Padrões de criação:

ProtoType

Exemplo de diagrama

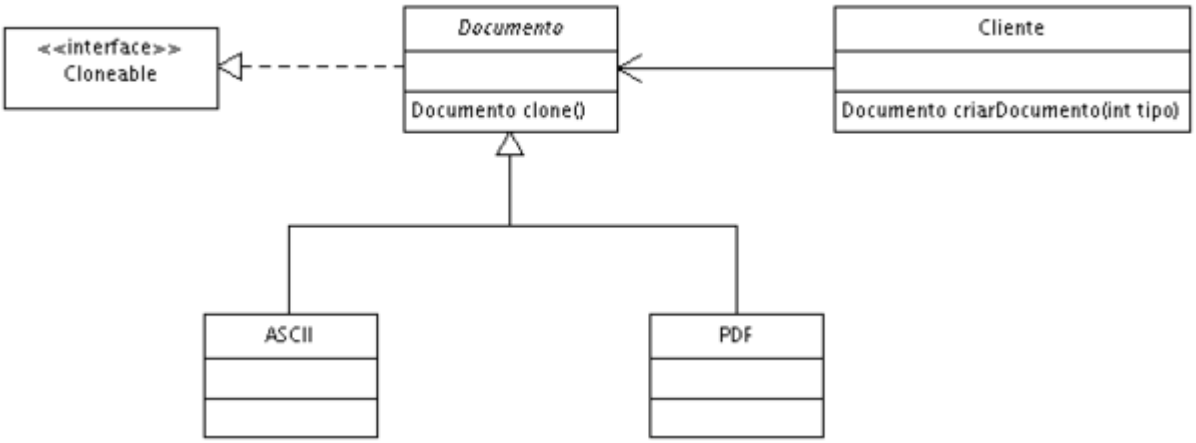
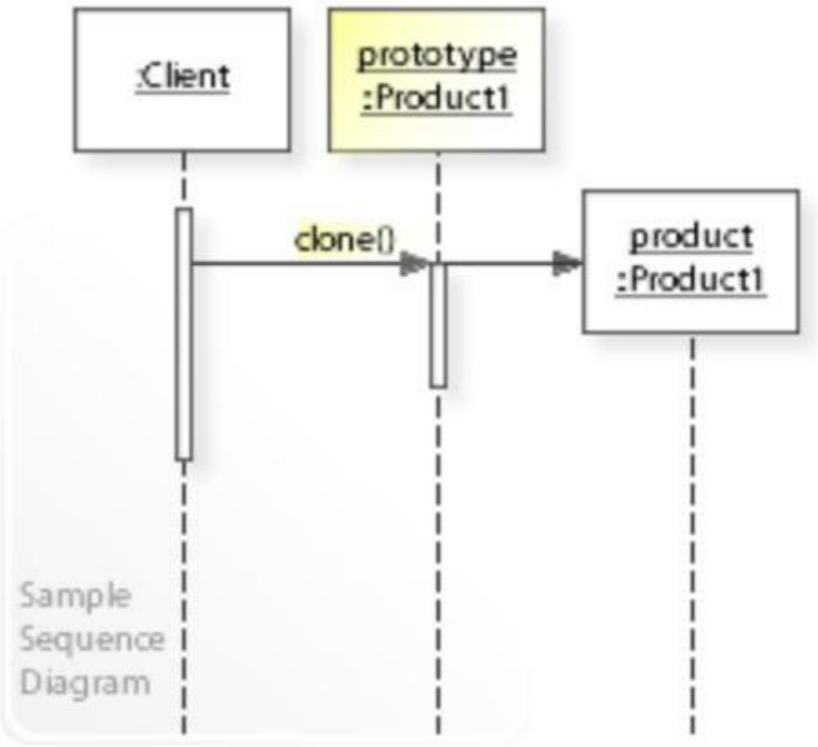


Diagrama de Sequência



Exemplo de código

```
public interface ICloneable {
        public Object clone();
}

abstract class Documento implements ICloneable {

    protected Documento clone() {

        Object clone = null;

        try {
             clone = super.clone();
        } catch (CloneNotSupportedException ex) {
             ex.printStackTrace();
        }

        return (Documento) clone;
    }
}

class ASCII extends Documento { }

class PDF extends Documento { }

class Cliente {

    static final int DOCUMENTO_TIPO_ASCII = 0;

    static final int DOCUMENTO_TIPO_PDF = 1;

    private Documento ascii = new ASCII();
```

```
    private Documento pdf = new PDF();

    public Documento criarDocumento(int tipo) {

        if (tipo == Cliente.DOCUMENTO_TIPO_ASCII) {
            return ascii.clone();
        } else {
            return pdf.clone();
        }
    }
}
```

Bibliotecas java

Java.lang.Cloneable

Singleton: (assegura que somente um objeto de uma determinada classe seja criado em todo o projeto)

