Design Pattern

Unit: I

Introduction to Design Pattern

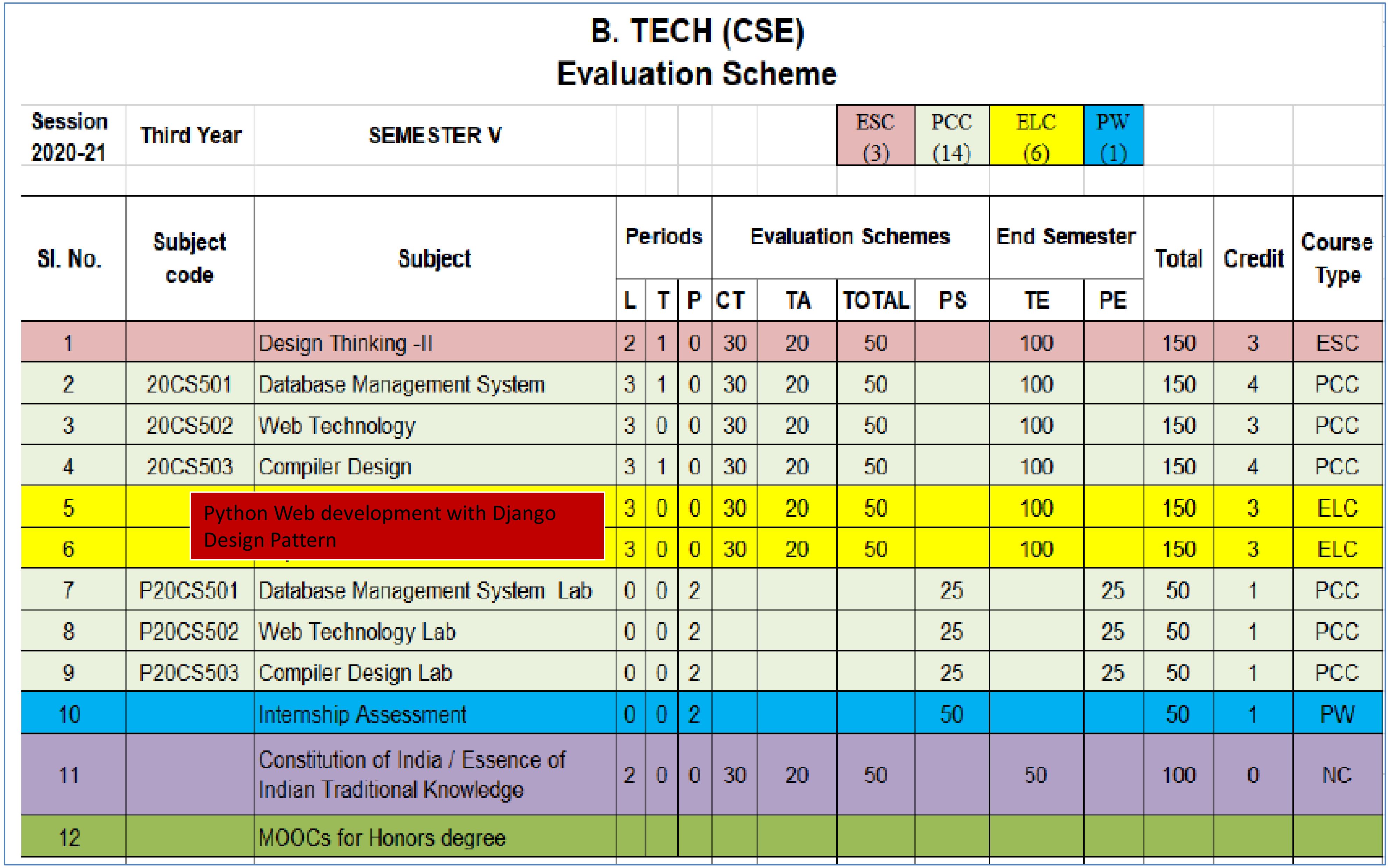
Course Details
(B. Tech. 5th Sem)

Ibrar Ahmed
(Asst. Professor)
CSE Department

Faculty Introduction

Name	Ibrar Ahmed
Qualification	M. Tech. (Computer Engineering)
Designation	Assistant Professor
Department	Computer Science & Engineering
Total Experience	4 years
NIET Experience	1 years
Subject Taught	Design & Analysis of Algorithm, Data Structures, Artificial Intelligence, Soft Computing, C Programming, Web Technology, Discrete Mathematics.

Evaluation Scheme



B. TECH (CSE)
Evaluation Scheme

Session 2020-21	Third Year	SEMESTER V						ESC (3)	PCC (14)	ELC (6)	PW (1)			
Sl. No.	Subject code	Subject	Periods			Evaluation Schemes				End Semester		Total	Credit	Course Type
			L	T	P	CT	TA	TOTAL	PS	TE	PE			
1		Design Thinking -II	2	1	0	30	20	50		100		150	3	ESC
2	20CS501	Database Management System	3	1	0	30	20	50		100		150	4	PCC
3	20CS502	Web Technology	3	0	0	30	20	50		100		150	3	PCC
4	20CS503	Compiler Design	3	1	0	30	20	50		100		150	4	PCC
5		Python Web development with Django Design Pattern	3	0	0	30	20	50		100		150	3	ELC
6			3	0	0	30	20	50		100		150	3	ELC
7	P20CS501	Database Management System Lab	0	0	2				25		25	50	1	PCC
8	P20CS502	Web Technology Lab	0	0	2				25		25	50	1	PCC
9	P20CS503	Compiler Design Lab	0	0	2				25		25	50	1	PCC
10		Internship Assessment	0	0	2				50			50	1	PW
11		Constitution of India / Essence of Indian Traditional Knowledge	2	0	0	30	20	50		50		100	0	NC
12		MOOCs for Honors degree												

Syllabus

UNIT-I: Introduction of Design Pattern

Describing Design Patterns, Design Patterns in Smalltalk MVC, The Catalogue of Design Patterns, Organizing The Cato log, How Design Patterns solve, Design Problems, How to Select a Design pattern, How to Use a Design Pattern. Principle of least knowledge.

Syllabus

UNIT-II: Creational Design Pattern

A Case Study: Designing a Document Editor

Creational Patterns: Abstract Factory, Builder , Factory Method, Prototype , Singleton Pattern,

Syllabus

UNIT-III: Structural Design Pattern

Structural Pattern Part-I, Adapter, Bridge, Composite.

Structural Pattern Part-II, Decorator, Facade, Flyweight, Proxy.

Syllabus

UNIT-IV: Behavioral Design Patterns Part: I

Behavioral Patterns Part: I, Chain of Responsibility, Command, Interpreter, Iterator Pattern.

Behavioral Patterns Part: II, Mediator, Memento, Observer, Patterns.

Syllabus

UNIT-V: Behavioral Design Patterns Part: II

Behavioral Patterns Part: III, State, Strategy, Template Method, Visitor, What to Expect from Design Patterns.

Branch Wise Application

1. Real time web analytics
2. Digital Advertising
3. E-Commerce
4. Publishing
5. Massively Multiplayer Online Games
6. Backend Services and Messaging
7. Project Management & Collaboration
8. Real time Monitoring Services
9.Live Charting and Graphing
10. Group and Private Chat

Course Objective

In this semester, the students will

- Study how to shows relationships and interactions between classes or objects..
- Study to speed up the development process by providing well-tested, proven development/design paradigms.
- Select a specific design pattern for the solution of a given design problem.
- Create a catalogue entry for a simple design pattern whose purpose and application is understood.

Course Outcomes (COs)

At the end of course, the student will be able to:

CO1 : Construct a design consisting of collection of modules.

CO2 : Exploit well known design pattern such as Factory, visitor etc.

CO3 : Distinguish between different categories of design patterns.

CO4 : Ability to common design pattern for incremental development.

CO5 : Identify appropriate design pattern for a given problem and design the software using pattern oriented architecture.

Program Outcomes (POs)

Engineering Graduates will be able to:

PO1 : Engineering Knowledge

PO2 : Problem Analysis

PO3 : Design/Development of solutions

PO4 : Conduct Investigations of complex problems

PO5 : Modern tool usage

PO6 : The engineer and society

Program Outcomes (POs)

Engineering Graduates will be able to:

PO7 : Environment and sustainability

PO8 : Ethics

PO9 : Individual and teamwork

PO10 : Communication

PO11 : Project management and finance

PO12 : Life-long learning

COs - POs Mapping

CO.K	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	2	2	3	3	-	-	-	-	-	-	-
CO2	3	2	3	2	3	-	-	-	-	-	-	-
CO3	3	2	3	2	3	-	-	-	-	-	-	-
CO4	3	2	3	2	3	-	-	-	-	-	-	-
CO5	3	2	3	3	3	-	-	-	-	-	-	-
AVG	2.8	2.0	2.8	2.4	3.0	-	-	-	-	-	-	-

Program Specific Outcomes(PSOs)

S. No.	Program Specific Outcomes (PSO)	PSO Description
1	PSO1	Understand to shows relationships and interactions between classes or objects of a pattern.
2	PSO2	Study to speed up the development process by providing well-tested, proven development
3	PSO3	Select a specific design pattern for the solution of a given design problem
4	PSO4	Create a catalogue entry for a simple design pattern whose purpose and application is understood.

COs - PSOs Mapping

CO.K	PSO1	PSO2	PSO3	PSO4
CO1	3	-	-	-
CO2	3	3	-	-
CO3	3	3	-	-
CO4	3	3	-	-
CO5	3	3	-	-

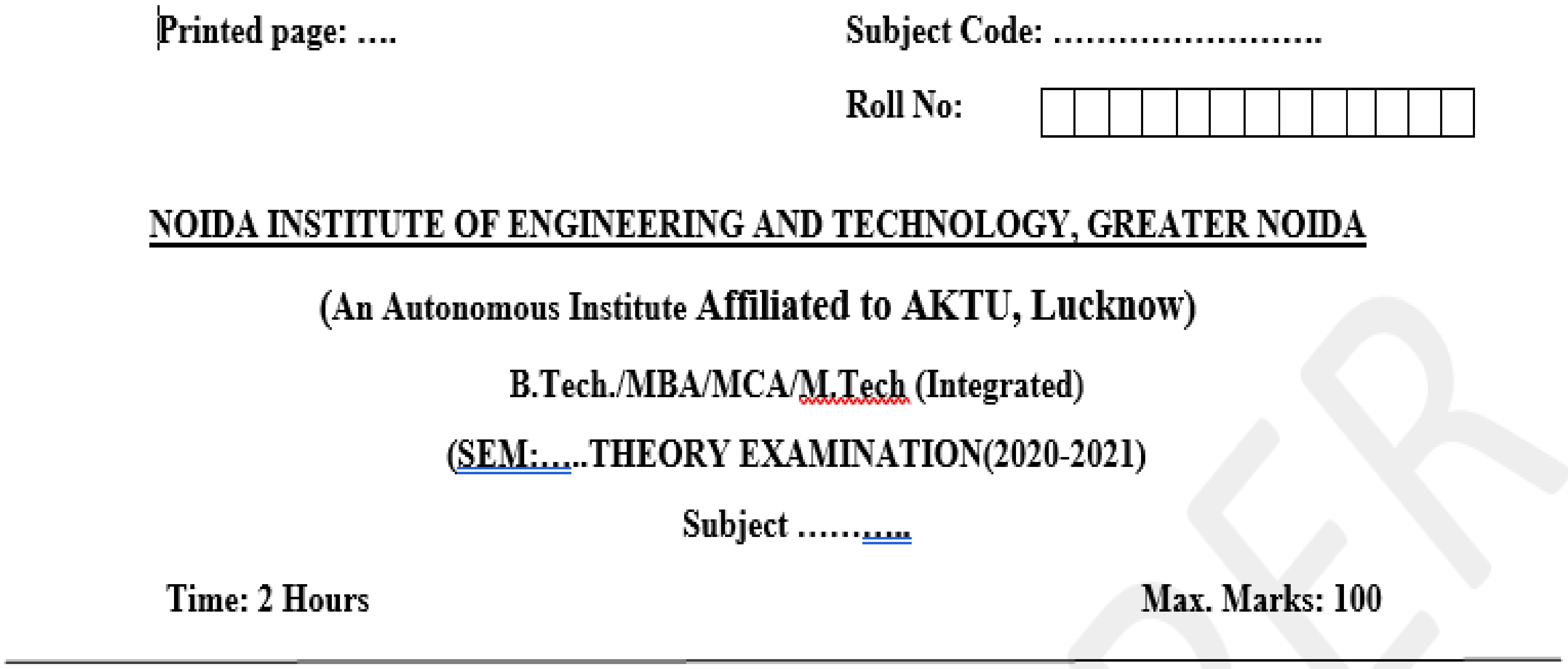
Program Educational Objectives (PEOs)

Program Educational Objectives (PEOs)	PEOs Description
PEOs	To have an excellent scientific and engineering breadth so as to comprehend, analyze, design and provide sustainable solutions for real-life problems using state-of-the-art technologies.
PEOs	To have a successful career in industries, to pursue higher studies or to support entrepreneurial endeavors and to face the global challenges.
PEOs	To have an effective communication skills, professional attitude, ethical values and a desire to learn specific knowledge in emerging trends, technologies for research, innovation and product development and contribution to society.
PEOs	To have life-long learning for up-skilling and re-skilling for successful professional career as engineer, scientist, entrepreneur and bureaucrat for betterment of society.

Result Analysis(Department Result & Subject Result & Individual result

Name of the faculty	Subject code	Result % of clear passed
Mr. Sanjay Nayak		

Pattern of Online External Exam Question Paper (100 marks)



Printed page: ....

