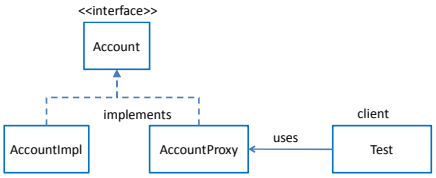
Design Patterns
Proxy, Adapter, Facade

B.Tech. (IT), Sem-6,
Applied Design Patterns and Application Frameworks (ADPAF)

Dharmsinh Desai University
Prof. H B Prajapati

The Proxy design pattern

- It is of type structural design pattern.
- In proxy design pattern, a class represents functionality of another class.



Example: The Proxy design pattern

- We have Account interface
- We implement Account interface in AccountImpl class, which contains actual business logic.
- We create AccountProxy class
  - AccountProxy does not know how methods (business logic) is implemented.
  - AccountProxy invokes method on AccountImpl class.

Example: The Proxy design pattern

```
package designpattern.proxy;
public interface Account {
   String withdraw(double amount);
   String deposit(double amount);
}
```


Example: The Proxy design pattern

```
package designpattern.proxy;

public class AccountImpl implements Account{
   double customerBalance;
   String customerName;

   public AccountImpl(String custName,double balance){
      this.customerName=custName;
      this.customerBalance=balance;
   }
```


Example: The Proxy design pattern

```
@Override
  public String withdraw(double amount) {
    String status;
    if(amount<=0)
      status = "Error: Negative Balance";
    else if((this.customerBalance-amount)<500.0)
      status = "Error: Insufficient Balance. Min. Rs. 500 needed in your
  account.";
    else{
      this.customerBalance-=amount;
      status = "Withdrawal Succefful, Updated
  balance="+this.customerBalance;
    }
    return status;
  }
```

Example: The Proxy design pattern

```
  @Override
  public String deposit(double amount) {
    String status;
    if(amount<=0)
      status = "Error: Negative Balance";
    else{
      this.customerBalance+=amount;
      status = "Deposit Succefful, Updated balance="+this.customerBalance;
    }
    return status;
  }
}
```

7

Example: The Proxy design pattern

```
package designpattern.proxy;
public class AccountProxy implements Account{
  private AccountImpl account;
  AccountProxy(String custName,double balance){
    account=new AccountImpl(custName, balance);
  }
  @Override
  public String withdraw(double amount) {
    return account.withdraw(amount);
  }
  @Override
  public String deposit(double amount) {
    return account.deposit(amount);
  }
}
```

8

Example: The Proxy design pattern

```
package designpattern.proxy;
public class Test {
  public static void main(String[] args) {
    Account myAcc=new AccountProxy("Kisan", 500.0);
    System.out.println("Deposit Rs. 10,000");
    System.out.println("Transaction Status: " + myAcc.deposit(10000));
    System.out.println("Widthdraw Rs. 10,300");
    System.out.println("Transaction Status: " + myAcc.withdraw(10300));
  }

}
```

9

Running the Example: The Proxy design pattern

Notifications | Output - DesignPattern (run) | Search Results

```
run:
Deposit Rs. 10,000
Transaction Status: Deposit Succefful, Updated balance=10500.0
Widthdraw Rs. 10,300
Transaction Status: Error: Insufficient Balance. Min. Rs. 500 needed in your account.
```

10

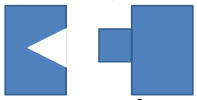
The Adapter design pattern

- It is of type structural design pattern.
- Adapter pattern works as a bridge between two incompatible interfaces.
- This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces.
- Example:
  - A card reader acts as an adapter between memory card and a computer.
  - HDMI to VGA adapter
  - USB to Mini USB

11

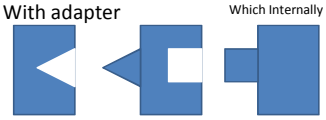
The Adapter design pattern

- Without adapter



Provider has method:
result(studentID, sem);
Consumer uses as :
getResult(sem, studentID);

