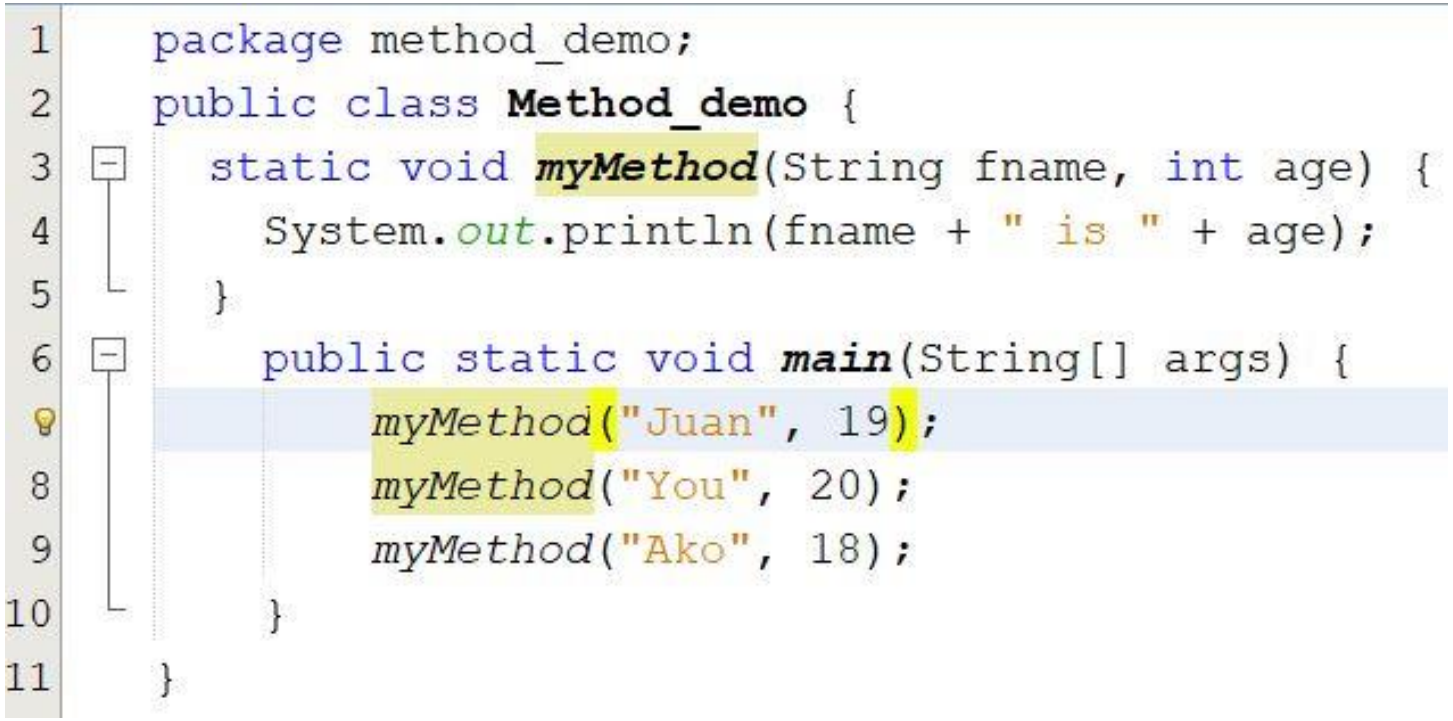
```
package method_demo;
public class Method_demo {
  static void myMethod(String fname) {
    System.out.println(fname + " Tamad");
  }
    public static void main(String[] args) {
        myMethod("Juan");
        myMethod("You");
        myMethod("Ako");
    }
}
```

Java Methods

- **Java Method Parameters**

- **Example:**
  - **Multiple Parameters**

Note: that when you are working with multiple parameters, the method call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.



