

## EXPERIMENT 6

### Wrapper to connect two applications with different architectures

37\_BECOMP-B\_Mihir-Gharat

**Learning Objective:** Student should be able to understand wrapper to connect two applications with different architectures.

#### **Theory:**

##### **Wrapper Architecture:**

In wrapper, there is a clear separation of concerns: the mediator deals with data source distribution while the wrappers deal with data source heterogeneity and autonomy.

This is achieved by using a common language between mediator and wrappers, and the translation to the data source language is done by the wrappers (as shown in the diagram).

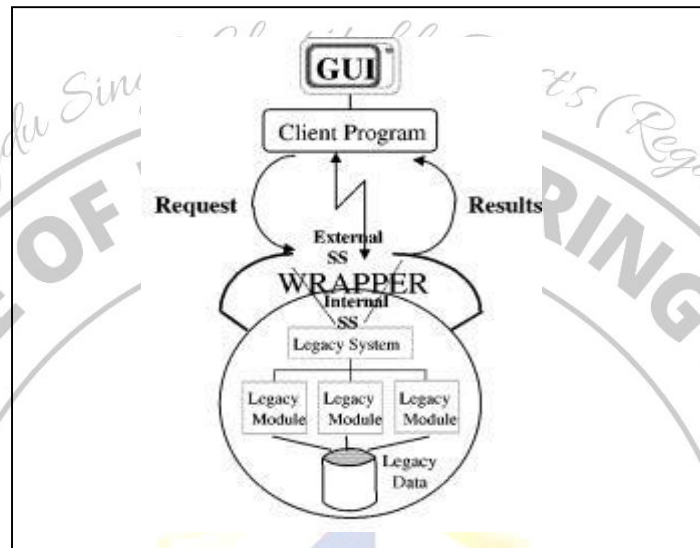
Each data source has an associated wrapper that exports information about the source schema, data and query processing capabilities. To deal with the heterogeneous nature of data sources, wrappers transform queries received from the mediator, expressed in a common query language, to the particular query language of the source. A wrapper supports the functionality of translating queries appropriate to the particular server, and reformatting answers (data) appropriate to the mediator. One of the major practical uses of wrappers has been to allow an SQL-based DBMS to access non-SQL databases.

The mediator centralizes the information provided by the the wrappers in a unified view of all available data. This unified view can be of two fundamental types local-as-view (LAV) and global-as-view (GAV). In LAV, the global schema definition exists, and each data source schema is treated as a view definition over it. In GAV on the other hand, the global schema is defined as a set of views over the data source schemas. These views indicate how the elements of the global schema can be derived, when needed, from the elements of the data source schemas.

The main functionality of the mediator is to provide uniform access to multiple data sources and perform query decomposition and processing using the wrappers to access the data sources.

Software encapsulation is based on the technology of wrapping. When asked what to do with the existing legacy software, in a new object-oriented architecture, his answer was to wrap it. Since then, there have been dozens of papers written on the subject in the object-oriented literature, but hardly any in the field of reverse- and re-engineering. One of the best technical discussions of the subject is to be found in the book “The Essential CORBA” by Mowbray and Zahari. According to these authors, an object wrapper provides access to a legacy system through an encapsulation layer. The encapsulation exposes only the attributes and operations desired by the software architect. The same authors go on to describe seven techniques for implementing a wrapper:

- remote procedure calls,
- file transfers,
- sockets or docking,
- application program interfaces,
- script procedures,
- macros,
- and – common headers.



Wrapper class between Client and server Architecture

These techniques can be implemented independently or in combination with one another to build a connection between the requester of a service and the service provider.

Database wrappers are gateways to existing databases. They allow client applications implemented in a modern object-oriented language to access data stored in a legacy database. System service wrappers provide a customized access to standard system services such as printing, sorting, routing and queuing. It is possible for a user program to invoke such services without knowledge of their internal interfaces. Application wrappers encapsulate batch processes or online transactions. They allow new client applications to include the legacy components as objects which can be called to perform certain tasks such as producing a report or updating a file. Function wrappers offer an interface to invoke individual functions within a wrapped program.

Not the program as a whole but only certain parts of it can be invoked from the client application. This amounts to a limited access.

All wrapper uses some kind of message passing mechanism to connect themselves to their clients. As a rule, the wrapper is in the same address space as the wrapped object. On the input side, it receives incoming requests, re-formats them, loads the object and invokes it with the reformatted arguments. On the output side, it takes the results from the wrapped object, re-formats them and sends them back to the requester. This is, in essence, what wrapping is all about.

