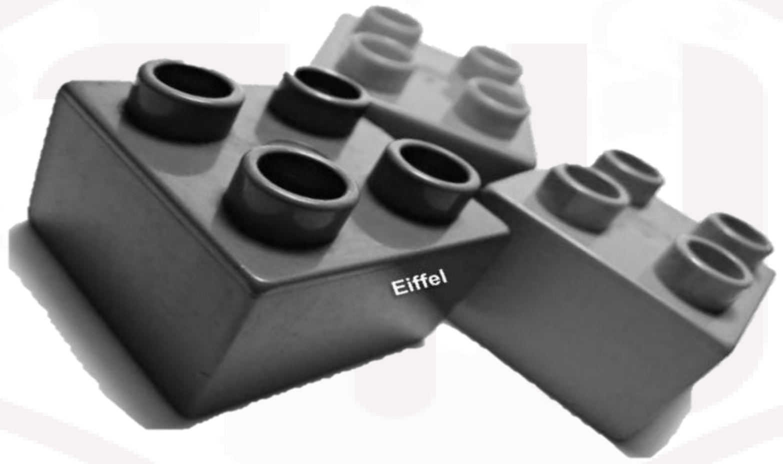
Methods in JAVA



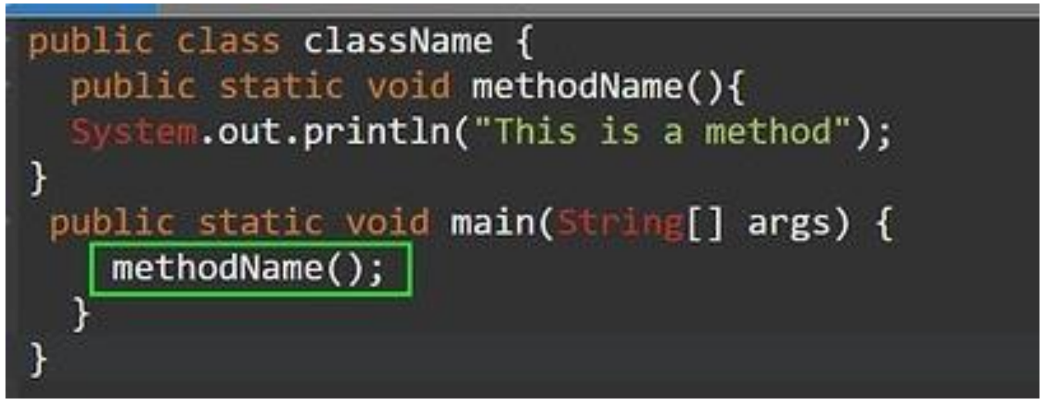
Computer Engineering

Yusramohammed@tiu.edu.iq

2022 - 2023

User-Defined Method

- The method written by the user or programmer is known as a **user-defined** method. These methods are modified according to the requirement.



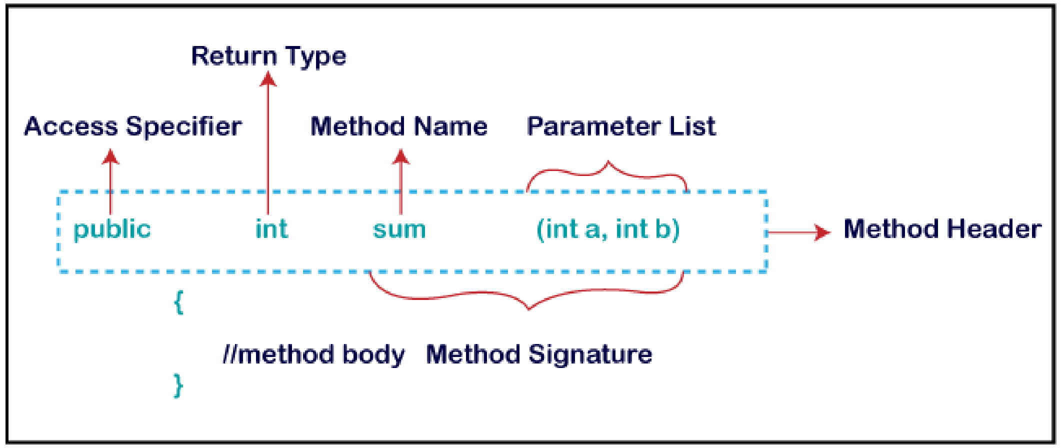
```
public class className {
  public static void methodName(){
  System.out.println("This is a method");
}
 public static void main(String[] args) {
    methodName();
  }
}
```

- The User Defined Methods are created in a class outside the main method. They can be called(invoked) from the main method to perform their required tasks.
- There are two ways to use methods in our projects:
  1. Create methods in the same class and call them in that class.
  2. Create methods in different classes and call them using objects. (OOP).

User-Defined Method

- **Method Signature:** Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.
- **Access Modifier:** Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier: public, private, protected, and default.

Method Declaration



- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

User-Defined Method

- **Return Type:** Return type is a data type that the method returns. If the method does not return anything, we use void keyword.

