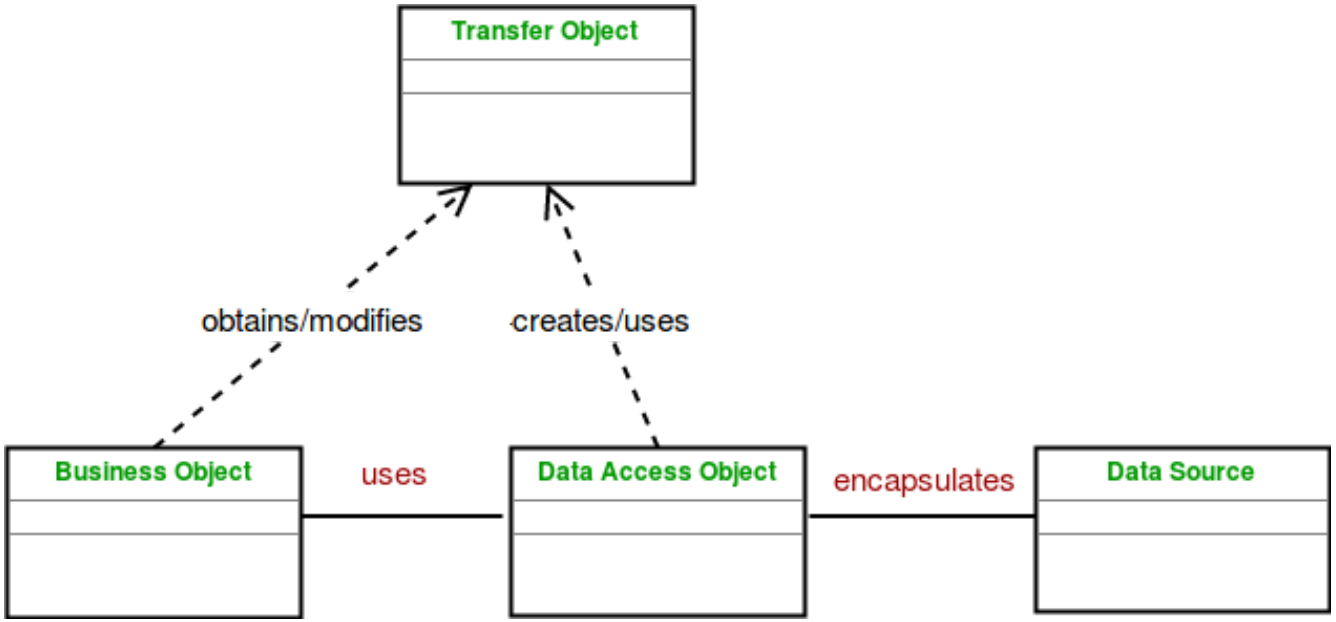
Data Access Object Pattern

Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services. Following are the participants in Data Access Object Pattern.

UML Diagram Data Access Object Pattern



Design components

- **BusinessObject :** The BusinessObject represents the data client. It is the object that requires access to the data source to obtain and store data. A BusinessObject may be implemented as a session bean, entity bean or some other Java object in addition to a servlet or helper bean that accesses the data source.
- **DataAccessObject :** The DataAccessObject is the primary object of this pattern. The DataAccessObject abstracts the underlying data access implementation for the BusinessObject to enable transparent access to the data source.
- **DataSource :** This represents a data source implementation. A data source could be a database such as an RDBMS, OODBMS, XML repository, flat file system, and so forth. A data source can also be another system service or some kind of repository.
- **TransferObject :** This represents a Transfer Object used as a data carrier. The DataAccessObject may use a Transfer Object to return data to the client. The DataAccessObject may also receive the data from the client in a Transfer Object to update the data in the data source.

