Design Patterns Assignment

D. Dinesh Koushik

Exercise - 1

Examine the listed Java APIs and identify some of the design patterns present. For each pattern found describe why the API follows it. Each bullet-point group corresponds to one pattern to be identified. Note that most of the patterns have not been covered on lectures. You may need to look them up on the web.

Creational

1. Singleton
2. Static factory method
3. Abstract factory

Structural

1. Flyweight
2. Adapter
3. Decorator

Behavioural

1. Chain of responsibility
2. Command
3. Iterator
4. Strategy
5. Template Method
6. Observer

Exercise 2

Consider a simple server implementation which uses the Singleton Pattern. The following code deals with the creation of new user sessions:

```
public class AccessChecker {
private static AccessChecker instance;
public static AccessChecker getInstance() {
if (instance == null) {
// create instance
}
return instance;
}
private ServerConfig config = ServerConfig.getInstance();
public AccessChecker() {
// initialization..
}
public boolean mayAccess(User user, String path) {
String userLevel = config.getAccessLevel(user);
// check if level suffices
}
// ...
}
public class ServerConfig {
private static ServerConfig instance;
private static String configFilePath = "...";
public static ServerConfig getInstance() {
if (instance == null) {
// create instance
}
return instance;
}
public ServerConfig() {
// load configuration from file
// validate
}
public String getAccessLevel(User u) { ... }
// ...
}
public class SessionManager {
private AccessChecker access = AccessChecker.getInstance();
public Session createSession(User user, String accessedPath) {
if (access.mayAccess(user, accessedPath)) {
return new Session(user);
} else {
throw new InsufficientRightsException(user, accessedPath);
}
}
// ...
}
```

1) Due to tightly coupling it is hard to create a proper unit test. In SessionManager class there is an object of AccessChecker class and in AccessChecker class there is an object of ServerConfig class which makes the tight coupling. in order to resolve this issue and make loose coupling we have to declare interfaces and do dependency injection.

2)

```
public interface ServerConfigInterface {
        public String getAccessLevel(User user);
 }
public interface AccessCheckerInterface {
        public boolean mayAccess(User user, String path);
}
```

3)

```
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
A web application can return one of many kinds of HTTP responses to the User-agent.

1.In the given implementation there is no object instance in any class, so they are loosely coupled.

2.

```
public class Responses {
	public static Response notFoundResponse() {
		return new NotFoundResponse();
	}

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

	public static Response fileResponse() {
		return new FileResponse();
	}
}
```

3.

```
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());
        }
}
```