

# Presentation

Java

## Topics

- Encapsulation and data hiding
- The notions of data abstraction and abstract data types (ADTs)
- Use of keyword this
- Use of static variables and methods
- To import static members of a class
- Controlling Access to Members
- Inheritance
- Polymorphism
- Packages

# Encapsulation and data hiding

**Encapsulation** in Java is a mechanism of **wrapping the data** (variables) and **code** acting on the data (methods) together as a single unit.

In encapsulation, the **variables of a class** will be **hidden** from other classes, and can be **accessed** only through the **methods** of their current class.

Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java –

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

# Advantage of Encapsulation in Java

1. By providing only a setter or getter method, make the class **read-only or write-only**.
2. Provides **control over the data**.
3. It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.
4. The encapsulate class is **easy to test**. So, it is better for unit testing.
5. The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

# Example of Encapsulation

Here is an example of Java class which is a fully encapsulated class. It has a private data member and getter and setter methods.

```
package com.testEncapsulation;  
public class Student{  
    private String name; //private data member  
  
    public String getName(){ //getter method for name  
        return name;  
    }  
  
    public void setName(String name){ //setter method for name  
        this.name=name  
    }  
}
```

## Example continued

//A Java class to test the encapsulated class

**package** com. testEncapsulation;

**class** Test{

**public static void** main(String[] args){

        Student s=**new** Student();                   //creating instance of the encapsulate  
        d class

        s.setName("Mitu");                         //setting value in the name member

        System.out.println(s.getName()); //getting value of the name member

    }

}

Output

Mitu

# The notions of data abstraction and abstract data types (ADTs)

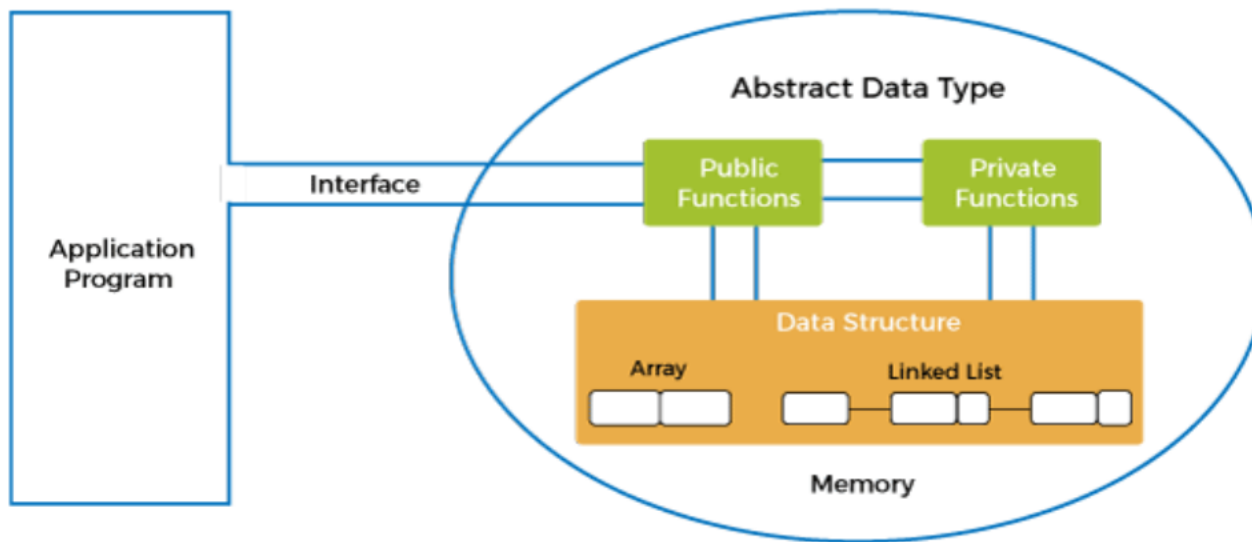
Abstraction: It is a technique of hiding the internal details from the user and only showing the necessary details to the user.

Creation of data structures along with their operations, and such data structures that are **not** in-built are known as Abstract Data Type (ADT).