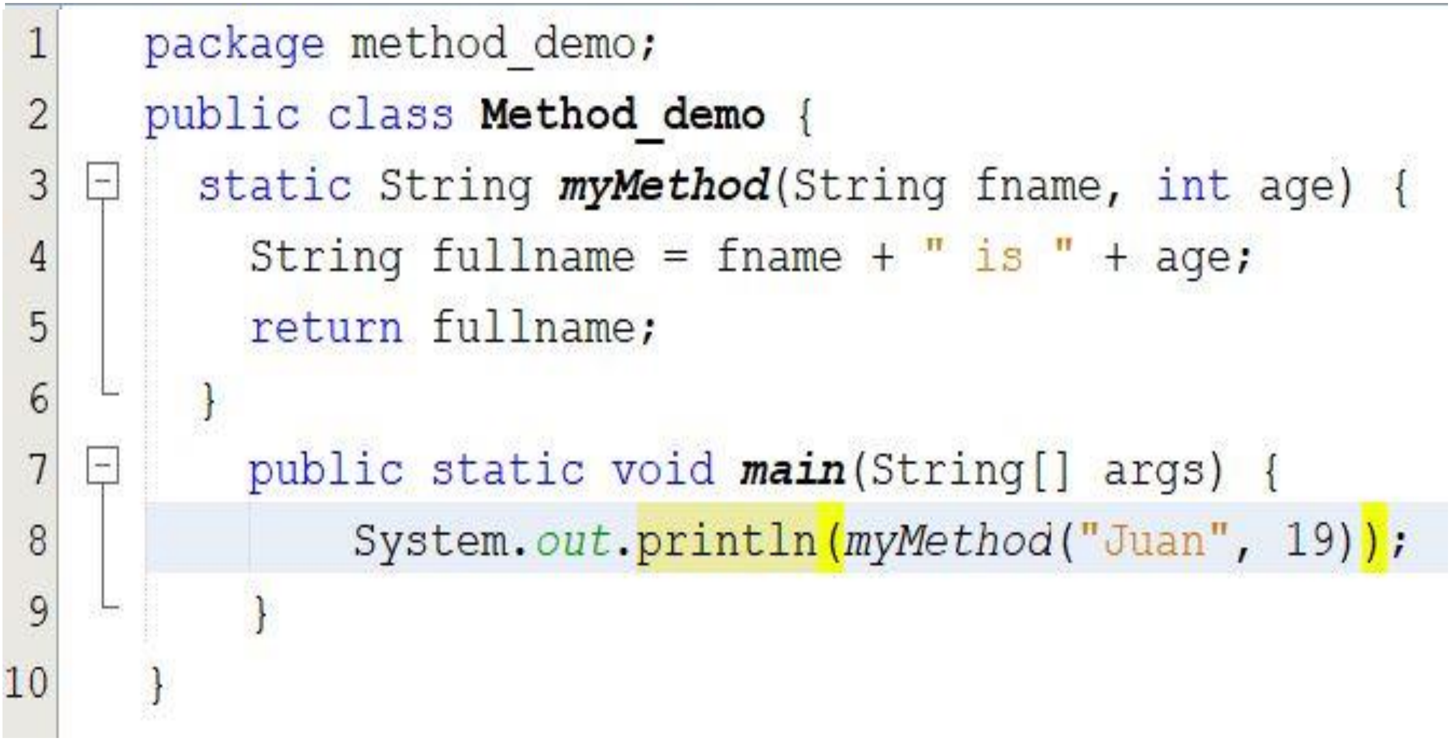
```java
package method_demo;
public class Method_demo {
  static void myMethod(String fname, int age) {
    System.out.println(fname + " is " + age);
  }
    public static void main(String[] args) {
        myMethod("Juan", 19);
        myMethod("You", 20);
        myMethod("Ako", 18);
    }
}
```

Java Methods

• Java Method Parameters

- Return Values

If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method



```java
package method_demo;
public class Method_demo {
  static String myMethod(String fname, int age) {
    String fullname = fname + " is " + age;
    return fullname;
  }
    public static void main(String[] args) {
        System.out.println(myMethod("Juan", 19));
    }
}
```