Let's see an example of Data Access Object Pattern.

```
// Java program to illustrate
// Data Access Object Pattern
import java.util.List;
import java.util.ArrayList;
import java.util.List;

class Developer
{
    private String name;
    private int DeveloperId;

    Developer(String name, int DeveloperId)
    {
        this.name = name;
        this.DeveloperId = DeveloperId;
    }

    public String getName()
    {
    return name;
    }

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

    public int getDeveloperId()
    {
    return DeveloperId;
    }

    public void setDeveloperId(int DeveloperId)
    {
    this.DeveloperId = DeveloperId;
    }
}
```

```
interface DeveloperDao
{
    public List<Developer> getAllDevelopers();
    public Developer getDeveloper(int DeveloperId);
    public void updateDeveloper(Developer Developer);
    public void deleteDeveloper(Developer Developer);
}

class DeveloperDaoImpl implements DeveloperDao
{
    List<Developer> Developers;

    public DeveloperDaoImpl()
    {
        Developers = new ArrayList<Developer>();
        Developer Developer1 = new Developer("Kushagra",0);
        Developer Developer2 = new Developer("Vikram",1);
        Developers.add(Developer1);
        Developers.add(Developer2);
    }
    @Override
    public void deleteDeveloper(Developer Developer)
    {
        Developers.remove(Developer.getDeveloperId());
        System.out.println("DeveloperId " + Developer.getDeveloperId()
        + ", deleted from database");
    }
    @Override
    public List<Developer> getAllDevelopers()
    {
        return Developers;
    }

    @Override
    public Developer getDeveloper(int DeveloperId)
    {
        return Developers.get(DeveloperId);
    }

    @Override
    public void updateDeveloper(Developer Developer)
```

```
        {
            Developers.get(Developer.getDeveloperId()).setName(Developer.getName());
            System.out.println("DeveloperId " + Developer.getDeveloperId()
            + ", updated in the database");
        }
    }

    class DaoPatternDemo
    {
        public static void main(String[] args)
        {
            DeveloperDao DeveloperDao = new DeveloperDaoImpl();

            for (Developer Developer : DeveloperDao.getAllDevelopers())
            {
                System.out.println("DeveloperId : " + Developer.getDeveloperId()
                + ", Name : " + Developer.getName());
            }

            Developer Developer =DeveloperDao.getAllDevelopers().get(0);
            Developer.setName("Lokesh");
            DeveloperDao.updateDeveloper(Developer);

            DeveloperDao.getDeveloper(0);
            System.out.println("DeveloperId : " + Developer.getDeveloperId()
            + ", Name : " + Developer.getName());
        }
    }
```

Output:

```
DeveloperId : 0, Name : Kushagra
DeveloperId : 1, Name : Vikram
DeveloperId 0, updated in the database
DeveloperId : 0, Name : Lokesh
```

Advantages :

- The advantage of using data access objects is the relatively simple and rigorous separation between two important parts of an application that can but should not know anything of each other, and which can be expected to evolve frequently and independently.
- if we need to change the underlying persistence mechanism we only have to change the DAO layer, and not all the places in the domain logic where the DAO layer is used from.

Disadvantages :

- Potential disadvantages of using DAO is leaky abstraction, code duplication, and abstraction inversion.

Reference :

http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

This article is contributed by **Saket Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Command Pattern

Observer Pattern | Set 1 (Introduction)

Observer Pattern | Set 2 (Implementation)

Singleton Design Pattern | Implementation

Decorator Pattern | Set 1 (Background)

The Decorator Pattern | Set 2 (Introduction and Design)

Decorator Pattern | Set 3 (Coding the Design)

Strategy Pattern | Set 1 (Introduction)

Strategy Pattern | Set 2 (Implementation)

Adapter Pattern

Iterator Pattern

Curiously recurring template pattern (CRTP)

Flyweight Design Pattern

Implementing Iterator pattern of a single Linked List

Singleton Design Pattern | Introduction

Article Tags : Design Pattern

Be the First to upvote.

To-do Done

3

Based on **1** vote(s)

Feedback/ Suggest Improvement | Add Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments