JAVA @17

Object as Method Argument
and Return type

Objective

After completing this session you will be able to understand,

- Passing Object as method argument
- Returning Object from the method

A Problem Developer Faces

Scenario: Assume you are developer and you need to create a method which accept the user details which includes the Name , Address ,Age ,Employment details and several other details. The method should calculate and return the tax rate and tax amount based on the user's age and salary and store the employee and tax details in database. The method should return user name and the tax.

Question: How can you pass all the details into the method as an argument ?

Answer: We can pass it as arguments to the method like

Method(String name ,String address ,int age………………….)

Disadvantages:

•This makes the method complex as the number of arguments increases.

•Any new user details addition will result in a change in the method name.

•We cannot return more than one value from a method.

What is the solution?

Solution is passing objects as method arguments and returning objects as return value

Types of methods argument

- **Passing Primitive Data Type :** Passing the primitive type data as method argument. **Example:** int, double, float etc.

- **Passing objects are argument :** In this case we pass objects as arguments to the method. **Example:** UserVO, ArrayList, Map

In this session we are going to learn on how to pass objects to methods?

Passing objects as method argument

▪Objects are passed as arguments to the method

▪Can make change **only** to the state of the object

▪Doesn't make any change to the reference variable hence we say that this

mechanism is not exactly same as "**Pass-By-Reference**" in C programming.

▪Since the state changes are done in objects some people argue this indirectly

achieves "**Pass by reference** ".

▪Can be used with constructors also.

Next slide explains how the state of objects changes when passed as arguments.

Passing Objects How it works?

Consider a class named MyClass with two fields a and b.

```
MyClass obj=new MyClass();
aMethod(obj);
```

Object Created

Heap Memory

Address: 1001

a=30
b=40

The **obj** contains the object memory address **1001** as value which is the address of the newly created object.

```
void aMethod(MyClass obj1)
{
 obj1.a=30;
obj1.b=40;
}
```

When the method **aMethod** is invoked by passing **obj** as argument. Now **obj1** in the invoked method contains the memory address value **1001** .

Since **obj1** contains the address **1001** , any changes in the state of this will also change the state of **obj** in the calling method.
So changes done to the obj1 state a=30 and b=40 will also change the state of **obj.**

Passing Object as Args.

```java
class Test {
    int a, b;
    Test(int i, int j) {
        a = i;
        b = j;
    }
    void alterPrimitive(int x, int y) {
        x = 60;
        y = 30;
    }
    void alterObject(Test o) {
        o.a = 25;
        o.b = 62;
    }
}
class PassOb {
    public static void main(String args[]) {
        Test ob1 = new Test(100, 22);
        System.out.println("Object State before alterPrimitive Method Call ob1.a : "
                        + ob1.a + " ob1.b : " + ob1.b);
        ob1.alterPrimitive(ob1.a, ob1.b);
        System.out.println("Object State after alterPrimitive  Method Call ob1.a : "
                        + ob1.a + " ob1.b : " + ob1.b);
        System.out.println("Object State before alterObject Method Call ob1.a : "
                        + ob1.a + " ob1.b : " + ob1.b);
        ob1.alterObject(ob1);
        System.out.println("Object State after alterObject  Method Call ob1.a : "
                        + ob1.a + " ob1.b : " + ob1.b);
    }
}
```

Accepts two primitive type values and tries to modify it.

Method accepts an object reference of the type Test and modifies the object's **State**

Passes the object members as primitive data

Passes the object as an argument

We will see how object passing works with help of the following code.

What is the Output?

```
Object State before alterPrimitive Method Call ob1.a : 100 ob1.b : 22
Object State after alterPrimitive  Method Call ob1.a : 100 ob1.b : 22
Object State before alterObject Method Call ob1.a : 100 ob1.b : 22
Object State after alterObject  Method Call ob1.a : 25 ob1.b : 62
```

The values will be same before and after the method call when passed as primitive type argument.

The state of the object is seen to be changed because we are passing the reference or the address of the object and making alteration in that memory area.

Returning Objects from a Method

- Instead of returning primitive data type from a method, we can return objects from a method.

- Works similar to methods with primitive return type.

- The return type should be either of class type or any of it's super classes/implemented interfaces.

- The methods can also return Collection objects like ArrayList, Map etc which you will learn in the later sessions.

Advantage:

If you need to return multiple values the values can be defined in a value object and returned.

Object as Return type

```
public class Student {
String studentID;
int mark1;
int mark2;
}
```

```
public class Result {
String studentID;
String grade;
}
```

```
public class ResultCalculator {
    public Result calculateResult(Student student) {
        int total = student.mark1 + student.mark2;
        Result result = new Result();
        result.studentID = student.studentID;
        if ((total / 2) < 60) {
            result.grade = "Fail";
        } else {
            result.grade = "pass";
        }
        return result;
    }
}
```

Note the methods return type is Result Object

Method accepts a Student object , calculates the result based on the marks , loads the result into a Result object and returns it.

Object as Args & Return type

Objective: In this lend a hand we will familiarize how objects can be passed as an argument to a method and how objects can be returned by methods.

Scenario : We are asked to develop a method which accepts the details of an student, which includes the marks of three subjects. The method should calculate the total mark and should rate the student pass/fail based on the average marks

if average >40 { Pass }

else { Fail }

Components

1. Student Class : To hold the student details

2. Result Class : To hold the student results

3. ResultCalculator class : Contains method to calculate the total mark and result of the student

4. MainClass class: Drives the application

Student.java

```java
public class Student {
    private String rollNo;
    private int mark1;
    private int mark2;
    private int mark3;
    public String getRollNo() {
        return rollNo;
    }
    public void setRollNo(String rollNo) {
        this.rollNo = rollNo;
    }
    public int getMark1() {
        return mark1;
    }
    public void setMark1(int mark1) {
        this.mark1 = mark1;
    }
    public int getMark2() {
        return mark2;
    }
    public void setMark2(int mark2) {
        this.mark2 = mark2;
    }
    public int getMark3() {
        return mark3;
    }
    public void setMark3(int mark3) {
        this.mark3 = mark3;
    }
}
```

Add the Accessor methods:

The methods starting with get and set are called *accessor* methods used for accessing the private members of the class.

The method starting with "*set*" is known as the setter used for setting the argument value to the member variable .

The method starting with "*get*" is known as the getter used for retrieving the value of the variable.

Result.java

```java
public class Result {
    private String rollNo;
    private String grade;

    public String getGrade() {
        return grade;
    }

    public void setGrade(String grade) {
        this.grade = grade;
    }

    public void setRollNo(String rollNo) {
        this.rollNo = rollNo;
    }

    public String getRollNo() {
        return rollNo;
    }
}
```
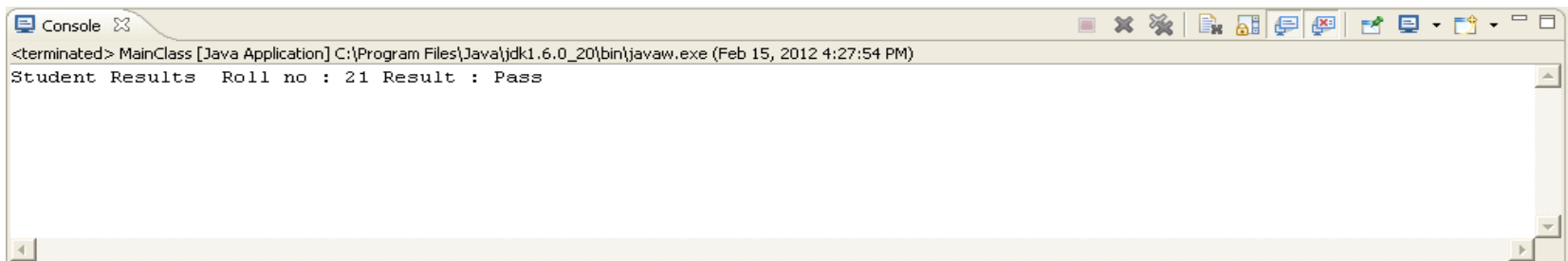
ResultCalculator.java

```java
public class ResultCalculator {
    public Result calculateResult(Student student) {
        int total = student.getMark1() + student.getMark2()
                + student.getMark3();
        int average = total / 3;
        Result result = new Result();
        result.setRollNo(student.getRollNo());
        if (average > 40) {
            result.setGrade("Pass");
        } else {
            result.setGrade("Fail");
        }
        return result;
    }
}
```

MainClass.java

```java
public class MainClass {
    public static void main(String args[]) {
        Student student = new Student();
        student.setRollNo("21");
        student.setMark1(65);
        student.setMark2(85);
        student.setMark3(40);
        ResultCalculator calculator = new ResultCalculator();
        Result result=calculator.calculateResult(student);
        System.out.println("Student Results  Roll no : "+result.getRollNo()+
                " Result : " + result.getGrade());
    }
}
```

Output

You have successfully completed

Object as Args & Return type