Subject Code: ....

Roll No:

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA

(An Autonomous Institute Affiliated to AKTU, Lucknow)

B.Tech./MBA/MCA/M.Tech (Integrated)

(SEM:.....THEORY EXAMINATION(2020-2021)

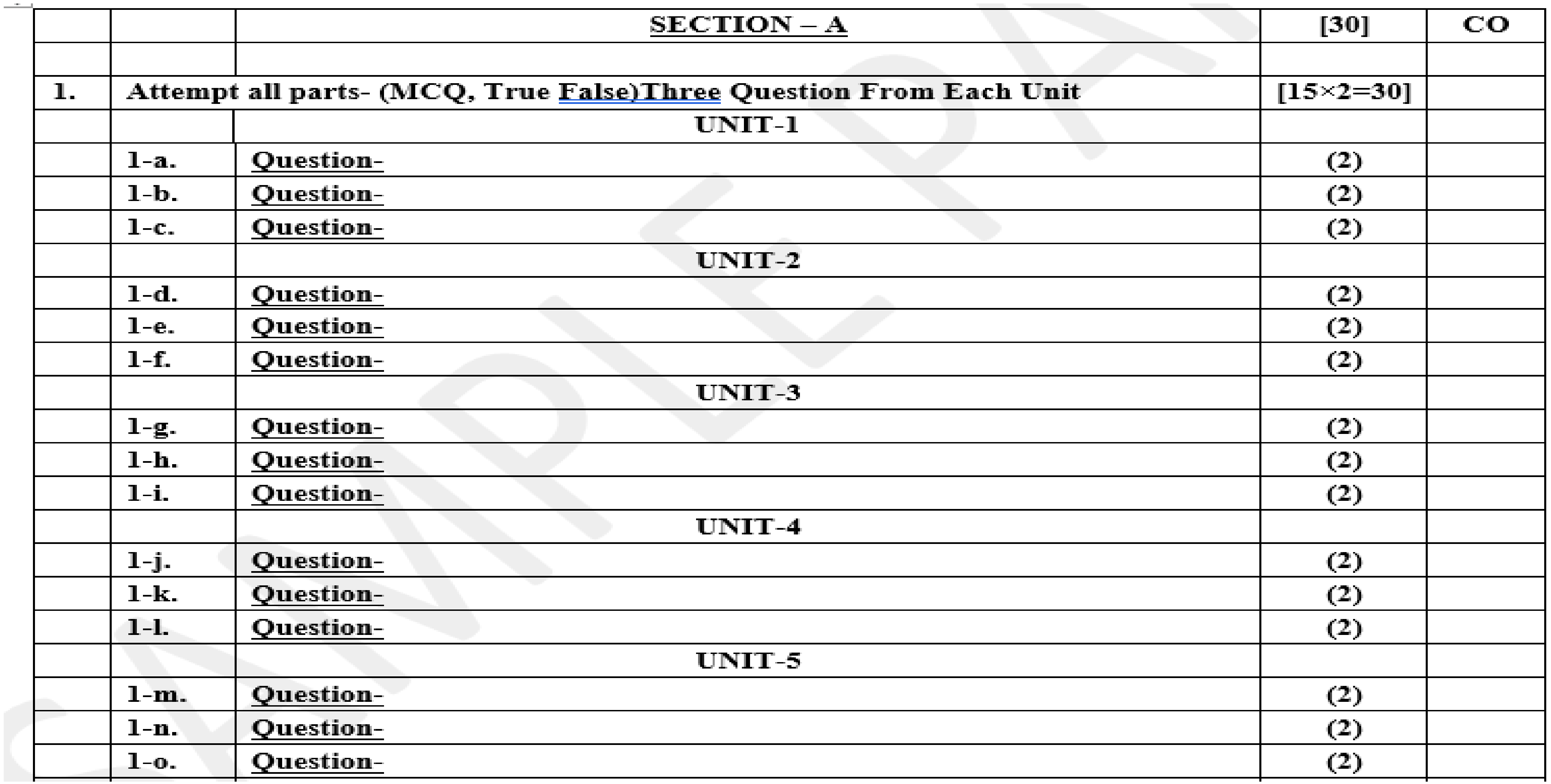
Subject ....

Time: 2 Hours

Max. Marks: 100

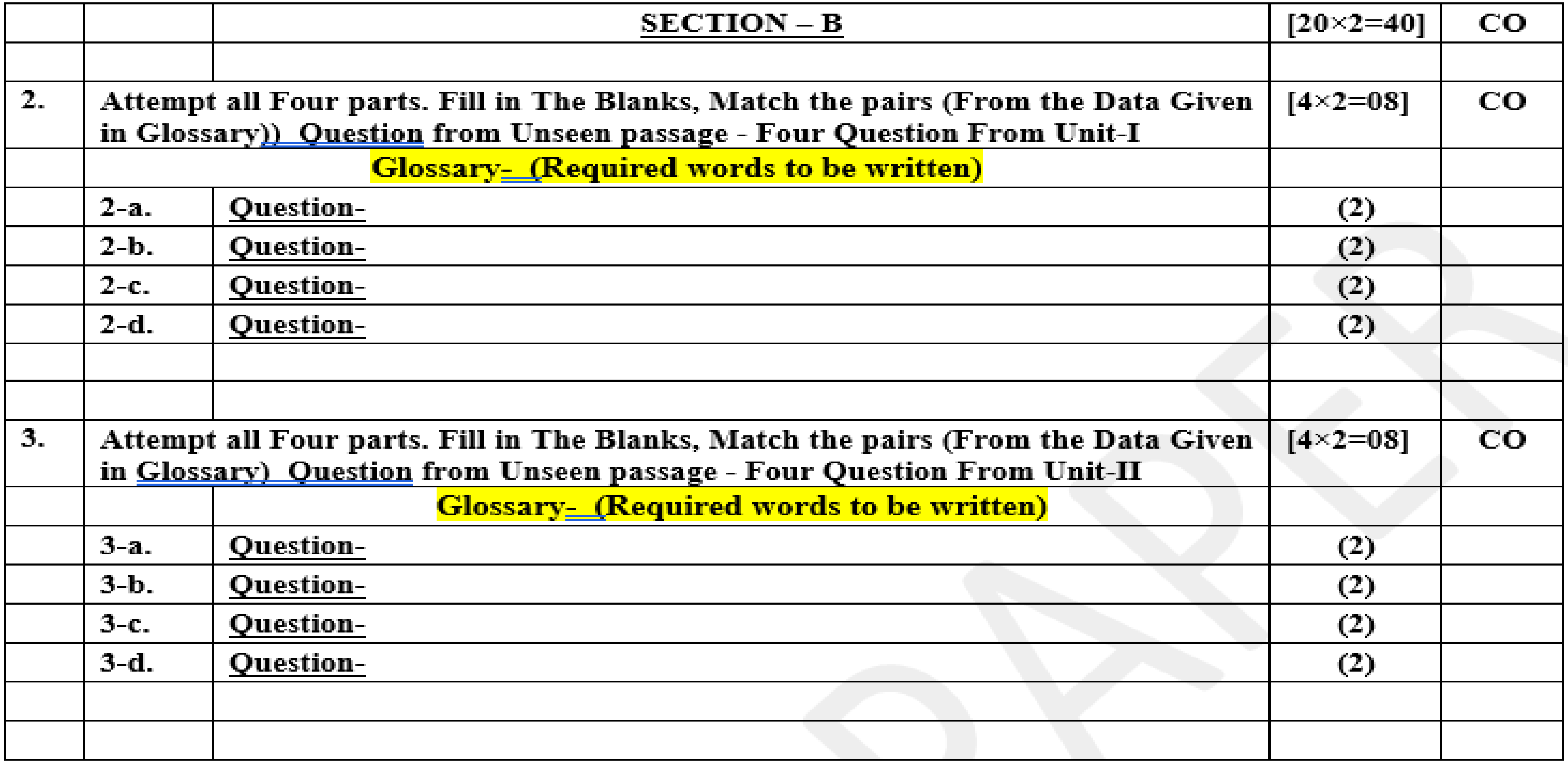
NIET Greater Noida
GET FUTURE READY
Estd 2001
AN AUTONOMOUS INSTITUTE

Pattern of Online External Exam Question Paper (100 marks)



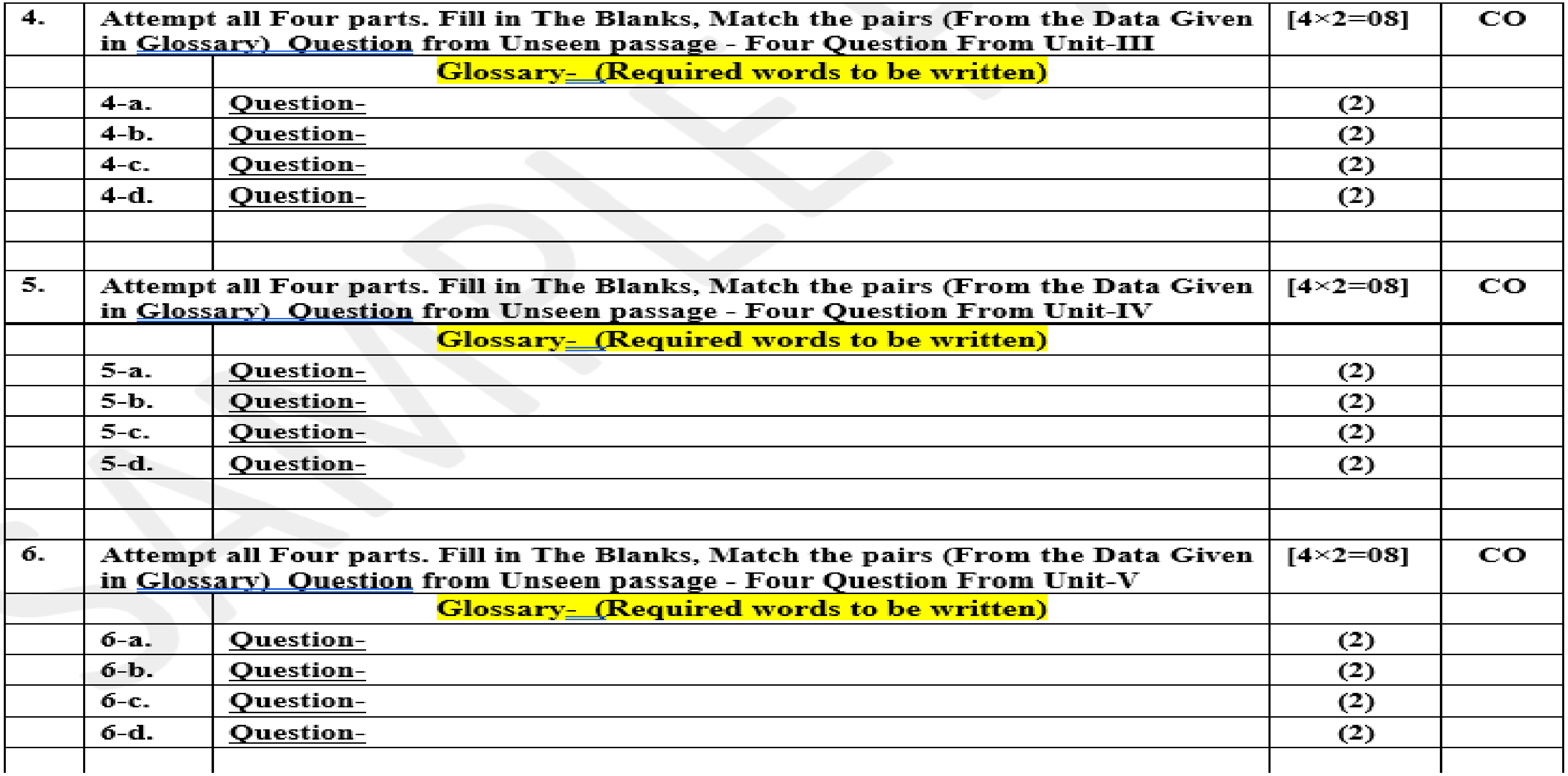
		SECTION – A	[30]	CO
1.	Attempt all parts- (MCQ, True False)Three Question From Each Unit		[15×2=30]	
		UNIT-1		
	1-a.	Question-	(2)	
	1-b.	Question-	(2)	
	1-c.	Question-	(2)	
		UNIT-2		
	1-d.	Question-	(2)	
	1-e.	Question-	(2)	
	1-f.	Question-	(2)	
		UNIT-3		
	1-g.	Question-	(2)	
	1-h.	Question-	(2)	
	1-i.	Question-	(2)	
		UNIT-4		
	1-j.	Question-	(2)	
	1-k.	Question-	(2)	
	1-l.	Question-	(2)	
		UNIT-5		
	1-m.	Question-	(2)	
	1-n.	Question-	(2)	
	1-o.	Question-	(2)	

Pattern of Online External Exam Question Paper (100 marks)