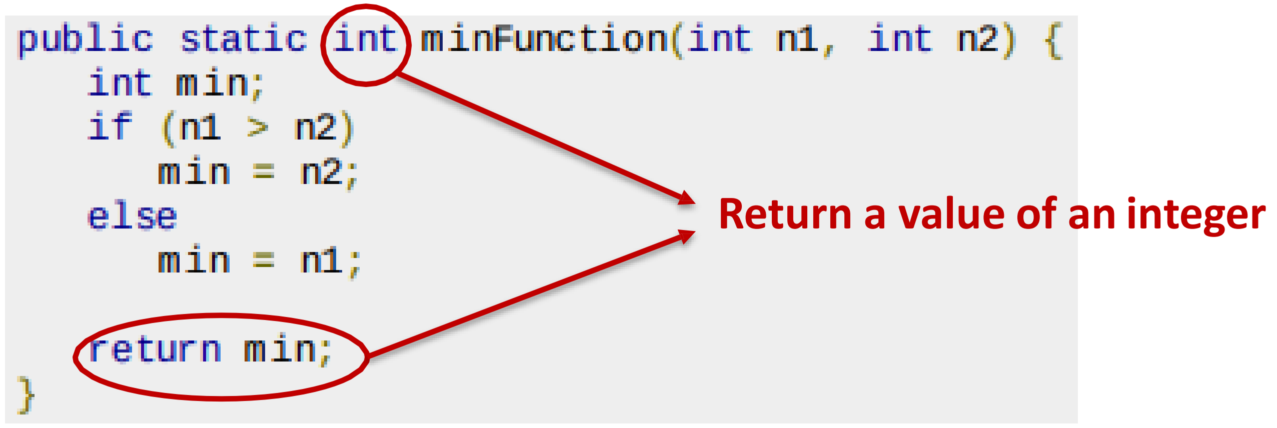
Method Type	Return Type	Example
void	Return nothing	public void display()
int	Return **int** value	public int display()
double	Return **double** value	Public double display()
String	Return a **String** statement	Public String display()
float	Return a **float** value	Public float display()
boolean	Return **true** or **false** value	Public Boolean display()

Method Return Value

- When a Method return a value after performing its task, the type of the method should be as the type of the value. We use:

```
return expression;
```

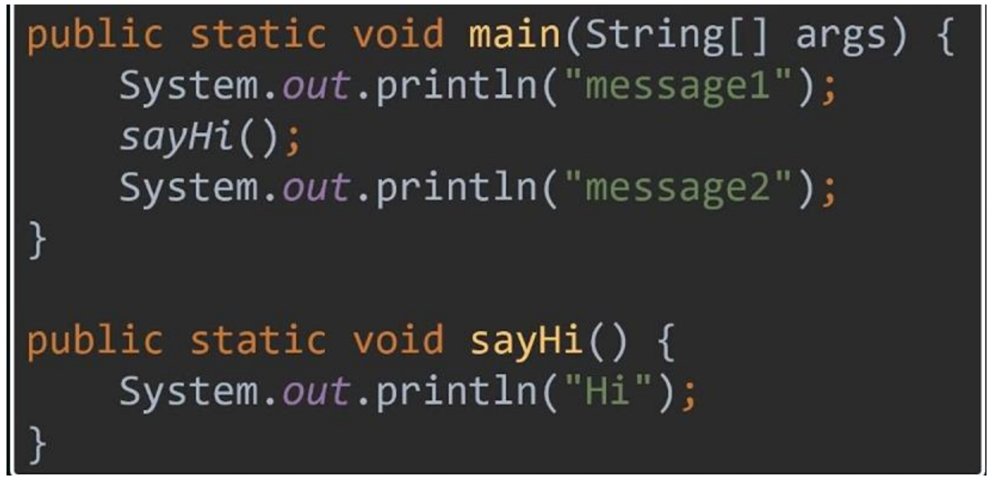
- **Example**:



```
public static int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

Return a value of an integer

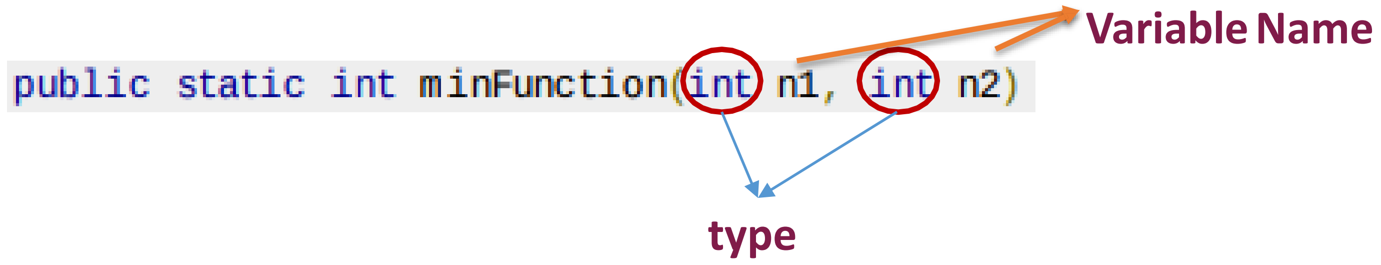
- When a Method doesn't return a value after performing its task, the type of the is ***void***