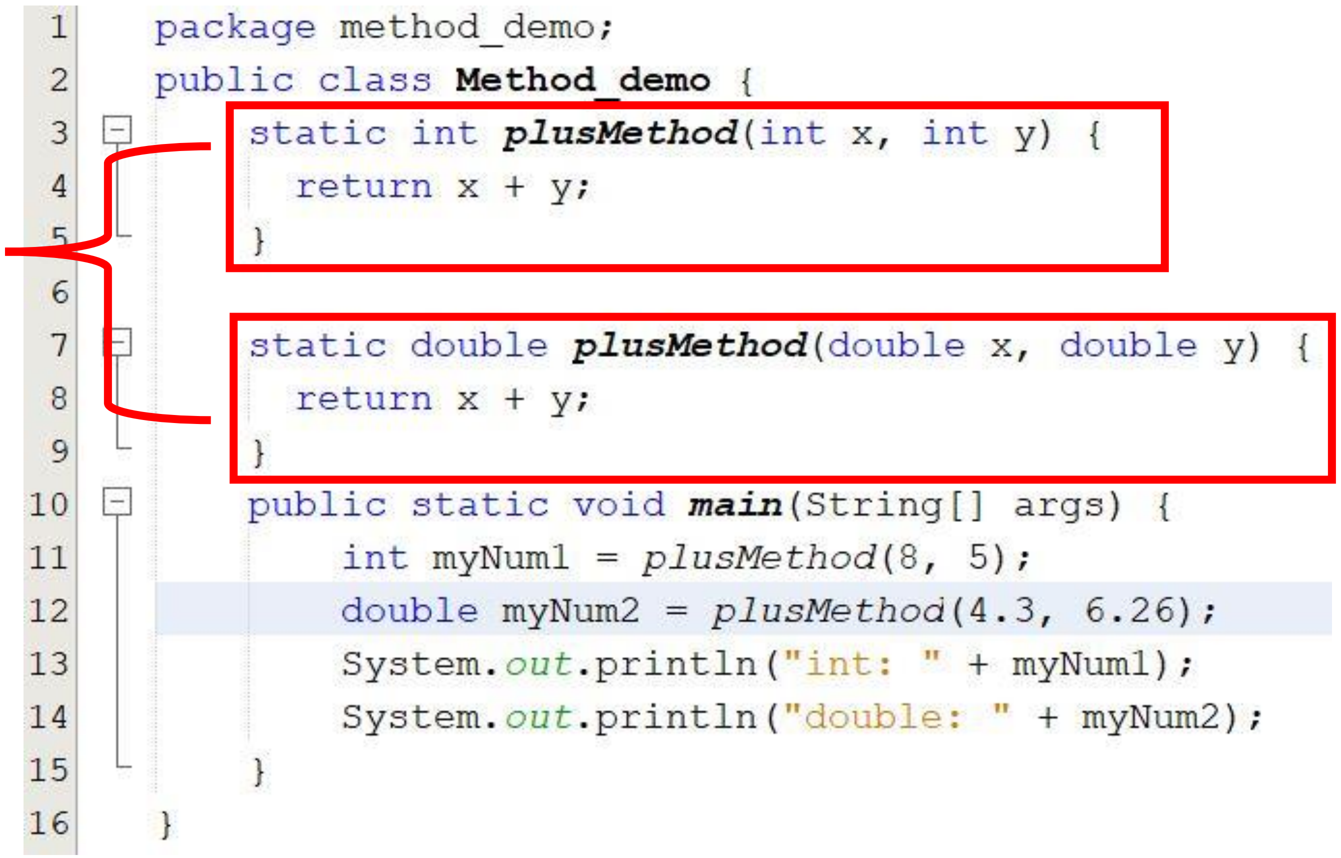
Java Methods

- **Method Overloading**

- With method overloading, multiple methods can have the same name with different parameters:

- Instead of defining two methods that should do the same thing, it is better to overload one.

Note: Multiple methods can have the same name as long as the number and/or type of parameters are different.



```java
package method_demo;
public class Method_demo {
    static int plusMethod(int x, int y) {
      return x + y;
    }

    static double plusMethod(double x, double y) {
      return x + y;
    }
    public static void main(String[] args) {
        int myNum1 = plusMethod(8, 5);
        double myNum2 = plusMethod(4.3, 6.26);
        System.out.println("int: " + myNum1);
        System.out.println("double: " + myNum2);
    }
}
```