# **Experiment No - 9**

***Aim:* Study Get and Set methods in java**

***Theory:***

We were able to create objects of the Student and initialise the variables of the object using the constructor. But after that, we can no longer change the values the variables hold. The following statement would result in an error since the variables were declared to be private.

s1.name = "Ravi";

public void setName ( String n ) {

name = n;

}

It is a convention to name the setter methods starting with the word 'set' followed by the variable name that is going to be set. If we wish to set the name for the first Student object s1 which we have created, we use the following statement:

s1.setName("Ravi");

public String getName() {

return name;

}

The value that is to be returned is specified by using the 'return' keyword. The variable or value that is to be returned follows the return keyword. We can specify not just a variable name but also a constant or an expression. the following return statements would also be equally fine. return "XYZ";

return "Name: "+name;

public String getName() {

return name;

name = "No name";

}

The statement name="No name" is an unreachable statement. This is because, once the return statement is encountered, the remaining lines of code in that method would be ignored. Therefore, the above code produces compilation errors. However, in certain circumstances, we can provide a return statement and also write code following that statement when we use decision making statements. We will look into it later on.

In the following statement, the value returned by getName is stored in the variable a Name.

String aName = s2.getName();

We hope that you are able to guess what the purpose of the above statements is. To make sure that you've got it right, let's look at them individually. If the condition stated in the parentheses of the if statement is true, then the code in the block is executed. In a similar way, for the if else statement, if the condition is true, the first block of code is executed. If the condition is false, the statements in the else block are executed. As already said, a block is a set of statements within a pair of curly braces. They indicate the ownership of a piece of code. Here, the blocks are used to indicate that the code contained in them is a part of the corresponding if or else statements. If the code which is to written for an if or else statement has just a single line or a single statement, there is no need to put the braces. Another important thing is that variables declared within a block are not accessible outside the block. Here, if we declare a new variable 'var 'in the if block and try to print its value in the else blocks, we would be getting a compilation error. The same holds true for methods also. For instance, if we declare a new variable in the setName() method and try to access it in the getName() method, we would be stopped bay compilation errors. The following code is erroneous.

Logical operator or ||

Operand 1 Operand 2 Result

true true true

true false true

false true true

false false false

Logical operator not !

Operand Result

true false

false true

Following code shows the use of logical operators

boolean a = true;

boolean b = false;

boolean res1 = a && b; // false

boolean res2 = a || b; //true

boolean res3 = !a; //false

boolean res4 = !false; // true

boolean res5 = true && a; //true

We again emphasize on the fact that Java is a free form language. You can break a statement into two lines at any valid point and also write more than one statement one single line. If you feel that reading the above code is inconvenient to you, we suggest the following alternative form.

public void setMarks( int m) {

if ( m>=0 && m<=100 ) {

marks1 = m;

} else {

marks1 = 0;

}

}

Also, in this case, since the if and else blocks contain only a single statement, you can omit the braces and write the code in the following way:

public void setMarks(int m) {

if ( m>=0 && m<=100 )

marks1 = m;

else

marks1 = 0;

}

And another remainder regarding eth accessibility of a variable: Variables defined within a block are accessible in that block only. These conditions are known as the scope and lifetime of variables. Scope refers to the part of the code in which the variable is accessible while lifetime refers to the duration for which a variable exists before it is destroyed by deallocating the memory allotted to it while declaring the variable. If we declare a variable within the if block, the lifetime ends once we come of if the if block. Any attempt to access that variable would give an error that the requested variable is not defined. If there is a need to declare variables inside such blocks and access them even after we move out of the blocks, then we need to declare these variables before the starting of the block and outside the block. Following pieces of code shows an incorrect way and a correct way of using such variables.

public void setMarks(int m) {

if ( m>=0 && m<=100 )

int temp = m;

else

int temp = 0;

marks1 =temp; // Incorrect: temp is not known here

}

public void setMarks(int m) {

int temp;

if (m>=0 && m<=100 )

temp = m;

else

temp = 0;

marks1 =temp; // Correct: temp is known here

}

***Conclusion:***

Hence understood the use of get and set operation.