DESIGN PATTERNS - BUSINESS DELEGATE PATTERN

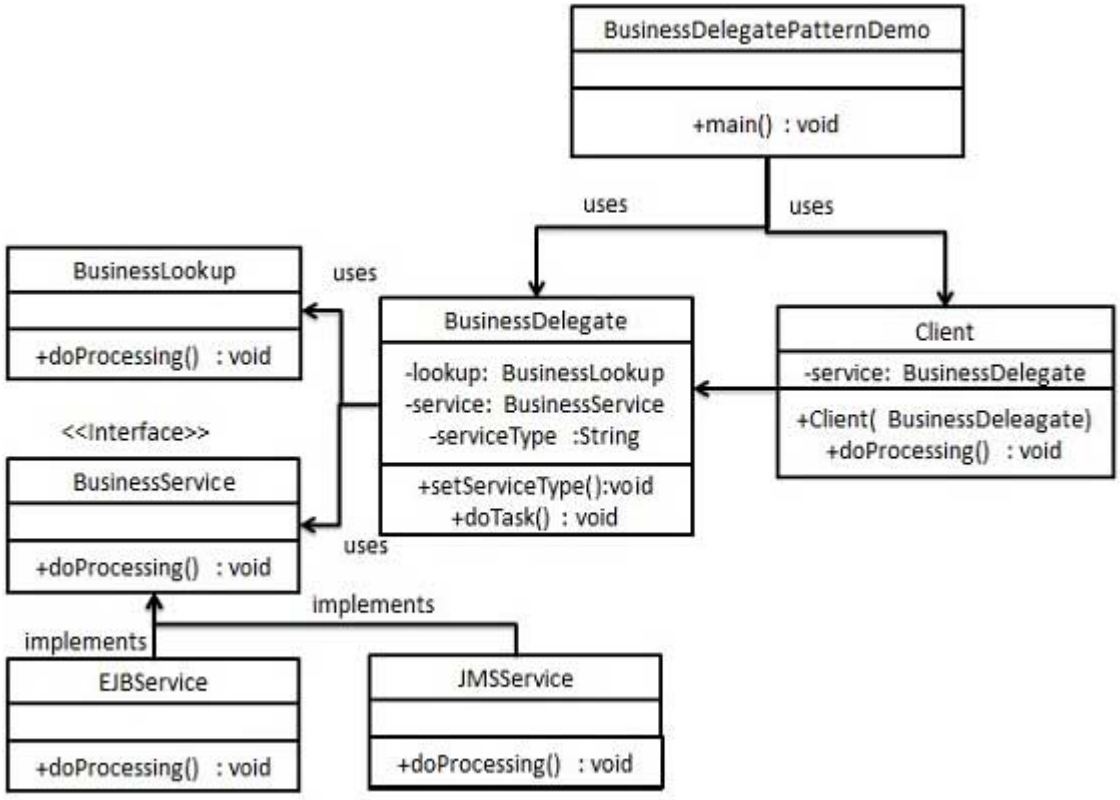
Business Delegate Pattern is used to decouple presentation tier and business tier. It is basically use to reduce communication or remote lookup functionality to business tier code in presentation tier code. In business tier we have following entities.

- **Client** - Presentation tier code may be JSP, servlet or UI java code.
- **Business Delegate** - A single entry point class for client entities to provide access to Business Service methods.
- **LookUp Service** - Lookup service object is responsible to get relative business implementation and provide business object access to business delegate object.
- **Business Service** - Business Service interface. Concrete classes implement this business service to provide actual business implementation logic.

Implementation

We are going to create a *Client*, *BusinessDelegate*, *BusinessService*, *LookUpService*, *JMSService* and *EJBService* representing various entities of Business Delegate patterns.

BusinessDelegatePatternDemo, our demo class, will use *BusinessDelegate* and *Client* to demonstrate use of Business Delegate pattern.



Step 1

Create BusinessService Interface.

BusinessService.java

```
public interface BusinessService {
   public void doProcessing();
}
```

Step 2

Create concrete Service classes.

EJBService.java

```
public class EJBService implements BusinessService {

   @Override
   public void doProcessing() {
      System.out.println("Processing task by invoking EJB Service");
   }
}
```

JMSService.java

```
public class JMSService implements BusinessService {

   @Override
   public void doProcessing() {
      System.out.println("Processing task by invoking JMS Service");
   }
}
```

Step 3

Create Business Lookup Service.

BusinessLookUp.java

```
public class BusinessLookUp {
   public BusinessService getBusinessService(String serviceType){

      if(serviceType.equalsIgnoreCase("EJB")){
         return new EJBService();
      }
      else {
         return new JMSService();
      }
   }
}
```

Step 4

Create Business Delegate.

BusinessDelegate.java

```
public class BusinessDelegate {
   private BusinessLookUp lookupService = new BusinessLookUp();
   private BusinessService businessService;
   private String serviceType;

   public void setServiceType(String serviceType){
      this.serviceType = serviceType;
   }

   public void doTask(){
      businessService = lookupService.getBusinessService(serviceType);
      businessService.doProcessing();
   }
}
```

Step 5

Create Client.

Client.java

```
public class Client {

   BusinessDelegate businessService;

   public Client(BusinessDelegate businessService){
      this.businessService  = businessService;
   }

   public void doTask(){
      businessService.doTask();
   }
}
```

Step 6

Use BusinessDelegate and Client classes to demonstrate Business Delegate pattern.

BusinessDelegatePatternDemo.java

```
public class BusinessDelegatePatternDemo {

   public static void main(String[] args) {

      BusinessDelegate businessDelegate = new BusinessDelegate();
      businessDelegate.setServiceType("EJB");

      Client client = new Client(businessDelegate);
      client.doTask();

      businessDelegate.setServiceType("JMS");
      client.doTask();
   }
}
```

Step 7

Verify the output.

```
Processing task by invoking EJB Service
Processing task by invoking JMS Service
```