Adapting a program for wrapping: It would be ideal if programs could be encapsulated without making any changes to them whatsoever. However, even in the case of module wrapping some change has to be made. Since the programs to be wrapped should also continue to operate in the normal mode, it is unlikely that they can be adapted manually. The risk of error is too high and the adaptation has to be repeated after every change to the program. So, if wrapping is to be done on a wide scale, the program adaptation has to be automated.

This will require four tools:

- a transaction wrapper,
- a program wrapper,
- a module wrapper, and
- a procedure wrapper.

### CODE AND OUTPUT:

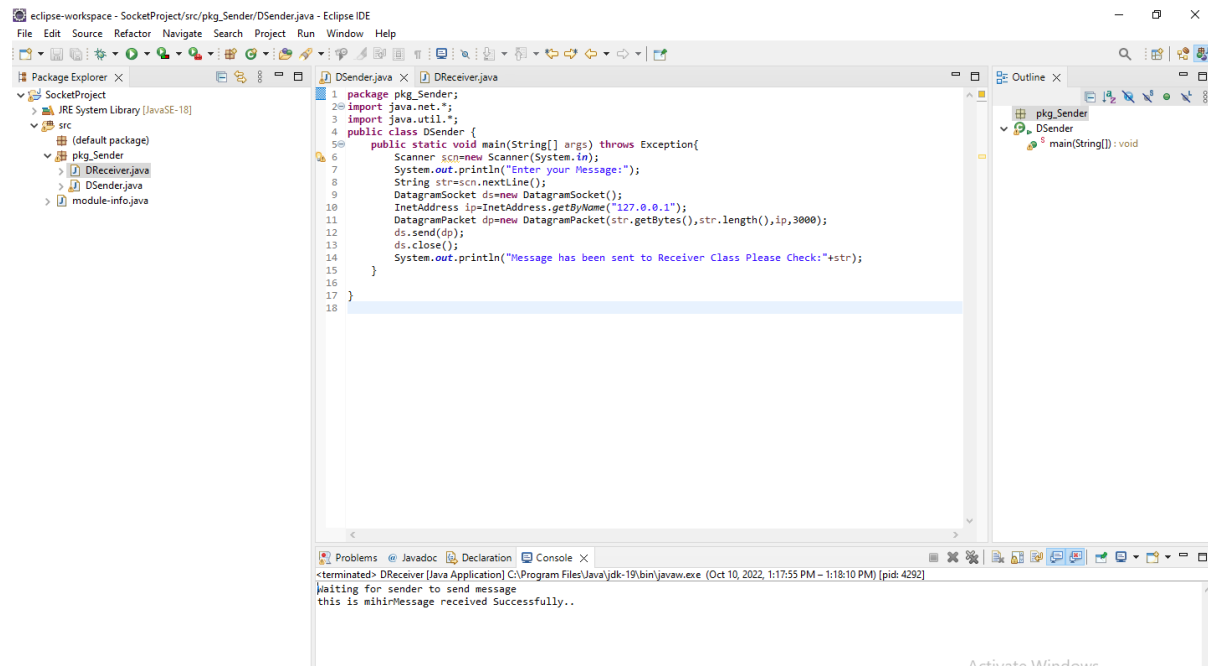
#### SENDER:

```
package pkg_Sender;
import java.net.*;
import java.util.*;
public class DSender {
    public static void main(String[] args) throws Exception{
        Scanner scn=new Scanner(System.in);
        System.out.println("Enter your Message:");
        String str=scn.nextLine();
        DatagramSocket ds=new DatagramSocket();
        InetAddress ip=InetAddress.getByName("127.0.0.1");
        DatagramPacket dp=new
        DatagramPacket(str.getBytes(),str.length(),ip,3000);
        ds.send(dp);
        ds.close();
        System.out.println("Message has been sent to Receiver Class
        Please Check:"+str);
    }
}
```

#### RECEIVER:

```
package pkg_Sender;
import java.net.*;
public class DReceiver {
    public static void main(String[] args) throws Exception{
        System.out.println("Waiting for sender to send
        message");
        DatagramSocket ds=new DatagramSocket(3000);
        byte[] buf=new byte[1024];
        DatagramPacket dp=new DatagramPacket(buf,1024);
        ds.receive(dp);
        String str=new String(dp.getData(),0,dp.getLength());
        System.out.print(str);
        ds.close();
        System.out.println("Message received Successfully..");
    }
}
```

