Design Patterns
Front controller, Interceptor

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

Dharmsinh Desai University
Prof. H B Prajapati

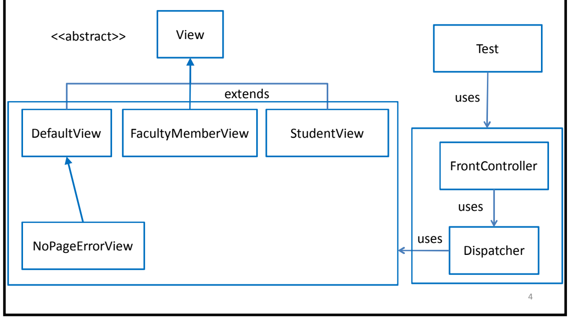
The Front Controller design pattern

- It is of type architectural design pattern.
- It provides a centralized request handling mechanism to handle all requests by a single handler.
- Front controller handler can do the following common activities:
  - Authentication
  - Authorization
  - Logging or tracking of request
- Following entities are involved in this design pattern:
  - Front controller: controls all request
  - Dispatcher: dispatches the request forwarded by front controller
  - View: response object for sent request

Example-The Front Controller design pattern

- We implement the front controller design pattern using the following entities:
  - Front controller, which performs the following
    - Logging or tracking of request
    - Authentication
    - Dispatching
  - Dispatcher:
    - Stores mapping between view-name and corresponding View object
    - Finds matching view or if error and executes that view
  - View:
    - Two Normal Views
      - StudentView
      - FacultyMemberView
    - Error View
      - NoPageErrorView

Example-The Front Controller design pattern



<<abstract>>
View
extends
DefaultView
FacultyMemberView
StudentView
NoPageErrorView
Test
uses
FrontController
uses
uses
Dispatcher

Example-The Front Controller design pattern

- View (Base class)

```
package designpattern.frontcontroller;
public abstract class View {
   abstract public void showView();
}
```

- DefaultView (Default error view)

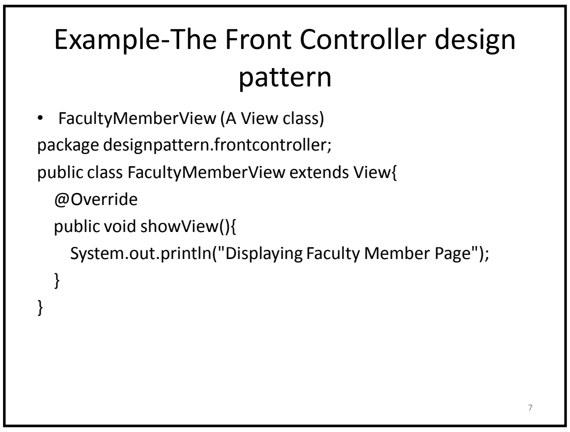
```
package designpattern.frontcontroller;
public class DefaultView extends View{
   @Override
   public void showView() {
      System.out.println("Error");
   }

}
```

Example-The Front Controller design pattern

- StudentView (A View class)

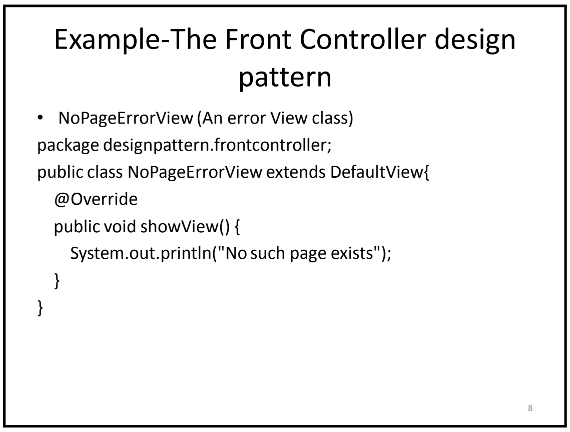
```
package designpattern.frontcontroller;
public class StudentView extends View{
   @Override
   public void showView(){
      System.out.println("Displaying Student Page");
   }
}
```



Example-The Front Controller design pattern

- FacultyMemberView (A View class)

