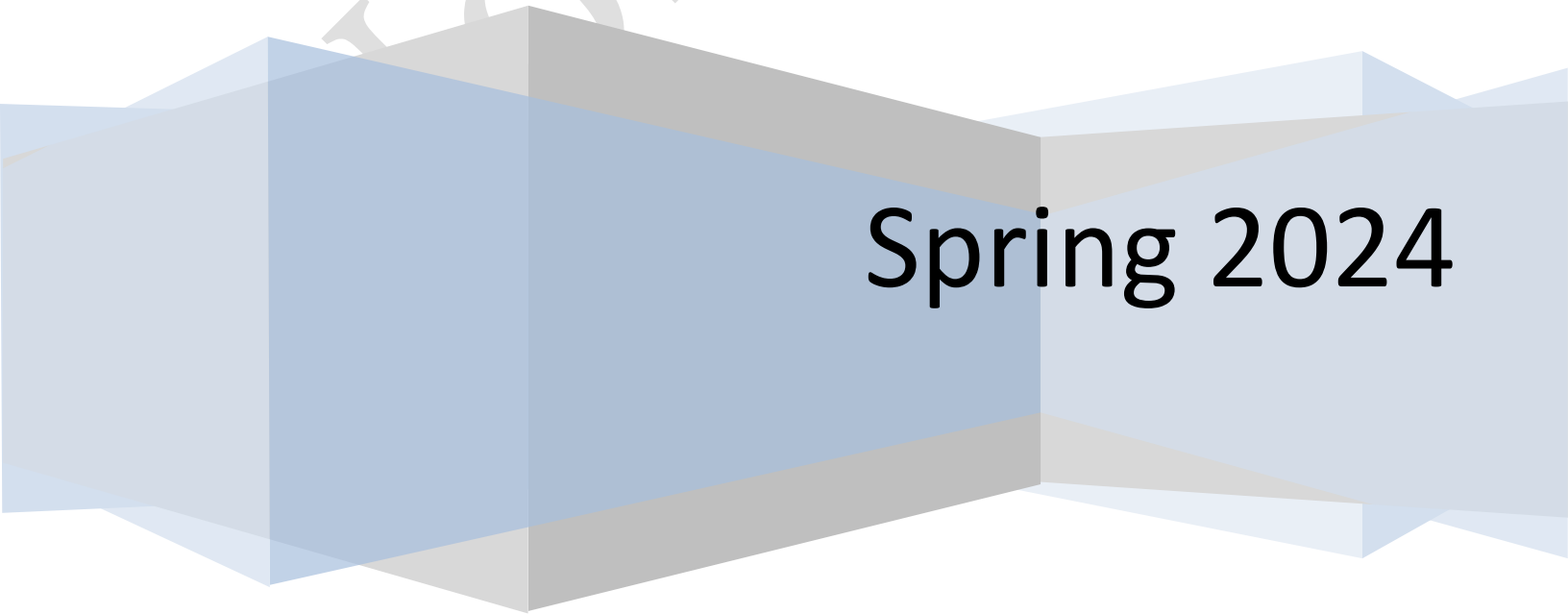
Northern University, Nowshera

Passing & returning objects from methods

Week # 07 - Lecture 13- 14



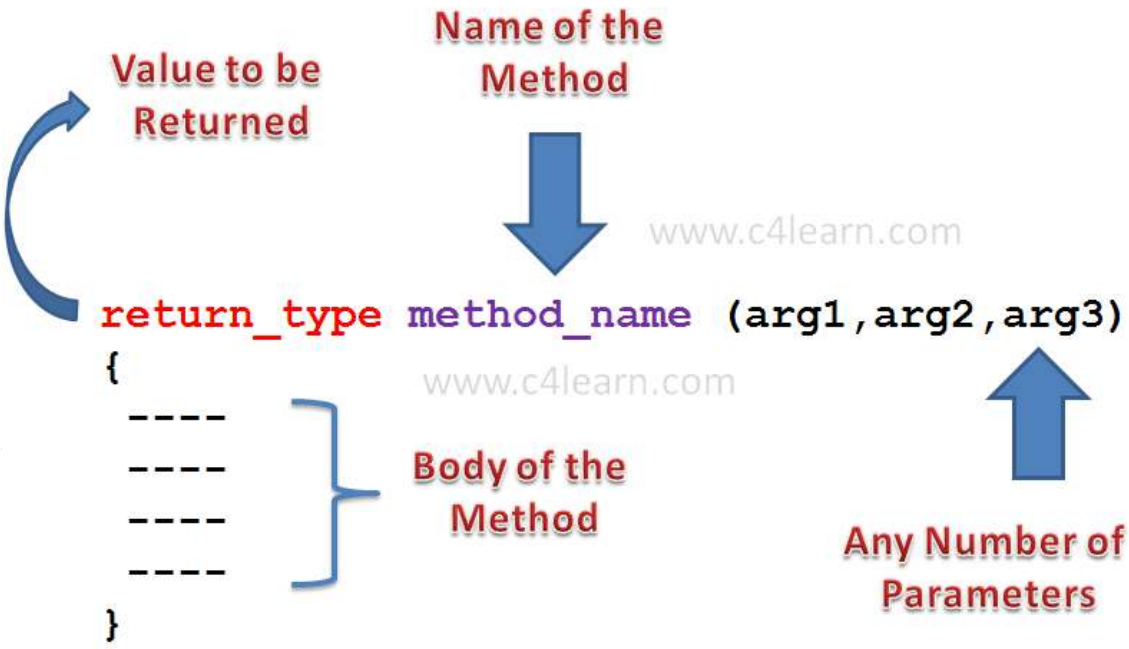
Spring 2024

Learning Objectives:

1. More with methods
2. returning values from methods
3. Passing objects to methods
4. returning objects from methods
5. Accessing private instance variables (outside the class)

1. More with methods

In Java Class, We can add user defined method which is equivalent to Functions in C/C++ Programming.



Syntax: Methods in Java Classes

return_type method_name (arg1 , arg2 , arg3)

return_type: is nothing but the value to be returned to a calling method.

method_name: is a name of method that we are going to call through any method.

arg1, arg2, arg3: are the different parameters that we are going to pass to a method.

Return Type of Method:

Method can return any type of value

Method can return any Primitive data type

int sum (int num1,unt num2);

Method can return Object of Class Type

Rectangle sum (int num1,unt num2);

Method sometimes may not return value

void sum (int num1,unt num2);

Method Name:

Method name must be valid identifier.

