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.

→ outside world don't know about any operation on the data.
→ outside world don't know data

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".

```
class Student{
    private int age;
    private String name;
```

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.

How to auto generate setters and getters in VSCode?
Sol^n → ① Declare the Private variables
② Inside class right click
③ Click source action
④ Click Generate setter and getter.

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;
		}
}
```

List of Concepts Involved:

Constructor

- Constructor
- DefaultConstructor
- Usage of Constructor
- Constructor Chaining
- Usage of this() and super()

Constructor

used when we have to do the initialization at the time of Object creation itself ∴, no need of using setter & getter as in case of Encapsulation.

- Object creation is not enough, compulsorily we should perform initialization then only the object is in a position to provide the response properly.
- Whenever we are creating an object some piece of the code will be executed automatically to perform initialization of an object. This piece of code is nothing but a constructor.
- Main objective of the constructor is nothing but initialisation of Object.

Rules for writing a constructor

- Name of the constructor and name of the class must be the same.
- Return type concept not applicable for constructor, even if we provide it won't result in compile time errors, if we do so then the Java language will treat this as "normal method".

Eg

```
class Test{
        void Test(){
                System.out.println("Hello");// It is not a constructor,it is a method.
        }
}
```

- It is not a good practice to take the method name same as that of the classname.
- The modifiers applicable for constructors are private,public,protected,default.
- The other modifiers if we use, it would result in compile time error.

```
class Test{
        static Test(){
}
}
```

Default constructor

- For every java class constructor concept is applicable.
- If we don't write any constructor, then the compiler will generate a default constructor.
- If we write at least one constructor then the compiler won't generate any default constructor, so we say every java class will have a compiler generated default constructor or programmer written constructor but not both simultaneously.

Prototype of default constructor

- There is always no argument constructor.
- The access modifier of the default constructor is the same as the class modifier. [applicable for public and default]
- Default constructor contains one line, super(). It is a call to super class constructor.

```
class Test{

}
```

```
class Test{
    Test(){
        super();
    }
}
```

```
public class Test{

}
```

```
public class Test{
    public Test(){
        super();
    }
}
```

```
class Test{
    void Test(){

    }
}
```

```
class Test{
   Test(){
      super();
   }
  void Test(){

  }
}
```

writing multiple constructors with same name but diff parameters.

Constructor Overloading/Constructor Chaining

- A class can contain more than one constructor and all these constructors have the same name they differ only in the type of argument, hence these constructors are considered as "Overloaded constructor".

Eg

```
class Test {
        Test(double d) {
                System.out.println("double argument constructor");
        }

        Test(int i) {
                this(10.5);
                System.out.println("int argument constructor");
        }

        Test() {
                this(10);
                System.out.println("no argument constructor");
        }
}
public class MainApp {
        public static void main(String[] args) throws Exception {
                Test t1= new Test();//double int no argument constructor
                Test t2= new Test(10);// double int argument constructor
                Test t3= new Test(10.5);//double argument constructor
        }
}
```

super() vs this()

1. The first line inside the constructor can be super()/ this().☆
2. If we are not writing anything then compiler will generate super();☆

case1:
We have to take super()/this() only in the first line of constructor, if we are writing anywhere else it would result in a compile time error.

eg1

```
class Test{
        Test(){
                System.out.println("Constructor");//CE
               super();
        }
}
```

eg2
we can either use super()/this() but not simultaneously

```
class Test{
        Test(){
              super();
              this();//CE
        }
}
```

eg3
we can use super()/this() only inside the constructor otherwise it would result in compile time error.

```
class Test{
        void methodOne(){
                super();
                this();
        }
}
```

Note

super()

- It should be the first line in the constructor.
- It should be used only in constructor.
- It will take control to the parent class constructor.

this()

- It should be the first line in the constructor.
- It should be used only in constructor.
- It will make the call of the current class constructor.

Difference b/w super(),this()?

super(),this()

- These are constructor calls
- These are used to invoke super class and current class constructor directly
- We should use only inside the constructor that to first line otherwise we get compile time error.