Diagrama de classe

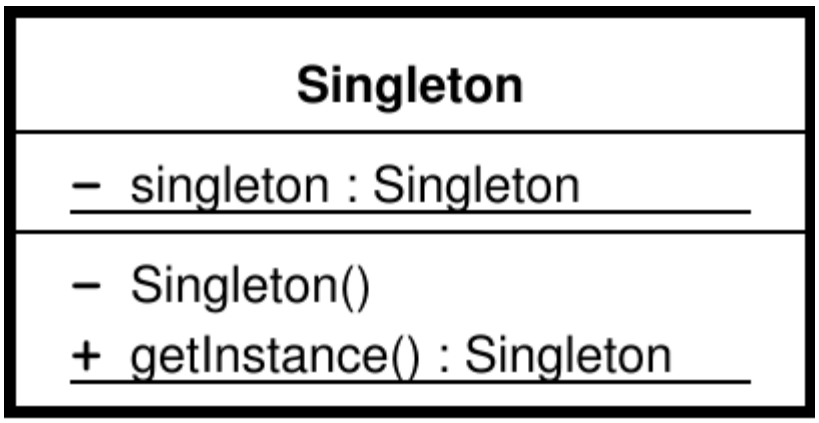
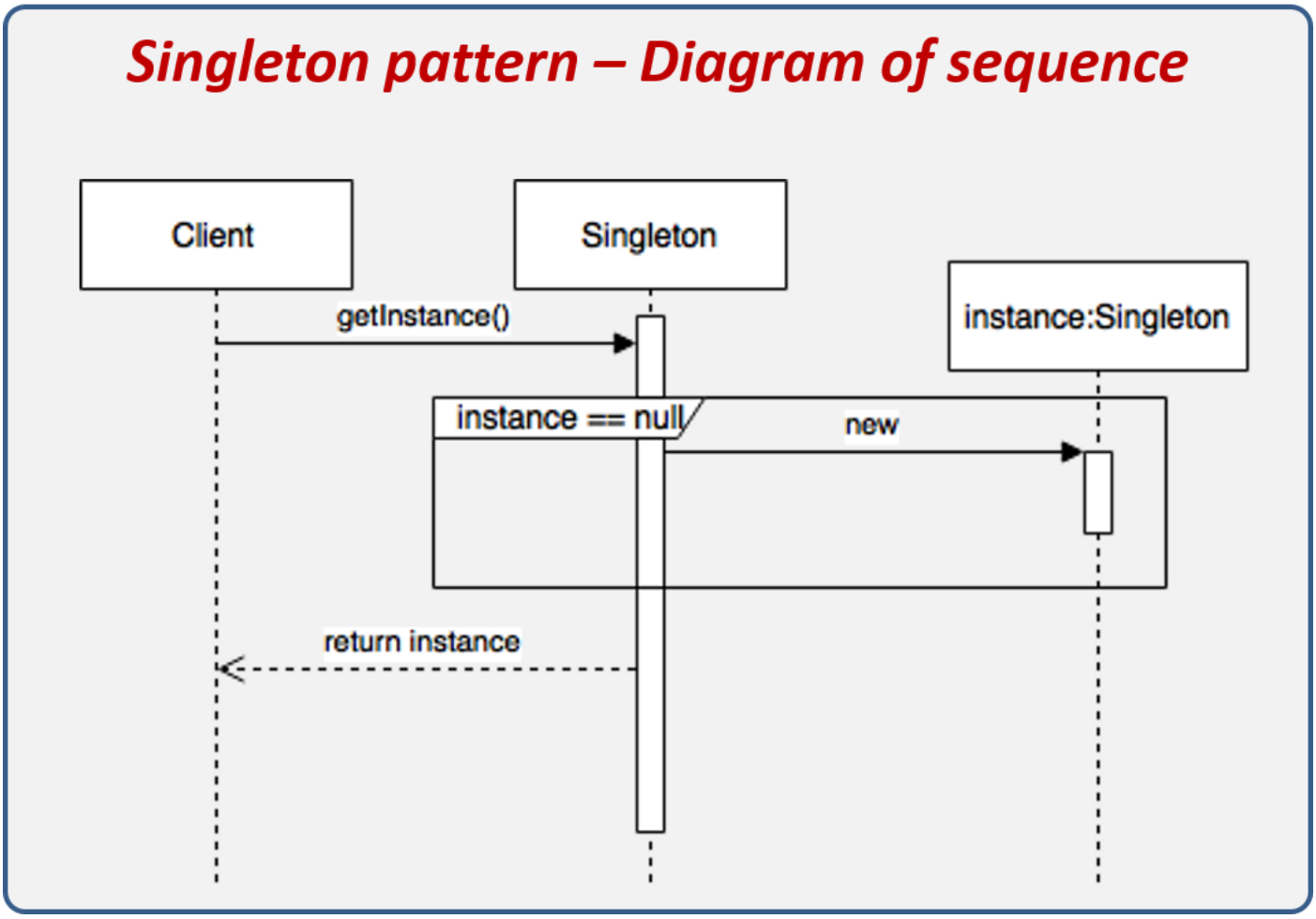


