Encapsulation and Abstraction in JAVA

Dr. Krishnendu Guha

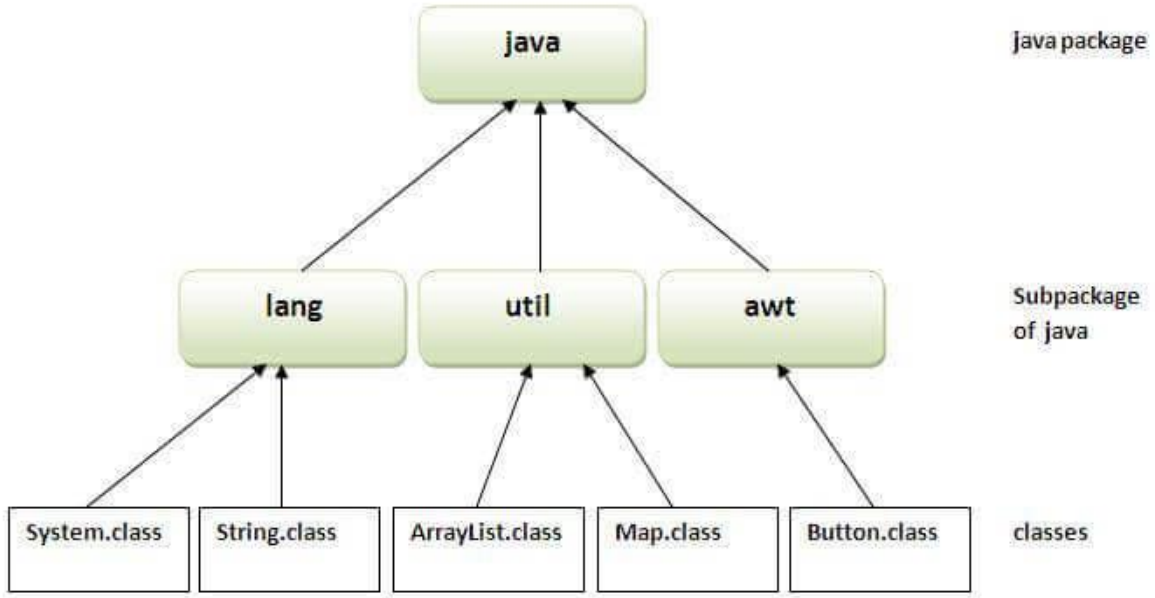
Assistant Professor/ Lecturer

School of Computer Science and Information Technology

University College Cork

Recap (Java Package)

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form, **built-in package** and **user-defined package**.

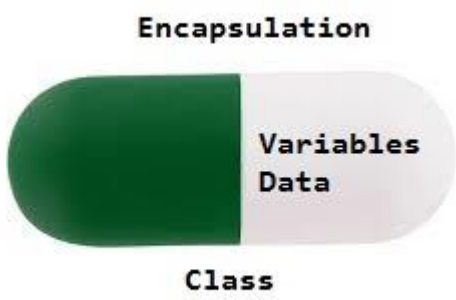


Recap (Access Modifiers)

- There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.
- The **access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class**.
- We can **change the access level of fields, constructors, methods, and class by applying the access modifier** on it.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

What is Encapsulation?



- Encapsulation: comes from **encapsulating**
- That is **wrapping up of code and data under a single unit**.

- It is a **protective shield that prevents the data from being accessed by code outside this shield.**
- So, **the variables or data of a class is hidden from any other class** and **can be accessed only through any member function of its own class** in which it is declared.
- It is also known as a **combination of data-hiding and abstraction**.
  - As in encapsulation, the data in a class is hidden from other classes using the **data hiding** concept which is achieved by **making the members or methods of a class private**, and
  - the class is **exposed to the end-user or the world without providing any details** behind implementation **using the abstraction concept**

How to perform Encapsulation

- Encapsulation can be achieved by
  - **Declaring all the variables in the class as private** and
  - writing **public methods** in the class to **set and get the values of variables**.
- It is more defined with the **setter and getter method**.

- The **Java Bean** class is the example of a fully encapsulated class.