```
public static void main(String[] args) {
    System.out.println("message1");
    sayHi();
    System.out.println("message2");
}

public static void sayHi() {
    System.out.println("Hi");
}
```

User-Defined Method

- **Parameter List:**
  - Information can be passed to methods as parameter.
  - It is an additional information sent to a method to perform a task.
  - A method can require one or more parameters that represent additional information it needs to perform its task.
  - Parameters are defined in a comma-separated **parameter list**
  - Each parameter must specify a *type* and a variable *name*.

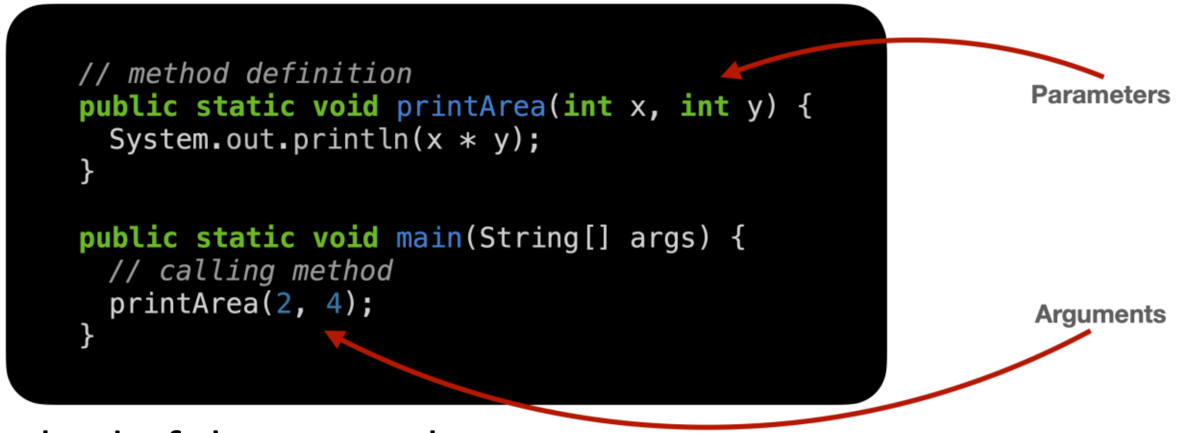


```
public static int minFunction(int n1, int n2)
```

- **Method Body:** It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Calling User Defined Methods

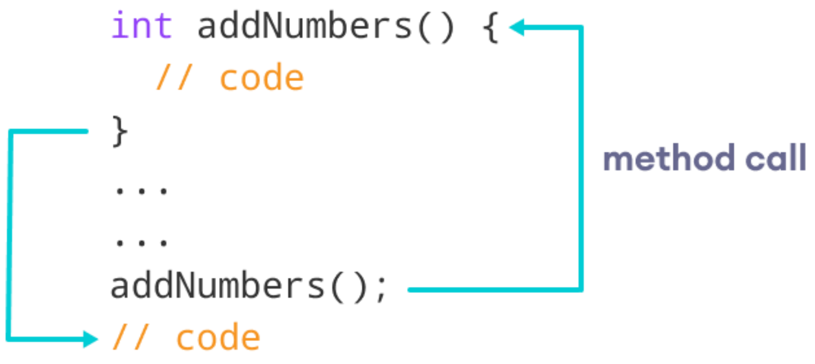
- Parameters are the variables we use in the method definition whereas arguments are the values we pass in the method call.



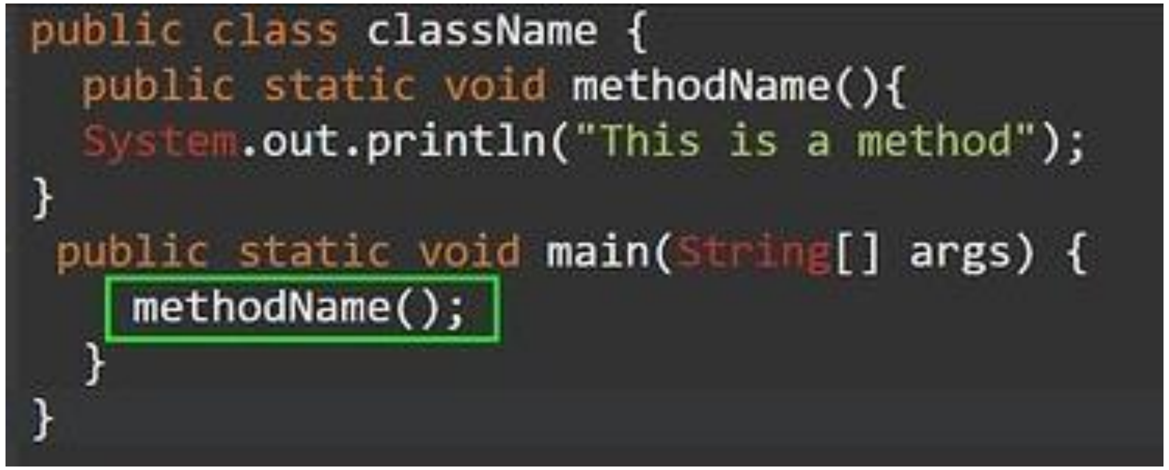
```java
// method definition
public static void printArea(int x, int y) {
  System.out.println(x * y);
}