Diagrama de sequencia



Exemplo 1

```
public class Singleton {

	private static Singleton uniqueInstance;

	private Singleton() {
	}

	public static synchronized Singleton getInstance() {
		if (uniqueInstance == null)
			uniqueInstance = new Singleton();
```

```
                return uniqueInstance;
        }
}
```

Exemplo 2

```
public class Singleton {

        private static Singleton uniqueInstance = new Singleton();

        private Singleton() {
        }

        public static Singleton getInstance() {
                return uniqueInstance;
        }
}
```

Exemplos de Singleton nas bibliotecas principais do Java:

- **java.lang.Runtime#getRuntime()**
- **java.awt.Desktop#getDesktop()**
- **java.lang.System#getSecurityManager()**

Padrão de estrutura

Bridge

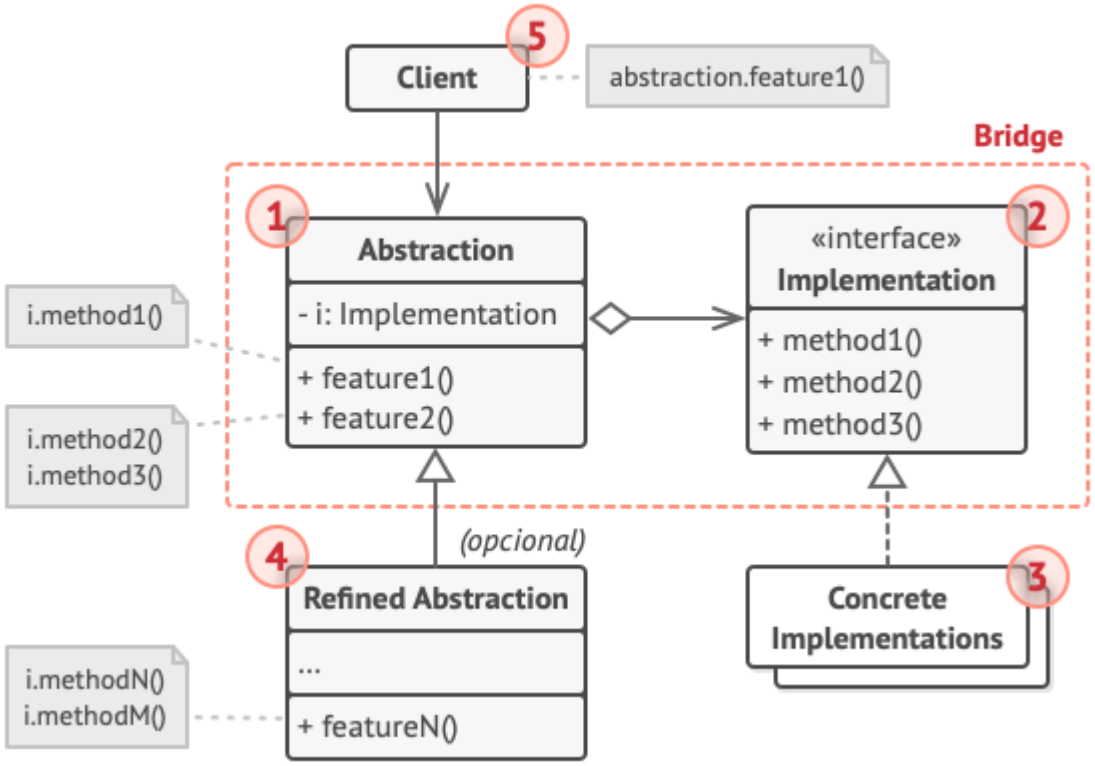


Diagrama de sequência

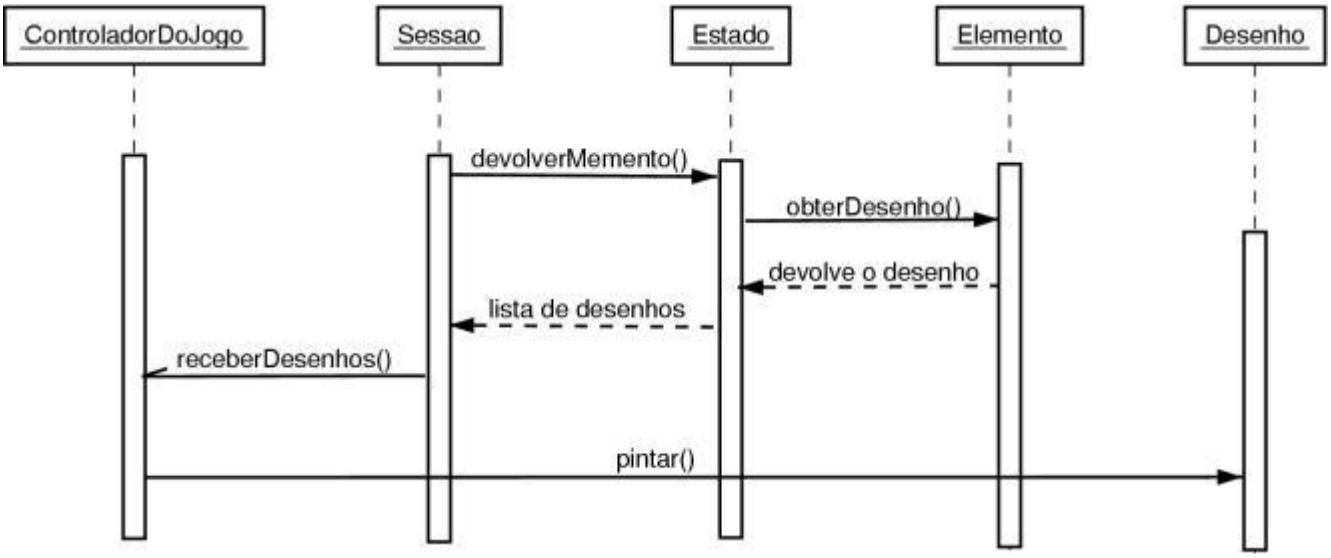
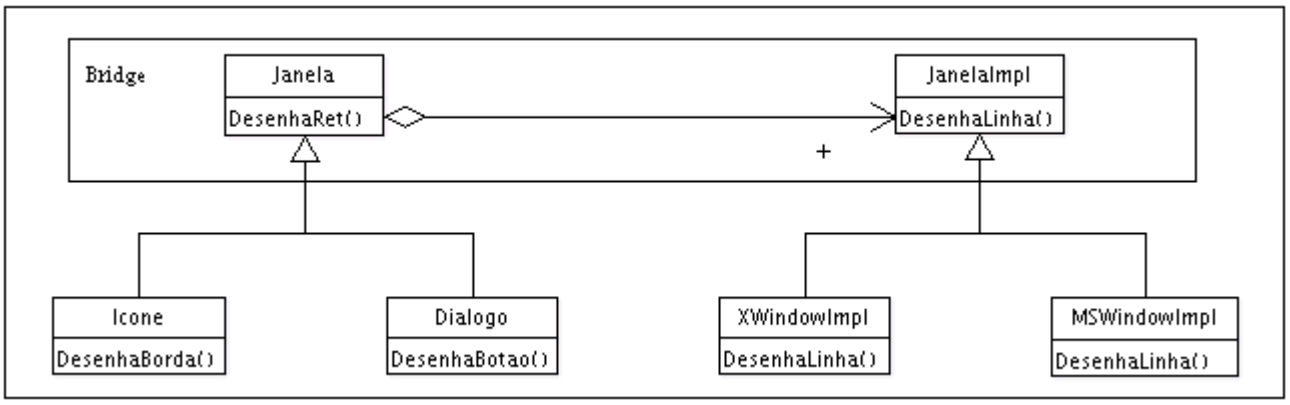


Diagrama de comunicação



Exemplo de código

```
class RemoteControl is
    protected field device: Device
    constructor RemoteControl(device: Device) is
        this.device = device
    method togglePower() is
        if (device.isEnabled()) then
            device.disable()
        else
            device.enable()
    method volumeDown() is
        device.setVolume(device.getVolume() - 10)
    method volumeUp() is
        device.setVolume(device.getVolume() + 10)
    method channelDown() is
        device.setChannel(device.getChannel() - 1)
    method channelUp() is
        device.setChannel(device.getChannel() + 1)


class AdvancedRemoteControl extends RemoteControl is
    method mute() is
        device.setVolume(0)
interface Device is
    method isEnabled()
    method enable()
    method disable()
    method getVolume()
    method setVolume(percent)
    method getChannel()
    method setChannel(channel)
class Tv implements Device is

class Radio implements Device is
tv = new Tv()
remote = new RemoteControl(tv)
remote.togglePower()

radio = new Radio()
remote = new AdvancedRemoteControl(radio)
```

