



## LEARN JAVA DESIGN PATTERNS

### problem solving approaches

#### Design Patterns Tutorial

- [Design Patterns - Home](#)
- [Design Patterns - Overview](#)
- [Design Patterns - Factory Pattern](#)
- [Abstract Factory Pattern](#)
- [Design Patterns - Singleton Pattern](#)
- [Design Patterns - Builder Pattern](#)
- [Design Patterns - Prototype Pattern](#)
- [Design Patterns - Adapter Pattern](#)
- [Design Patterns - Bridge Pattern](#)
- [Design Patterns - Filter Pattern](#)
- [Design Patterns - Composite Pattern](#)
- [Design Patterns - Decorator Pattern](#)
- [Design Patterns - Facade Pattern](#)
- [Design Patterns - Flyweight Pattern](#)
- [Design Patterns - Proxy Pattern](#)
- [Chain of Responsibility Pattern](#)
- [Design Patterns - Command Pattern](#)
- [Design Patterns - Interpreter Pattern](#)
- [Design Patterns - Iterator Pattern](#)
- [Design Patterns - Mediator Pattern](#)
- [Design Patterns - Memento Pattern](#)
- [Design Patterns - Observer Pattern](#)
- [Design Patterns - State Pattern](#)
- [Design Patterns - Null Object Pattern](#)
- [Design Patterns - Strategy Pattern](#)
- [Design Patterns - Template Pattern](#)
- [Design Patterns - Visitor Pattern](#)
- [Design Patterns - MVC Pattern](#)
- [Business Delegate Pattern](#)
- [Composite Entity Pattern](#)
- [Data Access Object Pattern](#)
- [Front Controller Pattern](#)
- [Intercepting Filter Pattern](#)
- [Service Locator Pattern](#)
- [Transfer Object Pattern](#)

#### Design Patterns Resources

- [Design Patterns - Questions/Answers](#)

## Data Access Object Pattern

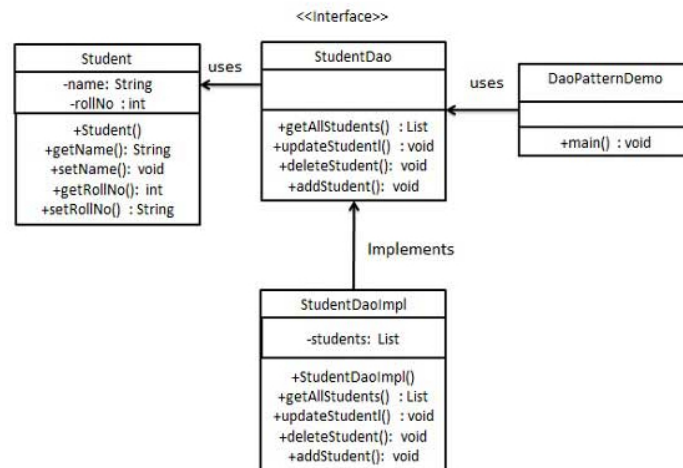
[Previous Page](#)
[Next Page](#)

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.

- **Data Access Object Interface** - This interface defines the standard operations to be performed on a model object(s).
- **Data Access Object concrete class** - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
- **Model Object or Value Object** - This object is simple POJO containing get/set methods to store data retrieved using DAO class.

### Implementation

We are going to create a *Student* object acting as a Model or Value Object. *StudentDao* is Data Access Object Interface. *StudentDaoImpl* is concrete class implementing Data Access Object Interface. *DaoPatternDemo*, our demo class, will use *StudentDao* to demonstrate the use of Data Access Object pattern.



### Step 1

Create Value Object.

*Student.java*

```

public class Student {
    private String name;
    private int rollNo;

    Student(String name, int rollNo){
        this.name = name;
        this.rollNo = rollNo;
    }

    public String getName() {
        return name;
    }

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

    public int getRollNo() {
        return rollNo;
    }

    public void setRollNo(int rollNo) {
    }
}
    
```

- Design Patterns - Quick Guide
- Design Patterns - Useful Resources
- Design Patterns - Discussion

#### Selected Reading

- UPSC IAS Exams Notes
- Developer's Best Practices
- Questions and Answers
- Effective Resume Writing
- HR Interview Questions
- Computer Glossary
- Who is Who

```
        this.rollNo = rollNo;
    }
}
```

## Step 2

Create Data Access Object Interface.

*StudentDao.java*

```
import java.util.List;

public interface StudentDao {
    public List<Student> getAllStudents();
    public Student getStudent(int rollNo);
    public void updateStudent(Student student);
    public void deleteStudent(Student student);
}
```

## Step 3

Create concrete class implementing above interface.

*StudentDaoImpl.java*

```
import java.util.ArrayList;
import java.util.List;

public class StudentDaoImpl implements StudentDao {

    //list is working as a database
    List<Student> students;

    public StudentDaoImpl(){
        students = new ArrayList<Student>();
        Student student1 = new Student("Robert",0);
        Student student2 = new Student("John",1);
        students.add(student1);
        students.add(student2);
    }

    @Override
    public void deleteStudent(Student student) {
        students.remove(student.getRollNo());
        System.out.println("Student: Roll No " + student.getRollNo() + ", deleted from database");
    }

    //retrive list of students from the database
    @Override
    public List<Student> getAllStudents() {
        return students;
    }

    @Override
    public Student getStudent(int rollNo) {
        return students.get(rollNo);
    }

    @Override
    public void updateStudent(Student student) {
        students.get(student.getRollNo()).setName(student.getName());
        System.out.println("Student: Roll No " + student.getRollNo() + ", updated in the database");
    }
}
```

## Step 4

Use the *StudentDao* to demonstrate Data Access Object pattern usage.

*DaoPatternDemo.java*

```
public class DaoPatternDemo {
    public static void main(String[] args) {
        StudentDao studentDao = new StudentDaoImpl();

        //print all students
        for (Student student : studentDao.getAllStudents()) {
            System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : " + student.getName() + "]");
        }
    }
}
```

```
//update student
Student student =studentDao.getAllStudents().get(0);
student.setName("Michael");
studentDao.updateStudent(student);

//get the student
studentDao.getStudent(0);
System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : " + student
}
}
```

## Step 5

Verify the output.

```
Student: [RollNo : 0, Name : Robert ]
Student: [RollNo : 1, Name : John ]
Student: Roll No 0, updated in the database
Student: [RollNo : 0, Name : Michael ]
```

[⏪ Previous Page](#) [🖨 Print Page](#)

[Next Page ⏩](#)



[🌐 About us](#)

[✱ Terms of use](#)

[🛡 Privacy Policy](#)

[❓ FAQ's](#)

[👉 Helping](#)

[📍 Contact](#)

© Copyright 2020. All Rights Reserved.