All Variable naming rules are applicable for writing Method Name.

Parameter List:

- Method can accept any number of parameters.
- Method can accept any data type as parameter.
- Method can accept Object as Parameter
- Method can accept no Parameter.
- Parameters are separated by Comma.
- Parameter must have Data Type

Example: Simple Method in Java Class

```
class Rectangle {
  private double length;
  private double breadth;
  public void setLength(int len)
  {
    length = len;
  }
```

```
  public void show()
    {
      System.out.println(length + " ,\t" + width);
    }

}
class RectangleDemo {
  public static void main(String args[]) {
  Rectangle r1 = new Rectangle();
  System.out.println("Before Function Length : " + r1.show());
  r1.setLength(20);
  System.out.println("After Function Length : " + r1.show());
  }
}
```

Output:

```
Before Function Length : 0.0 , 0.0
After Function Length : 20.0 , 0.0
```

Explanation: Calling a Method:

"r1" is an Object of Type Rectangle.

We are calling method "setLength()" by writing:

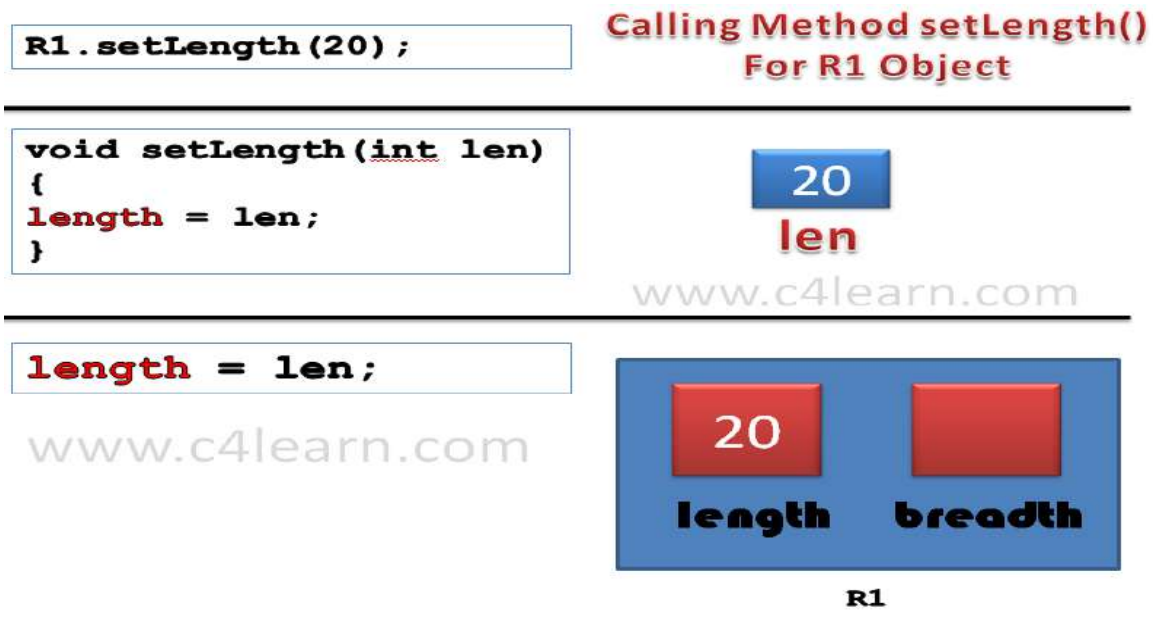
Object_Name [DOT] Method_Name (Parameter List) ;

Function call is always followed by Semicolon.

Method Definition:

Method Definition contains the actual body of the method.

Method can take parameters and can return a value.



2. Returning values from method

1. We can specify return type of the method as **"Primitive Data Type" or "Class name"**.
2. Return Type can be "Void" means **it does not return any value**.
3. Method can return a value by using "**return**" keyword.
4. Methods returning primitive data types can be called directly in sout.
5. we need a variable of same data type to store returned value

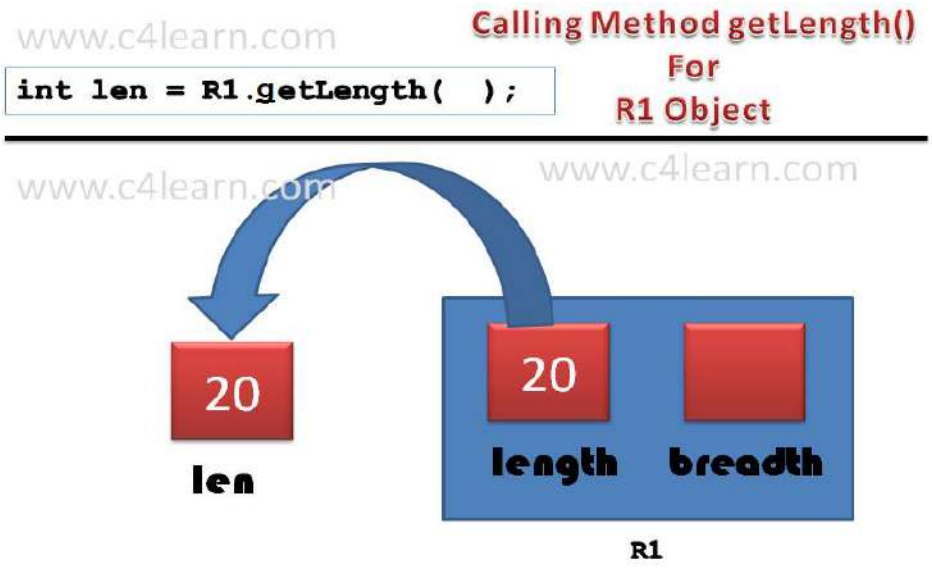
Example: Returning Value from the Method

```
class Rectangle {
  private int length;
  private int breadth;

  public void setLength(int len)
  {
  length = len;
  }
  public int getLength()
  {
```

```
  return length;
  }
}
class RectangleDemo {
  public static void main(String args[]) {

  Rectangle r1 = new Rectangle();
  r1.setLength(20);
  int len = r1.getLength();
  System.out.println("Length of Rectangle : " + len);
  System.out.println("Method called in System.out.println : " + r1.getLength() );
  }
}
```

Output :

```
Length of Rectangle : 20
Method called in System.out.println : 20
```



There are two important things to understand about returning values:

1. The **type of data returned by a method must be compatible with the return type specified by the method**. For example, if the return type of some method is boolean, you could not return an integer.

```
boolean getLength()
  {
  int length = 10;
  return(length);
  }
```

2. **The variable receiving the value returned by a method (such as len, in this case) must also be compatible with the return type specified for the method**.

```
public int getLength()
  {
  return length;
  }
boolean len = r1.getLength();
```

