Assignment 5 Solution

Software Architecture and Engineering

Spring 2014

Exercise 1

```
public class A {

    int aField;

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass()) // why not just check for A.class?
            return false;
        A other = (A) obj;
        if (aField != other.aField)
            return false;
        return true;
    }

    @Override
    public int hashCode() { ... }
}

public class B extends A {

    int bField;

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (!super.equals(obj))
```

```
            return false;
        if (getClass() != obj.getClass()) // why not just check for B.class?
            return false;
        B other = (B) obj;
        if (bField != other.bField)
            return false;
        return true;
    }

    @Override
    public int hashCode() { ... }
}
```

Exercise 2

- Creational
    1. Singleton
        - `java.lang.Runtime` – `#getRuntime()`
        - `java.lang.Desktop` – `#getDesktop()`
    2. Builder
        - `com.google.common.collect.MapMaker`
    3. Static factory method
        - `java.util.Calendar` – `#getInstance()`
        - `java.text.NumberFormat` – `#getInstance()`
        - `java.nio.charset.Charset` – `#forName()`
    4. Abstract factory
        - `javax.xml.parsers.DocumentBuilderFactory` – `#newInstance()`
        - `javax.xml.transform.TransformerFactory` – `#newInstance()`
        - `javax.xml.xpath.XPathFactory` – `#newInstance()`
- Structural
    1. Flyweight
        - `java.lang.Integer` – `#valueOf()`
        - `java.lang.Boolean` – `#valueOf()`
    2. Adapter
        - `java.io.InputStreamReader`
        - `java.io.OutputStreamWriter`
        - `java.util.Arrays` – `#asList()`
    3. Decorator

- `java.io.BufferedInputStream`
- `java.io.DataInputStream`
- `java.io.BufferedOutputStream`
- `java.util.zip.ZipOutputStream`
- `java.util.Collections#checkedList()`

- Behavioural

  1. Chain of responsibility
     - `javax.servlet.FilterChain` – `#doFilter()`
  2. Command
     - `java.lang.Runnable` – `#run()`
     - `java.util.concurrent.Callable` – `#call()`
  3. Iterator
     - `java.util.Iterator` – `#next()`, `#hasNext()`
  4. Strategy
     - `java.util.Comparator` – `#compare()`
     - `javax.servlet.Filter` – `#doFilter()`
  5. Template method
     - `java.util.AbstractList`, `java.util.AbstractSet`, `java.util.AbstractMap`
     - `java.io.InputStream`, `java.io.OutputStream`, `java.io.Reader`, `java.io.Writer`
  6. Observer
     - `java.util.EventListener`
     - `java.util.Observer`/`java.util.Observable`

Mostly taken from BalusC' answer at http://stackoverflow.com/questions/1673841/examples-of-gof-design-patterns

Exercise 3

Returning a specific response depends on knowledge of the actual class implementing that kind of response. If the implementation class needs to be changed later

1. either the clients of the hierarchy need to be changed to use the new implementation
2. or the old class should be kept but become a proxy object for the new implementation

The first point is clearly undesirable, while the second one keeps an api that is no longer relevant, thus complicating the code base.

```
public class Response {

    // constructor and accessors omitted for clarity

    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) {
        return response("200", Markdown.parse(body).toString());
    }
}
```