Padrão de estrutura

Proxy

Diagrama de classe

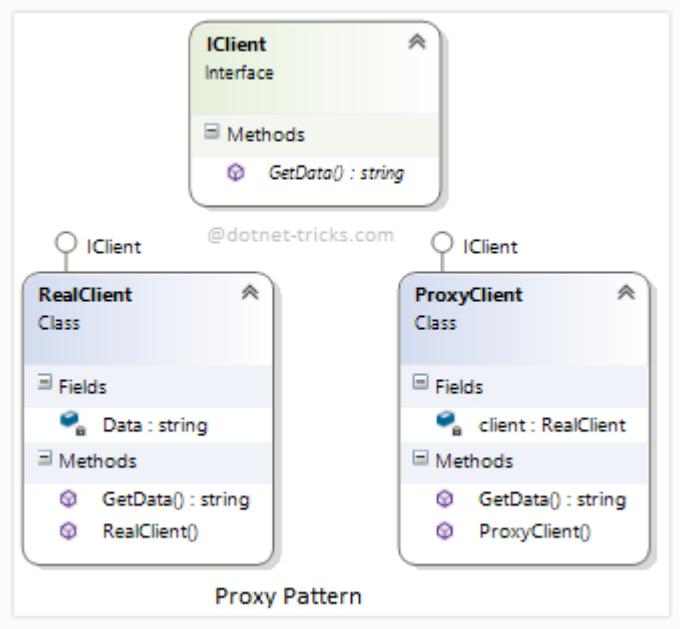
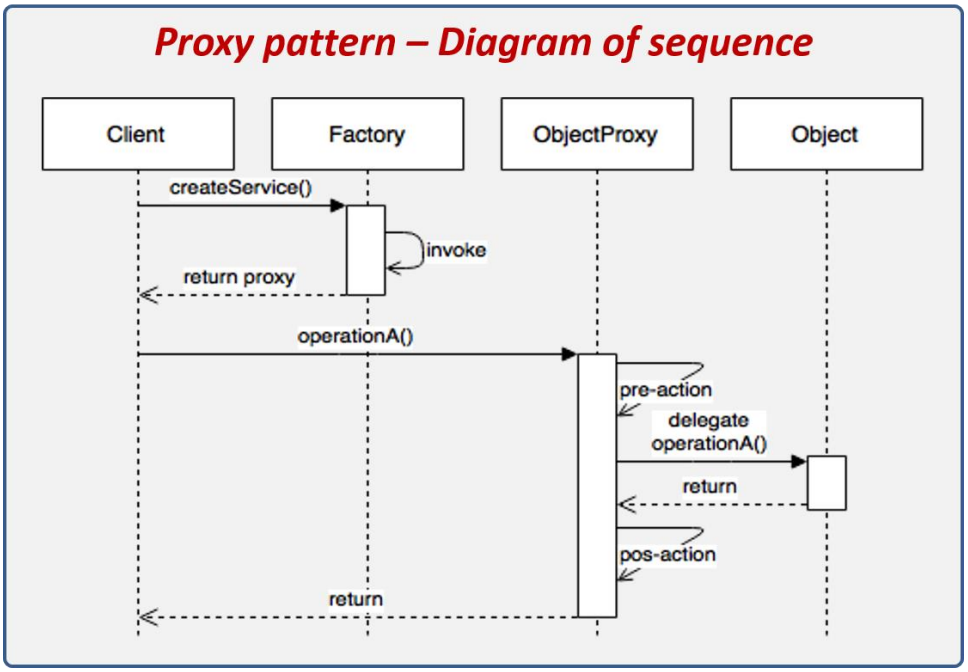


Diagrama de sequencia



Bibliotecas java

javax.persistence.PersistenceContext

java.lang.reflect.Proxy

javax.inject.Inject

Exemplo de código

```
interface ThirdPartyYouTubeLib is
    method listVideos()
    method getVideoInfo(id)
    method downloadVideo(id)

class ThirdPartyYouTubeClass implements ThirdPartyYouTubeLib is
    method listVideos() is

    method getVideoInfo(id) is

    method downloadVideo(id) is

class CachedYouTubeClass implements ThirdPartyYouTubeLib is
    private field service: ThirdPartyYouTubeLib
    private field listCache, videoCache
    field needReset

    constructor CachedYouTubeClass(service: ThirdPartyYouTubeLib) is
        this.service = service

    method listVideos() is
        if (listCache == null || needReset)
            listCache = service.listVideos()
        return listCache

    method getVideoInfo(id) is
        if (videoCache == null || needReset)
            videoCache = service.getVideoInfo(id)
        return videoCache

    method downloadVideo(id) is
        if (!downloadExists(id) || needReset)
            service.downloadVideo(id)
class YouTubeManager is
    protected field service: ThirdPartyYouTubeLib

    constructor YouTubeManager(service: ThirdPartyYouTubeLib) is
        this.service = service
```

```
    method renderVideoPage(id) is
        info = service.getVideoInfo(id)

    method renderListPanel() is
        list = service.listVideos()

    method reactOnUserInput() is
        renderVideoPage()
        renderListPanel()
class Application is
    method init() is
        aYouTubeService = new ThirdPartyYouTubeClass()
        aYouTubeProxy = new CachedYouTubeClass(aYouTubeService)
        manager = new YouTubeManager(aYouTubeProxy)
        manager.reactOnUserInput()
```

