**Design Pattterns**

**Design Patterns**

1. Creational Design Patterns- Create an object

* Singlteton
* Factory
* Abstract Factory
* Builder
* Prototype

1. Structural – Composing 2 objects

* Adapter
* Composite
* Proxy
* Fly Weight
* Façade
* Bridge
* Decorator

1. Behavioural – Communication bw 2 objects

* Template Method
* Mediator
* Chain of Responsibility
* Observer
* Strategy
* Command

**Factory Design Pattern –** Factory design pattern is a creational design pattern that defines an interface for creating an object but let subclass or methods decide which class to instantiate.

**A diagram of a car assembly

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Singleton Design Pattern**

* Allows only 1 instance of itself to be created and provides a global access to that instance.
* For eg, *propertyReader* class is used repeatedly by different classes to read properties of different files. It can be made singleton so that only one instance of that class is created.

**Lazy Initialization**

* Instance of the class is not created until it is needed.
* Not thread-safe by default
* Can make the getInstance method synchronized to ensure only one thread access it a time.
* But then every call to getInstance() is synchronized even after the object is created.
* Double-checked locking reduces the overhead.

**A computer screen shot of text

AI-generated content may be incorrect.**

**Double-checked Locking**

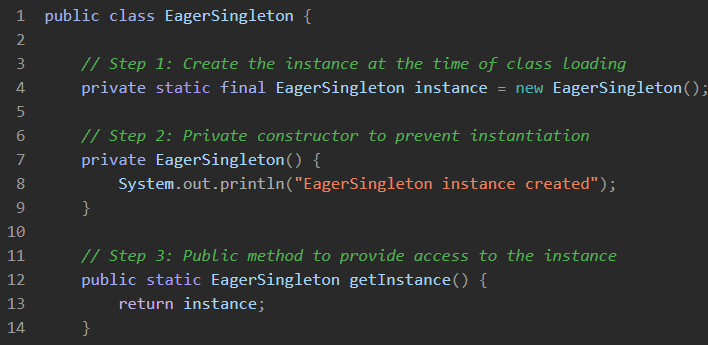
**synchronized (Singleton.class){…} – It means that thread is acquiring a lock on Singleton class object, not on instance of the class.**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**Eager Initialization**

* The instance is created as soon as the class is loaded, whether it is used or not.
* It is thread safe.

****

**Proxy Design Pattern –** It is a structural design pattern that provides a placeholder for another object to control access to it.

**Step by step code:**

1. **Subject Interface –** Implemented by real object as well as the proxy object

A screen shot of a computer

AI-generated content may be incorrect.

1. **Real object –** RealImage is the actual object that does the heavy work(loading an image from disk). Problem is that if you have 100 RealImage objects all of them will load immediately, even if you view 2 of them.

**A screen shot of a computer program

AI-generated content may be incorrect.**

1. **Proxy Object**

* ProxyImage does not load the image immediately.
* It stores the filename and waits until display() is called.
* Only when display() is called for the first time, it creates the RealImage and loads the image.
* On subsequent calls, it just displays the already-loaded image.

**A screen shot of a computer program

AI-generated content may be incorrect.**

1. **Client Code**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Command Design Pattern –** Command design pattern is a behavioural design pattern that turns a request into a standalone object. Provides a loose coupling between sender and the receiver.

Real world analogy – Each button on a remote control is a command (like turning on the TV, increasing volume, etc.). The remote does not know how the TV works – it just sends the command.

1. Command Interface – Common interface for all commands

A black screen with white text

AI-generated content may be incorrect.

1. Receiver – Actual object that performs the action

A screen shot of a computer program

AI-generated content may be incorrect.

1. Concrete Command Classes – They store a reference to the Light object and call the appropriate method when execute() is called.

A screen shot of a computer code

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

1. Invoker – It does not know what the command does – it just calls execute()

A computer screen with text

AI-generated content may be incorrect.

1. Client code

A screen shot of a computer program

AI-generated content may be incorrect.

**Observer Design Pattern –** The observer design pattern is a behavioural design pattern in which an object called as the subject, maintains a list of its dependents, known as observers. When the subject’s state changes, it automatically notifies all its observers.

Real world analogy – In YouTube, the channel is the subject, subscribers are observers. When a new video us uploaded all the subscribers get notified.

A screenshot of a black screen

AI-generated content may be incorrect.

Concrete Subject



A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

Decorator Design Pattern -The decorator design pattern allows us to add new functionality to an object dynamically without affecting behaviour of other objects from the same class. This is achieved by creating decorator classes that wrap up the original object and add new behaviours or attributes.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen shot of code

AI-generated content may be incorrect.