public static void main(String[] args) {
  // calling method
  printArea(2, 4);
}
```

- There are three ways to call a method:

1. Using a method name by itself to call another method of the *same* class
   - ***NameOfMethod(parameterList);***



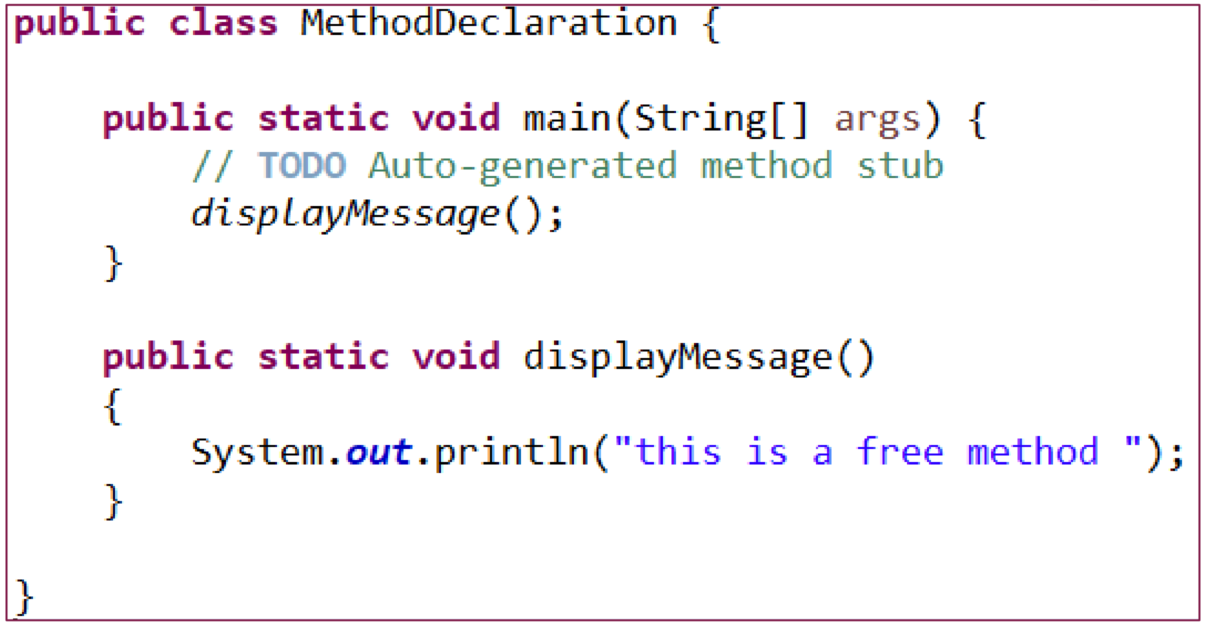
```java
int addNumbers() {
  // code
}
...
...
addNumbers();
// code
```



```java
public class className {
  public static void methodName(){
  System.out.println("This is a method");
 }
 public static void main(String[] args) {
    methodName();
   }
}
```

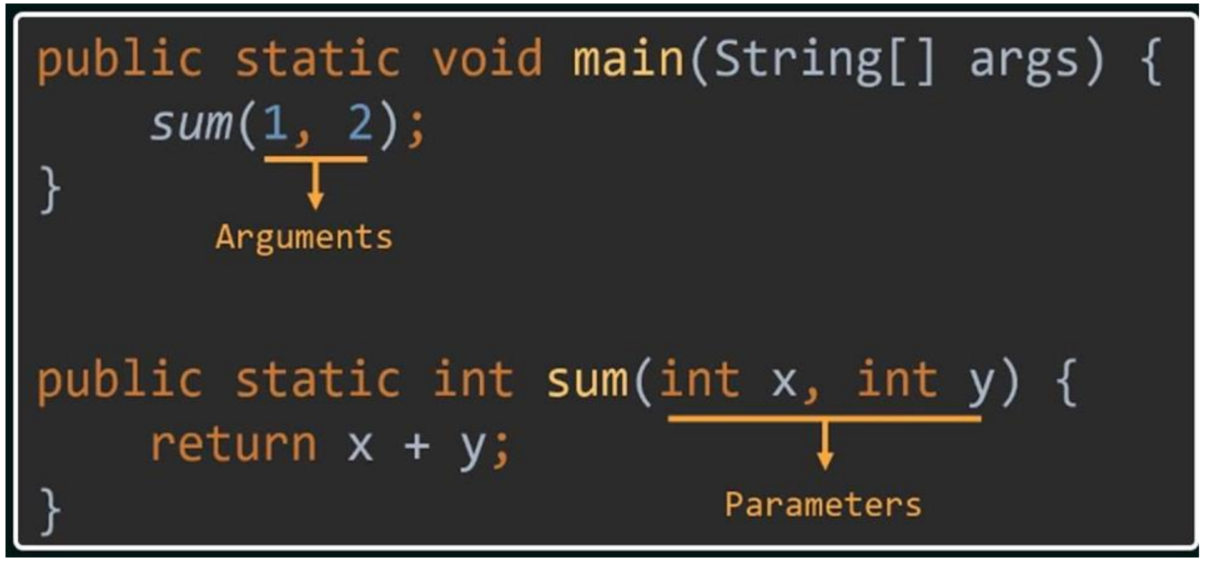
Calling User Defined Methods

- If the method doesn't have parameter list. It can be called using only the method name followed by the empty parentheses.