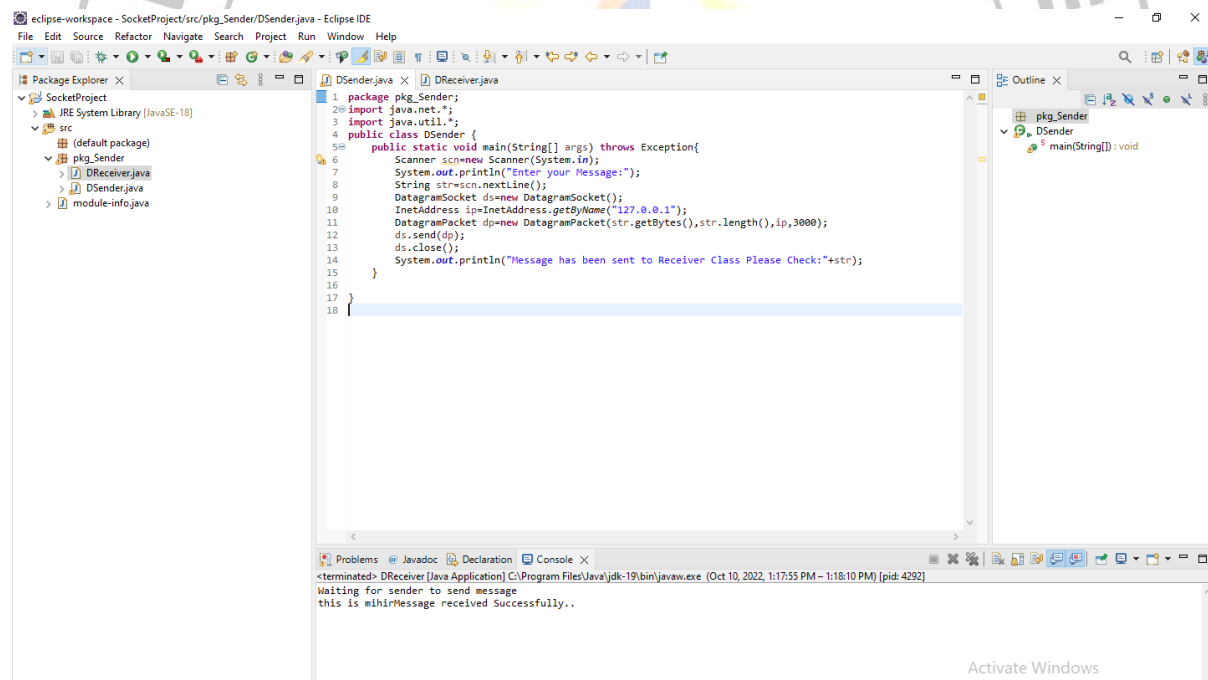
## OUTPUT:



```

1 package pkg_Sender;
2 import java.net.*;
3 import java.util.*;
4 public class DSender {
5     public static void main(String[] args) throws Exception{
6         Scanner scn=new Scanner(System.in);
7         System.out.println("Enter your Message:");
8         String str=scn.nextLine();
9         DatagramSocket ds=new DatagramSocket();
10        InetAddress ip=InetAddress.getByName("127.0.0.1");
11        DatagramPacket dp=new DatagramPacket(str.getBytes(),str.length(),ip,3000);
12        ds.send(dp);
13        ds.close();
14        System.out.println("Message has been sent to Receiver Class Please Check:"+str);
15    }
16 }
17
18
  
```

<terminated> DReceiver [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Oct 10, 2022, 1:17:55 PM ~ 1:18:10 PM) [pid: 4292]  
 Waiting for sender to send message  
 this is mihirMessage received Successfully..



```

1 package pkg_Sender;
2 import java.net.*;
3 import java.util.*;
4 public class DSender {
5     public static void main(String[] args) throws Exception{
6         Scanner scn=new Scanner(System.in);
7         System.out.println("Enter your Message:");
8         String str=scn.nextLine();
9         DatagramSocket ds=new DatagramSocket();
10        InetAddress ip=InetAddress.getByName("127.0.0.1");
11        DatagramPacket dp=new DatagramPacket(str.getBytes(),str.length(),ip,3000);
12        ds.send(dp);
13        ds.close();
14        System.out.println("Message has been sent to Receiver Class Please Check:"+str);
15    }
16 }
17
18
  
```

<terminated> DReceiver [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Oct 10, 2022, 1:17:55 PM ~ 1:18:10 PM) [pid: 4292]  
 Waiting for sender to send message  
 this is mihirMessage received Successfully..

## Result and Discussion:



### **Learning Outcomes:**

Students should have been

able to LO1: Define Wrapper.

LO2: Identify different types of wrappers.

LO3: Explain adapting a program for wrapping.

**Course Outcomes:** Upon completion of the course students will be able to understand wrapper to connect two applications with different architectures.

### **Conclusion:**

### **Viva Questions:**

1. Define wrapping.
2. Explain implementation of wrapping.
3. Explain implementation of wrapping of two different architectures.

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [ 40%]	Attendance / Learning Attitude [20%]	