



LEARN JAVA DESIGN PATTERNS

problem solving approaches

Design Patterns Tutorial

- [Design Patterns - Home](#)
- [Design Patterns - Overview](#)
- [Design Patterns - Factory Pattern](#)
- [Abstract Factory Pattern](#)
- [Design Patterns - Singleton Pattern](#)
- [Design Patterns - Builder Pattern](#)
- [Design Patterns - Prototype Pattern](#)
- [Design Patterns - Adapter Pattern](#)
- [Design Patterns - Bridge Pattern](#)
- [Design Patterns - Filter Pattern](#)
- [Design Patterns - Composite Pattern](#)
- [Design Patterns - Decorator Pattern](#)
- [Design Patterns - Facade Pattern](#)
- [Design Patterns - Flyweight Pattern](#)
- [Design Patterns - Proxy Pattern](#)
- [Chain of Responsibility Pattern](#)
- [Design Patterns - Command Pattern](#)
- [Design Patterns - Interpreter Pattern](#)
- [Design Patterns - Iterator Pattern](#)
- [Design Patterns - Mediator Pattern](#)
- [Design Patterns - Memento Pattern](#)
- [Design Patterns - Observer Pattern](#)
- [Design Patterns - State Pattern](#)
- [Design Patterns - Null Object Pattern](#)
- [Design Patterns - Strategy Pattern](#)
- [Design Patterns - Template Pattern](#)
- [Design Patterns - Visitor Pattern](#)
- [Design Patterns - MVC Pattern](#)
- [Business Delegate Pattern](#)
- [Composite Entity Pattern](#)
- [Data Access Object Pattern](#)
- [Front Controller Pattern](#)
- [Intercepting Filter Pattern](#)
- [Service Locator Pattern](#)
- [Transfer Object Pattern](#)

Design Patterns Resources

- [Design Patterns - Questions/Answers](#)

Design Patterns - Template Pattern

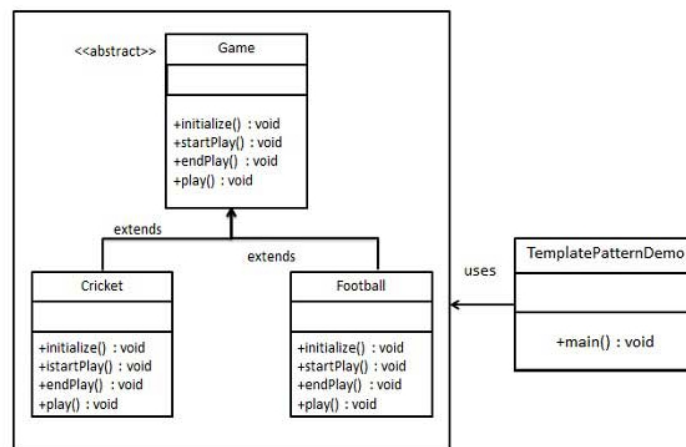
[Previous Page](#)
[Next Page](#)

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

```

public abstract class Game {
    abstract void initialize();
    abstract void startPlay();
    abstract void endPlay();

    //template method
    public final void play(){

        //initialize the game
        initialize();

        //start game
        startPlay();

        //end game
        endPlay();
    }
}
    
```

Step 2

Create concrete classes extending the above class.

Cricket.java

```

public class Cricket extends Game {

    @Override
    void endPlay() {
        System.out.println("Cricket Game Finished!");
    }
}
    
```

- Design Patterns - Quick Guide
- Design Patterns - Useful Resources
- Design Patterns - Discussion
- Selected Reading
- UPSC IAS Exams Notes
- Developer's Best Practices
- Questions and Answers
- Effective Resume Writing
- HR Interview Questions
- Computer Glossary
- Who is Who

```
}

@Override
void initialize() {
    System.out.println("Cricket Game Initialized! Start playing.");
}

@Override
void startPlay() {
    System.out.println("Cricket Game Started. Enjoy the game!");
}
}
```

Football.java

```
public class Football extends Game {

    @Override
    void endPlay() {
        System.out.println("Football Game Finished!");
    }

    @Override
    void initialize() {
        System.out.println("Football Game Initialized! Start playing.");
    }

    @Override
    void startPlay() {
        System.out.println("Football Game Started. Enjoy the game!");
    }
}
```

Step 3

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

TemplatePatternDemo.java

```
public class TemplatePatternDemo {
    public static void main(String[] args) {

        Game game = new Cricket();
        game.play();
        System.out.println();
        game = new Football();
        game.play();
    }
}
```

Step 4

Verify the output.

```
Cricket Game Initialized! Start playing.
Cricket Game Started. Enjoy the game!
Cricket Game Finished!

Football Game Initialized! Start playing.
Football Game Started. Enjoy the game!
Football Game Finished!
```

[Previous Page](#) [Print Page](#)

[Next Page](#)

