Structured Programming

Lecture 4

Dr. Obead Alhadreti

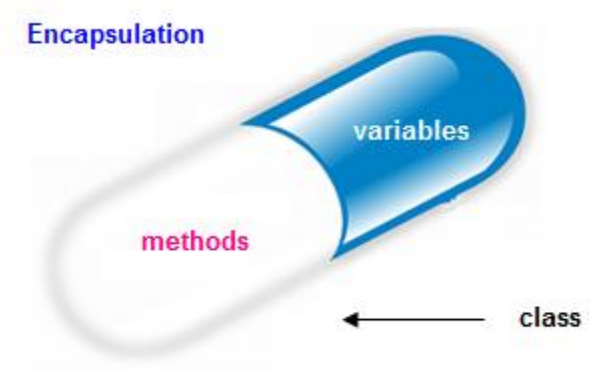
Outline

- Encapsulation
- Data Hiding
- Get Method
- Set Method

Encapsulation

Encapsulation

- ***Encapsulation*** is a process of wrapping of data (variables) and methods in a single unit (i.e., class). Classes (and their objects) encapsulate, i.e., encase, their variables and methods.



- For example, take a Student class where we will have students instance variables and methods acting on those instance variables at one place (i.e., Student class).

Main advantages of Encapsulation

1. **Simplicity and clarity:** As all data and methods are stored in the classes (and their objects), it becomes very easy to understand the purpose of each data member and method in a class. Furthermore, everyone will be able to understand whole scenario by simple looking into object diagrams.

2. **Low complexity:** As data members and methods are hidden in classes (and their objects), and each object has a specific behavior so there is less complexity in code there will be no such situations that a method is using some other method and that method is using some other function.

Data Hiding

Data Hiding

- Encapsulation also enables us to secure data from other classes, when we make variables (data) of a class ***private*** then these variables can be accessed only through the methods of their current class (this technique is known as ***data hiding***).

- Example: A bank application forbids a client to change an account's balance.

- Real life example of data hiding: Your name and other personal information is stored in your brain we can't access this information directly. For getting this information we need to ask you about it and it will be up to you how much details you would like to share with us.

Data Hiding

- Benefits of data hiding:

1. A class can have total control over what is stored in its data.
2. The users of a class do not know how the class stores its data. A class can change the data type of a field and users of the class do not need to change any of their code.
3. The data of a class can be made read-only or write-only.

- To achieve data hiding in Java:
  1. Declare the variables of a class as *private*.
  2. Provide public getter and setter methods to modify and view the variables values.

Get & Set Methods

Get Methods

- When an instance variable is declared private, we need a way to access that variable.

- A method used to obtain the value of a private instance variable is referred to as a get method (also known as getter or accessor methods).

- The naming scheme of getter should be as follows:

```
public returnType getVariableName ()
{
return variableName;
}
```

Get Methods

- The method's return type specifies the type of data returned to a method's caller.
- Empty parentheses following a method name indicate that the method does not require any parameters to perform its task.
- The return statement passes a value from a called method back to its caller.
- To call a method of an object, follow the object name with a dot separator, the method name and a set of parentheses containing the method's arguments.

Set Methods

- A method used to set the value of a private instance variable is referred to as a set method (also known as setter or mutators methods).

- The naming scheme of setter should as follows:

```
public void setVariableName (Type localVariable)
                {
        VariableName = localVariable;
                }
```

- Keyword void indicates that a method will perform a task but will not return any information.

Set Methods

- A method call supplies values—known as arguments—for each of the method's parameters.
- Each argument's value is assigned to the corresponding parameter in the method header.
- The number of arguments in a method call must match the number of parameters in the method declaration's parameter list.
- The argument types in the method call must be consistent with the types of the corresponding parameters in the method's declaration.

Example 1

Start Page | Main.java | Employee.java

```
1  package employeerecords;
2
3  public class Main {
4
5      public static void main(String[] args) {
6          Employee e = new Employee();
7
8
9          e.name   = "Mhamad";
10         e.age    = 21;
11         e.salary = 1500000;
12
13
14         System.out.println("Name: "   +e.name);
15         System.out.println("Age: "    +e.age);
16         System.out.println("Salary: " +e.salary);
17
18     }
19
20 }
```

tart Page | Main.java | Employee.java

```
1  package employeerecords;
2
3  public class Employee {
4
5      String name;
6       int    age;
7       double salary;
8
9  }
0
```

Output - EmployeeRecords (run)

```
run:
Name: Mhamad
Age: 21
Salary: 1500000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Example 2

Start Page | Main.java | **Employee.java**

```
1   package employeerecords;
2
3   public class Employee {
4
5       private String name;
6       private int    age;
7       private double salary;
8
9
10      public String getName() {
11          return name;
12      }
13
14      public int getAge() {
15          return age;
16      }
17
18      public double getSalary() {
19          return salary;
20      }
21
22
23      public void setName(String n) {
24          name = n;
25      }
26
27      public void setAge(int a) {
28          age = a;
29      }
30
31      public void setSalary(double s) {
32          salary = s;
33      }
34
35  }
```

Example 2

Start Page | Main.java | Employee.java

```
1   package employeerecords;
2
3   public class Main {
4
5       public static void main(String[] args) {
6           Employee e = new Employee();
7
8
9           e.setName("Mhamad");
10          e.setAge(21);
11          e.setSalary(1500000);
12
13
14          System.out.println("Name: "    +e.getName());
15          System.out.println("Age: "     +e.getAge());
16          System.out.println("Salary: " +e.getSalary());
17      }
18
19  }
```

Output - EmployeeRecords (run)

```
run:
Name: Mhamad
Age: 21
Salary: 1500000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Get & Set Methods

- By using getter and setter, the programmer can control how his important variables are accessed and updated in a correct manner, such as changing value of a variable within a specified range.

Example 3

Start Page | Main.java | Employee.java

```
1   package employeerecords;
2
3   public class Employee {
4
5       private String name;
6       private int    age;
7       private double salary;
8
9
10      public String getName() {
11          return "Name: " +name;
12      }
13
14      public int getAge() {
15          return age;
16      }
17
18      public double getSalary() {
19          return salary;
20      }
21
22
23      public void setName(String n) {
24          if (n.length() < 3) {
25              System.out.println("Name is too short, name can't be less then 3 characters!");
26          }
27          else {
28              name = n;
29          }
30      }
31
32      public void setAge(int a) {
33          age = a;
34      }
35
36      public void setSalary(double s) {
37          salary = s;
38      }
39  }
```

Example 3

Start Page | Main.java | Employee.java

```
1   package employeerecords;
2
3   public class Main {
4
5       public static void main(String[] args) {
6           Employee e = new Employee();
7
8           e.setName("dj");
9           e.setAge(21);
10          e.setSalary(1500000);
11
12          System.out.println(e.getName());
13          System.out.println("Age: "     +e.getAge());
14          System.out.println("Salary: " +e.getSalary());
15      }
16
17  }
```

Output - EmployeeRecords (run)

```
run:
Name is too short, name can't be less then 3 characters!
Name: null
Age: 21
Salary: 1500000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```