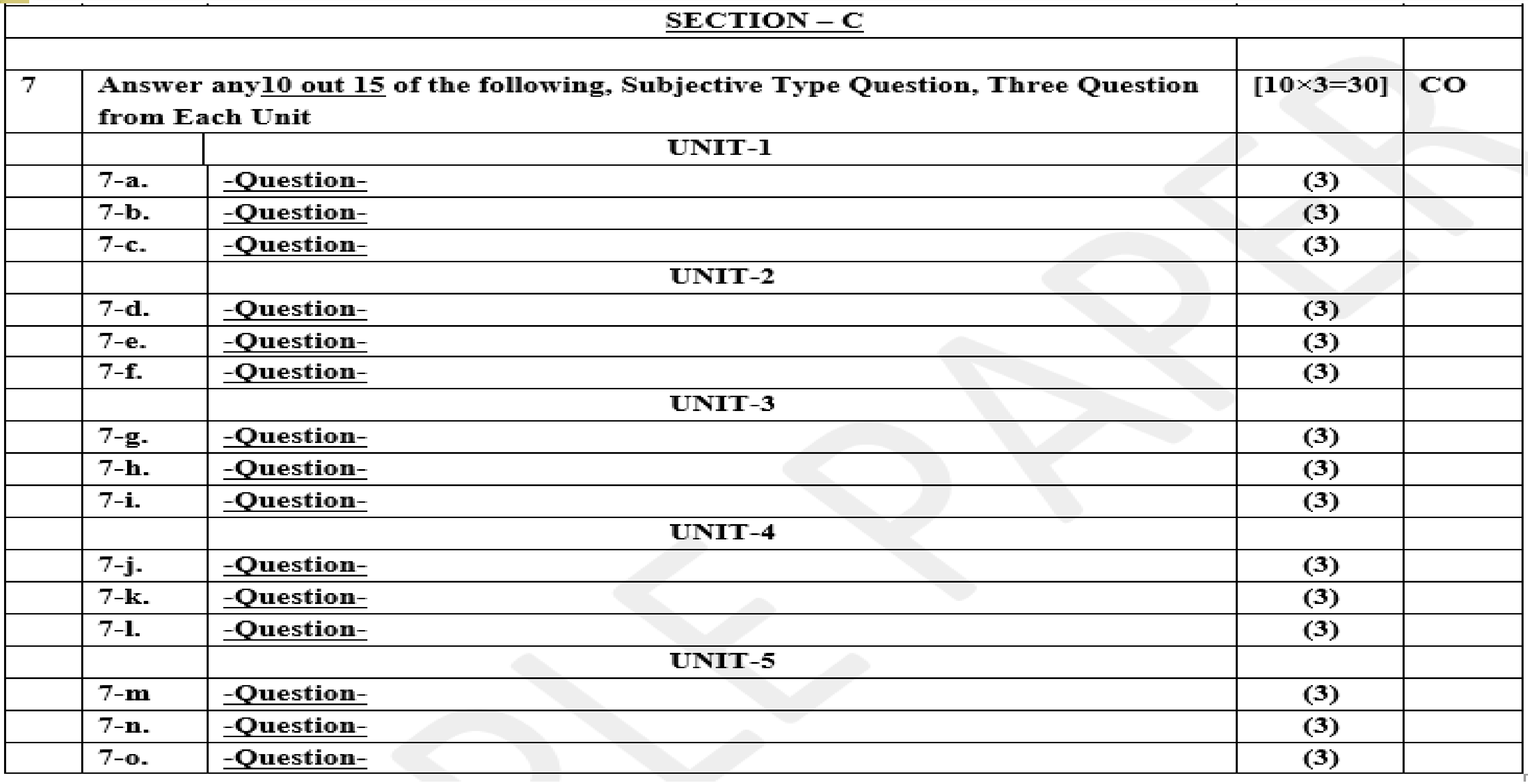
		SECTION – B	[20×2=40]	CO
2.	Attempt all Four parts. Fill in The Blanks, Match the pairs (From the Data Given in Glossary)) Question from Unseen passage - Four Question From Unit-I		[4×2=08]	CO
	Glossary- (Required words to be written)			
	2-a.	Question-	(2)	
	2-b.	Question-	(2)	
	2-c.	Question-	(2)	
	2-d.	Question-	(2)	
3.	Attempt all Four parts. Fill in The Blanks, Match the pairs (From the Data Given in Glossary) Question from Unseen passage - Four Question From Unit-II		[4×2=08]	CO
		Glossary- (Required words to be written)		
	3-a.	Question-	(2)	
	3-b.	Question-	(2)	
	3-c.	Question-	(2)	
	3-d.	Question-	(2)	

Pattern of Online External Exam Question Paper (100 marks)



4.	Attempt all Four parts. Fill in The Blanks, Match the pairs (From the Data Given in Glossary) Question from Unseen passage - Four Question From Unit-III		[4×2=08]	CO
		Glossary- (Required words to be written)		
	4-a.	Question-	(2)	
	4-b.	Question-	(2)	
	4-c.	Question-	(2)	
	4-d.	Question-	(2)	
5.	Attempt all Four parts. Fill in The Blanks, Match the pairs (From the Data Given in Glossary) Question from Unseen passage - Four Question From Unit-IV		[4×2=08]	CO
		Glossary- (Required words to be written)		
	5-a.	Question-	(2)	
	5-b.	Question-	(2)	
	5-c.	Question-	(2)	
	5-d.	Question-	(2)	
6.	Attempt all Four parts. Fill in The Blanks, Match the pairs (From the Data Given in Glossary) Question from Unseen passage - Four Question From Unit-V		[4×2=08]	CO
		Glossary- (Required words to be written)		
	6-a.	Question-	(2)	
	6-b.	Question-	(2)	
	6-c.	Question-	(2)	
	6-d.	Question-	(2)	

Pattern of Online External Exam Question Paper (100 marks)



SECTION – C				
7	Answer any 10 out 15 of the following, Subjective Type Question, Three Question from Each Unit		[10×3=30]	CO
		UNIT-1		
	7-a.	-Question-	(3)	
	7-b.	-Question-	(3)	
	7-c.	-Question-	(3)	
		UNIT-2		
	7-d.	-Question-	(3)	
	7-e.	-Question-	(3)	
	7-f.	-Question-	(3)	
		UNIT-3		
	7-g.	-Question-	(3)	
	7-h.	-Question-	(3)	
	7-i.	-Question-	(3)	
		UNIT-4		
	7-j.	-Question-	(3)	
	7-k.	-Question-	(3)	
	7-l.	-Question-	(3)	
		UNIT-5		
	7-m	-Question-	(3)	
	7-n.	-Question-	(3)	
	7-o.	-Question-	(3)	

Prerequisite / Recap

- Student should have knowledge of object oriented analysis and design.
- Knowledge of Data structure and algorithm.
- knowledge of Programing language such as C/C++ etc.
- Good problem solving Skill .

Brief Introduction about the Subject with videos

YouTube /other Video Links

- https://youtu.be/rI4kdGLaUiQ?list=PL6n9fhu94yhUbctIoxoVTrklN3LMwTCmd
- https://youtu.be/v9ejT8FO-7I?list=PLrhzvIcii6GNjpARdnO4ueTUAVR9eMBpc
- https://youtu.be/VGLjQuEQgkI?list=PLt4nG7RVVk1h9lxOYSOGI9pcP3I5oblbx

Unit V Content

- Behavioral Design Patterns Part-I :
- State Pattern.
- Strategy Pattern.
- Template Pattern
- Visitor Pattern

Unit V Objective

In Unit V, the students will be able to find

- Definitions of terms and concepts.
- The idea of a pattern.
- The origins of all design patterns.
- How Patterns Work in software design.
- Scope of development activity: applications, toolkits, frameworks.
- All Behavioral Pattern and their need.

Topic Objective

Topic : State Pattern.

- In this topic, the students will gain , The idea of a Behavioral design pattern, In these design patterns A State Pattern says that "the class behavior changes based on its state". In State Pattern, we create objects which represent various states and a context object whose behavior varies as its state object changes.

State Pattern

State Pattern:-

- A State Pattern says that "the class behavior changes based on its state". In State Pattern, we create objects which represent various states and a context object whose behavior varies as its state object changes.

- The State Pattern is also known as Objects for States. It keeps the state-specific behaviour. It makes any state transitions explicit.

- When the behavior of object depends on its state and it must be able to change its behavior at runtime according to the new state.

- It is used when the operations have large, multipart conditional statements that depend on the state of an object.

Implementation of (State Pattern)

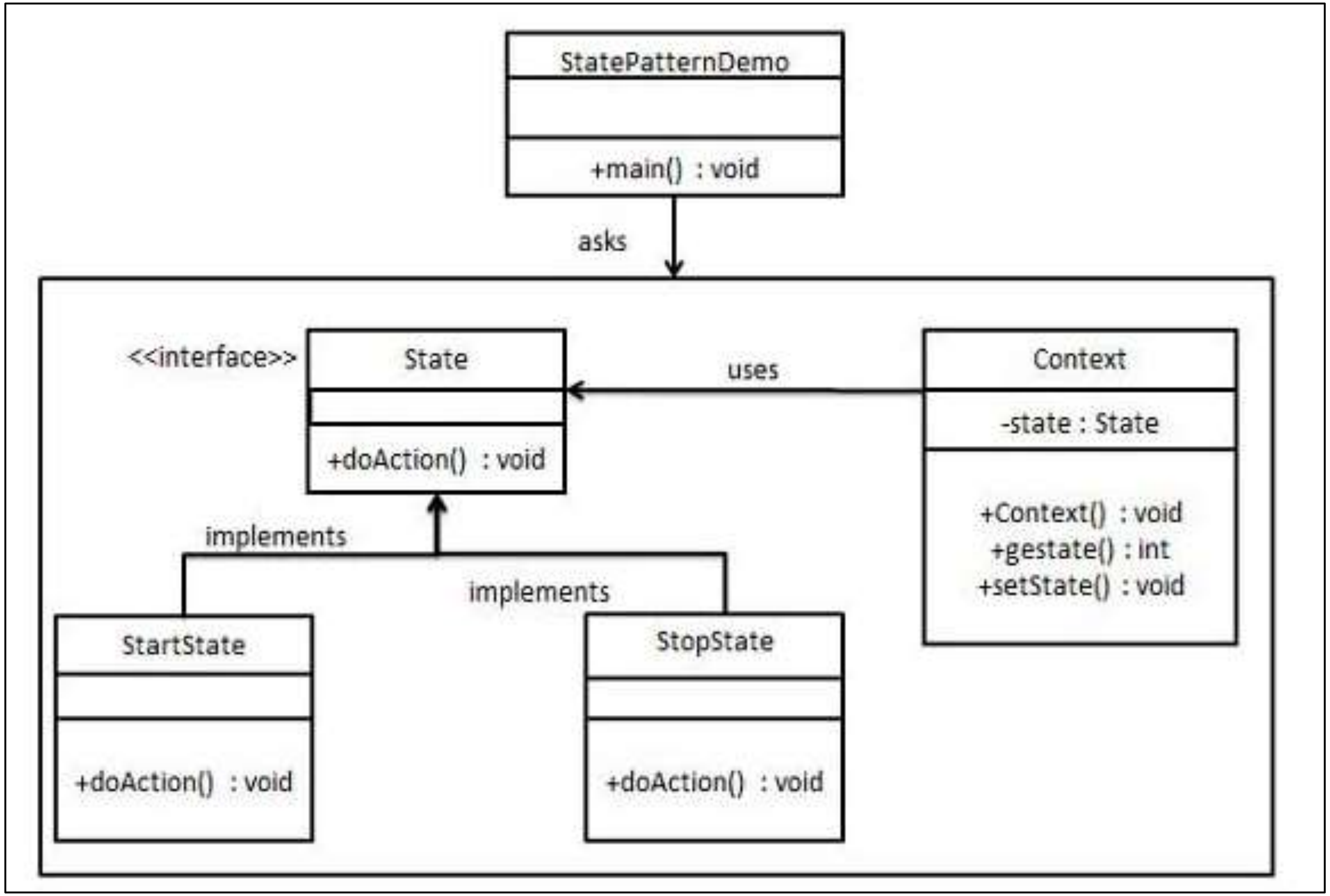
- We are going to create a State interface defining an action and concrete state classes implementing the State interface. Context is a class which carries a State.

- StatePatternDemo, our demo class, will use Context and state objects to demonstrate change in Context behavior based on type of state it is in.

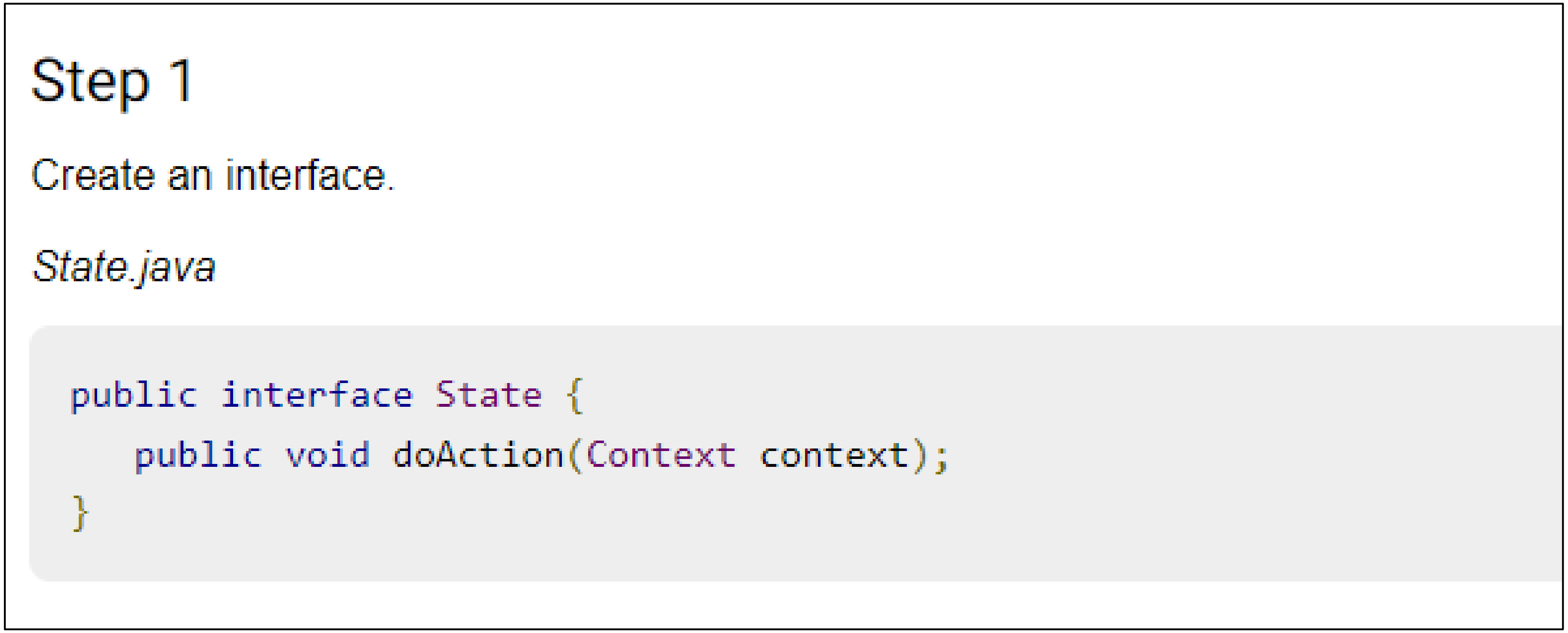
- In State pattern a class behavior changes based on its state. This type of design pattern comes under behavior pattern.