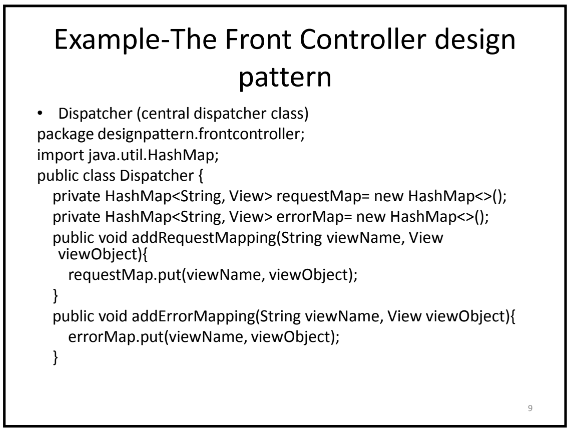
```
package designpattern.frontcontroller;
public class FacultyMemberView extends View{
   @Override
   public void showView(){
      System.out.println("Displaying Faculty Member Page");
   }
}
```



Example-The Front Controller design pattern

- NoPageErrorView (An error View class)

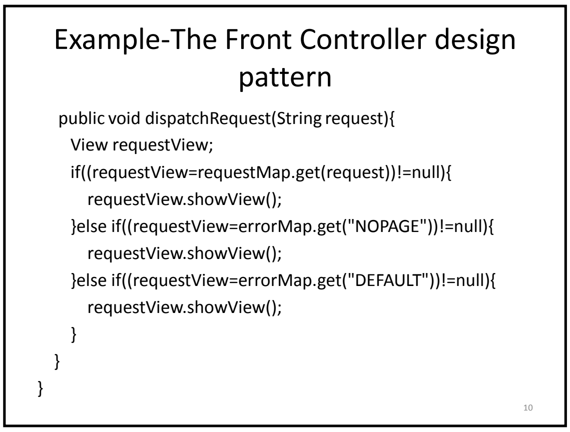
```
package designpattern.frontcontroller;
public class NoPageErrorView extends DefaultView{
   @Override
   public void showView() {
      System.out.println("No such page exists");
   }
}
```



Example-The Front Controller design pattern

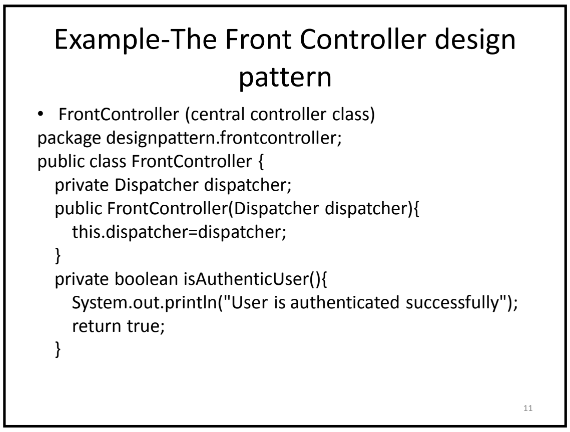
- Dispatcher (central dispatcher class)

```
package designpattern.frontcontroller;
import java.util.HashMap;
public class Dispatcher {
   private HashMap<String, View> requestMap= new HashMap<>();
   private HashMap<String, View> errorMap= new HashMap<>();
   public void addRequestMapping(String viewName, View
    viewObject){
      requestMap.put(viewName, viewObject);
   }
   public void addErrorMapping(String viewName, View viewObject){
      errorMap.put(viewName, viewObject);
   }
```



Example-The Front Controller design pattern

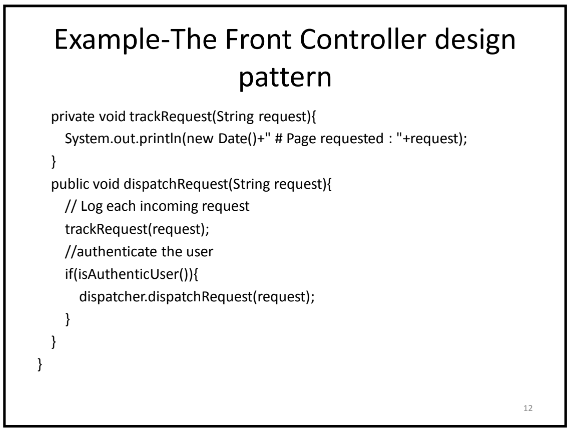
```
   public void dispatchRequest(String request){
     View requestView;
     if((requestView=requestMap.get(request))!=null){
        requestView.showView();
     }else if((requestView=errorMap.get("NOPAGE"))!=null){
        requestView.showView();
     }else if((requestView=errorMap.get("DEFAULT"))!=null){
        requestView.showView();
     }
   }
}
```