```java
public class MethodDeclaration {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        displayMessage();
    }

    public static void displayMessage()
    {
        System.out.println("this is a free method ");
    }

}
```

- If a method is created with parameters, we need to pass the corresponding values while calling the method. For example,



```java
public static void main(String[] args) {
    sum(1, 2);
}

public static int sum(int x, int y) {
    return x + y;
}
```

Calling User Defined Methods

2. Using the class name and a dot (.) to call a static method of a class—such

- as ***Math.sqrt(900.0)***

3. Using a variable that contains a reference to an object, followed by a dot (.) and the method name to call a non-static method of the referenced object ***(OOP)***

Methods hierarchy in the same class

Without Parameter

- Method declaration

```
public static void displayMessage()
```

- Method Calling

```
displayMessage();
```

- Return Value: the type of method is ***void.*** So there is no return value.

NO Parameter list

```
dpublic static void displayMessage()
 {
     System.out.println("this is a free method ");
 }
```

With Parameter List

- Method declaration

```
public static void CourseName(String name)
```

- Method Calling

```
CourseName(yourName);
```

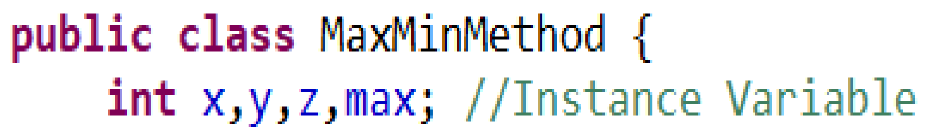
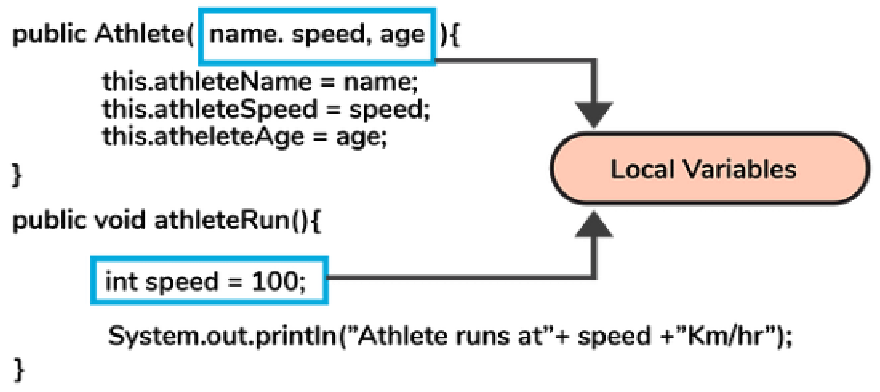
- Return Value
  1. No return value if the type is ***void***
  2. return the type of the method

Parameter list

```
public static void CourseName(String name)
{

    System.out.println("Welcome to OOP class "+name);
}
```

Local and Instance Variables

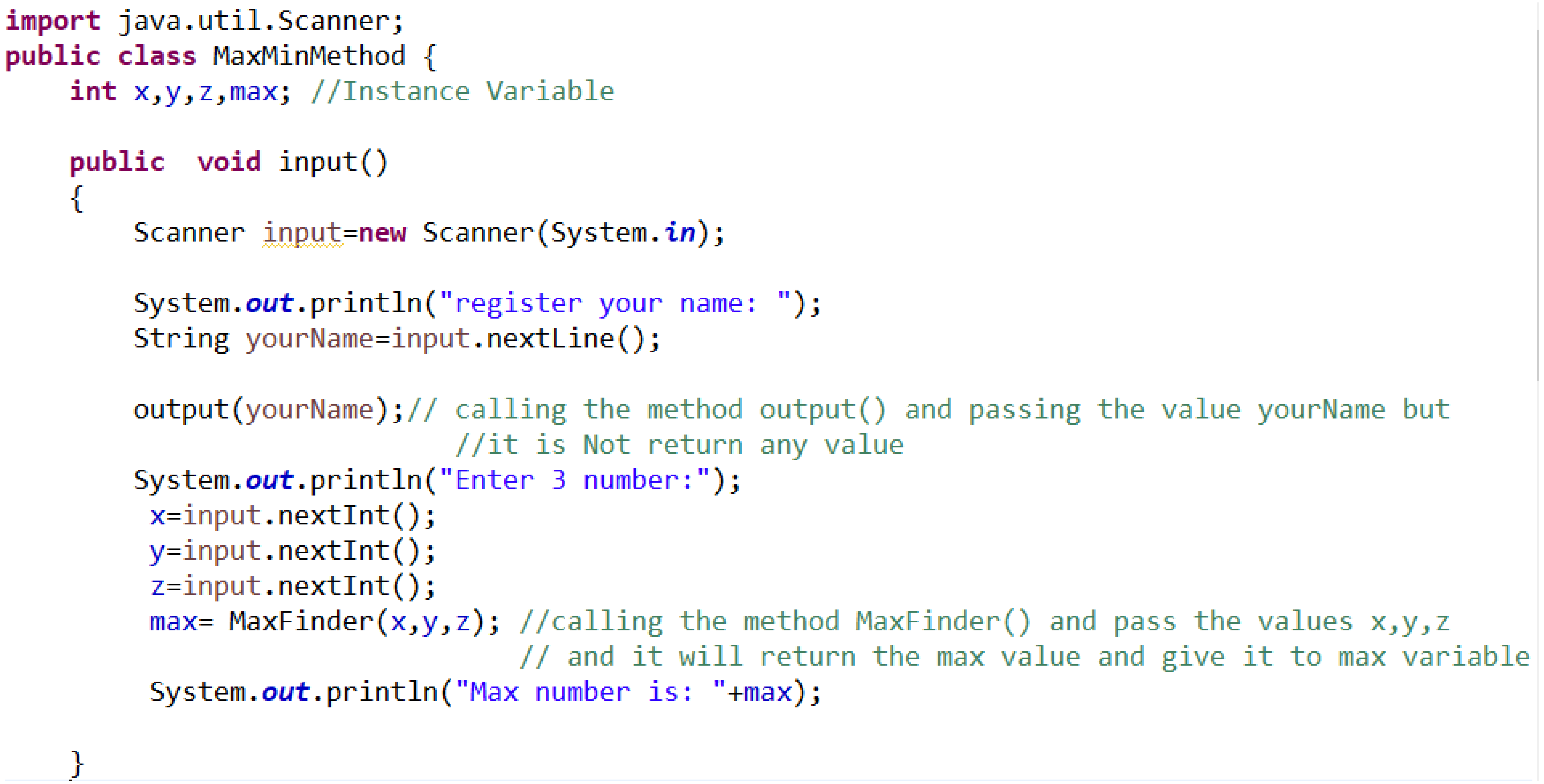
- There are two types of variables in a class

Instance Variable	Local variable
• Declared inside the class and outside the method. • It can be used in all the methods of the class. • It is declared immediately when the body of the class open. 	• Declared inside the method of the class. • It can only used inside the method declared not outside the method. • It is declared immediately when the body of the method open. 