- In State pattern, we create objects which represent various states and a context object whose behavior varies as its state object changes.

UML\Structure for State Pattern



Implementation of (State Pattern)



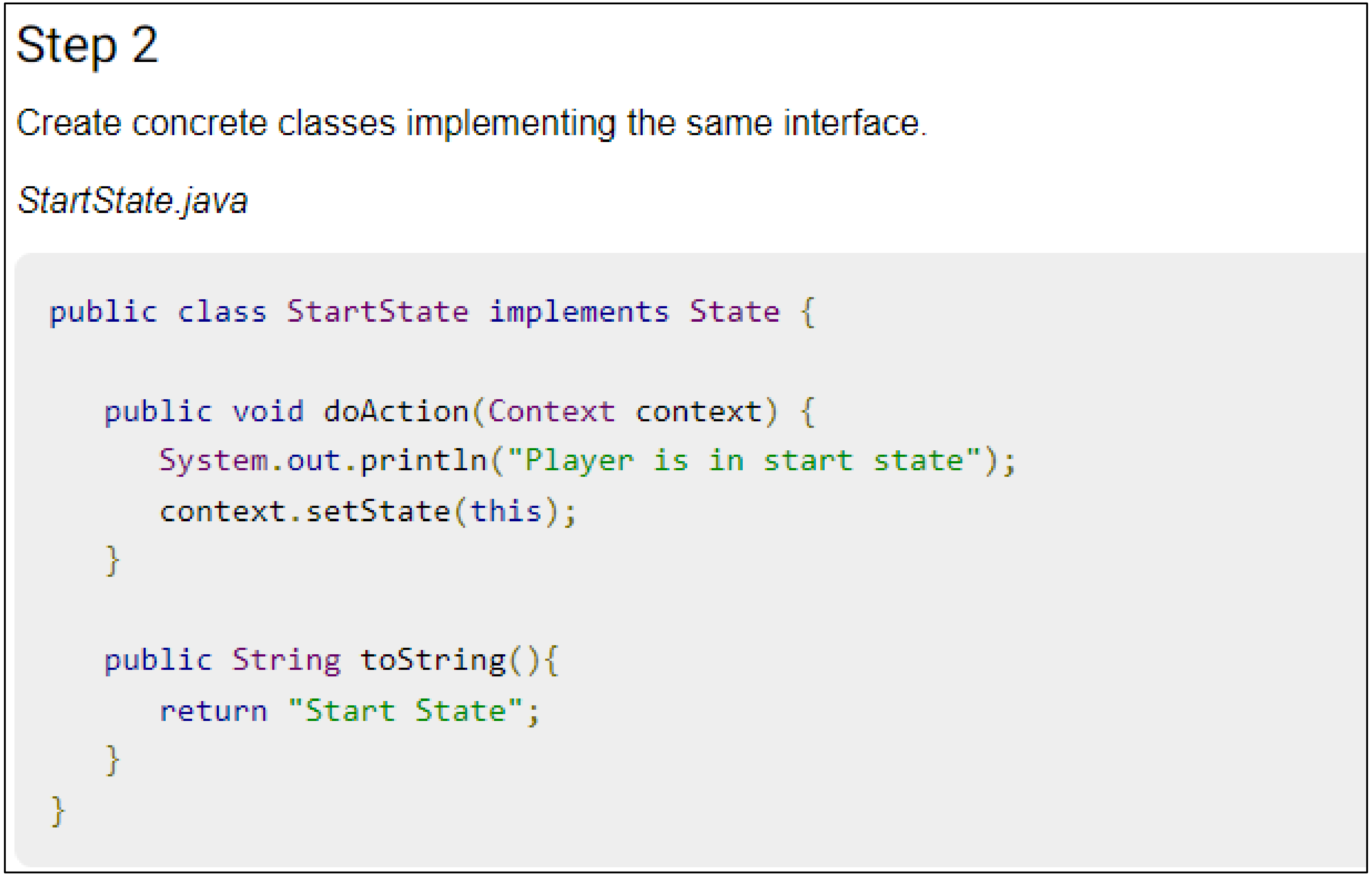
Step 1

Create an interface.

State.java

```
public interface State {
   public void doAction(Context context);
}
```

Implementation of (State Pattern)



Step 2

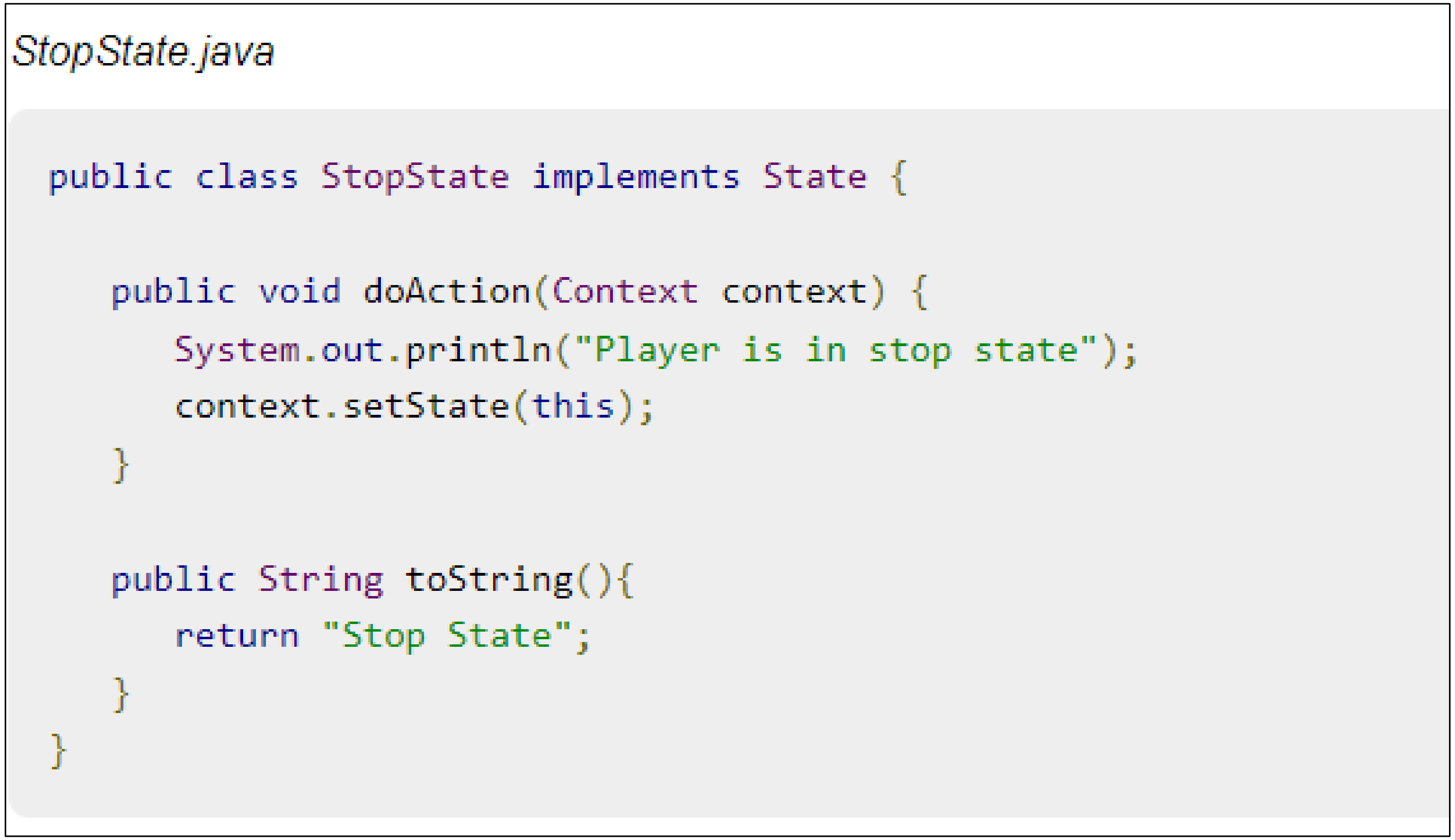
Create concrete classes implementing the same interface.

StartState.java

```
public class StartState implements State {

   public void doAction(Context context) {
      System.out.println("Player is in start state");
      context.setState(this);
   }

   public String toString(){
      return "Start State";
   }
}
```

Implementation of (State Pattern)

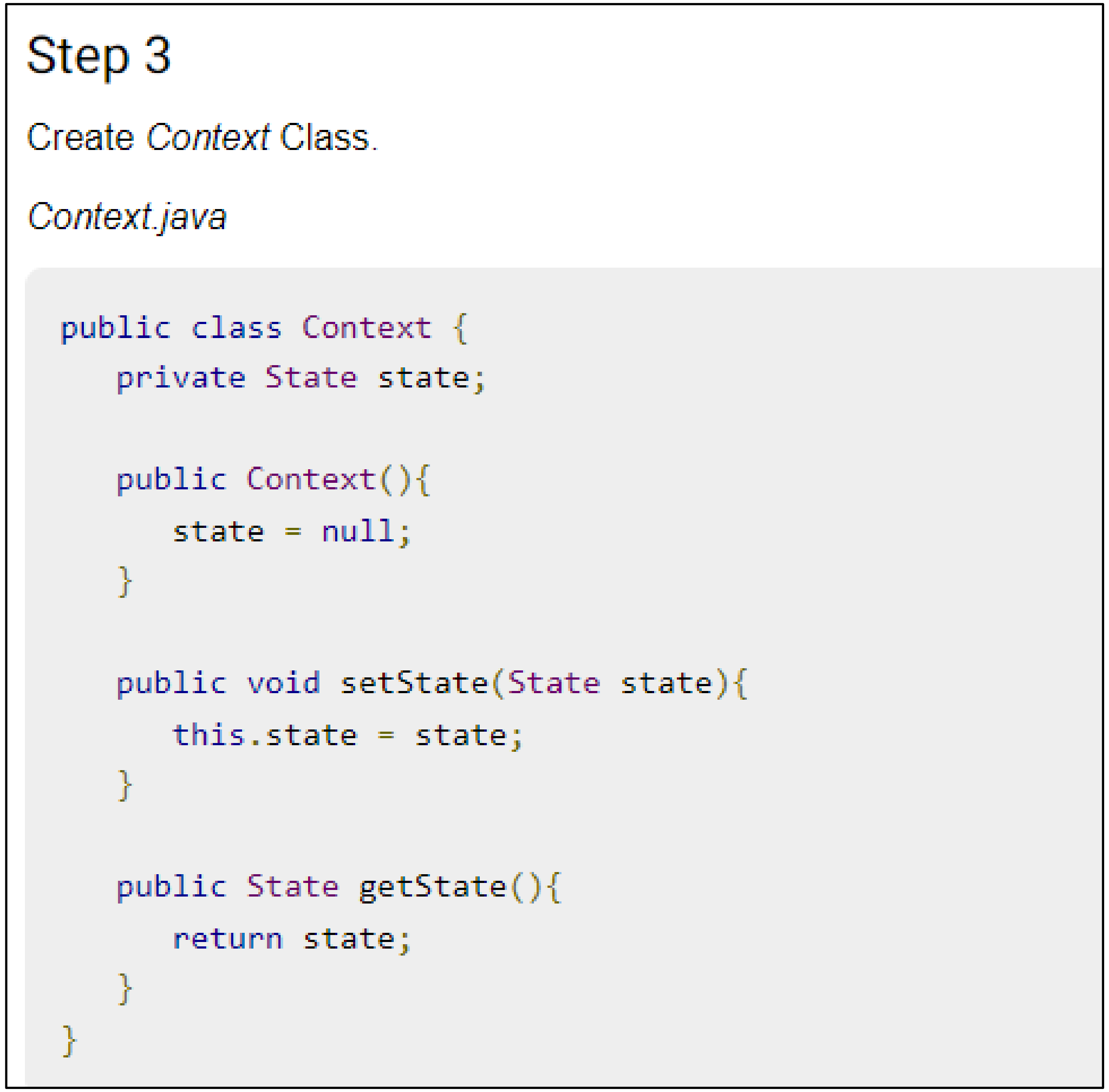
Step -2 Cont.........