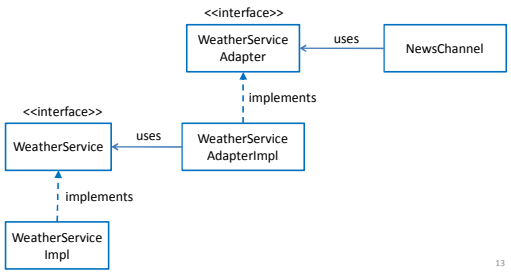
- With adapter



Adapter provides method:
getResult(sem, studentID);
Which Internally calls result()

12

The Adapter design pattern



13

Example: The Adapter design pattern

- WeatherService interface

```
package designpattern.adapter.weatherinfo;
interface WeatherService {
   public float temperature();
   public float humidity();
}
```

- WeatherServiceAdapter interface

```
package designpattern.adapter.weatherinfo;
public interface WeatherServiceAdapter {
   public float getTemperature();
   public float getHumidity();
}
```

14

Example: The Adapter design pattern

```
package designpattern.adapter.weatherinfo;
public class WeatherServiceImpl implements WeatherService
   {
   @Override
   public float temperature() {
      return 15.5f;
   }
   @Override
   public float humidity() {
      return 65.8f;
   }
}
```

15

Example: The Adapter design pattern

```
package designpattern.adapter.weatherinfo;
public class WeatherServiceAdapterImpl implements
    WeatherServiceAdapter {
   WeatherService weatherService;
   public WeatherServiceAdapterImpl(WeatherService
    weatherService) {
      this.weatherService = weatherService;
   }
   @Override
   public float getTemperature() {
      return weatherService.temperature();
   }
```

16

Example: The Adapter design pattern

```
   @Override
   public float getHumidity() {
      return weatherService.humidity();
   }

}
```

17

Example: The Adapter design pattern

```
package designpattern.adapter.weatherinfo;

public class NewsChannel {
   public static void main(String[] args) {

      WeatherServiceAdapter weatherService=new
     WeatherServiceAdapterImpl(new WeatherServiceImpl());

      System.out.println("Temperature = "+weatherService.getTemperature());

      System.out.println("Humidity = "+weatherService.getHumidity());
   }
}
```

18

Running the Example: The Adapter design pattern

Notifications | Output - DesignPattern (run)

```
run:
Temperature = 15.5
Humidity = 65.8
```

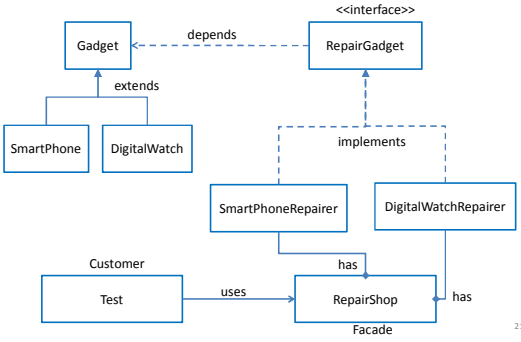
19

The Facade design pattern

- It is of type structural design pattern.
- Facade pattern hides the complexities of the system and provides an interface to the user using which the user can access the system.
- This design pattern involves a single class which provides simplified methods required by user and delegates calls to methods of existing system classes.

20

The Facade design pattern



21

Example-The Facade design pattern

```
package designpattern.facade;
public class Gadget {
   private String name;
   private String status;
   public String getStatus() {
      return status;
   }
   public void setStatus(String status) {
      this.status = status;
   }
```

22

Example-The Facade design pattern

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

23

Example-The Facade design pattern

- SmartPhone (a gadget)

```
package designpattern.facade;
public class SmartPhone extends Gadget{
}
```

- DigitalWatch (a gadget)

```
package designpattern.facade;
public class DigitalWatch extends Gadget{
}
```

24

Example-The Facade design pattern

- RepairGadget (specifies how repairing is done)

```
package designpattern.facade;
public interface RepairGadget {
   public void repair(Gadget gadget);
}
```