Padrão de comportamento

State

Diagrama

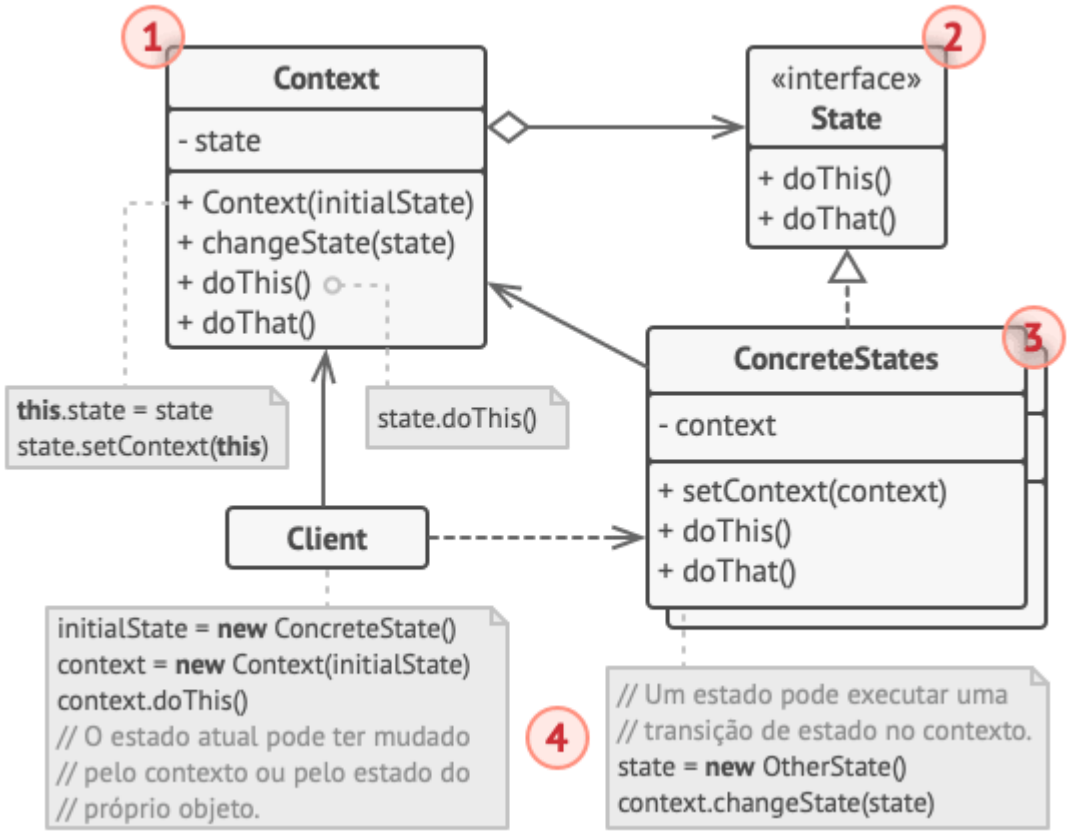
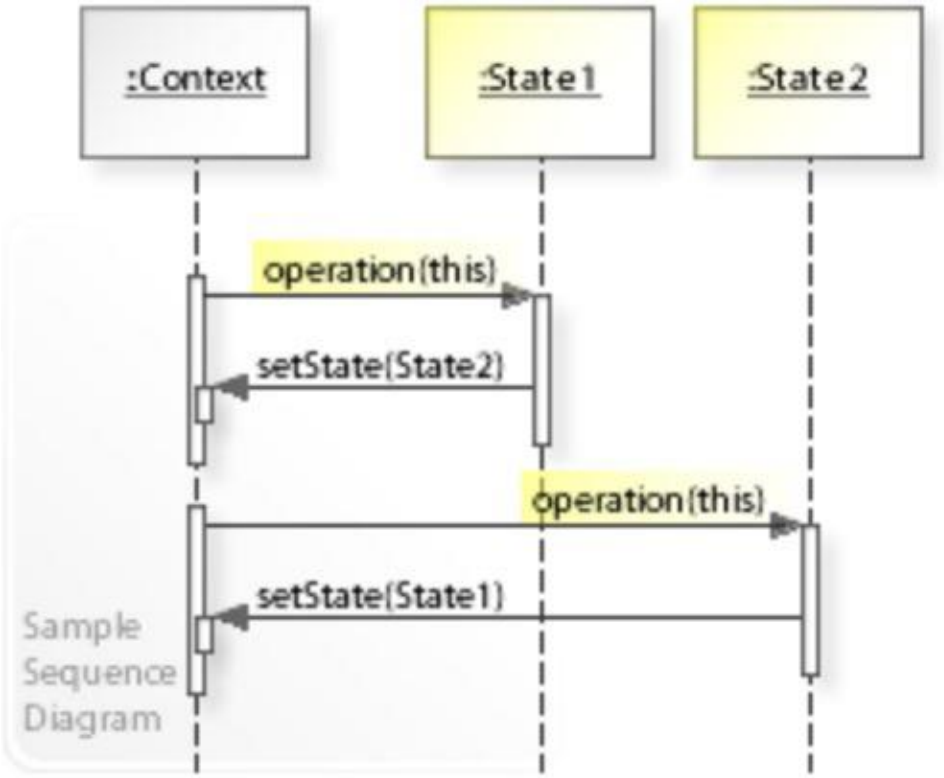