3. **Parameters should be passed in sequence and they must be accepted by method in the same sequence**.

```
public void setParameters(String str,int len)
  {
  -----
  -----
  -----
  }

 r1.setParameters(12,"Pritesh");
```

```
public void setParameters(int length,String str)
   {
   -----
```

```
  -----
  -----
  }

r1.setParameters(12,"Pritesh");
```

3. Passing objects as parameter

Objects of same class can be passed as parameters to methods that can help to process multiple objects of same class. For example: to compare the length of two objects of rectangle class say r1 and r2, we need a method "compareLength(rectangle obj)" where obj is a reference of parameterized object. Private instance variables of parameterized object can accessed directly within function using dot operator (of same class).

1. We can pass Object of any class as parameter to a method in java.
2. We can also access private instance variables of the parameterized objects.

```
area = r1.length * r1.width //Here r1 is reference of class object
```

3. It is good practice to initialize instance variables of an object before passing object as parameter to method otherwise it will take default initial values.

Example: Comparing the length of two rectangle class objects

```
package BIIT;
class Rectangle {
    private int length;
    private int width;
    public void setRectangle (int l, int b) {
        length = l;
        width = b;
    }
    void compare(Rectangle obj) {
       if(length > obj.length)
       {
           System.out.println("r1 has greater length: ");
       }
```

```
        else
            System.out.println("r2 has greater length ");
    }
}
class RectangleDemo {
    public static void main(String args[]) {
      Rectangle r1 = new Rectangle();
      r1.setRectangle(10, 20);

      Rectangle r2 = new Rectangle();
      r2.setRectangle(12, 17);

      r1.compare(r2);
    }
}
```

Output:

```
r2 has greater length:
```

4. Returning objects from methods:

Example 5.1: Add two objects - without returning object

```
public class Rectangle {
        private int length;
        private int width;
        public void setRectangle (int l, int b) {
            length = l;
            width = b;
        }
        void addRect(Rectangle obj) {
            Rectangle sum=new Rectangle();
            sum.length=length+obj.length;
            sum.width=width+obj.width;
            System.out.println("Sum of r1 and r2 is : " +sum.length+"   "+sum.width);
            //consider this call that is more better than above line of code
           // sum.showRect();
```

```
        }
        public void showRect()
        {
            System.out.println(length+" , "+width);
        }
    }
    class RectangleDemo {
        public static void main(String args[]) {
        Rectangle r1 = new Rectangle();
        r1.setRectangle(8, 12);
        System.out.println("r1 = ");
        r1.showRect();

        Rectangle r2 = new Rectangle();
        r2.setRectangle(12, 17);
        System.out.println("r2 = ");
        r2.showRect();
        r1.addRect(r2);   //will add objects r1, r2
        }
    }
```

Output:

```
r1 = 8  12
r2 = 12  17
Sum of r1 and r2 is : 20   29
```

In the above example we cannot add three objects with addRect() mehod. For this we need to modify function definition of addRect() like this:

```
void addRect(Rectangle obj1, Rectangle obj2)
{-----------------------------}
```

The following example is a generic way to add more than two objects.

Example 5.2: : Add two objects - with return object

```
class Rectangle {
```

```
    private int length;
    private int width;
    public void setRectangle (int l, int b) {
        length = l;
        width = b;
    }
   public Rectangle addRect(Rectangle obj) {
        Rectangle sum=new Rectangle();
        sum.length=length+obj.length;
        sum.width=width+obj.width;
        return sum;


    }
   public void showRect()
    {
        System.out.println(length+" , "+width);
    }
}
class RectangleDemo {
    public static void main(String args[]) {
   Rectangle r1 = new Rectangle();
   r1.setRectangle(8, 12);
   System.out.print("r1 = ");
   r1.showRect();

   Rectangle r2 = new Rectangle();
   r2.setRectangle(12, 17);
   System.out.print("r2 = ");
   r2.showRect();

   Rectangle result=new Rectangle();
   result=r1.addRect(r2);
   System.out.print("Sum of r1 and r2 is :");
   result.showRect();

   Rectangle r3 = new Rectangle();
```

```
        r3.setRectangle(15, 14);
        System.out.print("r3 = ");
        r3.showRect();

        result=result.addRect(r3);
        System.out.print("Sum of r1 and r2 and r3 is :");
        result.showRect();
        }
    }
```

Output:

```
r1 = 8  12
r2 = 12  17
Sum of r1 and r2 is :20  29
r3 = 15  14
Sum of r1 r2 and r3 is : 35  43
```

5. Accessing private instance variables outside the class

As we have already discussed that private attributes can only be accessed within class not outside the class. What if, we required these private properties outside the class?