25

Example-The Facade design pattern

- SmartPhoneRepairer (knows how to repair a smart phone)

```
package designpattern.facade;
public class SmartPhoneRepairer implements RepairGadget{

   @Override
   public void repair(Gadget gadget) {
      SmartPhone smartPhone=(SmartPhone)gadget;
      System.out.println("Repairing your "+smartPhone.getClass().getName());
      smartPhone.setStatus("Working");
      System.out.println("Your "+smartPhone.getClass().getName()+" now
    working");
   }
}
```

26

Example-The Facade design pattern

- DigitalWatchRepairer (knows how to repair a digital watch)

```
package designpattern.facade;
public class DigitalWatchRepairer implements RepairGadget{

   @Override
   public void repair(Gadget gadget) {
      DigitalWatch digitalWatch=(DigitalWatch)gadget;
      System.out.println("Repairing your "+digitalWatch.getClass().getName());
      digitalWatch.setStatus("Working");
      System.out.println("Your "+digitalWatch.getClass().getName()+" now
    working");
   }
}
```

27

Example-The Facade design pattern

- RepairShop (where repairing of various gadgets is done)

```
package designpattern.facade;
//This is a facade class
public class RepairShop {
   private RepairGadget smartPhoneRepairer;
   private RepairGadget digitalWatchRepairer;
   public RepairShop(){
      smartPhoneRepairer=new SmartPhoneRepairer();
      digitalWatchRepairer=new DigitalWatchRepairer();
   }
```

28

Example-The Facade design pattern

```
   public void repairSmartPhone(SmartPhone smartPhone){
      smartPhoneRepairer.repair(smartPhone);
   }
   public void repairDigitalWatch(DigitalWatch digitalWatch){
      digitalWatchRepairer.repair(digitalWatch);
   }
}
```

29

Example-The Facade design pattern

- Test (Shows a scenario of using RepairShop-facade)

```
package designpattern.facade;
public class Test {
   public static void main(String[] args) {
      RepairShop repairShop=new RepairShop();
      SmartPhone sp1=new SmartPhone();
      sp1.setName("Samsung Note 3");
      sp1.setStatus("Working");
      DigitalWatch dw1=new DigitalWatch();
      dw1.setName("Sony SpeedX");
      dw1.setStatus("Working");
```

30

Example-The Facade design pattern

```
System.out.println("Using devices");
System.out.println("Devices' Status:");
System.out.println(sp1.getName()+" is "+sp1.getStatus());
System.out.println(dw1.getName()+" is "+dw1.getStatus());
System.out.println("Devices got problem");
sp1.setStatus("Not Working");
dw1.setStatus("Not Working");
System.out.println("Devices' Status:");
System.out.println(sp1.getName()+" is "+sp1.getStatus());
System.out.println(dw1.getName()+" is "+dw1.getStatus());
```

31

Example-The Facade design pattern

```
    System.out.println("Devices sent for repairing");
    repairShop.repairSmartPhone(sp1);
    repairShop.repairDigitalWatch(dw1);
    System.out.println("Devices' Status after repairing");
    System.out.println(sp1.getName()+" is "+sp1.getStatus());
    System.out.println(dw1.getName()+" is "+dw1.getStatus());
  }
}
```

32

Running the example-The Facade design pattern

Notifications | Output - DesignPattern (run) | Search Results

```
run:
Using devices
Devices' Status:
Samsung Note 3 is Working
Sony SpeedX is Working
Devices got problem
Devices' Status:
Samsung Note 3 is Not Working
Sony SpeedX is Not Working
Devices sent for repairing
Repairing your designpattern.facade.SmartPhone
Your designpattern.facade.SmartPhone now working
Repairing your designpattern.facade.DigitalWatch
Your designpattern.facade.DigitalWatch now working
Devices' Status after repairing
Samsung Note 3 is Working
Sony SpeedX is Working
```

33

References

- Java Design Patterns, problem solving approaches, tutorials point, www.tutorialspoint.com

34