StopState.java

```
public class StopState implements State {

   public void doAction(Context context) {
      System.out.println("Player is in stop state");
      context.setState(this);
   }

   public String toString(){
      return "Stop State";
   }
}
```

Implementation of (State Pattern)



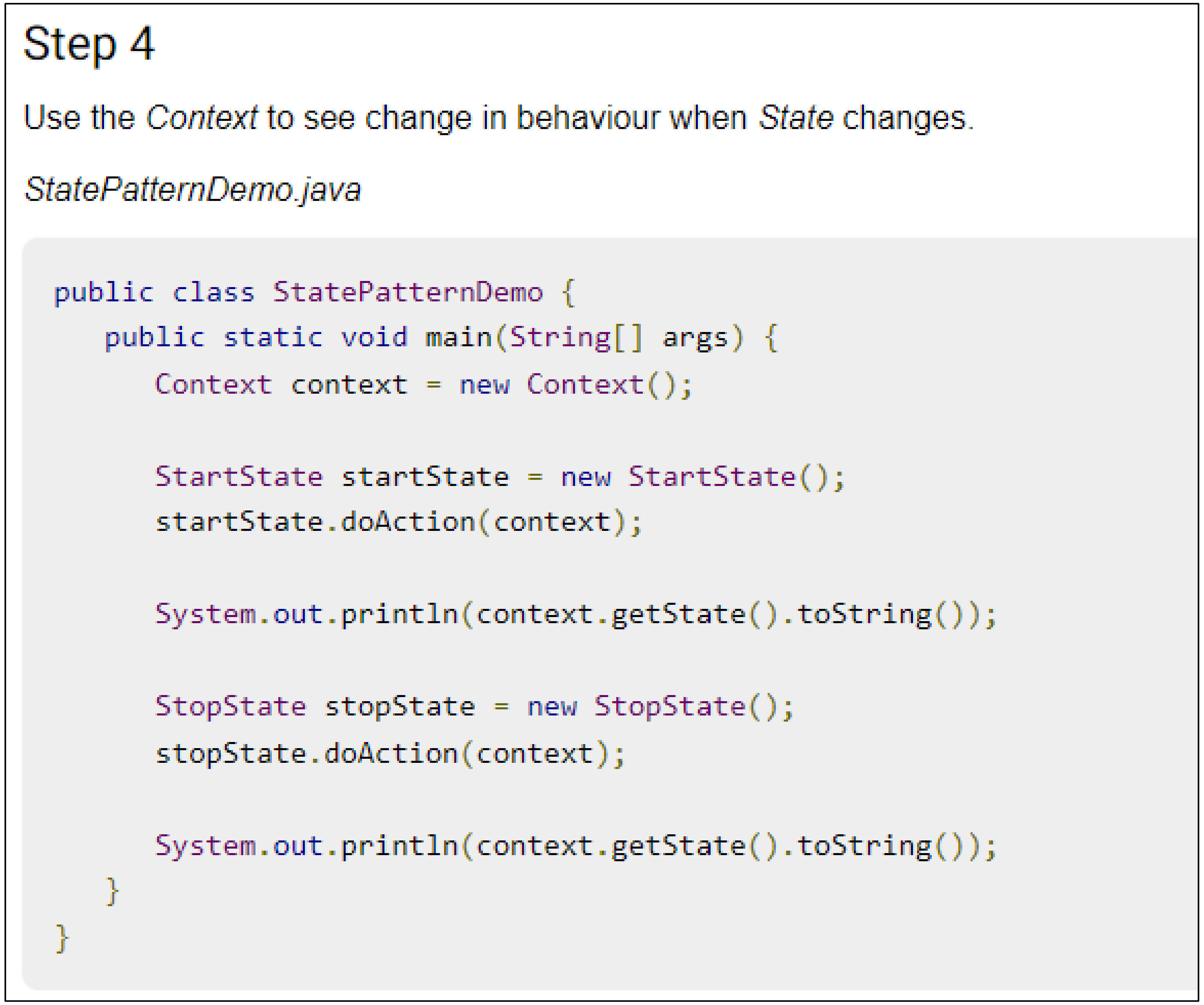
Step 3

Create *Context* Class.

Context.java

```
public class Context {
   private State state;

   public Context(){
      state = null;
   }

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

   public State getState(){
      return state;
   }
}
```

Implementation of (State Pattern)



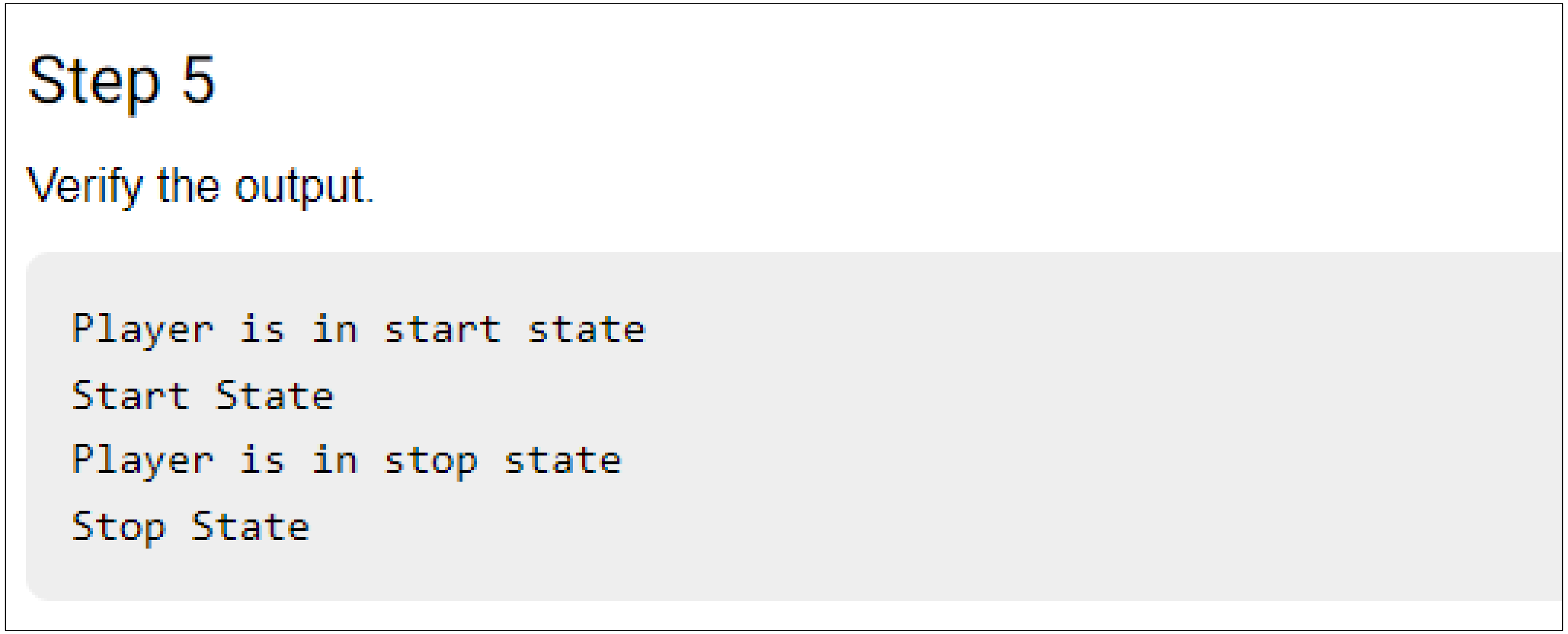
Step 4

Use the *Context* to see change in behaviour when *State* changes.

StatePatternDemo.java

```java
public class StatePatternDemo {
   public static void main(String[] args) {
      Context context = new Context();

      StartState startState = new StartState();
      startState.doAction(context);

      System.out.println(context.getState().toString());

      StopState stopState = new StopState();
      stopState.doAction(context);

      System.out.println(context.getState().toString());
   }
}
```

Implementation of (State Pattern)



Step 5

Verify the output.

```
Player is in start state
Start State
Player is in stop state
Stop State
```

Topic Objective

Topic : Strategy Pattern.

- In this topic, the students will gain , The idea of a Behavioral design pattern, In these design patterns i.e Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern.

Strategy Pattern

Strategy Pattern:-

- In Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.
- In Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern.
- A Strategy Pattern says that "defines a family of functionality, encapsulate each one, and make them interchangeable". The Strategy Pattern is also known as Policy.
- It provides a substitute to sub classing.

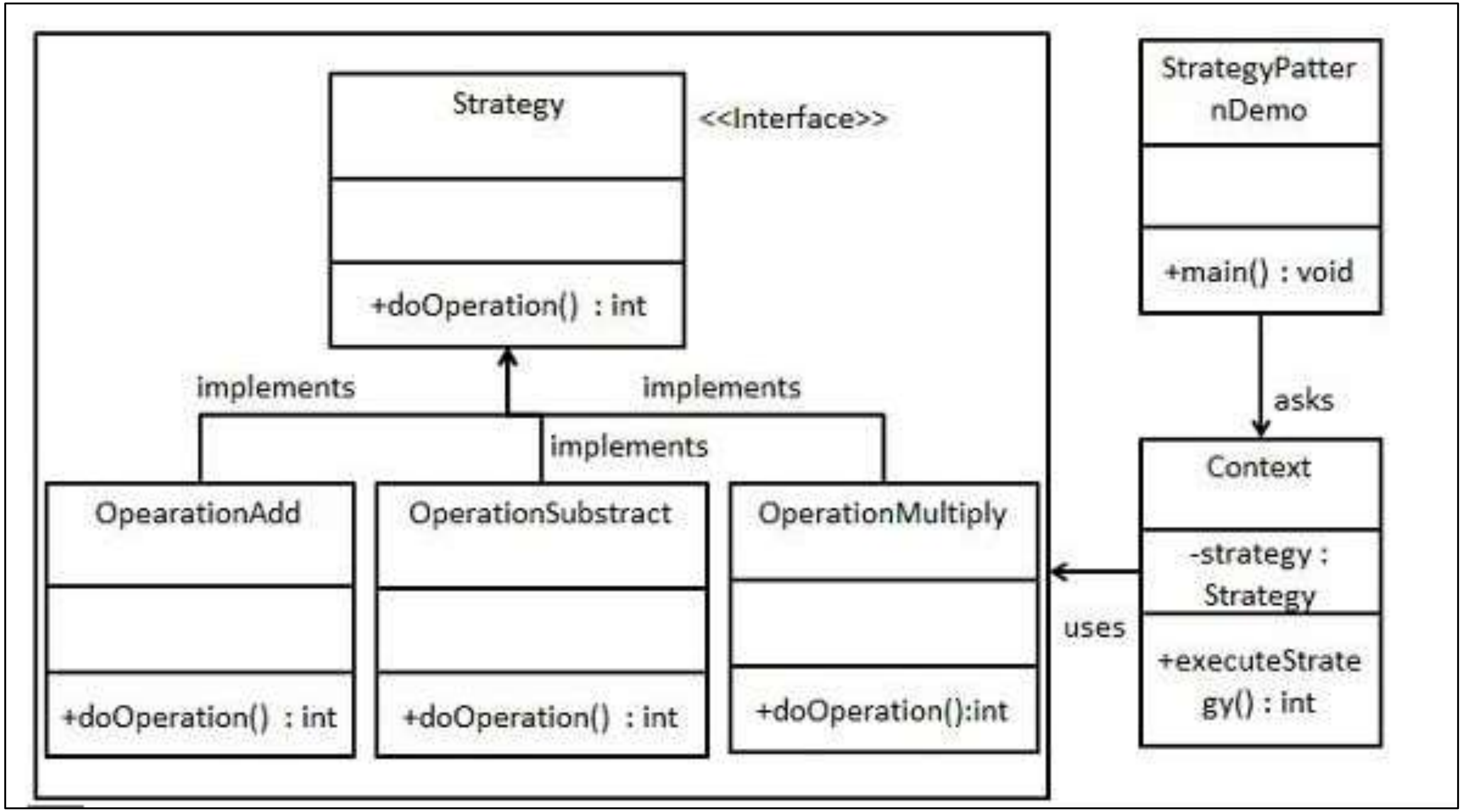
Implementation of (Strategy Pattern)

- We are going to create a Strategy interface defining an action and concrete strategy classes implementing the Strategy interface. Context is a class which uses a Strategy.