Example-The Front Controller design pattern

- FrontController (central controller class)

```
package designpattern.frontcontroller;
public class FrontController {
   private Dispatcher dispatcher;
   public FrontController(Dispatcher dispatcher){
      this.dispatcher=dispatcher;
   }
   private boolean isAuthenticUser(){
      System.out.println("User is authenticated successfully");
      return true;
   }
```



Example-The Front Controller design pattern

```
   private void trackRequest(String request){
      System.out.println(new Date()+" # Page requested : "+request);
   }
   public void dispatchRequest(String request){
      // Log each incoming request
      trackRequest(request);
      //authenticate the user
      if(isAuthenticUser()){
         dispatcher.dispatchRequest(request);
      }
   }
}
```

Example-The Front Controller design pattern

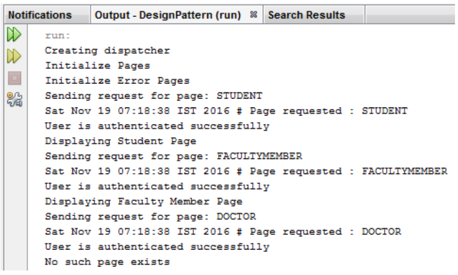
```
package designpattern.frontcontroller;
public class Test {
   public static void main(String[] args) {
      System.out.println("Creating dispatcher");
      Dispatcher dispatcher=new Dispatcher();
      System.out.println("Initialize Pages");
      dispatcher.addRequestMapping("STUDENT", new StudentView());
      dispatcher.addRequestMapping("FACULTYMEMBER", new
    FacultyMemberView());

      System.out.println("Initialize Error Pages");
      dispatcher.addErrorMapping("DEFAULT", new DefaultView());
      dispatcher.addErrorMapping("NOPAGE", new NoPageErrorView());
```

Example-The Front Controller design pattern

```
      FrontController frontController=new FrontController(dispatcher);
      System.out.println("Sending request for page: STUDENT");
      frontController.dispatchRequest("STUDENT");
      System.out.println("Sending request for page: FACULTYMEMBER");
      frontController.dispatchRequest("FACULTYMEMBER");
      System.out.println("Sending request for page: DOCTOR");
      frontController.dispatchRequest("DOCTOR");
   }
}
```

Running the Example-The Front Controller design pattern

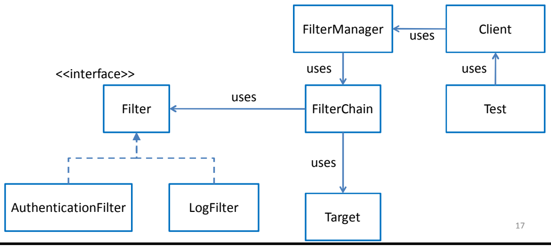


```
run:
Creating dispatcher
Initialize Pages
Initialize Error Pages
Sending request for page: STUDENT
Sat Nov 19 07:18:38 IST 2016 # Page requested : STUDENT
User is authenticated successfully
Displaying Student Page
Sending request for page: FACULTYMEMBER
Sat Nov 19 07:18:38 IST 2016 # Page requested : FACULTYMEMBER
User is authenticated successfully
Displaying Faculty Member Page
Sending request for page: DOCTOR
Sat Nov 19 07:18:38 IST 2016 # Page requested : DOCTOR
User is authenticated successfully
No such page exists
```

The Interceptor design pattern

- It is of type architectural design pattern.
- It is also referred as intercepting filter design pattern.
- It is used when we want to allow some pre-processing with request and some post-processing with the response.
- This design pattern involves the following entities:
  - Filter: Performs certain task prior or after execution of request
  - Filter Chain: Contains multiple filters and executes them in defined order
  - Target: It is the request handler
  - Filter Manager: It managers the filters and filter chain
  - Client: It sends request to the target object