The solution is: we will return these values through functions. The above example can also be solved by returning private attributes. For this, we need an additional method that will send private property where it is being called. The following example demonstrates the concept.

Example: returning private attributes

```
class Rectangle {
    private int length;
    private int width;
    public void setRectangle (int l, int b) {
        length = l;
        width = b;
    }
    public void show()
    {
```

```
      System.out.println(length + "\t" + width);
    }
    public int returnLength () {
       return length;
}
class RectangleDemo {
    public static void main(String args[]) {
      Rectangle r1 = new Rectangle();
      r1.setRectangle(10, 20);
      Rectangle r2 = new Rectangle();
      r2.setRectangle(12, 17);
      System.out.println("Rectangle with greater length is : \n ");
      // if (r1.length > r2. length) not possible because length is private so we
                                    need a method
       if(r1.returnLength() > r2. returnLength() )
        r1.show();
      else
        r2.show();
    }}
```

Output:

```
Rectangle with greater length is : 12   17
```

Note: We cannot access length attribute with dot operator because of private access modifier. As you can see that length property is now available in main() for comparison.

Assignment #6

READ THE FOLLOWING INSTRUCTIONS CAREFULLY:

1. *Attempt the Assignment after reading the lecture notes, watching the videos lectures and doing self-learning of the topic from web.*
2. *Submit your assignment via email /Google class room to respective teacher by the end of this week, dated 12 April 2020 before : 11:59 PM*
3. *Your Assignment should be a single file either pdf or MS Word (handwritten scanned document) and must follow the naming format, your Name, Reg#, Section, subject and assignment number, i.e. AliAhmed_2018-Arid-0001-CS2A_OOP_Asgn4*

File: TestAccount.java

```
1. //Java Program to demonstrate the working of a banking-system
2. //where we deposit and withdraw amount from our account.
3. //Creating an Account class which has deposit() and withdraw() methods
4. class Account{
5. private int acc_no;
6. private String name;
7. private float amount;
8. //Method to initialize object
9. public void insert(int a,String n,float amt){
10.         acc_no=a;
11.         name=n;
12.         amount=amt;
13.         }
14.         //deposit method
15.         public void deposit(float amt){
16.         amount=amount+amt;
17.         System.out.println(amt+" deposited");
18.         }
19.         //withdraw method
20.         public void withdraw(float amt){
21.         if(amount<amt){
22.         System.out.println("Insufficient Balance");
23.         }else{
24.         amount=amount-amt;
25.         System.out.println(amt+" withdrawn");
26.         }
27.         }
28.         //method to check the balance of the account
29.         public void checkBalance()
30.         {System.out.println("Balance is: "+amount);}
31.         //method to display the values of an object
32.         void display(){System.out.println(acc_no+" "+name+" "+amount);}
33.         }
34.         //Creating a test class to deposit and withdraw amount
35.         class TestAccount{
36.         public static void main(String[] args){
37.         Account a1=new Account();
38.         a1.insert(832345,"Ali",1000);
39.         a1.display();
40.         a1.checkBalance();
```

```
41.         a1.deposit(40000);
42.         a1.checkBalance();
43.         a1.withdraw(15000);
44.         a1.checkBalance();
45.         }
46.         }
```

Output:

```
832345 Ali 1000.0
Balance is: 1000.0
40000.0 deposited
Balance is: 41000.0
15000.0 withdrawn
Balance is: 26000.0
```

Q#1: Modify the above bank example in a way that user (bank employee) can perform following operations:

- *Create new bank account (maximum five)*
- *Display account balance (based on account number)*
- *Display the name and amount of customer having maximum account balance (find max amount among all account holders)*

Note: Use the concepts discussed in lesson so far i.e, access modifiers, passing & returning objects

Q#2: Consider we already created **Employee** Class as given in UML diagram. Create **Employee Management** class which have an **array 10 of Employees (**Array of Employee must be a data member**)**. Create following functions as instructed:

1. **InputAll:** this will receive the values (first name, last name, salary) for each Employee and assign to respective objects. ID should be assign incrementally starting from 100.
2. **Display:** this will ask user for employee ID and it will display the information of the employee.
3. **SortBySalary:** this will sort the array in ascending order, based the value of the salary.