Data types such as int, float, double, long, etc. are considered to be in-built data types and we can perform basic operations.

ADT defines what operations are to be performed but not how these operations will be implemented.

# The notions of data abstraction and abstract data types (ADTs)





# The notions of data abstraction and abstract data types (ADTs)

three ADTs namely List ADT, Stack ADT, Queue ADT.

**Stack ADT:** The program allocates memory for the *data* and *address* is passed to the stack ADT. The head node and the data nodes are encapsulated in the ADT. The calling function can only see the pointer to the stack.

The stack head structure also contains a pointer to *top* and *count* of number of entries currently in stack.

- `push()` – Insert an element at one end of the stack called *top*.
- `pop()` – Remove and return the element at the top of the stack, if it is not empty.
- `peek()` – Return the element at the top of the stack without removing it, if the stack is not empty.
- `size()` – Return the number of elements in the stack.
- `isEmpty()` – Return true if the stack is empty, otherwise return false.
- `isFull()` – Return true if the stack is full, otherwise return false.

# Use of keyword this

this is a **reference variable** that refers to the current object of a method or a constructor.

The main purpose of using this keyword in Java is to remove the confusion between class attributes and parameters that have same names.

Following are various uses of 'this' keyword in Java:

this can also be used to:

- Invoke current class constructor
- Invoke current class method
- Return the current class object
- Pass an argument in the method call
- Pass an argument in the constructor call

# Use of keyword this

```
class Account{  
    int a;  
    int b;  
    public void setData(int a ,int b){  
  
        a = a;  
        b = b;  
    }  
    public void showData(){  
        System.out.println("Value of A =" +a);  
        System.out.println("Value of B =" +b);  
    }  
  
    public static void main(String args[]){  
        Account obj = new Account();  
        obj.setData(2,3);  
        obj.showData();  
    }  
}
```

a = a;  
b = b;



this.a =a;  
this.b = b;

Output:  
Value of A = 2  
Value of B = 3

# Use of static variables and methods

Static variables are also called class variables.

That is, they belong to a class and not a particular instance(object).

As a result, **class initialization will initialize static variables.**

```
public class StaticVariableDemo {  
    public static int i;  
    public static int j = 20;  
  
    public StaticDemo() {}  
}
```

Here i and j both are static variables  
They belong to the class  
StaticVariableDemo

When **static** keyword with any method, it is known as static method.

A static method belongs to the class rather than the object of a class.

A static method can be invoked without the need for creating an instance of a class.

A static method can access static data member and can change the value of it.

# To import static members of a class

The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly.

There is no need to qualify it by the class name.

Less coding is required if you have access any static member of a class.

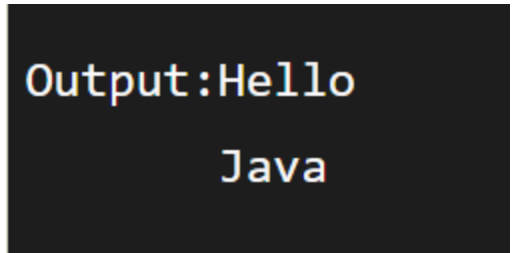
Example:

```
import static java.lang.System.*;
```

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

```
        out.println("Hello");//Now no need of System.out  
        out.println("Java");
```

```
    }  
}
```



Output:Hello  
Java

# Controlling Access to Members

We use access specifiers in Java to protect both a class's **variables** and its **methods** when you declare them. The Java language supports four distinct access levels for member variables and methods: private, protected, public, and, if left unspecified, package.

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
<i>no modifier*</i>	✓	✓	✗	✗
private	✓	✗	✗	✗

# Inheritance

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

Why use inheritance in java?

1. For Method Overriding (so runtime polymorphism can be achieved).
2. For Code Reusability

Terms used in Inheritance

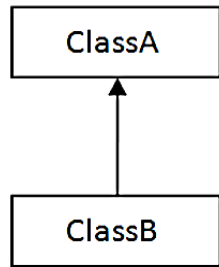
**Class:** A class is a group of objects which have **common properties**. It is a template or blueprint from which objects are created.

