State Design Pattern in Java

Introduction

Over the course of this article, we will examine State design pattern in java with help of realtime examples.

The State design pattern belongs to the behavioral family of pattern that deals with the runtime object behavior based on the current state. The definition of State Design Pattern as per the original Gang of Four book is; "Allows an object to alter its behavior when its internal state changes. The object will appear to change its class".

How State pattern works?

1. Define an object that represent various states that object can be. Remember state machine.
2. Define a context object, whose behavior varies as its state object changes.

Real time use case

State pattern is one of the heavily used pattern in game development. The game character can be in different states such as healthy, surviving and dead. When character is healthy, it allows user to fires at enemies with different weapons. When surviving state its health gets critical, and when its health reaches to 0, the character is said to be in dead state where the game is over.

Let us implement this use case without using State design pattern. It can be achieved by using set of if else conditional checks, as shown in the following code snippets.

Player.java

The Player class defines the different actions a player can perform.

```
public class Player {

	public void attack() {
		System.out.println("Attack");
	}

	public void fireBumb() {
		System.out.println("Fire Bomb");
```

```
	}

	public void fireGunblade() {
		System.out.println("Fire Gunblade");
	}

	public void fireLaserPistol() {
		System.out.println("Laser Pistols");
	}

	public void firePistol() {
		System.out.println("Fire Pistol");
	}

	public void survive() {
		System.out.println("Surviving!");
	}

	public void dead() {
		System.out.println("Dead! Game Over");
	}

}
```

Now let us define our game context class which defines the different actions conditionally depends on the state of the player.

GameContext.java

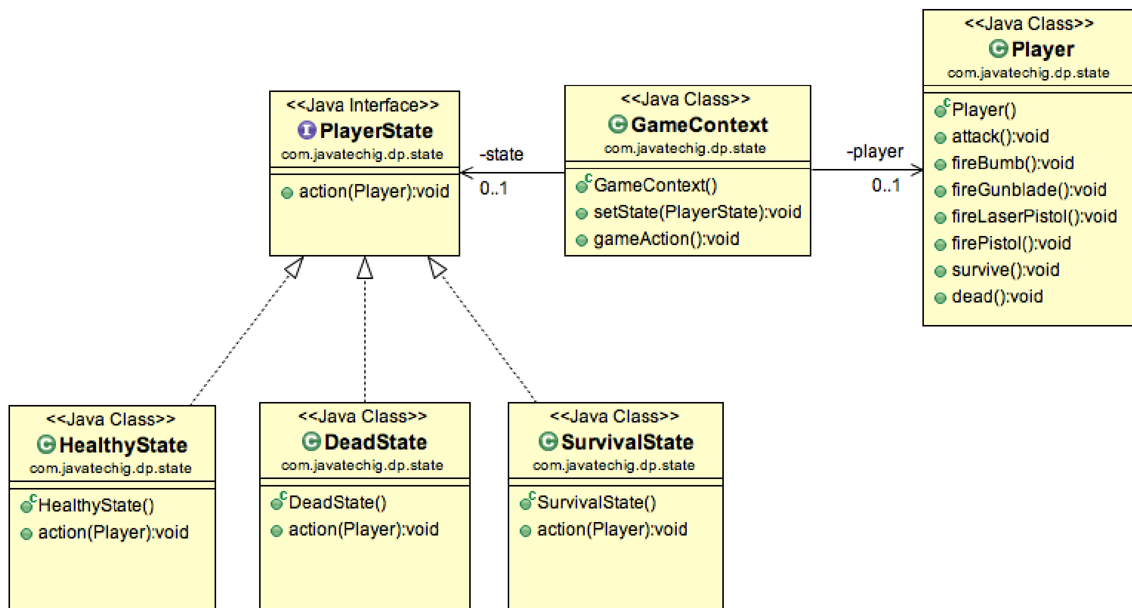
```
public class GameContext {

	private Player player = new Player();

	public void gameAction(String state) {
		if (state == "healthy") {
			player.attack();
			player.fireBumb();
			player.fireGunblade();
			player.fireLaserPistol();
		} else if (state == "survival") {
			player.survive();
			player.firePistol();
		} else if (state == "dead") {
			player.dead();
		}
```

```
    }
}
```

In the above code snippet, the gameAction method contains too many conditional blocks for performing different game action based on the state of player. This is a real problem of code maintainability. this can be avoided using State design pattern.

State Design Pattern Example

Before we begin with the state design pattern, let us have a look into the class design.



1. Define an interface named PlayerState that defines action method. The access() method takes the instance of Player class. This is required to perform player action.

```
public interface PlayerState {
    void action(Player p);
}
```

2. Define three different classes that represents the different states. In this example, I have named them as `HealthyState`, `SurvivalState`, `DeadState`. All of three classes implements `PlayerState`interface and provides the specific `action()` method implementation.

```
}

public class HealthyState implements PlayerState {

	@Override
	public void action(Player p) {
		p.attack();
		p.fireBumb();
		p.fireGunblade();
		p.fireLaserPistol();
	}
}

public class SurvivalState implements PlayerState {

	@Override
	public void action(Player p) {
		p.survive();
		p.firePistol();
	}
}

public class DeadState implements PlayerState {

	@Override
	public void action(Player p) {
		p.dead();
	}
}
```

3. The `GameContxt` class contains two `setState()` method composition. Now we will remove all of the code to conditional logic.

```
public class GameContext {

	private PlayerState state = null;
	private Player player = new Player();

	public void setState(PlayerState state) {
		this.state = state;
	}

	public void gameAction() {
		state.action(player);
	}
}
```

4. Let’s test our code using below `GameTest` class.

```
public class GameTest {

	public static void main(String[] args) {

		GameContext context = new GameContext();

		context.setState(new HealthyState());
		context.gameAction();
		System.out.println("*****");

		context.setState(new SurvivalState());
		context.gameAction();
		System.out.println("*****");

		context.setState(new DeadState());
		context.gameAction();
		System.out.println("*****");

	}
}
```

Output

```
Attack
Fire Bomb
Fire Gunblade
Laser Pistols
*****
Surviving!
Fire Pistol
*****
Dead! Game Over
*****
```