```
public class MaxMinMethod {
    int x,y,z,max; //Instance Variable
```

```
public Athlete( name. speed, age ){
        this.athleteName = name;
        this.athleteSpeed = speed;
        this.atheleteAge = age;
}
public void athleteRun(){

        int speed = 100;

         System.out.println("Athlete runs at"+ speed +"Km/hr");
}
```

Sample Development


```java
import java.util.Scanner;
public class MaxMinMethod {
    int x,y,z,max; //Instance Variable

    public  void input()
    {
        Scanner input=new Scanner(System.in);

        System.out.println("register your name: ");
        String yourName=input.nextLine();

        output(yourName);// calling the method output() and passing the value yourName but
                            //it is Not return any value
        System.out.println("Enter 3 number:");
         x=input.nextInt();
         y=input.nextInt();
         z=input.nextInt();
         max= MaxFinder(x,y,z); //calling the method MaxFinder() and pass the values x,y,z
                                // and it will return the max value and give it to max variable
         System.out.println("Max number is: "+max);

    }
```

Sample Development

```
    public  int MaxFinder(int a, int b, int c)//method of type int and return a value
    {
        if(a>b&&a>c)
            return a; //return a value
        else if(b>a&&b>c)
            return b; //return a value
        else
            return c; //return a value
    }

    public void output(String name)// method of type void and return only an output statement
    {
        System.out.println("Welcome to the Math class "+name+"\n"
                + "this class is to find maximum number\n");
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        MaxMinMethod mx=new MaxMinMethod();  //Object of the class MAxMinMethod
        mx.input();  //calling the input() method using the object mx
    }
}
```

static Variable & ***static*** Methods

- ***static*** **Variable:**

- Every object has its own copy of all the instance variables of the class. In such cases, a ***static*** field— called a class variable—is used.
- Static variables belong to the class. It is called class variables.
- The declaration of a static variable begins with the keyword static.

Sometimes No need to the Access Modifier

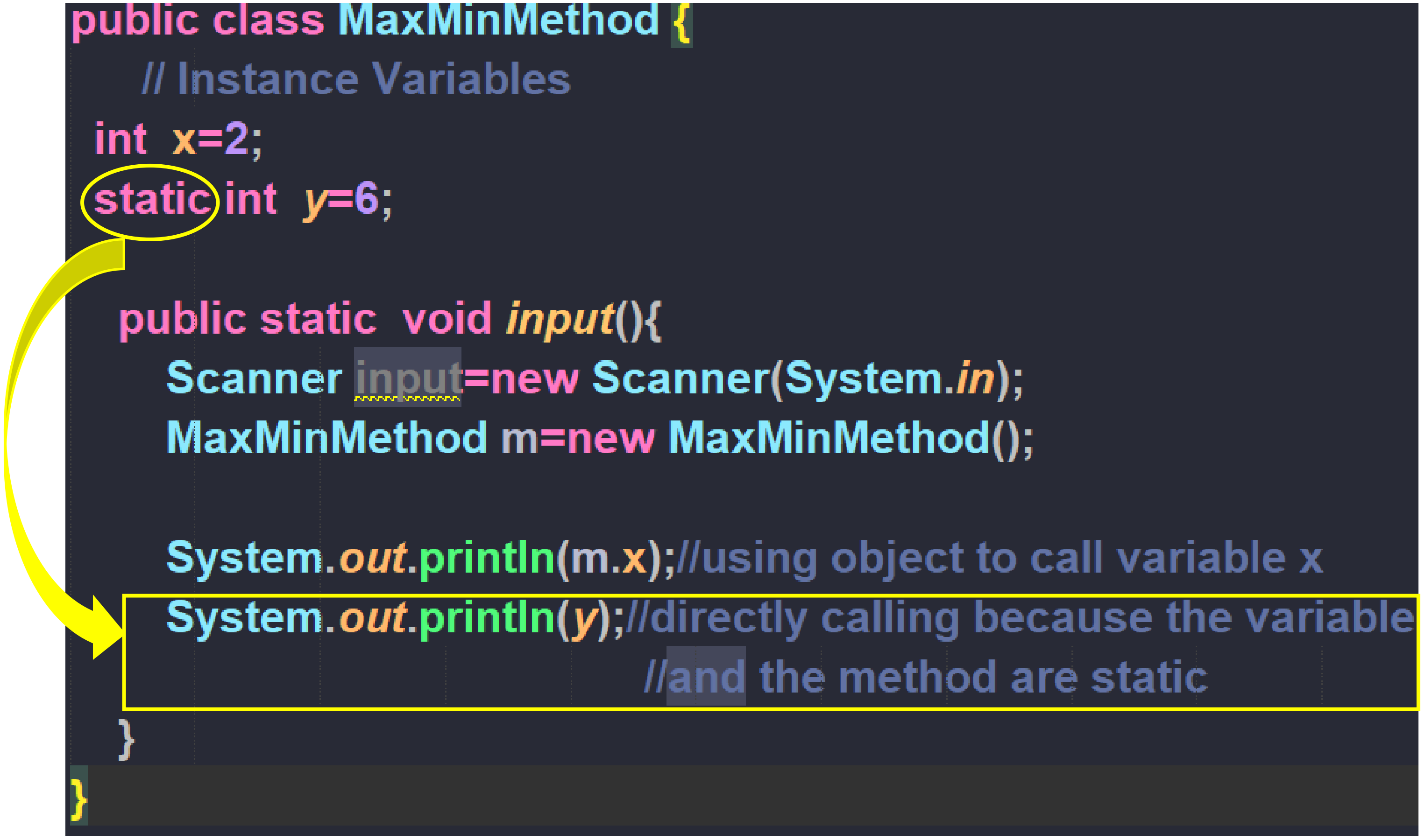
AccessModifier *static* type name;

- A static Variables can call *only* other static methods of the same class using the variable. If the variable is not static we create an object of type class and call the variable in the class.

ClassName objectName=new ClassName();

objectName.VariableName

Example