Advantages

- Data Hiding
- Better control of class attributes and methods
- Class attributes can be made read-only (only using the get method) or write only (only using the set method)
- Reusability
- Testing Code is easy

Get and Set Methods

private variables can only be accessed within the same class (an outside class has no access to it).

However, it is possible to access them if we provide public **get** and **set** methods.

The get method returns the variable value, and the set method sets the value.

Syntax for both is that they start with either get or set, followed by the name of the variable, with the first letter i

```
public class Person {
        private String name; // private = restricted access

        // Getter
        public String getName() {
                return name;
        }

        // Setter
        public void setName(String newName) {
                this.name = newName;
        }
}
```

The get method returns the value of the variable name.

The set method takes a parameter (newName) and assigns it to the name variable.
The this keyword is used to refer to the current object.

```
public class Person {
        private String name; // private = restricted access

        // Getter
        public String getName() {
                return name;
        }

        // Setter
        public void setName(String newName) {
                this.name = newName;
        }
}

public class Main {
        public static void main(String[] args) {
                Person myObj = new Person();
                myObj.name = "John"; // error
                System.out.println(myObj.name); // error
        }
}
```

However, as the name variable is declared as private, we **cannot** access it from outside this class

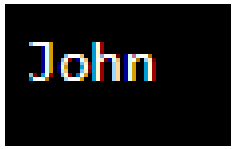
If the variable was declared as public, we would get the output John

```
public class Person {
		private String name; // private = restricted access

		// Getter
		public String getName() {
			return name;
		}

		// Setter
		public void setName(String newName) {
			this.name = newName;
		}
}

public class Main {
	public static void main(String[] args) {
		Person myObj = new Person();
		myObj.setName("John"); // Set the value of the name variable to "John"
		System.out.println(myObj.getName());
	}
}
```



```java
// fields to calculate area
class Area {

  int length;
  int breadth;

  // constructor to initialize values
  Area(int length, int breadth) {
    this.length = length;
    this.breadth = breadth;
  }

  // method to calculate area
  public void getArea() {
    int area = length * breadth;
    System.out.println("Area: " + area);
  }
}

class Main {
  public static void main(String[] args) {

    Area rectangle = new Area(2, 16);
    rectangle.getArea();
  }
}
```

```
Area: 32
```

Abstraction

- Data **abstraction** is the process of hiding certain details and showing only essential information to the user.
- Recap (Abstract Classes and Methods)

•**Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

•**Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

The abstract keyword is a non-access modifier, used for classes and methods

An abstract class can have both abstract and regular methods:

```
abstract class UCC {
        public abstract void acct();
        public void csit() {
                System.out.println(“study java");
        }
}
```

it is not possible to create an object of the UCC class:

```
UCC myObj = new UCC(); // will generate an error
```

To access the abstract class, it must be inherited from another class.

```
// Abstract class
abstract class UCC {
        // Abstract method (does not have a body)
        public abstract void acct();
        // Regular method
        public void csit() {
                System.out.println(“study java");
        }
}

// Subclass (inherit from UCC)
class SEFS extends UCC {
        public void acct() {
                // The body of acct() is provided here
                System.out.println(“Account dept is here");
        }
}

class Main {
        public static void main(String[] args) {
                SEFS mySEFS = new SEFS(); // Create a SEFS object
                mySEFS.acct();
                mySEFS.csit();
        }
}
```

```
Account dept is here
study java
```

Difference between Abstraction and Encapsulation in Java

Abstraction	Encapsulation
Abstraction is the process or method of gaining the information.	While encapsulation is the process or method to contain the information.
In abstraction, problems are solved at the design or interface level.	While in encapsulation, problems are solved at the implementation level.
Abstraction is the method of hiding the unwanted information.	Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.
We can implement abstraction using abstract class and interfaces.	Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public.
In abstraction, implementation complexities are hidden using abstract classes and interfaces.	While in encapsulation, the data is hidden using methods of getters and setters.
The objects that help to perform abstraction are encapsulated.	Whereas the objects that result in encapsulation need not be abstracted.
Abstraction provides access to specific part of data.	Encapsulation hides data and the user can not access same directly (data hiding.
Abstraction focus is on “what” should be done.	Encapsulation focus is on “How” it should be done.

Thank you

Any question?