# **Lazy Loading Design Pattern Example**

**Multimedia Player Application**

**Introduction**

The multimedia player application showcases various design patterns to efficiently manage media sources, implement lazy loading for media playback, and demonstrate observer and strategy patterns for dynamic behavior.

**Design Patterns Used**

**Singleton Pattern**

* Implemented in MediaPlayer class
* Ensures only one instance of MediaPlayer exists throughout the application.
* Accessed through the static method getInstance().

**Factory Pattern**

* Implemented in MediaFactory class
* Creates different MediaSource instances based on the media type (audio, video).
* Returns the appropriate media source using createMediaSource().

**Proxy Pattern (Lazy Loading)**

* Implemented in MediaProxy class
* Acts as a placeholder for the actual media source.
* Loads the real media (RealMedia) lazily when play() is called.

**Observer Pattern**

* Implemented with MediaDisplay interface and concrete classes (AudioDisplay, VideoDisplay)
* Observers registered with MediaPlayback to update media displays.
* MediaPlayback notifies registered observers about media playback changes.

**Strategy Pattern**

* Implemented with PlaybackStrategy interface and concrete classes (ShuffleStrategy, RepeatStrategy, DefaultStrategy)
* Different strategies for media playback implemented by classes adhering to PlaybackStrategy.
* Strategies can be dynamically selected and applied.

**Code Overview**

**MediaPlayer**

* Singleton pattern implementation ensuring a single instance of the media player.

**MediaFactory**

* Factory pattern creating different MediaSource instances based on media type.

**MediaSource (Interface)**

* Interface defining the behavior for media sources (play() method).

**MediaProxy**

* Proxy pattern acting as a placeholder and performing lazy loading for actual media playback.

**MediaDisplay**

* Interface defining the behavior for media displays as observers.

**AudioDisplay, VideoDisplay**

* Concrete classes implementing MediaDisplay for audio and video display updates.

**MediaPlayback**

* Manages observers (media displays) and notifies them about media playback changes.

**RealMedia**

* Represents the actual media source performing media playback operations.

**PlaybackStrategy**

* Interface defining different strategies for media playback (apply() method).

**ShuffleStrategy, RepeatStrategy, DefaultStrategy**

* Concrete classes implementing PlaybackStrategy for various playback strategies.

**How Proxy Pattern Relates to Lazy Loading**

1. **Deferred Object Creation:** In the Music Player example, the MediaProxy class serves as a placeholder for the actual MediaSource object (RealMedia). When the play() method is called on the MediaProxy, it internally creates the actual RealMedia object only when necessary (i.e., when play() is invoked for the first time).
2. **Lazy Initialization:** The RealMedia object, representing the actual media source, is loaded lazily inside the MediaProxy when play() is called. If the media has not been loaded yet, the MediaProxy will create the RealMedia object and invoke its play() method.

**Relation to Lazy Loading**

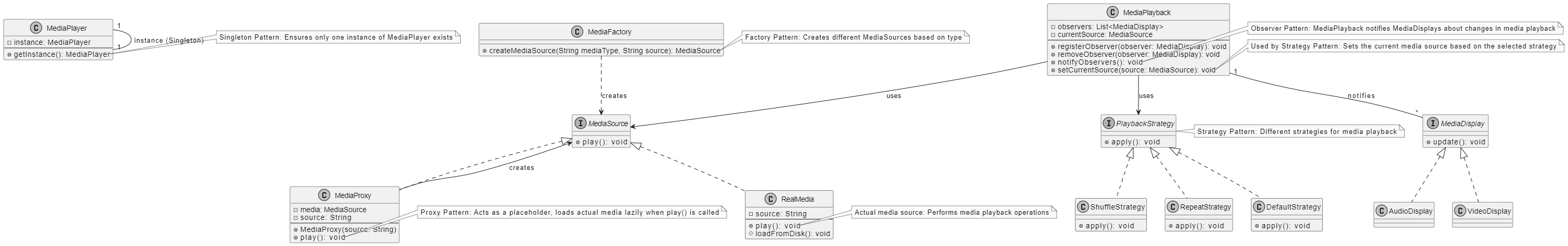
This usage of the Proxy pattern aligns with the concept of lazy loading:

**Lazy Object Creation:** The RealMedia object is only created when it's needed, avoiding unnecessary initialization during the Music Player's startup.

**Deferred Initialization**: The actual loading of the media is delayed until the play() method is called on the MediaProxy, which signifies the need for that media source.

Therefore, while the pattern is named Proxy in the example, its behavior of deferring the creation and loading of the RealMedia object until it's actually needed aligns closely with the principles of lazy loading.

**Class Diagram:**

****