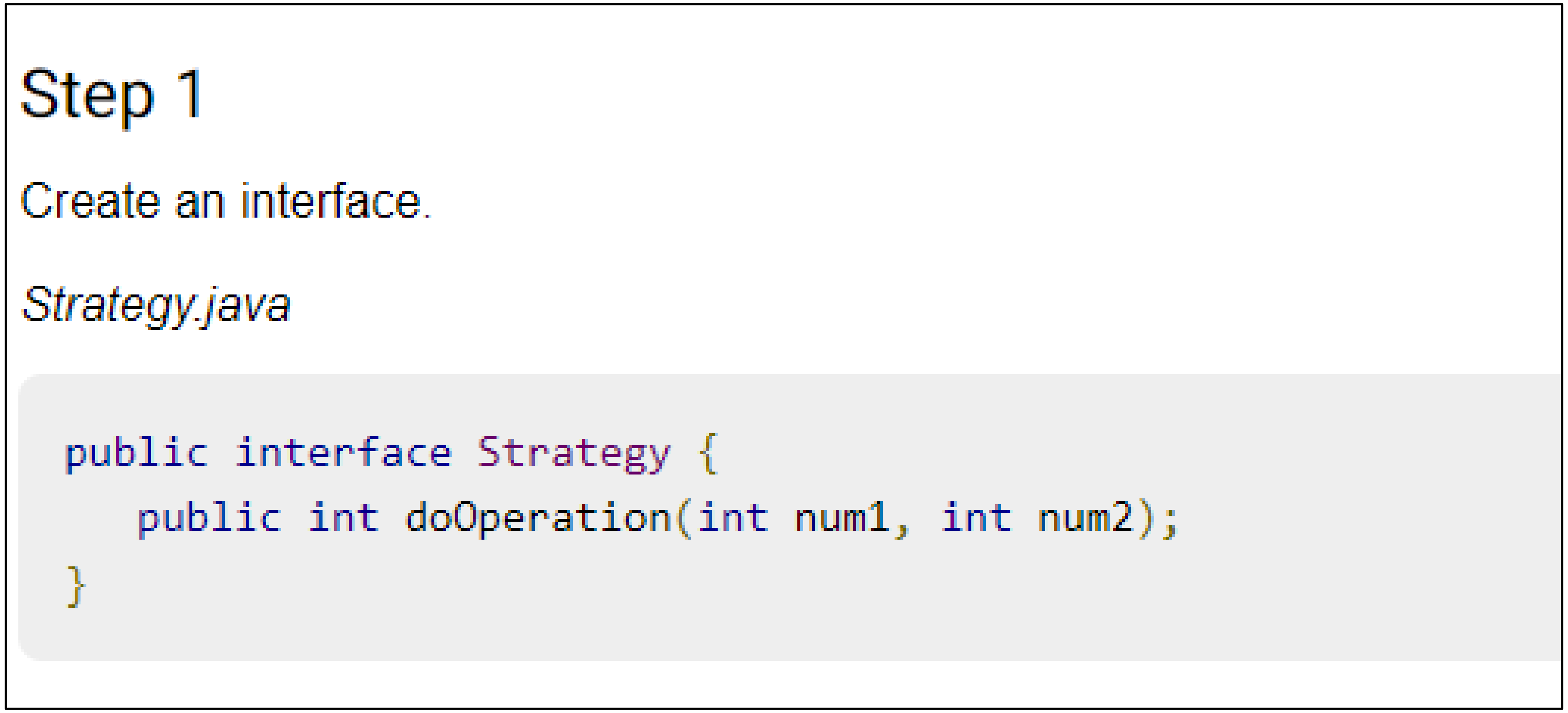
- StrategyPatternDemo, our demo class, will use Context and strategy objects to demonstrate change in Context behaviour based on strategy it deploys or uses.
- When the multiple classes differ only in their behaviors. e.g. Servlet API.It is used when you need different variations of an algorithm.

- It makes it easier to extend and incorporate new behavior without changing the application.

UML\Structure for Strategy Pattern



Implementation of (Strategy Pattern)



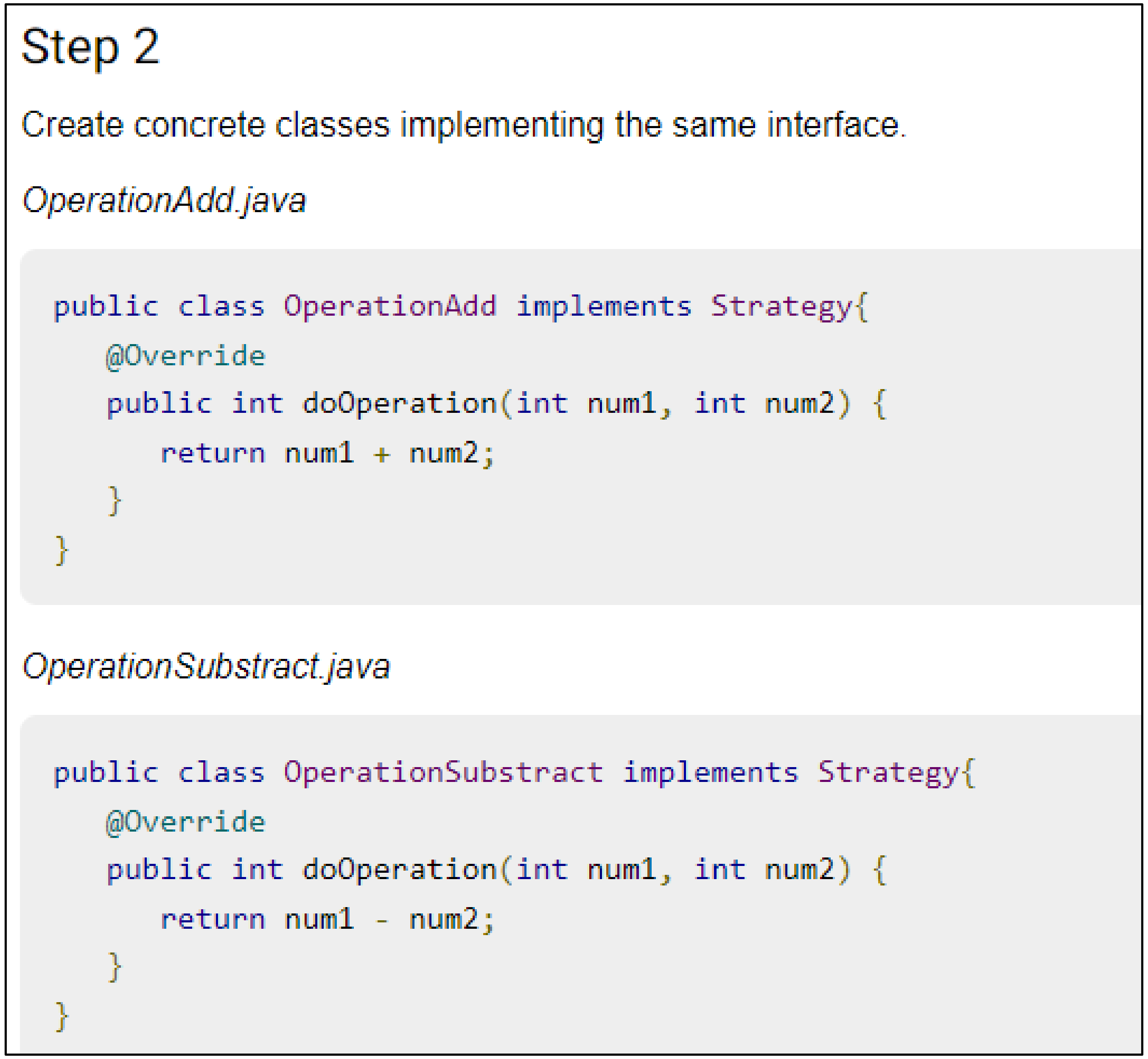
Step 1

Create an interface.

Strategy.java

```
public interface Strategy {
   public int doOperation(int num1, int num2);
}
```

Implementation of (Strategy Pattern)



Step 2

Create concrete classes implementing the same interface.

OperationAdd.java

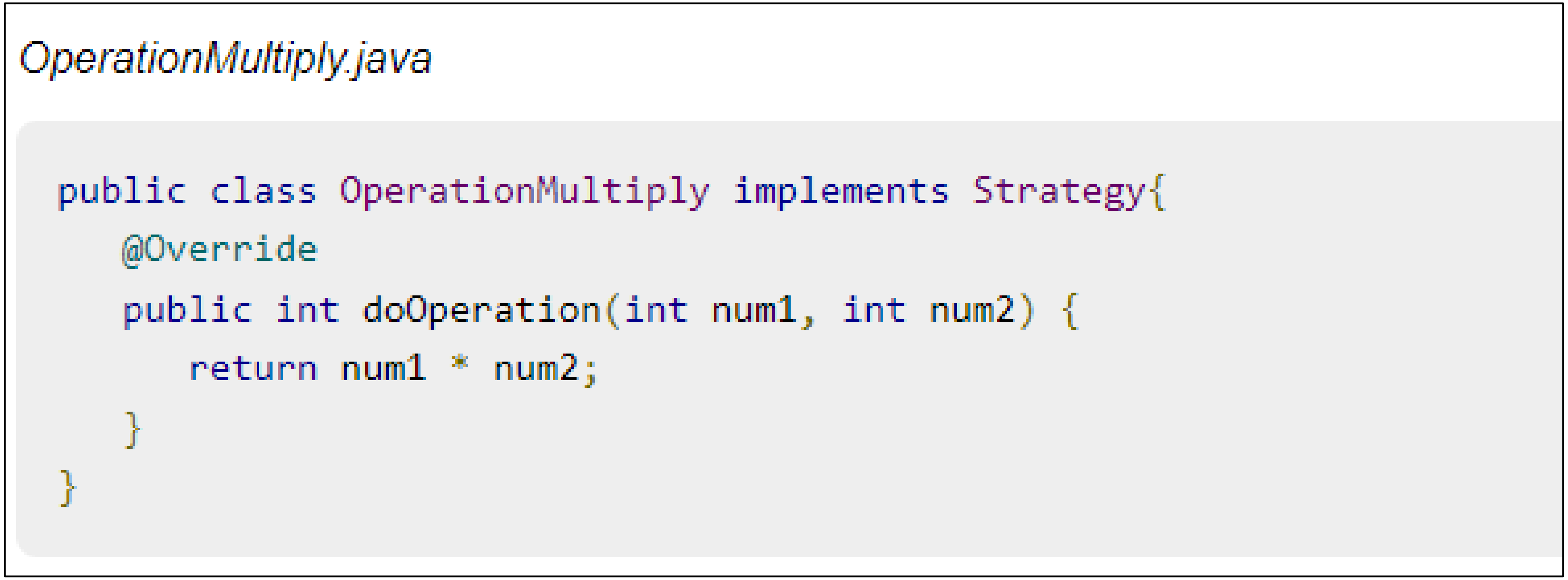
```
public class OperationAdd implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 + num2;
   }
}
```

OperationSubstract.java

```
public class OperationSubstract implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 - num2;
   }
}
```

Implementation of (Strategy Pattern)

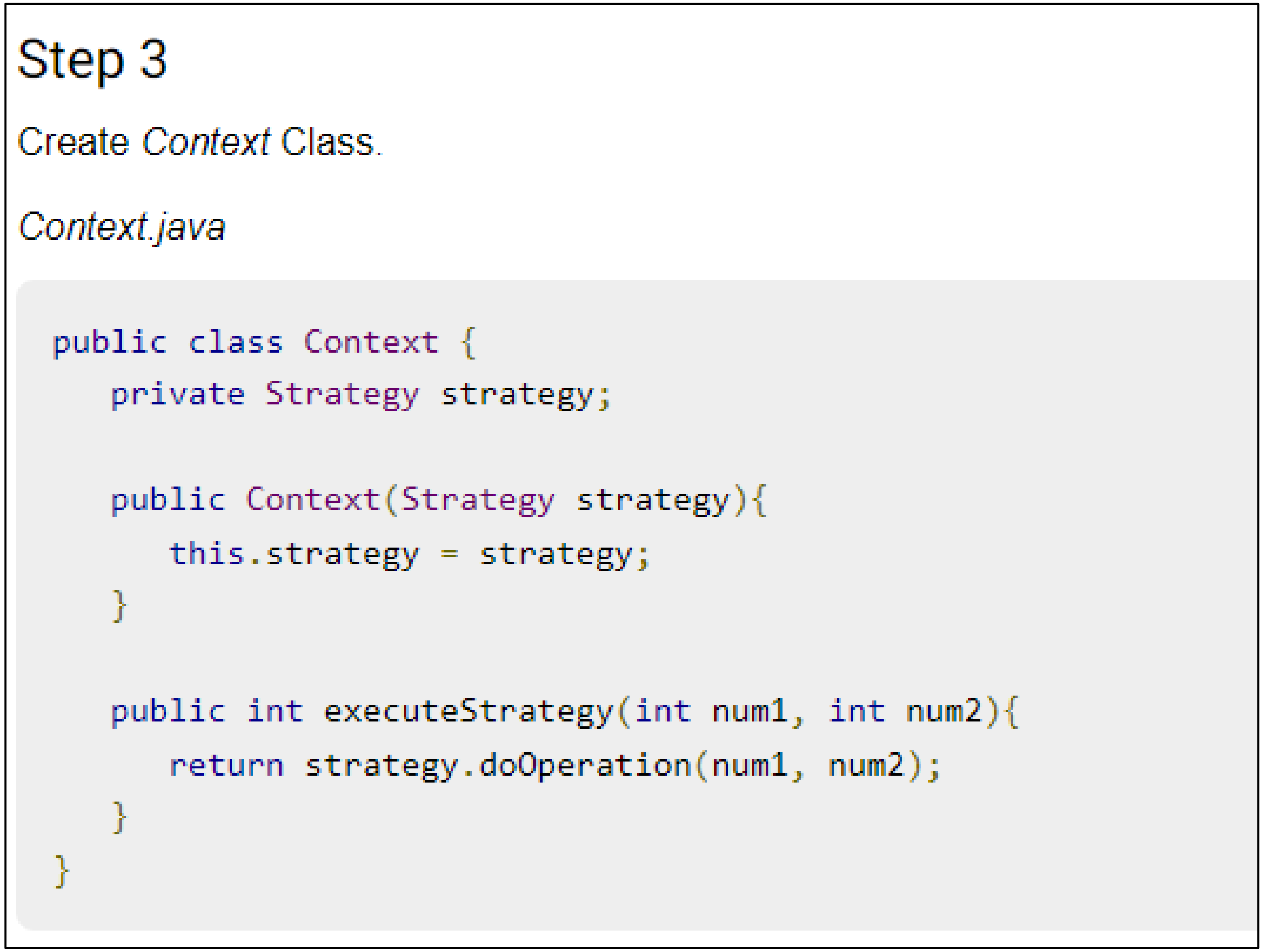
Step -2 Cont.........



