

**COMSATS UNIVERSITY ISLAMABAD, ABBOTTABAD**

Design Pattern

Assignment # 02

***Submitted by:***

Laiba binte tahir FA21-BSE-019

***Submitted to:***

Mam Faiza Hameed

Contents

[Q1: Design patterns in java APIs 3](#_Toc185277578)

[1. Creational design Patterns 3](#_Toc185277579)

[2. Structural Design Patterns 3](#_Toc185277580)

[3. Behavioral Design Pattern 4](#_Toc185277581)

[Q2: Design Patterns (Intent and Explanation) 5](#_Toc185277582)

[1. Front Controller Pattern 6](#_Toc185277583)

[2. Application Controller Pattern 7](#_Toc185277584)

[3. Dependency Injection Pattern 8](#_Toc185277585)

[4. Data Mapper Pattern 9](#_Toc185277586)

[5. Domain Object Factory Pattern 10](#_Toc185277587)

[6. Adaptive Design Pattern 11](#_Toc185277589)

[7. Null Object Pattern 12](#_Toc185277590)

[8. Service Locator Pattern 13](#_Toc185277591)

# Q1: Design patterns in java APIs

## 1. Creational design Patterns

1. ***java.lang.Runtime and java.lang.Desktop***
   * **Singleton Pattern:** Above java APIs follows the singleton design pattern because they provide a single instance of object and that’s globally accessible. GetDesktop () and GetRuntime() ensure that only one instance of the runtime environment is accessible to all application throughout.
2. ***com.google.common.collect.MapMaker***
   * **Builder Pattern:** The aboveclass from Google's Guava library implements the Builder pattern to create customized ConcurrentMap instances. It allows incremental configuration of features.
3. ***java.util.Calendar, java.text.NumberFormat, java.nio.charset.Charset***
   * **Factory Pattern:** These provide static factory methods to create instances. Calender.getInstance() returns an appropriate calender instance. Without exposing the exact implementation. NumberFormat.getInstance() returns a locale-specific instance for formatting numbers. Charset.forName() returns a Charset object for the specified charset name, hiding the instantiation details.
4. ***javax.xml.parsers.DocumentBuilderFactory, javax.xml.transform.TransformerFactory, javax.xml.xpath.XPathFactory***
   * **Abstract Factory pattern:** Each creates specific objects (DocumentBuilder, Transformer, XPath) that belong to XML processing domain also the implementations are abstracted from the client. Also, can be configured to use specific implementations via configuration files.

## 2. Structural Design Patterns

1. ***java.lang.Integer and java.lang.Boolean***
   * **Flyweight pattern:** Wrapper classes like integer and Boolean reuse objects Integer.ValueOf() caching small integers b/w -128 and 127. This reduces memory usage by sharing the instances.
2. ***java.io.InputStreamReader, java.io.OutputStreamWriter, java.util.Arrays***
   * **Adapter Pattern:** InputStreamRender and OutputStreamWriter adapt streams to character streams. And Array.asList() adapts array into List making it usable with collection framework.
3. ***java.io.BufferedInputStream, java.io.DataInputStream, java.io.BufferedOutputStream, java.util.zip.ZipOutputStream, java.util.Collections#checkedList()***
   * **Decorator Pattern:** the above classes wrap existing streams to add new functionality without changing their structure. Buffered, data, and Zip stream or type-checking enhance base functionality without altering the underlying object.
   * Decorators like Buffered, Data, and Zip streams or type-checking for Collections enhance base functionality without altering the underlying object.

## 3. Behavioral Design Pattern

1. ***javax.servlet.FilterChain***
   * **Chain of Responsibility Pattern:** Filterchain allows multiple filters to process requests sequentially. Each filter passes the request to the next filter in chain until final resource is reached i-e Servlet
2. ***java.lang.Runnable and java.util.concurrent.Callable***
   * **Command Pattern:** Runnable and callable do encapsulation of a task or command in object to be executed later, which promotes decoupling as tasks can be executed by any thread.
3. ***java.util.Iterator***
   * **Iterator Pattern:** The Iterator provides a way to traverse collections without exposing their underlying structure. It encapsulates traversal logic and simplifies access.
4. ***java.util.Comparator and javax.servlet.Filter***
   * **Strategy Pattern:** Comparator allows you to define different comparison strategies for sorting objects. Filter provides a strategy for processing requests or responses in servlets.
5. ***java.util.AbstractList, java.util.AbstractSet, java.util.AbstractMap***
   * **Template Method Pattern:** These abstract classes define a skeleton (template) for specific collection implementations. Subclasses must implement specific methods, but the overall flow remains defined in the abstract class.
6. ***java.io.InputStream, java.io.OutputStream, java.io.Reader, java.io.Writer***
   * **Template Method Pattern:** These I/O classes define an abstract flow for input and output operations, where concrete subclasses implement specific behavior.
7. ***java.util.EventListener and java.util.Observer/java.util.Observable***
   * **Observer Pattern:** EventListener is used to observe and respond to events (like UI events). Observer/Observable follow the observer pattern, where the Observable object notifies all registered observers when its state changes.

# Q2: Design Patterns (Intent and Explanation)

**All diagrams have been made using tool:** [**https://www.mermaidchart.com/**](https://www.mermaidchart.com/)

## 1. Front Controller Pattern

A close-up of a document

Description automatically generatedA diagram of a computer

Description automatically generated

## 2. Service Locator Pattern

A black and white text on a white background

Description automatically generated

A diagram of a service

Description automatically generated

## 3. Dependency Injection Pattern

A black and white text on a white background

Description automatically generated

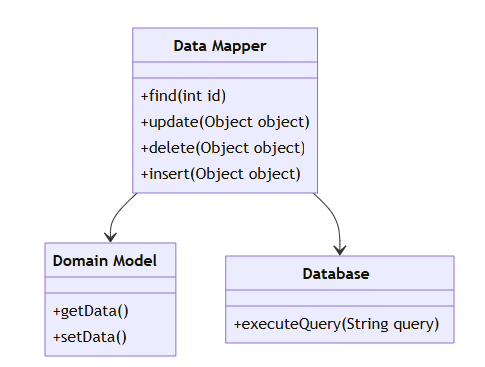
A diagram of a service

Description automatically generated

## 4. Data Mapper Pattern

A white text with black text

Description automatically generated



## 5. Domain Object Factory Pattern

## A close-up of a design pattern Description automatically generated

A diagram of a domain

Description automatically generated

## 6. Adaptive Design Pattern

A diagram with text on it

Description automatically generated

A diagram of a computer

Description automatically generated

## 7. Null Object Pattern

A black and white text on a white background

Description automatically generated

A diagram of a diagram

Description automatically generated with medium confidence

## 8. Application Controller Pattern

A white text on a black background

Description automatically generated

A diagram of a computer

Description automatically generated