The Interceptor design pattern



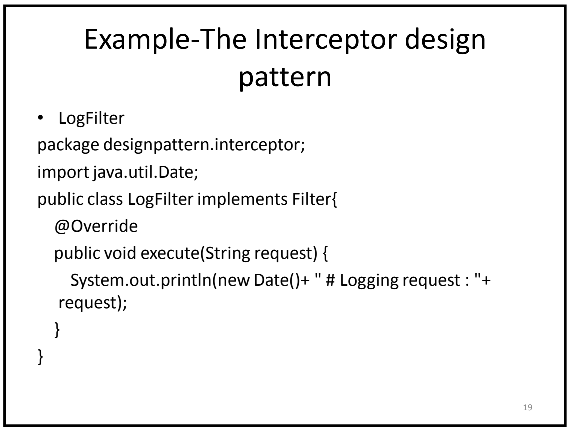
Example-The Interceptor design pattern

- Filter

```
package designpattern.interceptor;
public interface Filter {
   public void execute(String request);
}
```

- AuthenticationFilter

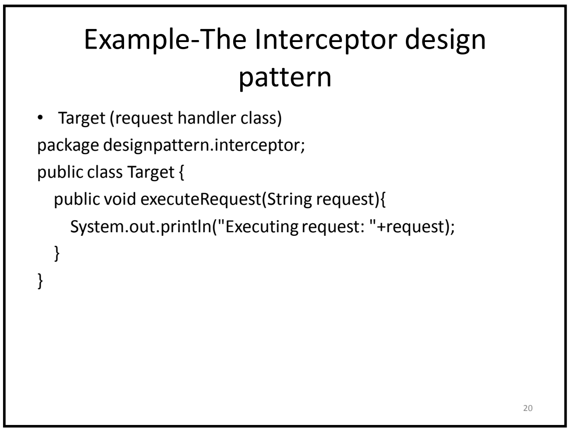
```
package designpattern.interceptor;
public class AuthenticationFilter implements Filter{
   @Override
   public void execute(String request) {
      System.out.println("Authenticating request : "+request);
   }
}
```



Example-The Interceptor design pattern

- LogFilter

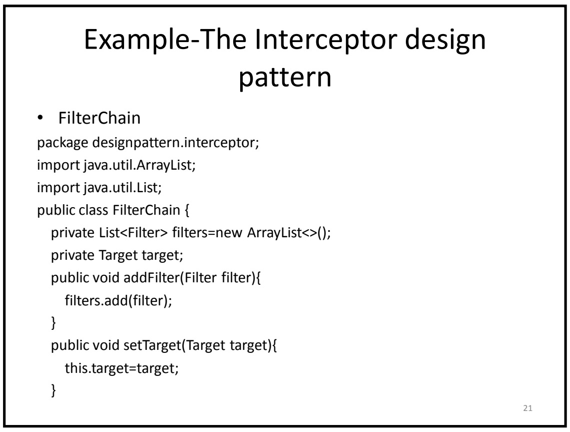
```
package designpattern.interceptor;
import java.util.Date;
public class LogFilter implements Filter{
   @Override
   public void execute(String request) {
      System.out.println(new Date()+ " # Logging request : "+
    request);
   }
}
```



Example-The Interceptor design pattern

- Target (request handler class)

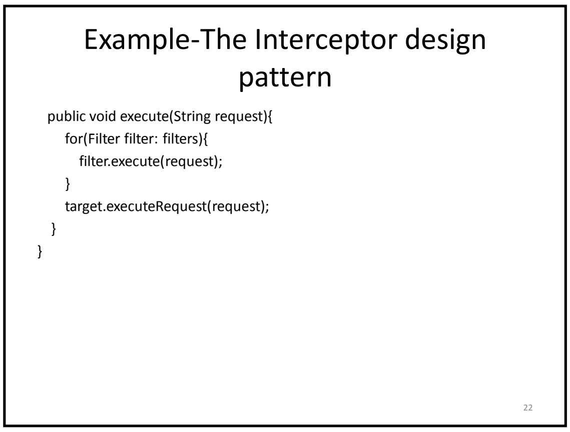
```
package designpattern.interceptor;
public class Target {
   public void executeRequest(String request){
      System.out.println("Executing request: "+request);
   }
}
```