OperationMultiply.java

```
public class OperationMultiply implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 * num2;
   }
}
```

Implementation of (Strategy Pattern)



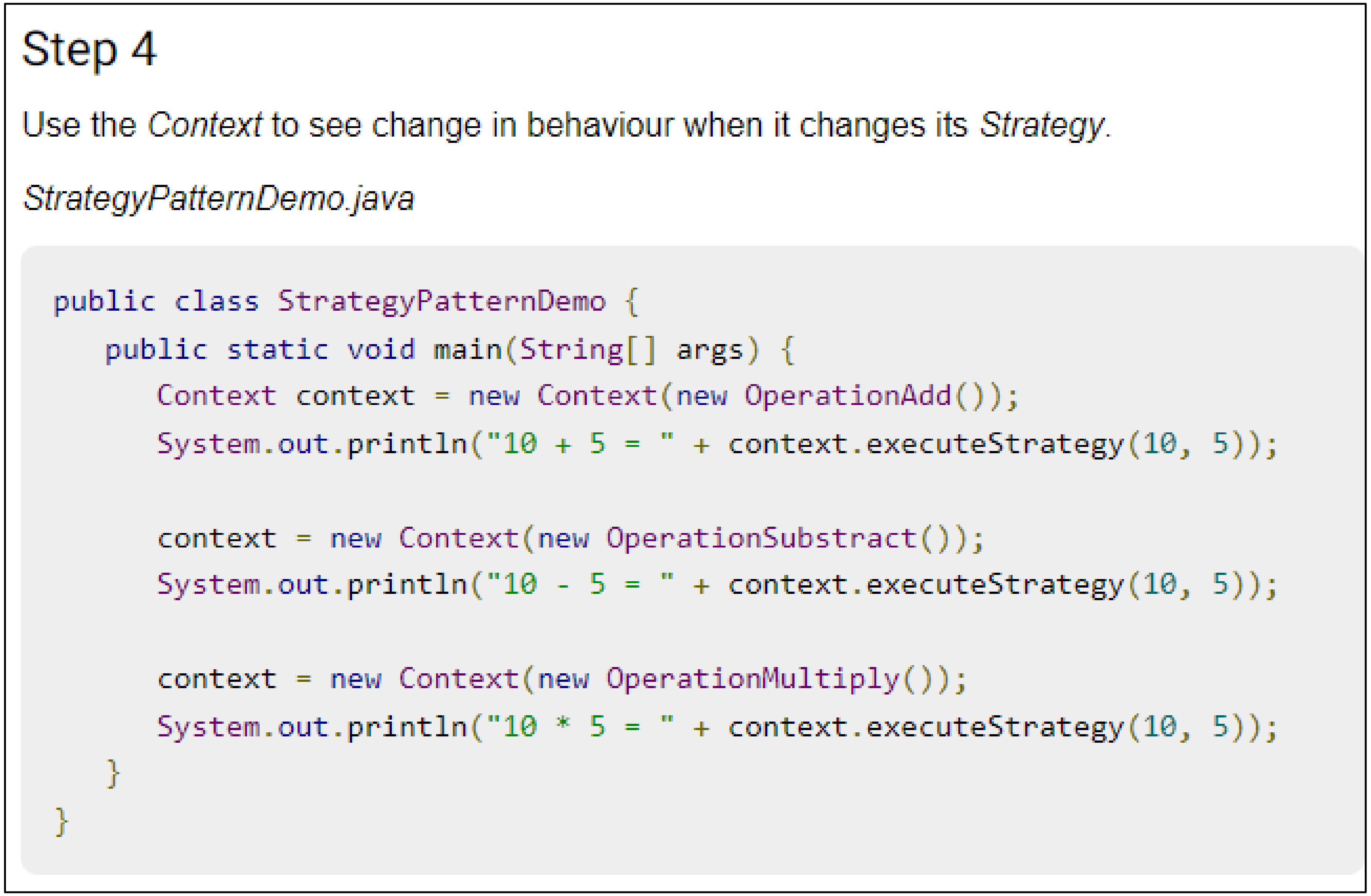
Step 3

Create *Context* Class.

Context.java

```java
public class Context {
   private Strategy strategy;

   public Context(Strategy strategy){
      this.strategy = strategy;
   }

   public int executeStrategy(int num1, int num2){
      return strategy.doOperation(num1, num2);
   }
}
```

Implementation of (Strategy Pattern)



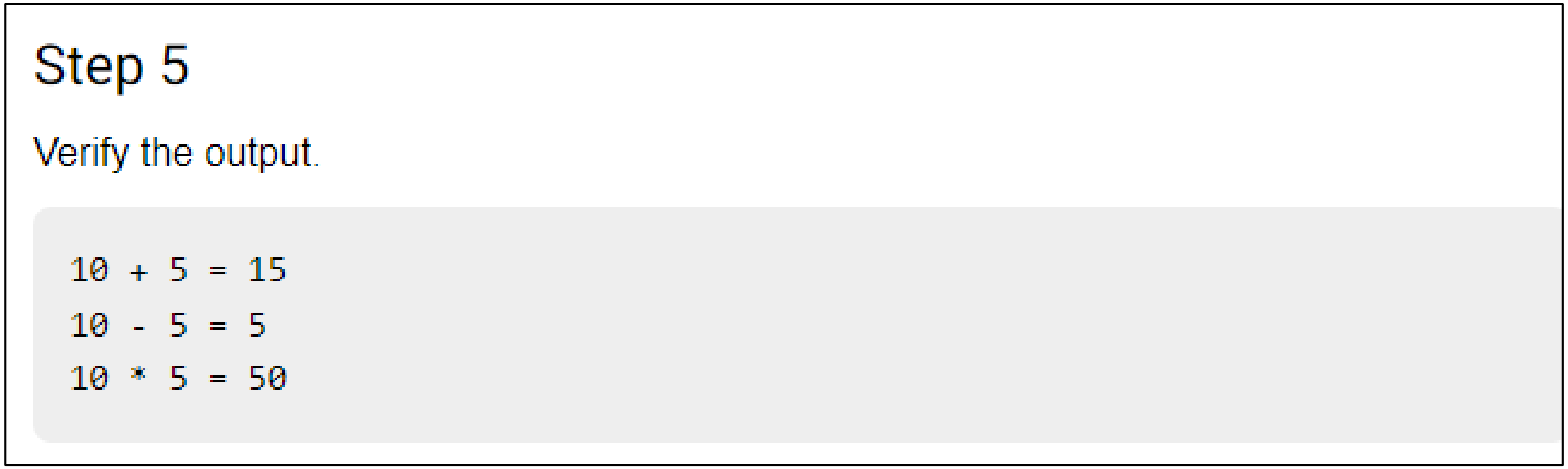
Step 4

Use the *Context* to see change in behaviour when it changes its *Strategy*.

StrategyPatternDemo.java

```
public class StrategyPatternDemo {
   public static void main(String[] args) {
      Context context = new Context(new OperationAdd());
      System.out.println("10 + 5 = " + context.executeStrategy(10, 5));

      context = new Context(new OperationSubstract());
      System.out.println("10 - 5 = " + context.executeStrategy(10, 5));

      context = new Context(new OperationMultiply());
      System.out.println("10 * 5 = " + context.executeStrategy(10, 5));
   }
}
```

Implementation of (Strategy Pattern)


Step 5

Verify the output.

```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
```

Topic Objective

Topic : Template Pattern.

- In this topic, the students will gain , The idea of a Behavioral design pattern, In these design patterns i.e In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods

Template Pattern

<u>Template Pattern:-</u>

- 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.

- A Template Pattern says that "just define the skeleton of a function in an operation, deferring some steps to its subclasses".

- It is used when the common behavior among sub-classes should be moved to a single common class by avoiding the duplication.

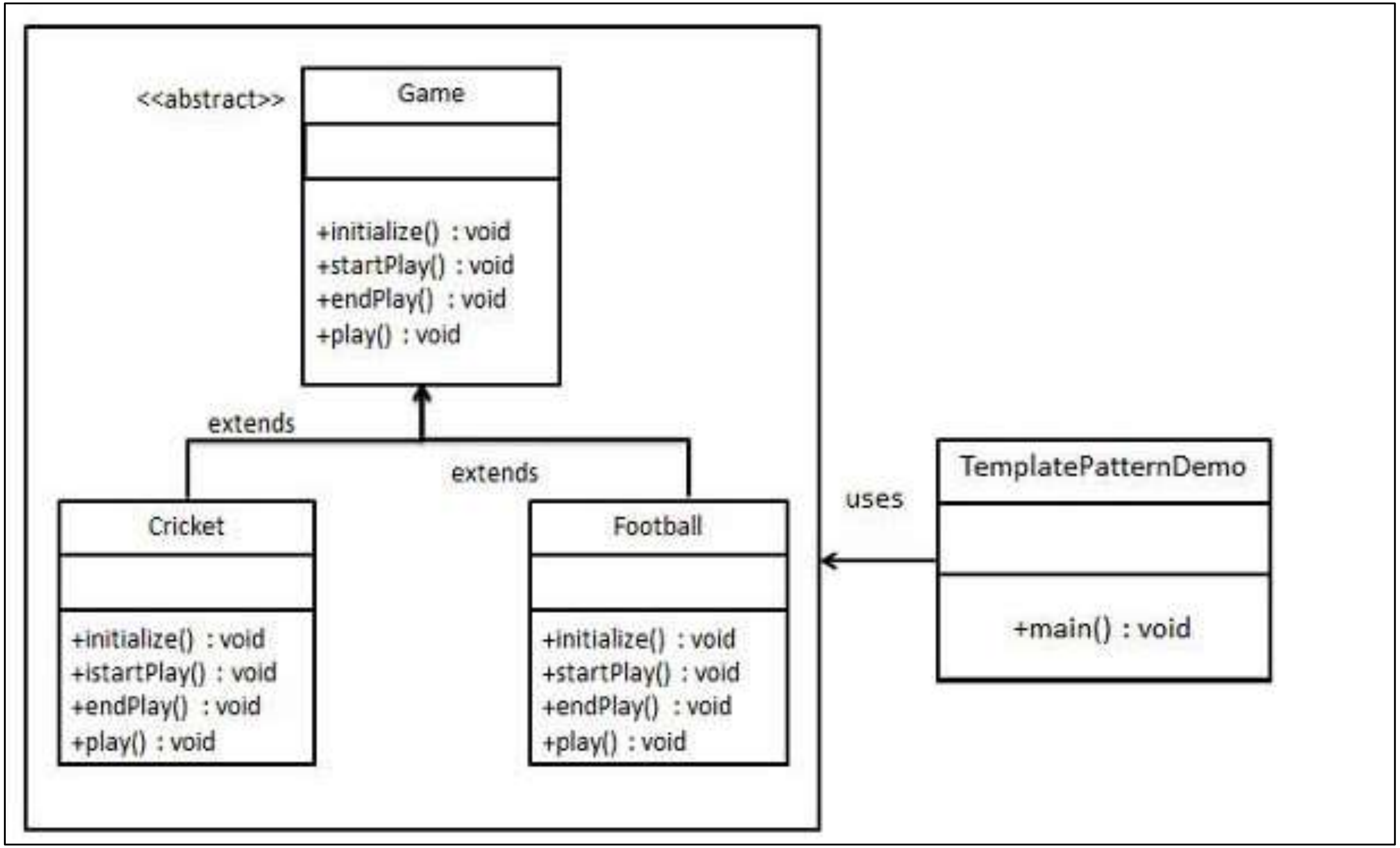
Implementation of (Template Pattern)

- 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.

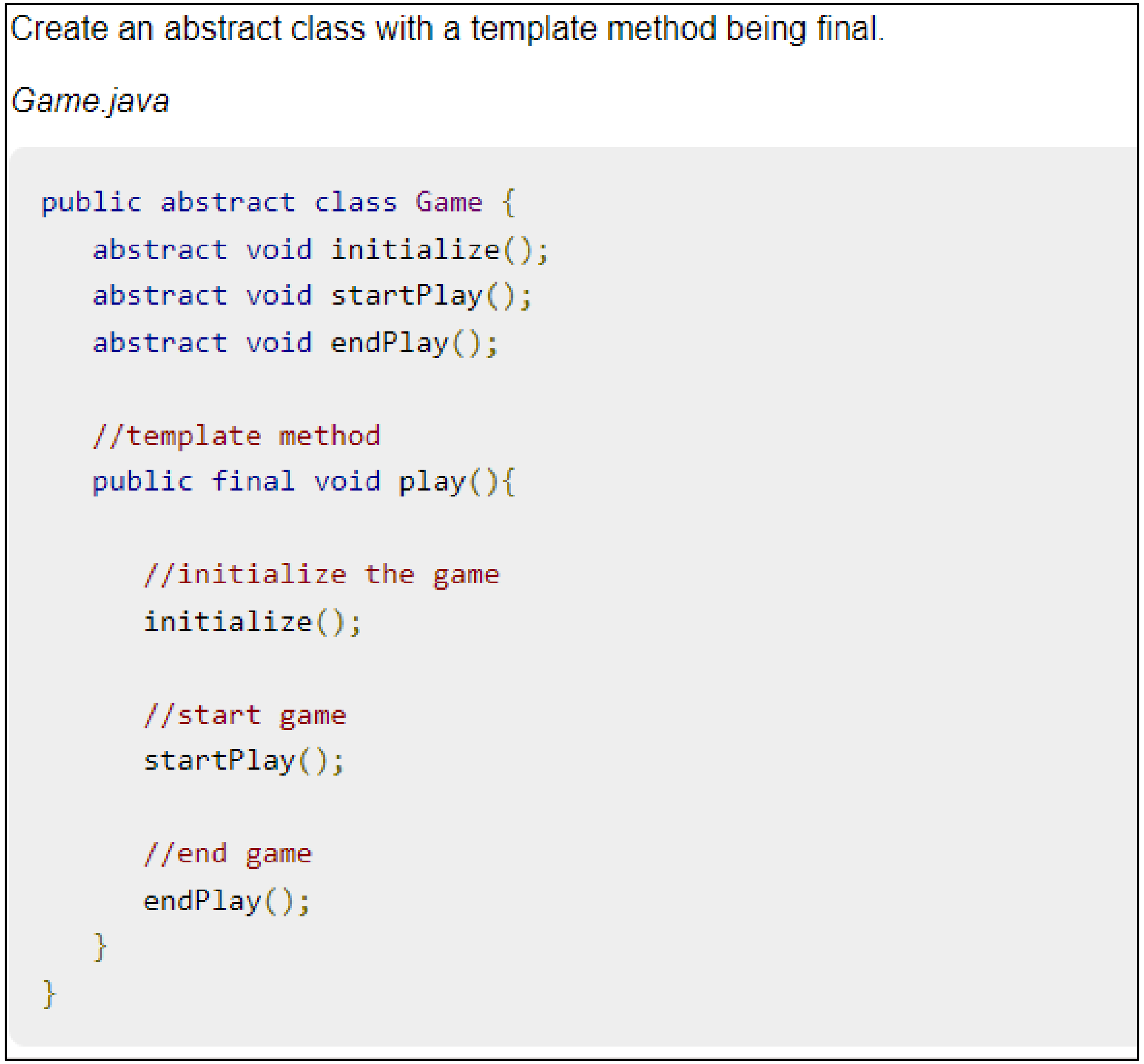
- In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class. This pattern comes under behavior pattern category.