**Sub Class/Child Class:** Subclass is a class **which inherits the other class**. It is also called a derived class, extended class, or child class.

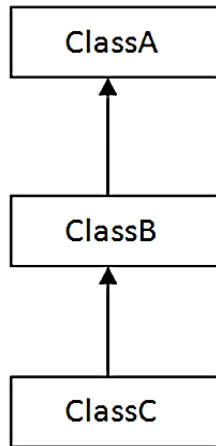
**Super Class/Parent Class:** Superclass is the **class from where** a subclass inherits the features. It is also called a base class or a parent class.

**Reusability:** A mechanism which facilitates to reuse the fields and methods of the existing class when you create a new class. The same fields and methods already defined in the previous class.

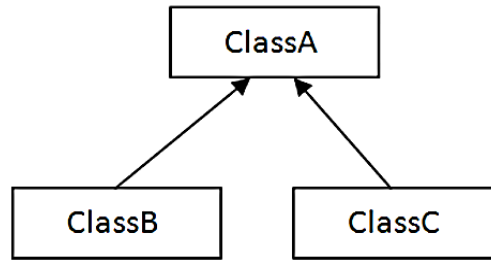
# Types of Inheritance



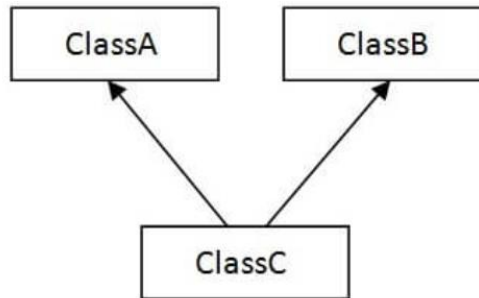
1) Single



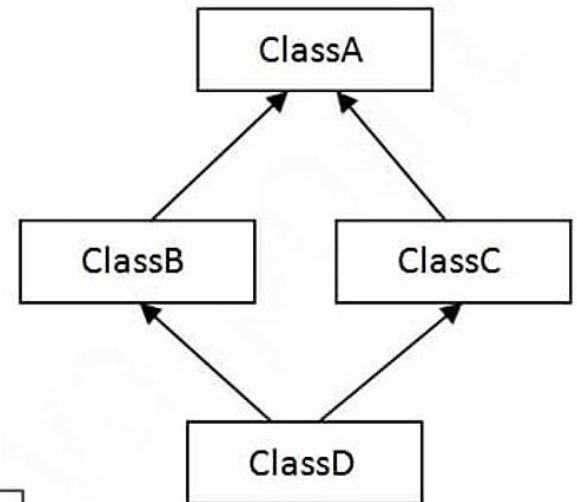
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid



# Inheritance

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

Why use inheritance in java?

1. For Method Overriding (so runtime polymorphism can be achieved).
2. For Code Reusability

Terms used in Inheritance

**Class:** A class is a group of objects which have **common properties**. It is a template or blueprint from which objects are created.

**Sub Class/Child Class:** Subclass is a class **which inherits the other class**. It is also called a derived class, extended class, or child class.

**Super Class/Parent Class:** Superclass is the **class from where** a subclass inherits the features. It is also called a base class or a parent class.

**Reusability:** A mechanism which facilitates to reuse the fields and methods of the existing class when you create a new class. The same fields and methods already defined in the previous class.

# Example:

```
class Animal{  
void eat(){System.out.println("eating...");} }  
class Dog extends Animal{  
void bark(){System.out.println("barking...");  
} }  
class TestInheritance{  
    public static void main(String args[]){  
        Dog d=new Dog();  
        d.bark();  
        d.eat();  
    }  
}
```

Output:

```
barking...  
eating...
```

A class can extends one class at a time

A class implements multiple interfaces at a time

An interface is a special type of class which implements a complete abstraction and only contains abstract methods.

# Packages

A package in Java is used to group related classes. Think of it as **a folder in a file directory**. We use packages to avoid name conflicts, and to write a better maintainable code.

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.

Packages are divided into two categories:

- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)

Example of Built-in Packages :

```
import java.util.Scanner; // Import a single class
```

```
import java.util.*; // Import the whole package
```

Thank you