Diagrama de sequência



Exemplo de código

```
class AudioPlayer is
    field state: State
    field UI, volume, playlist, currentSong

    constructor AudioPlayer() is
        this.state = new ReadyState(this)
        UI = new UserInterface()
        UI.lockButton.onClick(this.clickLock)
        UI.playButton.onClick(this.clickPlay)
        UI.nextButton.onClick(this.clickNext)
        UI.prevButton.onClick(this.clickPrevious)

    method changeState(state: State) is
        this.state = state
    method clickLock() is
        state.clickLock()
    method clickPlay() is
        state.clickPlay()
    method clickNext() is
        state.clickNext()
    method clickPrevious() is
        state.clickPrevious()

    method startPlayback() is
        // ...
    method stopPlayback() is
        // ...
    method nextSong() is
        // ...
    method previousSong() is
        // ...
    method fastForward(time) is
        // ...
    method rewind(time) is
        // ...

abstract class State is
    protected field player: AudioPlayer

    constructor State(player) is
        this.player = player

    abstract method clickLock()
    abstract method clickPlay()
    abstract method clickNext()
    abstract method clickPrevious()


class LockedState extends State is
    method clickLock() is
        if (player.playing)
            player.changeState(new PlayingState(player))
        else
            player.changeState(new ReadyState(player))

    method clickPlay() is
```

```
    method clickNext() is

    method clickPrevious() is

class ReadyState extends State is
    method clickLock() is
        player.changeState(new LockedState(player))

    method clickPlay() is
        player.startPlayback()
        player.changeState(new PlayingState(player))

    method clickNext() is
        player.nextSong()

    method clickPrevious() is
        player.previousSong()


class PlayingState extends State is
    method clickLock() is
        player.changeState(new LockedState(player))

    method clickPlay() is
        player.stopPlayback()
        player.changeState(new ReadyState(player))

    method clickNext() is
        if (event.doubleclick)
            player.nextSong()
        else
            player.fastForward(5)

    method clickPrevious() is
        if (event.doubleclick)
            player.previous()
        else
            player.rewind(5)
```

Bibliotecas

javax.faces.lifecycle.LifeCycle#execute()

Observer

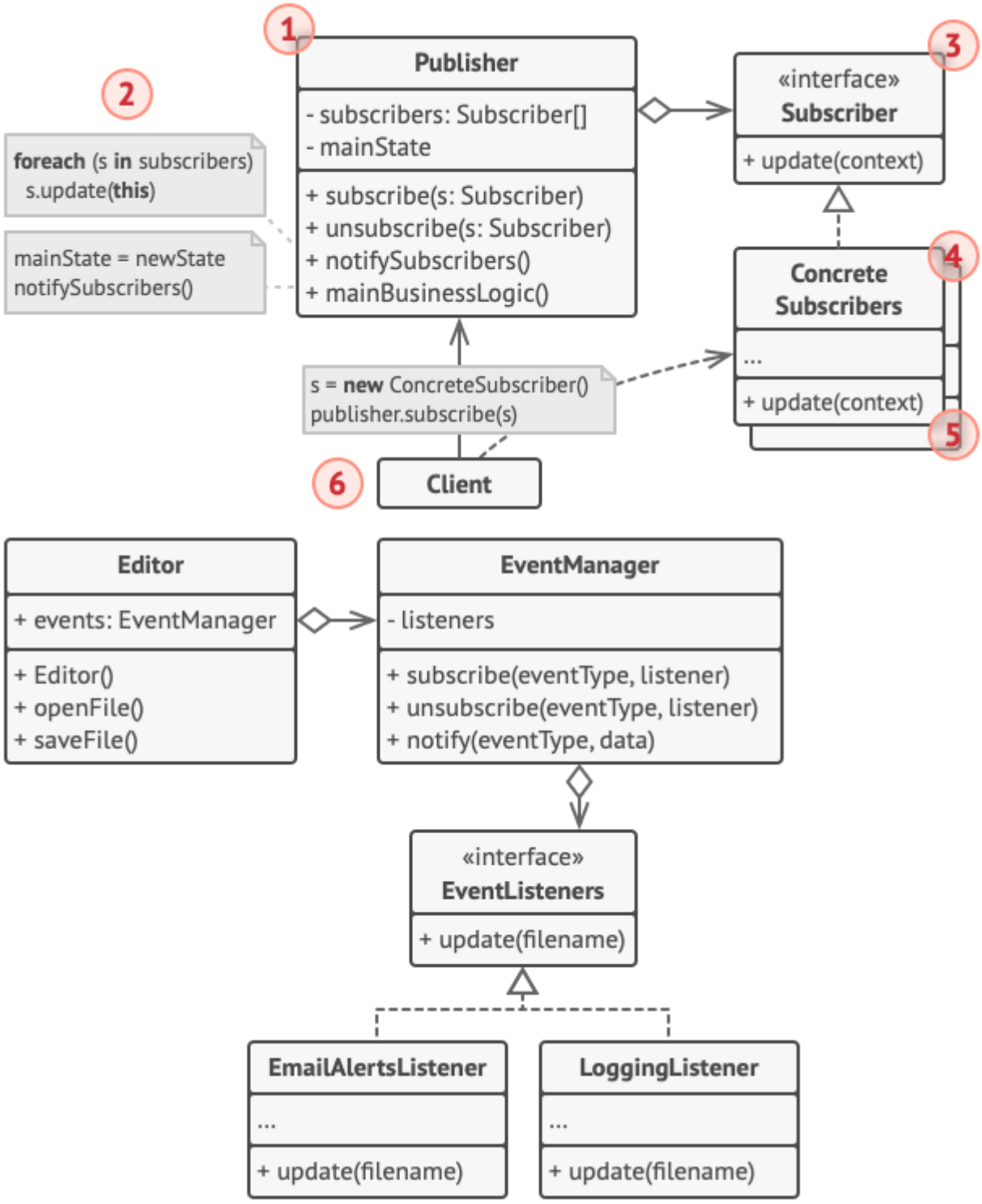
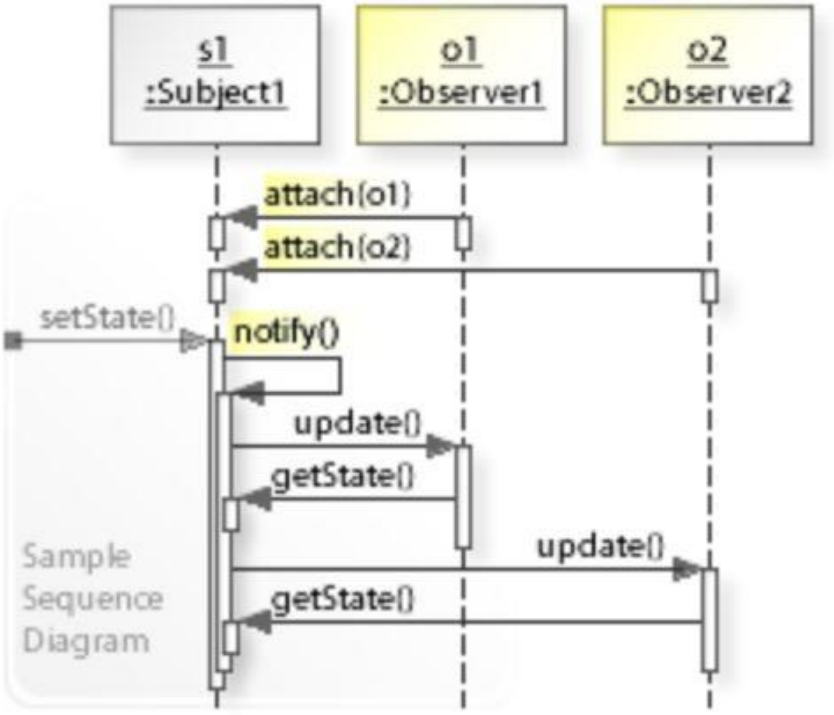


