

**Sukkur Institute of Business Administration University**

Department of Computer Science

**Object Oriented Programming using Java**

BS – II (CS/AI/SE)

Spring-2024

**Lab # 06: Let’s learn about class association**

**Instructor:** Nimra Mughal

# Objectives

After performing this lab, students will be able to understand:

* Pass by value, pass by reference
* Return Objects
* Recursion
* Association in java
* Java Documentation basics

# Pass by Reference

/\*In this example we have passed the object as a parameter\*/

class Rectangle {

    int length;

    int width;

    Rectangle(int length, int width) {

        this.length = length;

        this.width = width;

    }

}

public class PassObjectExample {

    public static void main(String[] args) {

        Rectangle rect = new Rectangle(5, 4);

        System.out.println("Before modifying: Length = " + rect.length + ", Width = " + rect.width);

        modifyRectangle(rect);

        System.out.println("After modifying: Length = " + rect.length + ", Width = " + rect.width);

    }

    public static void modifyRectangle(Rectangle r) {

        r.length = 10;

        r.width = 8;

    }

}

/\*In this example, we have passed the array as an object which is another example of pass-by reference\*/

public class Example {

  public static void modifyArray(int[] arr) {

    arr[0] = 10;

  }

  public static void main(String[] args) {

    int[] numbers = {1, 2, 3};

    modifyArray(numbers);

    System.out.println(numbers[0]); // Output: 10

  }

}

# Return object

Implement a method to search for a product and return the product object containing details like name, price, and availability. Ensure proper handling of object return and manipulation.

import java.util.Scanner;

class Product {

    String name;

    double price;

    boolean available;

    Product(String name, double price, boolean available) {

        this.name = name;

        this.price = price;

        this.available = available;

    }

    void displayInfo() {

        System.out.println("Name: " + name);

        System.out.println("Price: $" + price);

        System.out.println("Availability: " + (available ? "Available" : "Not Available"));

    }

}

public class ShoppingCart {

    static Product[] products = new Product[3]; // Array to hold products

    static int cartSize = 0;

    public static void main(String[] args) {

        // Adding sample products

        products[0] = new Product("Laptop", 999.99, true);

        products[1] = new Product("Smartphone", 599.99, true);

        products[2] = new Product("Headphones", 99.99, false);

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter product name to search: ");

        String searchName = scanner.nextLine();

        Product foundProduct = searchProductByName(searchName);

        if (foundProduct != null) {

            System.out.println("Product found:");

            foundProduct.displayInfo();

            System.out.println("Adding product to cart...");

            addToCart(foundProduct);

            System.out.println("Product added to cart successfully!");

        } else {

            System.out.println("Product with name '" + searchName + "' not found.");

        }

    }

    public static Product searchProductByName(String name) {

        for (Product product : products) {

            if (product != null && product.name.equalsIgnoreCase(name)) {

                return product;

            }

        }

        return null; // Return null if product not found

    }

}

# Final keyword

Develop a Java program for a **banking system**. Use static variables and methods to keep track of the total number of accounts and the bank's interest rate. **Utilize final** keyword to create immutable classes for important entities like Account, ensuring their values cannot be changed once initialized.

class Account {

    private final int accountNumber;

    private final String accountHolderName;

    private double balance;

    public Account(int accountNumber, String accountHolderName, double balance) {

        this.accountNumber = accountNumber;

        this.accountHolderName = accountHolderName;

        this.balance = balance;

    }

    public int getAccountNumber() {

        return accountNumber;

    }

    public String getAccountHolderName() {

        return accountHolderName;

    }

    public double getBalance() {

        return balance;

    }

    public void deposit(double amount) {

        balance += amount;

        System.out.println(amount + " deposited successfully. Current balance: " + balance);

    }

    public void withdraw(double amount) {

        if (balance >= amount) {

            balance -= amount;

            System.out.println(amount + " withdrawn successfully. Current balance: " + balance);

