Lesson:

Encapsulation

List of Concepts Involved:

- Need of Encapsulation
- What is Encapsulation?
- Private members
- Shadowing Problem and this keyword
- Setters and Getters

Need of Encapsulation

- To the outside world, the data should not be exposed directly.
- In order to provide the controlled access, we need to use “Encapsulation”.

What is Encapsulation?

- Binding of data and corresponding methods into a single unit is called "Encapsulation".
- If any java class follows data hiding and abstraction then such class is referred as "Encapsulated class".

Encapsulation = Data Hiding + abstraction.

Every data member inside the class should be declared as private, and to access this private data we need to have setter and getter methods.

Advantages of Encapsulation

a. We can achieve security.
b. Enhancement becomes easy.
c. Maintainability and modularisation becomes easy.
d. It provides flexibility to the user to use the system very easily.

Private members

- Our internal data should not go to the outside world directly, that is, outside people should not access our internal data directly.
- By using private modifiers we can implement "data hiding".

Example

```
class Account
{
 private double balance;
 }
```

- Advantage of Data Hiding is security.
- Recommended modifier for data members is private.

Shadowing Problem and this keyword

- If both local variable and instance variable have the same name inside the method then it would result in a name-clash and jvm will always give preference for local variable.

This approach is called the **"Shadowing problem".**

Example

```
class Student
{
        private String name;
        private Integer id;
        private String address;

       Student(String name,Integer id, String address){
                name = name;
                id = id;
                address = address;
      }
      public void display()
     {
              System.out.println("Name is    :: "+name);
              System.out.println("Id is      :: "+id);
              System.out.println("Address is :: "+address);
    }
}

public class Demo
{
         public static void main(String[] args)
        {
                Student std = new Student("sachin",10,"MI");
                std.display();
       }
}
```

Output

Name is :: null
Id is :: null
Address is :: null

- As noticed in the above program, the variables name,id,address are local variables and these values should be assigned to instance variables of student class.
- Inside the method the jvm will always give preference only for local variables, this problem is termed as "Shadowing".
- To resolve this problem we need to use , "this" keyword.

Note:
this keyword would always point to current object, and this variable would hold the address the active object present in the heap memory.

Program to resolve the problem of shadowing

```
class Student
{
        private String name;
        private Integer id;
        private String address;

       Student(String name,Integer id, String address){
                this.name = name;
                this.id = id;
                this.address = address;
      }
      public void display()
     {
              System.out.println("Name is    :: "+name);
              System.out.println("Id is      :: "+id);
              System.out.println("Address is :: "+address);
      }
}

class Demo
{
       public static void main(String[] args)
       {
              Student std = new Student("sachin",10,"MI”);
              std.display();
      }
}
```

Output

Name is   :: sachin
Id is   :: 10
Address is :: MI

Setters and Getters

Setter methods are used to set the value to the instance variables of the class.
Syntax for setter method
a. compulsory the method name should start with set.
b. it should be public.
c. return type should be void.
d. compulsorily it should have some argument.

Getter methods are used to get the value from the instance variables of the class.
Syntax for getter method
a. compulsory the method name should start with get.
b. it should be public.
c. return type should not be void.
d. compulsorily it should not have any argument.

Program to demonstrate the usage of setters and getters

```
class Student
{
        private String name;
        private Integer id;
        private String address;

      //setters
      public void setName(String name){
                        this.name = name;
      }
      public void setId(Integer id){
                         this.id = id;
      }
      public void setAddress(String address){
                        this.address = address;
      }

      //getters
      public Integer getId(){
                return id;
     }
     public String getName(){
                return name;
    }
    public String getAddress(){
                return address;
    }
}
```

```
public class Demo
{
	public static void main(String[] args)
	{
		Student std = new Student();
		std.setId(10);
		std.setName("sachin");
		std.setAddress("MI");

		System.out.println("Id is      :: "+std.getId());
		System.out.println("Name is    :: "+std.getName());
		System.out.println("Address is :: "+std.getAddress());
	}
}
```

Output

Name is :: sachin
Id is :: 10
Address is :: MI

Note

if the property is of type boolean then for getter method we can prefix with either "is/get".

Example

```
public class Student{
	private boolean married;
	public void setMarried(boolean married){
		this.married=married;
	}

		public boolean isMarried()(){
				return married;
		}
}
```