

Design Patterns Assignment

Exercise-1:

Creational (abstract factory, builder, singleton, static factory method)

1. (a) java.lang.Runtime (b) java.lang.Desktop

Follows Singleton design pattern.

2. (a) com.google.common.collect.MapMaker

Follows Builder design pattern

3. (a) java.util.Calendar (b) java.text.NumberFormat
(c) java.nio.charset.Charset

Following the Static Factory design pattern.

4. (a) javax.xml.parsers.DocumentBuilderFactory (b) javax.xml.transform.TransformerFactory
(c) javax.xml.xpath.XPathFactory

Follows Abstract Factory design pattern.

Structural (adapter, decorator, flyweight)

1. (a) java.lang.Integer

- (b) java.lang.Boolean

Follows Flyweight design pattern

2. (a) java.io.InputStreamReader
(b) java.io.OutputStreamWriter
(c) java.util.Arrays

Follows Adapter design pattern

3. (a) java.io.BufferedInputStream
(b) java.io.DataInputStream
(c) java.io.BufferedOutputStream
(d) java.util.zip.ZipOutputStream
(e) java.util.Collections#checkedList()

Follows Decorator design pattern

Behavioural (chain of responsibility, command, iterator, observer, strategy, template method)

1. (a) javax.servlet.FilterChain

Follows Chain of responsibility design pattern

2. (a) java.lang.Runnable

- (b) java.util.concurrent.Callable

Follows Command design pattern

3. (a) java.util.Iterator

Follows Iterator design pattern

4. (a) java.util.Comparator

(b) javax.servlet.Filter

Follows Strategy design pattern

5. (a) java.util.ArrayList, java.util.AbstractSet, java.util.AbstractMap

(b) java.io.InputStream, java.io.OutputStream, java.io.Reader, java.io.Writer

Follows Template design pattern

6. (a) java.util.EventListener

(b) java.util.Observer/java.util.Observable

Follows Observer design pattern

Exercise 2

1. it is hard to create a proper unit test, because there is tight coupling in the given implementation.

2.

public interface ServerConfigInterface

```
{  
    public String getAccessLevel(User user);  
}
```

public interface AccessCheckerInterface

```
{  
    public boolean mayAccess(User user, String path);  
}
```

```
public interface Response { String getStatus();  
    Map<String, String> getHeaders(); String getBody();  
}
```

```
public class FileResponse implements Response { public FileResponse(String path) {  
    this.path = Paths.get(path);  
}
```

```
    @Override
```

```
    public String getStatus() { return "200";  
}
```

```
    @Override
```

```
    public Map<String, String> getHeaders() {  
        HashMap<String, String> headers = new HashMap<String, String>(); headers.put("content-  
        type", Files.probeContentType(path));  
        return headers;  
}
```

```
    @Override
```

```
    public String getBody() {  
        byte[] bytes = Files.readAllBytes(path); String body = new String(bytes);
```

```

}
private Path path;
}
public class NotFoundResponse extends FileResponse { public NotFoundResponse() {
super(app.Assets.getInstance().getNotFoundPage());
}
@Override
public String getStatus() { return "404";
}
}
public class MarkdownResponse implements Response { public MarkdownResponse(String
body) {
this.body = body;
}
@Override
public String getStatus() { return "200"
}
@Override
public Map<String, String> getHeaders() {
HashMap<String, String> headers = new HashMap<String, String>(); headers.put("content-
type", "text/html");
return headers;
}
@Override
public String getBody() {
return Markdown.parse(body).toHtml();
}
private String body;
}
public class Test {
public static void main(String[] args) { Module module = new AbstractModule() { @Override
protected void configure() {
bind(AccessCheckerInterface.class).to(AccessCheckerMock.class);
}
};
SessionManager maneger =
Guice.createInjector(module).getInstance(SessionManager.class); User user = new User();
maneger.createSession(user, "path");
}
}

```

Exercise 3

1.)

Applying static factory method public class Responses {
public static Response notFoundResponse() {

```

return new NotFoundResponse();
}
public static Response markdownResponse() { return new MarkdownResponse();
}
public static Response fileResponse() { return new FileResponse();
}
}

```

2.)

```

public class Response { private String status;
private Map<String, String> headers; private String body;
}
public class Responses {
public static Response response(String status, Map<String, String> headers, String body) {
return new Response(status, headers, body);
}
public static Response file(String status, String path) { Path filePath = Paths.get(path);
HashMap<String, String> headers = new HashMap<String, String>(); headers.put("content-
type", Files.probeContentType(filePath));
byte[] bytes = Files.readAllBytes(filePath); String body = new String(bytes);
return response(status, headers, body);
}
public static Response notFound() {
return file("404", app.Assets.getInstance().getNotFoundPage());
}
public static markdown(String body) {
HashMap<String, String> headers = new HashMap<String, String>(); headers.put("content-
type", "text/html");
return response("200", headers, Markdown.parse(body).toHtml());
}
}

```