        } else {

            System.out.println("Insufficient balance. Withdrawal failed.");

        }

    }

}

public class BankingSystem {

    private static int totalAccounts = 0;

    private static final double INTEREST\_RATE = 0.05;

    public static void main(String[] args) {

        Account account1 = new Account(101, "Alice", 1000);

        Account account2 = new Account(102, "Bob", 2000);

        totalAccounts += 2; // Increment totalAccounts when new accounts are

        System.out.println("Total number of accounts: " + totalAccounts);

        System.out.println("Bank's interest rate: " + INTEREST\_RATE);

        // Perform operations on accounts

        account1.deposit(500);

        account2.withdraw(300);

    }

}

# Recursion

**Base case** (i.e., when to stop), where the function directly computes an answer without calling itself.

**Recursive case** (i.e., call ourselves), where the function calls itself as part of the computation.

Perhaps the simplest example is calculating factorial: 𝑛! = 𝑛 ∙ (𝑛 − 1) ∙ ⋯∙ 2 ∙ 1. However, we can also see that 𝑛! = 𝑛 ∙ (𝑛 − 1)!. Thus, factorial is defined in terms of itself. For example,

factorial( 5 ) = 5 \* factorial( 4 )

= 5 \* ( 4 \* factorial( 3 ) )

= 5 \* ( 4 \* (3 \* factorial( 2 ) ) )

= 5 \* ( 4 \* (3 \* (2 \* factorial( 1 ) ) ) )

= 5 \* ( 4 \* (3 \* (2 \* ( 1 \* factorial( 0 ) ) ) ) )

= 5 \* ( 4 \* (3 \* (2 \* ( 1 \* 1 ) ) ) )

= 5 \* 4 \* 3 \* 2 \* 1 \* 1 = 120

We can trace this computation in precisely the same way that we trace any sequence of function calls.

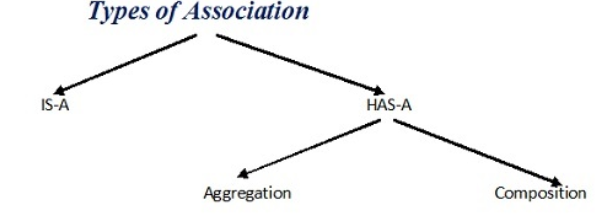
|  |
| --- |
| factorial(5)  factorial(4)  factorial(3)  factorial(2)  factorial(1)  return 1  return 2\*1 = 2  return 3\*2 = 6  return 4\*6 = 24  return 5\*24 = 120 |

*When a recursive call is made, new storage locations for variables are allocated on the stack*

# Association in JAVA

Association establishes relationship between two separate classes through their objects. Association relationship indicates how objects know each other and how they are using each other’s functionality. The relationship can be one to one, One to many, many to one and many to many.

In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object. **Composition** and **Aggregation** are the two forms of association.



**Composition**

It is a “belongs-to” type of association. It simply means, it is a part or member of the larger object. Alternatively, it is often called a “has-a” relationship.

For example, a building has a room, or in other words, a room belongs to a building. Composition is a strong kind of “has-a” relationship because the objects’ lifecycles are tied. It means that if we destroy the owner object, its members also will be destroyed with it.

//Car must have Engine

public class Car {

   //engine is a mandatory part of the car

   private final Engine engine;

   public Car () {

     engine = new Engine();

   }

}

//Engine Object

class Engine {}

**Aggregation**

Aggregation is also a “has-a” relationship, but, what distinguishes it from composition, is that the lifecycles of the objects are not tied. Both the entries can survive individually which means ending one entity will not affect the other entity. Both of them can exist independently of each other. Therefore, it is often referred to as week association.

For example: A player who is a part of the team can exist even when the team ceases to exist. The main reason why you need Aggregation is to maintain code reusability.

//Team

public class Team {

   //players can be 0 or more

   private int players[];

   public Team () {

      players = new int[10];

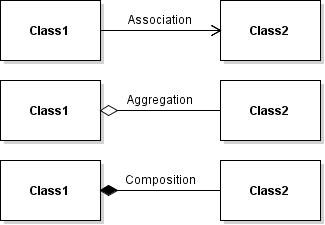
   }

Player p=new Player();

}

//Player Object

class Player {}



// Java program to illustrate the concept of Association

import java.io.\*;

class Bank

{

    private String name;

    Bank(String name)

    {

        this.name = name;

    }

    public String getBankName()

    {

        return this.name;

    }

}

// employee class

class Employee

{

    private String name;

    // employee name

    Employee(String name)

    {

        this.name = name;

    }

    public String getEmployeeName()

    {

        return this.name;

    }

}

// Association between both the classes in main method

class Association

{

    public static void main (String[] args)

    {

        Bank bank = new Bank("Axis");

        Employee emp = new Employee("Neha");

        System.out.println(emp.getEmployeeName() +

            " is employee of " + bank.getBankName());

    }

}

*In above example two separate classes Bank and Employee are associated through their Objects. Bank can have many employees, So it is a one-to-many relationship.*

Summary

# Java Documentation Comments

import java.io.\*;

/\*\*

\* Add Two Numbers!

\* The AddNum program implements an application that

\* simply adds two given integer numbers and Prints

\* the output on the screen.

\* <p>

\* <b>Note:</b> Giving proper comments in your program makes it more

\* user friendly and it is assumed as a high quality code.

\*

\* @author  Zara Ali

\* @version 1.0

\* @since   2014-03-31

\*/

public class Account {

   /\*\*

   \* This is the main method which makes use of addNum method.

   \* @param args Unused.

   \* @exception IOException On input error.

   \* @see IOException

   \*/

   public static void main(String args[]) throws IOException {

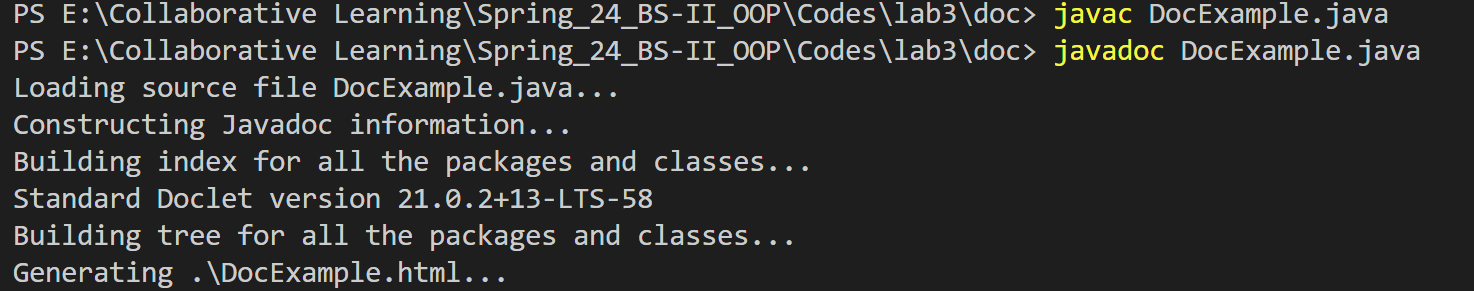
      AddNum obj = new AddNum();

      int sum = obj.addNum(10, 20);

      System.out.println("Sum of 10 and 20 is :" + sum);

   }

}



**The javadoc tool recognizes the following tags:**

<https://www.tutorialspoint.com/java/java_documentation.htm>

<https://www.javatpoint.com/java-comments>

# Exercises

Mega Exercise: Solve Exercises from the HackerRank **(OOP track topics). As many as you can. I have listed here two basic challenges)**

<https://www.hackerrank.com/domains/java>

## Exercise 1 Pass by reference MedicalInventoryManager

Create a Java program to manage medical inventory. The program should include two classes namely **Medicine** and **MedicalInventoryManager**. The medicine class has **MedicineName, strength, and Quantity** attributes and a Method displayDetails() to display the details of medicines.  **MedicalIventoryManager** class has a method to update the quantity of a particular medicine by passing its reference and to display the details of a medicine by passing its value.

**Method Names:**

updateMedicineQuantity(Medicine medicine, int quantityToAdd)

This method updates the quantity of a particular medicine by passing its reference.

Parameters:

medicine: Reference to the medicine object whose quantity needs to be updated.

quantityToAdd: The quantity to add to the current quantity of the medicine.

displayDetails()

This method displays the details of a medicine by passing its value.

## Exercise 2 (Return object)

In previous classes, you implemented a program where you created an array of Students. Please extend that code, Implement a method to search for a student by ID, and return the student object.

* **Student** class represents a student with attributes such as id, name, and age.
* **StudentDatabase** class holds the main method. It also contains a static array **students** to store student objects.
* In the **main** method, **students** array is created and values are initilizaed for each student.
* A scanner object is created to get user input for searching a student by ID.
* The **searchStudentById(int ID)** takes the **ID** as a parameter and iterates through the **students** array to find a student with the given ID. If found, it returns the student object; otherwise, it returns null.
* The result of the search is displayed in the **main** method. If the student is found, their information is displayed; otherwise, a message indicating that the student was not found is displayed.

## Exercise 3 Even Fibonacci Sum (Recursion) EvenFibSum.java

Write a method that returns the sum of all Fibonacci numbers using recursion. Consider

all Fibonacci numbers that are less than or equal to n.

Each new element in the Fibonacci sequence is generated by adding the previous two

elements.

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Method should be:

public Integer evenFibonacciSum(Integer n)

## Exercise 4 (Recursion)

Write a recursive function to compute power of a number (Xn ). Test and trace for 4 5 ? Hint: 45 =4 \* 44 ; 40=1.

## Exercise 5(a) (Association & Aggregation) *(Time.java, Passenger.java, Flight.java)*

Based On the class diagram given below, you are required to write complete program for Flight's class, Time's class and Passenger's class with the concept of association and aggregation. Functions information also been given in the table below. Program should display information supplied to flight object.

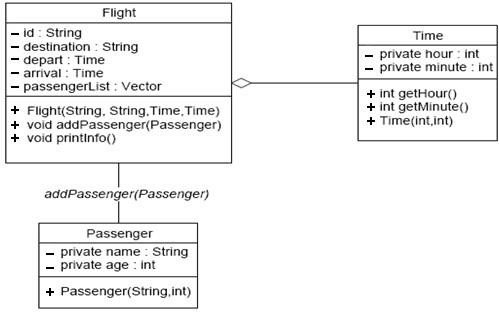


Table of Methods for Flight Class

|  |  |
| --- | --- |
| **Method** | **Description** |
| addPassenger(Passenger) | This method will add Passenger's object to vector passengerList. |
| printInfo() | This method will display all flight information namely ID (Flight number), destination, departure time, arrival time and number of passengers. For Example:  Flight no : PK-203  Destination : Sukkur  Departure : 8:10  Arrival : 9:00  Number of passenger :3 |
| getHour() | This method will return the value of attribute **hour** |
| getMinute() | This method will return the value of attribute **minute** |

## Exercise 5(b) *(FlightTester.java)*

Use the above classes to create the objects in the FlightTester class and call methods:

/\*

\* FlightTester class

\*/ public class FlightTester { public static void main(String arg[])

{

Time dep=new Time (8,10);

Time arr=new Time (9,00);

Flight f = new Flight("PK-303","Lahore",dep,arr);

Passenger psg1= new Passenger("Aariz", 30);

Passenger psg2= new Passenger("Arsham", 44); f.addPassenger(psg1);

f.addPassenger(psg2);

f.printInfo();

}

}