In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class. This pattern comes under behavior pattern category.

## Implementation

We are going to create a *Game* abstract class defining operations with a template method set to be final so that it cannot be overridden. *Cricket* and *Football* are concrete classes that extend *Game* and override its methods.

*TemplatePatternDemo*, our demo class, will use *Game* to demonstrate use of template pattern.



/\*

\* Step 1

\* create an abstract class with a template method being final

\* Game.java

\*/

abstract class Game{

abstract void initialize();

abstract void startPlay();

abstract void endPlay();

//template methods , order may be changed

public final void play() {

//initialize the game

initialize();

//start game

startPlay();

//end game

endPlay();

}

}

/\*

\* Step 2

\* Create concrete classes extending the above class

\*/

class Cricket extends Game{

@Override

void initialize() {

System.out.println("Cricket game initialized , start playing");

}

@Override

void startPlay() {

System.out.println("Cricket game started, Enjoy the game");

}

@Override

void endPlay() {

System.out.println("Cricket Game finished !!");

}

}

class FootBall extends Game{

@Override

void initialize() {

System.out.println("Football game initialized , start playing");

}

@Override

void startPlay() {

System.out.println("Football game started, Enjoy the game");

}

@Override

void endPlay() {

System.out.println("Football Game finished !!");

}

}

/\*

\* step 3

\* Use the Game's template method play() to demonstrate a defined way of playing game.

\*/

public class TestTemplatePatternDemo {

public static void main(String[] args) {

Game game = new Cricket();

game.play();

game = new FootBall();

game.play();

}

}