Diagrama de sequência



Exemplo de código

```
// A classe publicadora base inclui o código de gerenciamento de
// inscrições e os métodos de notificação.
class EventManager is
    private field listeners: hash map of event types and listeners

    method subscribe(eventType, listener) is
        listeners.add(eventType, listener)

    method unsubscribe(eventType, listener) is
        listeners.remove(eventType, listener)

    method notify(eventType, data) is
        foreach (listener in listeners.of(eventType)) do
            listener.update(data)

// O publicador concreto contém a verdadeira lógica de negócio
// que é de interesse para alguns assinantes. Nós podemos
// derivar essa classe a partir do publicador base, mas isso nem
// sempre é possível na vida real devido a possibilidade do
// publicador concreto já ser uma subclasse. Neste caso, você
// pode remendar a lógica de inscrição com a composição, como
// fizemos aqui.
class Editor is
    public field events: EventManager
    private field file: File
```

```
    constructor Editor() is
        events = new EventManager()

    // Métodos da lógica de negócio podem notificar assinantes
    // acerca de mudanças.
    method openFile(path) is
        this.file = new File(path)
        events.notify("open", file.name)

    method saveFile() is
        file.write()
        events.notify("save", file.name)

    // ...


// Aqui é a interface do assinante. Se sua linguagem de
// programação suporta tipos funcionais, você pode substituir
// toda a hierarquia do assinante por um conjunto de funções.
interface EventListener is
    method update(filename)

// Assinantes concretos reagem a atualizações emitidas pelo
// publicador a qual elas estão conectadas.
class LoggingListener implements EventListener is
    private field log: File
    private field message: string

    constructor LoggingListener(log_filename, message) is
        this.log = new File(log_filename)
        this.message = message

    method update(filename) is
        log.write(replace('%s',filename,message))

class EmailAlertsListener implements EventListener is
    private field email: string
    private field message: string

    constructor EmailAlertsListener(email, message) is
        this.email = email
        this.message = message

    method update(filename) is
        system.email(email, replace('%s',filename,message))


// Uma aplicação pode configurar publicadores e assinantes
// durante o tempo de execução.
class Application is
    method config() is
        editor = new Editor()

        logger = new LoggingListener(
            "/path/to/log.txt",
            "Someone has opened the file: %s")
        editor.events.subscribe("open", logger)
```

```
        emailAlerts = new EmailAlertsListener(
            "admin@example.com",
            "Someone has changed the file: %s")
        editor.events.subscribe("save", emailAlerts)
```

Bibliotecas

javax.servlet.http.HttpSessionBindingListener

javax.servlet.http.HttpSessionAttributeListener

javax.faces.event.PhaseListener