Example-The Interceptor design pattern

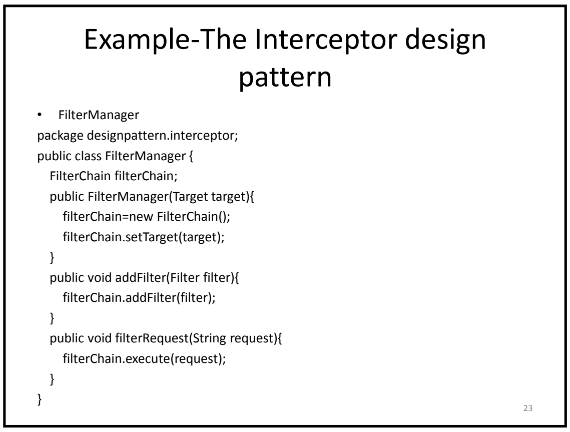
- FilterChain

```
package designpattern.interceptor;
import java.util.ArrayList;
import java.util.List;
public class FilterChain {
   private List<Filter> filters=new ArrayList<>();
   private Target target;
   public void addFilter(Filter filter){
      filters.add(filter);
   }
   public void setTarget(Target target){
      this.target=target;
   }
```



Example-The Interceptor design pattern

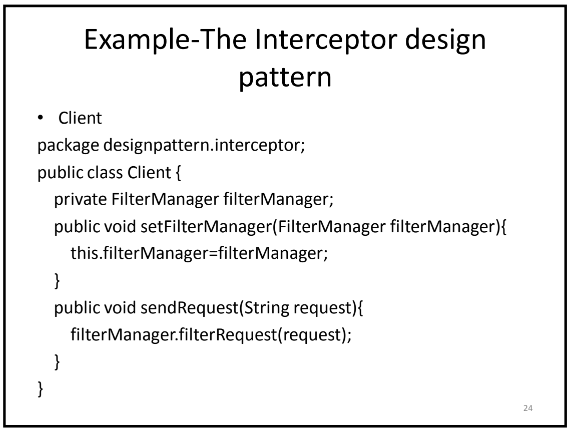
```
  public void execute(String request){
     for(Filter filter: filters){
        filter.execute(request);
     }
     target.executeRequest(request);
   }
}
```



Example-The Interceptor design pattern

- FilterManager

```
package designpattern.interceptor;
public class FilterManager {
   FilterChain filterChain;
   public FilterManager(Target target){
      filterChain=new FilterChain();
      filterChain.setTarget(target);
   }
   public void addFilter(Filter filter){
      filterChain.addFilter(filter);
   }
   public void filterRequest(String request){
      filterChain.execute(request);
   }
}
```



Example-The Interceptor design pattern

- Client

```
package designpattern.interceptor;
public class Client {
   private FilterManager filterManager;
   public void setFilterManager(FilterManager filterManager){
      this.filterManager=filterManager;
   }
   public void sendRequest(String request){
      filterManager.filterRequest(request);
   }
}
```

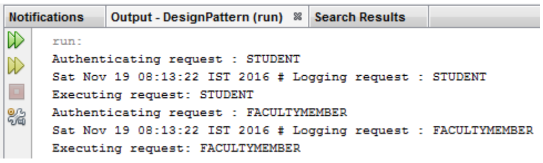
Example-The Interceptor design pattern

- Test

```
package designpattern.interceptor;
public class Test {
   public static void main(String[] args) {
      FilterManager filterManager=new FilterManager(new Target());
      filterManager.addFilter(new AuthenticationFilter());
      filterManager.addFilter(new LogFilter());

      Client client=new Client();
      client.setFilterManager(filterManager);
      client.sendRequest("STUDENT");
      client.sendRequest("FACULTYMEMBER");
   }
}
```

Running the Example-The Interceptor design pattern



```
run:
Authenticating request : STUDENT
Sat Nov 19 08:13:22 IST 2016 # Logging request : STUDENT
Executing request: STUDENT
Authenticating request : FACULTYMEMBER
Sat Nov 19 08:13:22 IST 2016 # Logging request : FACULTYMEMBER
Executing request: FACULTYMEMBER
```

References

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