CREATIONAL

i. Singleton
ii. Builder
iii. Static factory
iv. Abstract factory

Structural

i. Flyweight
ii. Adaptor
iii. Decorator

Behavioral

i. Chain of responsibility
ii. Command
iii. Iterator
iv. Strategy
v. Template method
vi. Observer

Question 2-The ServerConfig class has methods like setting and loading the configuration file, etc, so it might not be practical to create instances of them for the tests. AS the ServerConfig follows the Singleton design, we can override its getInstance method. So, instead of going with the classes, we should go with declaring the interfaces.We can use abstract factory pattern or dependency injection to directly instantiate referencing classes.

2- ServerConfig -interface:

```
public interface

   ServerConfigInterface { public

   String getAccessLevel(User u);

}
```

AccessChecker-Interface:

```
public interface AccessCheckerInterface {
```

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


    }


3. public class MainTest {
      public static void main(String[] args) {

          Module module = new AbstractModule() {
             @Override
             protected void configure() {
                bind(AccessCheckerInterface.class).to(AccessCheckerMock.class);
              }
          };
           SessionManager mgr =
          Guice.createInjector(module).getInstance(SessionManager.class); User user =
          new User();
          mgr.createSession(user, "any path");
       }
}
```

Question 3

Part 1 and 2 :

static factory class creation

```
public class Responses {

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

We can use single implementation:

```
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());

    }
    }
```