```
public class MaxMinMethod {
    // Instance Variables
 int  x=2;
 static int  y=6;

   public static  void input(){
      Scanner input=new Scanner(System.in);
      MaxMinMethod m=new MaxMinMethod();

      System.out.println(m.x);//using object to call variable x
      System.out.println(y);//directly calling because the variable
                                     //and the method are static
   }
}
```

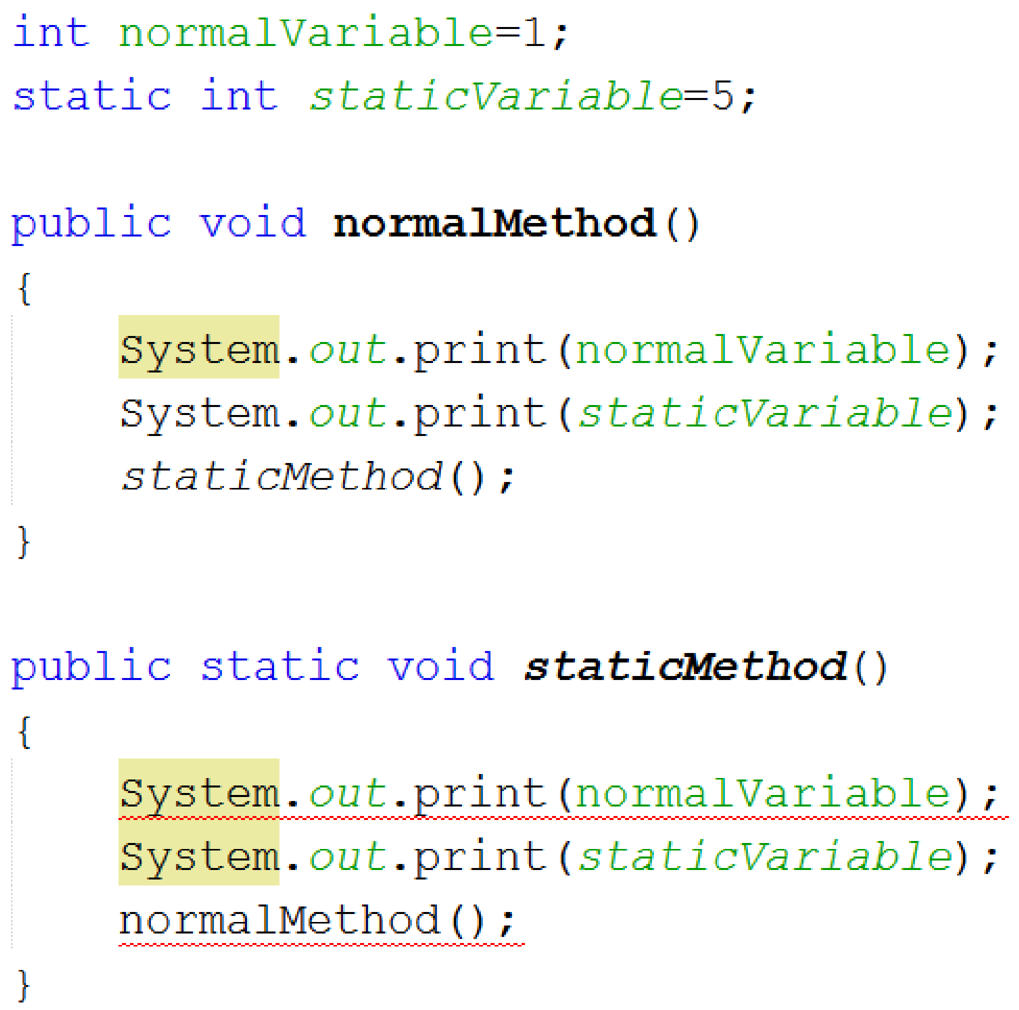
static Methods

- ***static* Method:**

- A static method can call *only* other static methods of the same class using the method name by itself and can manipulate *only* static variables in the same class directly. If the method is not static we create an object of type class and call the methods in the class.

  - **A *non-static* method can reference a static variable or static method.**

  - **A *static* method can reference a static variable but not an instance variable directly. We should use objects**

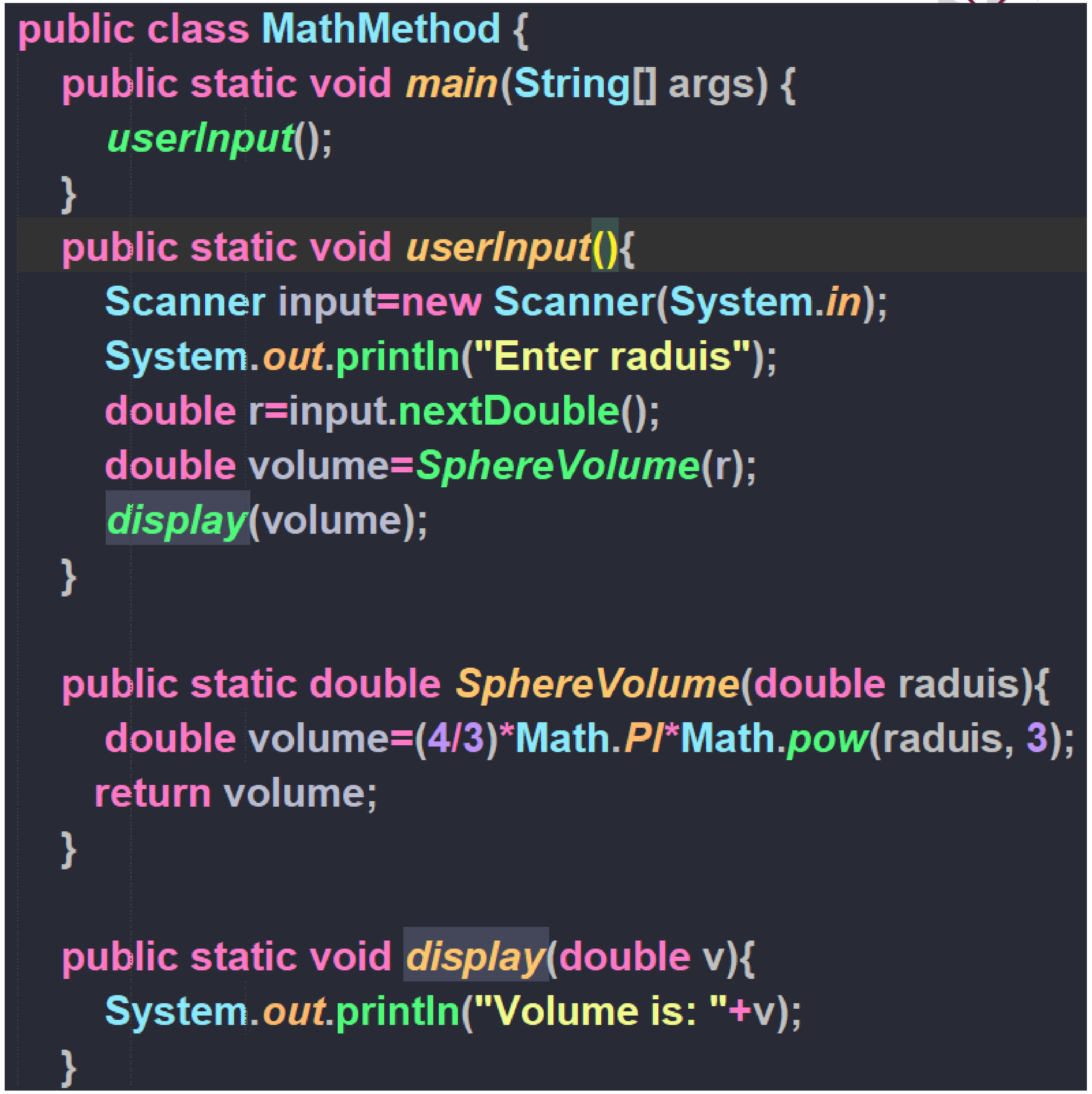


```
int normalVariable=1;
static int staticVariable=5;

public void normalMethod()
{
    System.out.print(normalVariable);
    System.out.print(staticVariable);
    staticMethod();
}

public static void staticMethod()
{
    System.out.print(normalVariable);
    System.out.print(staticVariable);
    normalMethod();
}
```

Sample Development

Write a complete Java application to prompt the user for the ***double radius*** of a sphere, and call method ***sphereVolume*** to calculate and return its value to the method ***display*** to show the volume of the sphere. Use the following statement to calculate the volume:

$$double\ volume = \left({}^{4}\!/_{3}\right) * Math.PI * MAth.pow(raduis, 3)$$



```java
public class MathMethod {
    public static void main(String[] args) {
        userInput();
    }
    public static void userInput(){
        Scanner input=new Scanner(System.in);
        System.out.println("Enter raduis");
        double r=input.nextDouble();
        double volume=SphereVolume(r);
        display(volume);
    }

    public static double SphereVolume(double raduis){
        double volume=(4/3)*Math.PI*Math.pow(raduis, 3);
       return volume;
    }

    public static void display(double v){
        System.out.println("Volume is: "+v);
    }
```