A **Design Pattern** in Java is a well-proven solution to a common problem that software developers face during the design phase of a project. They represent best practices and are typically used to solve common object-oriented design problems. These patterns are not specific to Java but are commonly applied in Java programming due to its object-oriented nature.

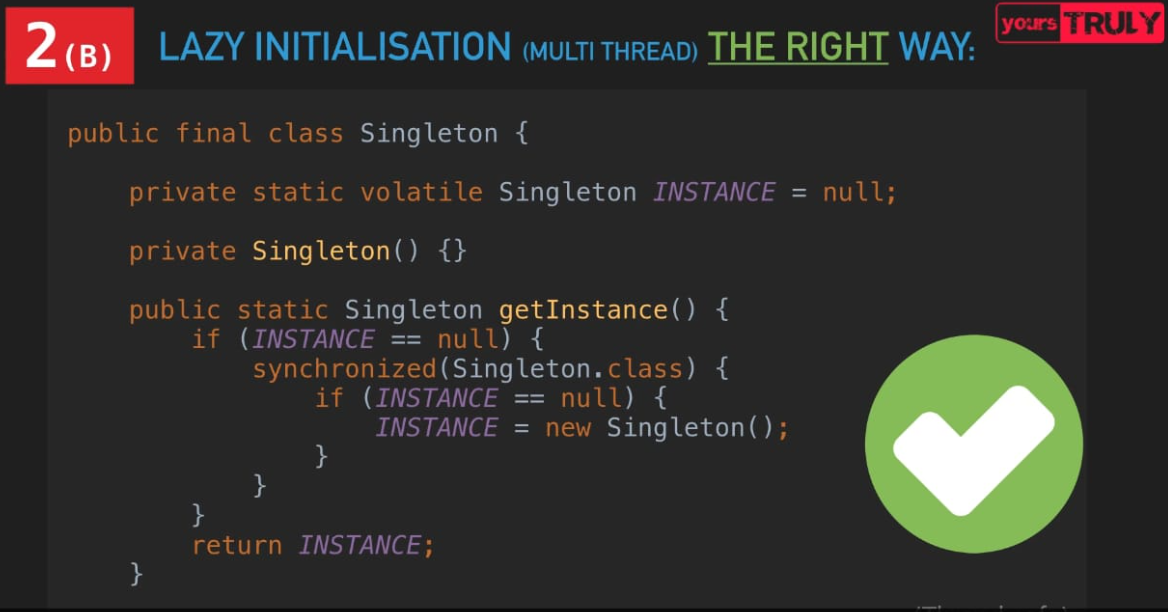
Design patterns can be classified into three main categories:

1. **Creational Patterns** – Deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.
2. **Structural Patterns** – Deal with object composition or how objects interact to form larger structures.
3. **Behavioral Patterns** – Deal with communication between objects and how they interact with each other.

Here are detailed examples of each type, starting with simple patterns that are easy for beginners to understand.

**1. Creational Design Pattern: Singleton Pattern**

The **Singleton Pattern** ensures that a class has only one instance and provides a global point of access to that instance. It is useful when exactly one object is needed to coordinate actions across the system.



**Example:**

A screen shot of a computer program

Description automatically generated

**A computer screen shot of a program

Description automatically generated**

**Explanation:**

* The constructor is private to prevent creating new instances.
* The static method getInstance() controls the instance creation.
* Once an instance is created, any subsequent calls to getInstance() will return the same object.

**2. Structural Design Pattern: Adapter Pattern**

The **Adapter Pattern** is used to allow two incompatible interfaces to work together. It acts as a bridge between two incompatible classes.

**Example:**

A screenshot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

**Explanation:**

* The **Adapter** (MediaAdapter) allows the AudioPlayer to use AdvancedMediaPlayer to play VLC or MP4 files, while the AudioPlayer can play MP3 files directly.
* This pattern allows classes with incompatible interfaces to collaborate by creating a bridge between them.

**3. Behavioral Design Pattern: Observer Pattern**

The **Observer Pattern** defines a one-to-many relationship between objects so that when one object changes state, all its dependents are notified and updated automatically.

**Example:**

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Explanation:**

* **Subject** holds a list of observers and notifies them when its state changes.
* Observers (BinaryObserver and HexObserver) update themselves based on the subject’s state.
* When the subject’s state changes (via setState()), it automatically notifies all observers, and each observer takes appropriate action.

**Key Points for Beginners:**

* **Singleton** ensures a single instance of a class.
* **Adapter** allows two incompatible interfaces to work together.
* **Observer** establishes a one-to-many dependency between objects, enabling one object to notify multiple observers of changes in state.

These design patterns provide reusable solutions to common programming challenges, which will help you write better, more flexible, and maintainable code.