Sukkur Institute of Business Administration University
Department of Computer Science
BS – II (CS/SE/AI) Spring 2024

Object Oriented Programming

Lab # 06: To become familiar with Association in JAVA

Instructor: Engr. Zainab Umair Kamangar

Lab Report Rubrics (Add the points in each column, then add across the bottom row to find the total score)					**Total Marks**
S.No	**Criterion**	**0.5**	**0.25**	**0.125**	
1	Accuracy	☐ Desired output	☐ Minor mistakes	☐ Critical mistakes	
2	Timing	☐ Submitted within the given time	☐ 1 day late	☐ More than 1 day late	

Submission Profile

Name: Submission date (dd/mm/yy):
Enrollment ID: Receiving authority name and signature:
Comments:

_____
Instructor Signature

Note: Submit this lab hand-out in the next lab with attached solved activities and exercises

Objectives

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

- Pass by Value and Pass by reference Example
- Association in JAVA

Pass by Value (Example)

```
/*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 PassByValue {
   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.length,rect.width);
      System.out.println("After modifying: Length = " + rect.length + ", Width = " + rect.width);
   }

   public static void modifyRectangle(int length, int width) {
      length = 10;
      width = 8;
        System.out.println("During modifying: Length = " + length + ", Width = " + width);
   }
}
```

Pass by Reference (Example)

```
/*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 PassByReference {
   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;
          System.out.println("After modifying: Length = " + r.length + ", Width = " + r.width);
   }
}
```

```
/*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.

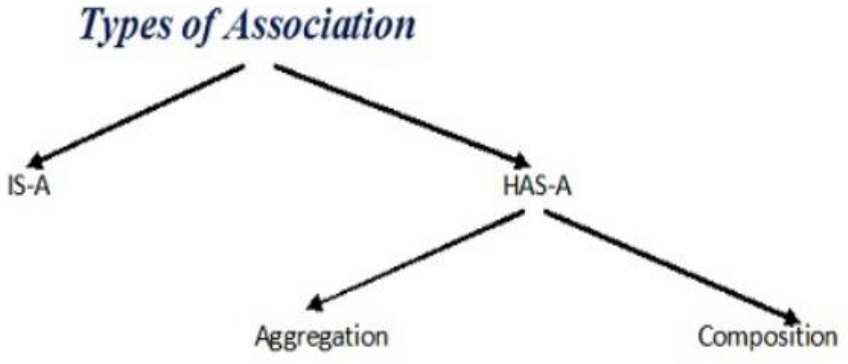
```
class Account {
    private final int accountNumber;
    private final String accountHolderName;
    private double balance;

    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);
    }
}
```

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.

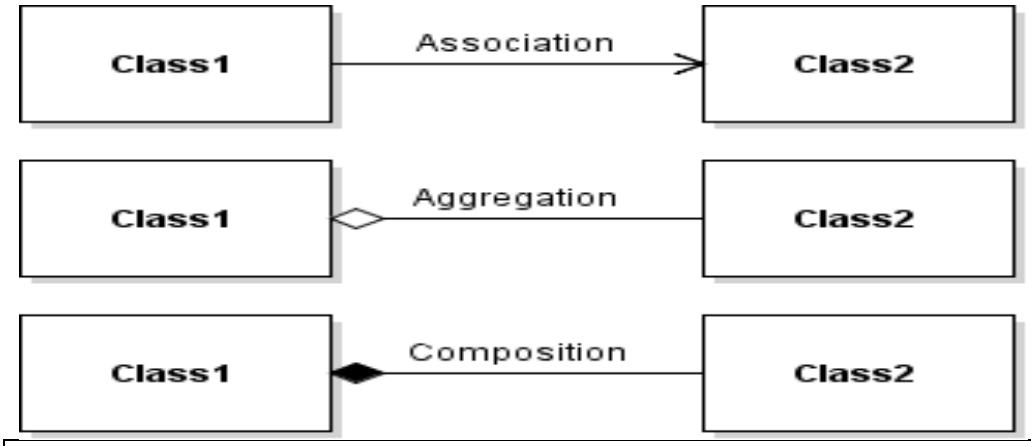
```
//Car must have Engine
public class Car {
   //engine is a mandatory part of the car
   private final Engine engine;

   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 weak 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.

