**In this document, First, I will tell you the basics. And basic benefits of having a DAO.**

DAO stands for data access object. A DAO is only only responsible for storing data. Why why should we need a class which only stores data and nothing else? And why do we need other classes to operate on DAO or set of DAOs? Because of cohesion and coupling. The classes should be highly cohesive and loosely coupled. And those should have a single purpose.

**Consider the following class which is very badly designed:**

import java.uitl.\*;

public class Book

{

private static Map<String,Book> bookStore;

//storage:: extra responsibilitty

private string isbn;

private string title;

private string author;

public collection<Book> findAllBooks()

{

return bookStore.value();

}

public Book findBookByIsbn(String isbn)

{

return bookStore.get(isbn);

}

public void create()

{

bookStore.put(isbn,this);

}

public void delete()

{

bookStore.remove(isbn);

}

public void update()

{

//no operation-for an in memory database

//we will update automatically in real time

}

//omitted getters and setters

}

Now, we already omit the getters and setters. Still the Book class is around 50 lines, and it hardly does anything. A real Book class would have a lot more fields. A Bookstore needs to tell you when the book was written, the edition, the price, and all sorts of other information. A bookstore also needs to be able to keep track of books somewhere more than a map. After all, we don’t want our bookstore to forget about everything when we reboot.

**Solution:**

The DAO pattern has us split up these two responsibilities. The storage responsibility of a book is separated from the actual book class.

public class Book

{

private string isbn;

private string title;

private string author;

//omitted getters and setters

}

There can be methods in in the Book class such as toString(), hashcode() and equals.

(Overriding hashcode() and equals() could be beneficial if you want to store Book class’s object in hash based collections.

Methods that have to do with a Bookstore or database are now gone.

Now,

import java.uitl.\*;

public class InMemoryBookDao

{

private static Map<String,Book> bookstore=new HashMap<String,Book> ();

public Collection<Book> findAllBooks()

{

return bookstore.values();

}

public book findBookByIsbn(Book book)

{

return bookstore.get(book.getIsbn());

}

public void create(Book book)

{

bookstore.put(book.getIsbn(),book);

}

public void delete(Book book)

{

bookstore.remove(book.getIsbn());

//using Book's Isbn as this is used as key in the bookstore

}

public void update(Book book)

{

//no operation for an in memory database

//we update automatically in real time

}

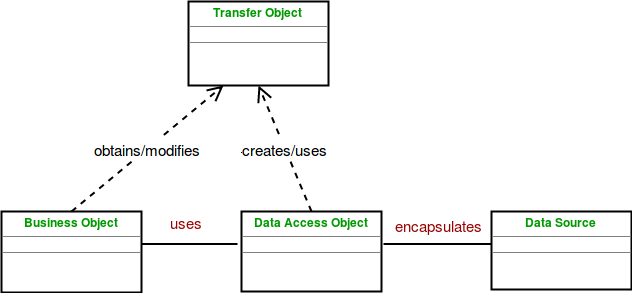
}

Now, alongside with the solutions of already mentioned problems, This DAO pattern can provide you some additional flexibility. For instance we can create an interface BookDao which is implemented by two classes. InMemoryBookDao and DatabaseBookDao. **We can follow abstractFactory design pattern to support more families of DAO over book**

**Now, upto this was just basics.**

**Why is DAO used?**

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.

****

(Now, low level data accessing APIs are separated from high end business services)

**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.

**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.

(How, DAO suffers from leaky abstraction and abstraction inversion, I need to learn)