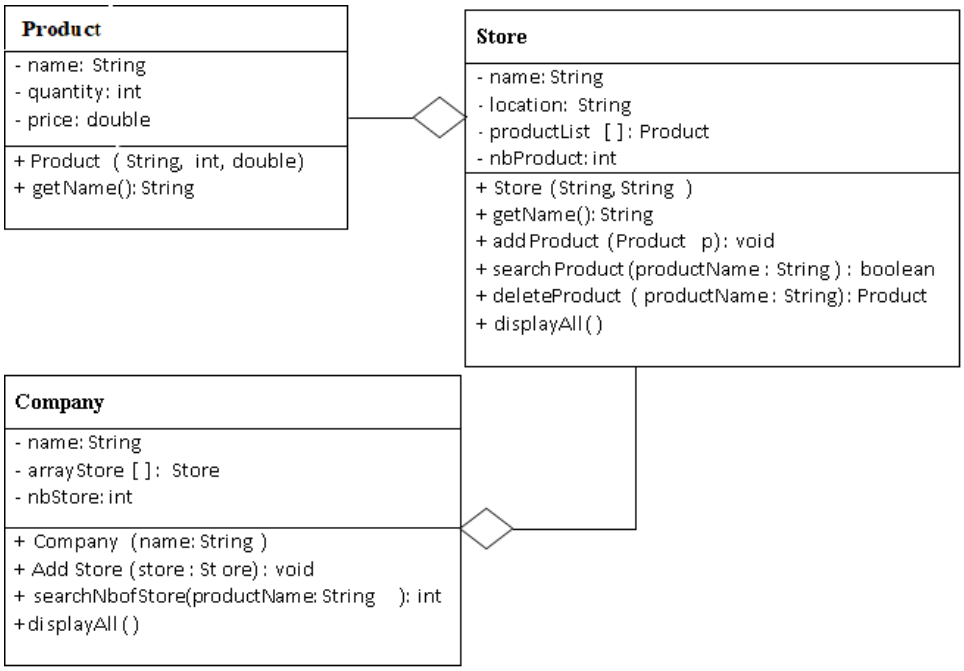
Summary

- Association follows many-to-many relationships.
- Aggregation follows a one-to-one relationship.
- The Composition follows a one-to-many relationship.

Exercises

Question 1: (Ass: & Agggregation) **(Product.java, Store.java, Company.java)**

A company manages many stores. Each Store contains many Products. The UML diagram of this company system is represented as follow:



Class Store:

Attribute: name, location, productList, nbProduct
Constructor: Store (name: String, location: String):
Method:

addProduct() that adds a new product. Maximum 100 products can be added. searchProduct() that accepts the name of product and return **True** if exist, **False** otherwise. deleteProduct() that accepts the name of product that has to be deleted and returns the deleted object.

displayAll() prints the name of products available in store.

Class Company:

Attribute: name, arrayStore, nbStore
Constructor: Company (name: string):
Method:

addStore() that adds a new Store. Maximum 10 stores can be added.

searchNbofStore() that accepts the name of product and returns the number of stores containing the product. displayAll() prints the name of stores belongs to company.

Question 2: **(TestCompany.java)**

```
public class TestCompany {
    public static void main(String [] args){
        Product p1 = new Product("TV",4,34000);
        Product p2 = new Product("Bicycle", 4, 5500);
        Product p3 = new Product("Oven", 3,70000);

        Store s1 = new Store("Makro", "Karachi");
        Store s2 = new Store("Hypermart","Karachi");
        s1.addProduct(p1);
        s1.addProduct(p2);
        s1.addProduct(p3);
        s1.displayAll();
        Product tempProduct = s1.deleteProduct("Bicycle");
        if (tempProduct!=null)
            System.out.println("Product "+tempProduct.getName()+ " is
deleted");
        else
            System.out.println("There is no product to delete");
         s1.displayAll();
         s2.addProduct(p1);
         s2.addProduct(p2);
         s2.addProduct(p3);
         s2.displayAll();
         Company c1 = new Company("Unilever");
         c1.addStore(s1);
         c1.addStore(s2);
         c1.displayAll();
         int n= c1.searchNbofStore("TV");
         System.out.println("Number of stores have TV "+n);

    }
}
```

Implement Product, Store and Company classes and use the following class to test.