UML\Structure for Template Pattern



Implementation of (Template Pattern)

Step 1

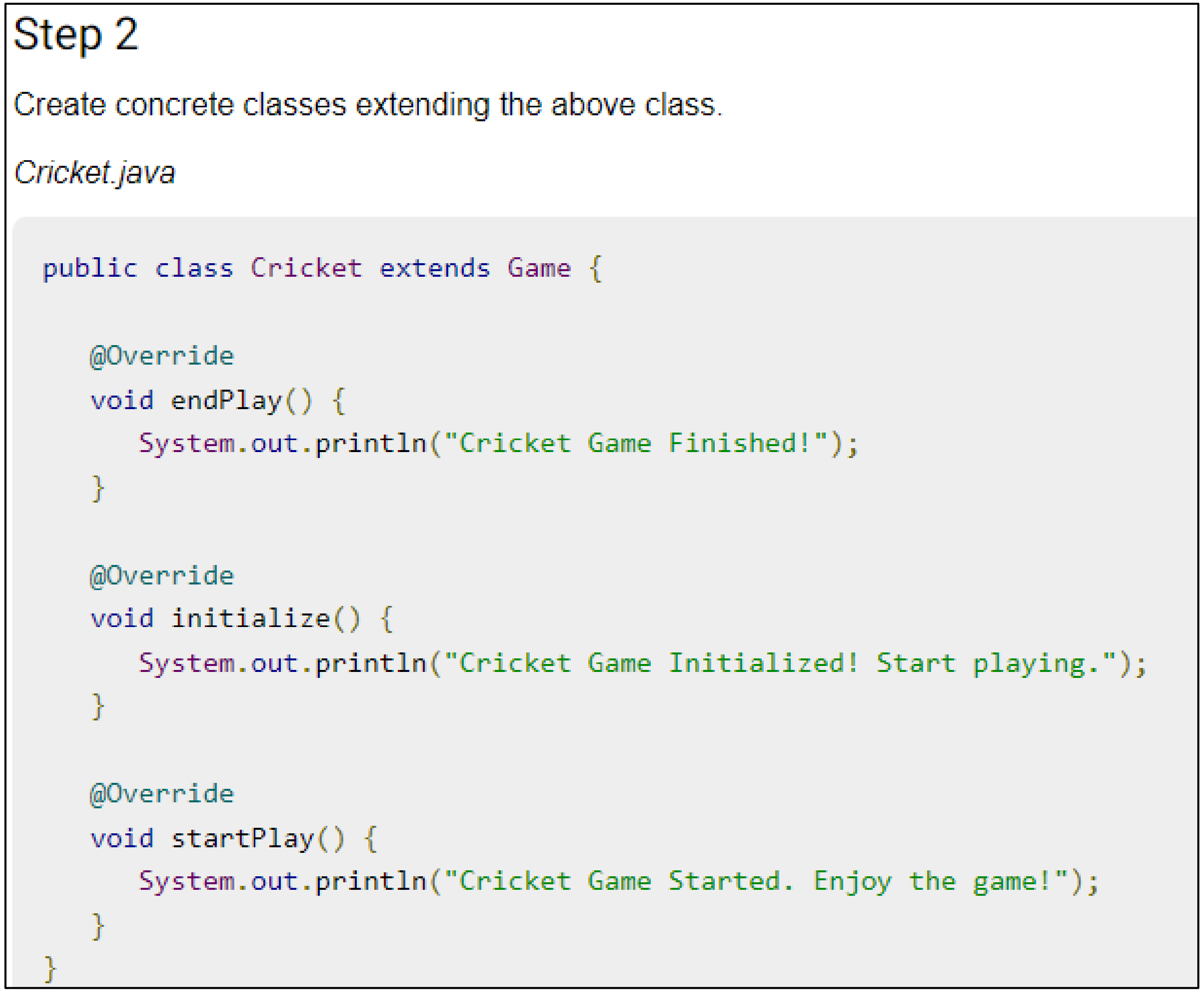


Create an abstract class with a template method being final.

Game.java

```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();
   }
}
```

Implementation of (Template Pattern)



Step 2

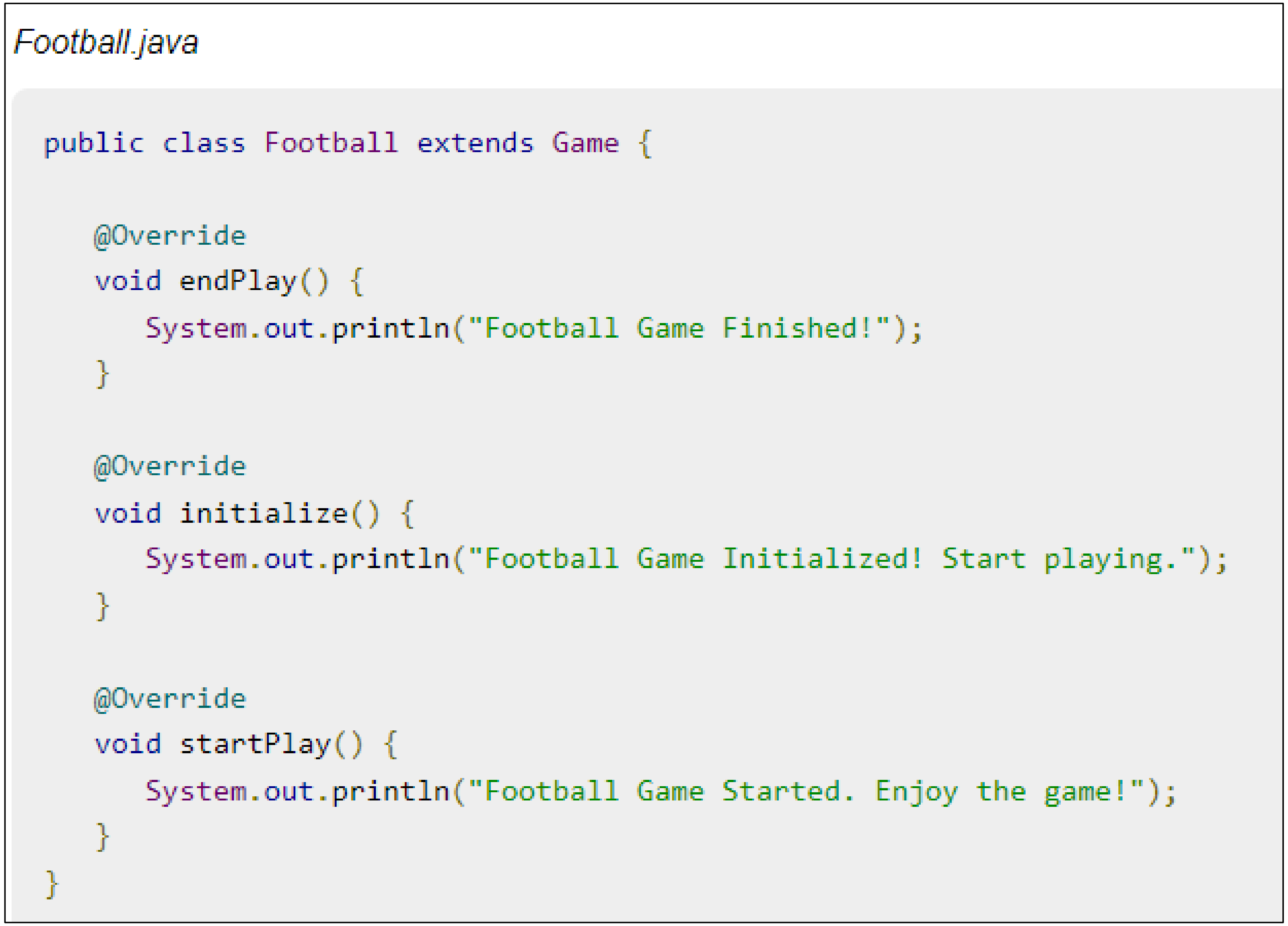
Create concrete classes extending the above class.

Cricket.java

```
public class Cricket extends Game {

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

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

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

Implementation of (Template Pattern)

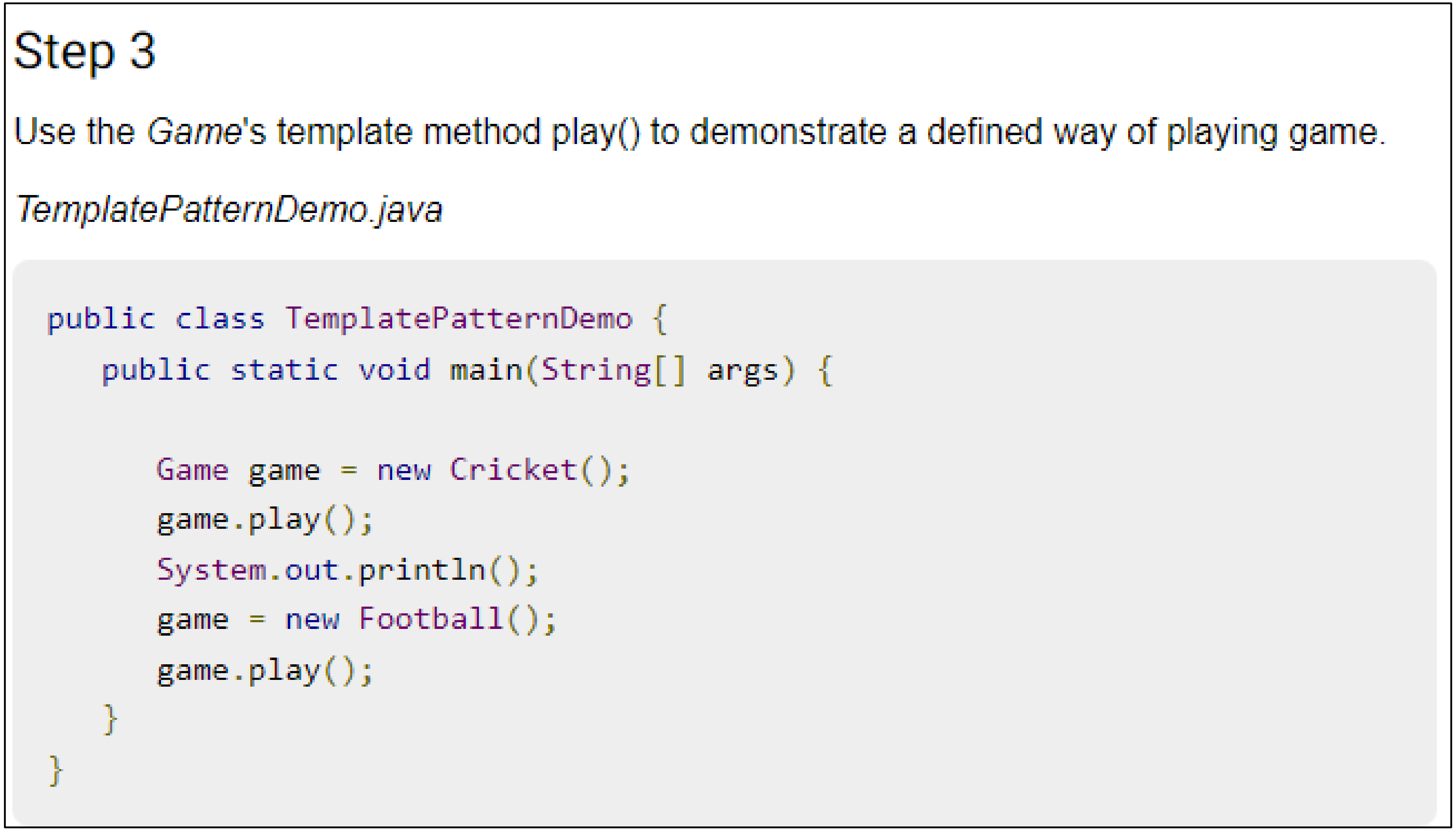
Step -2 Cont………



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!");
   }
}
```

Implementation of (Template Pattern)



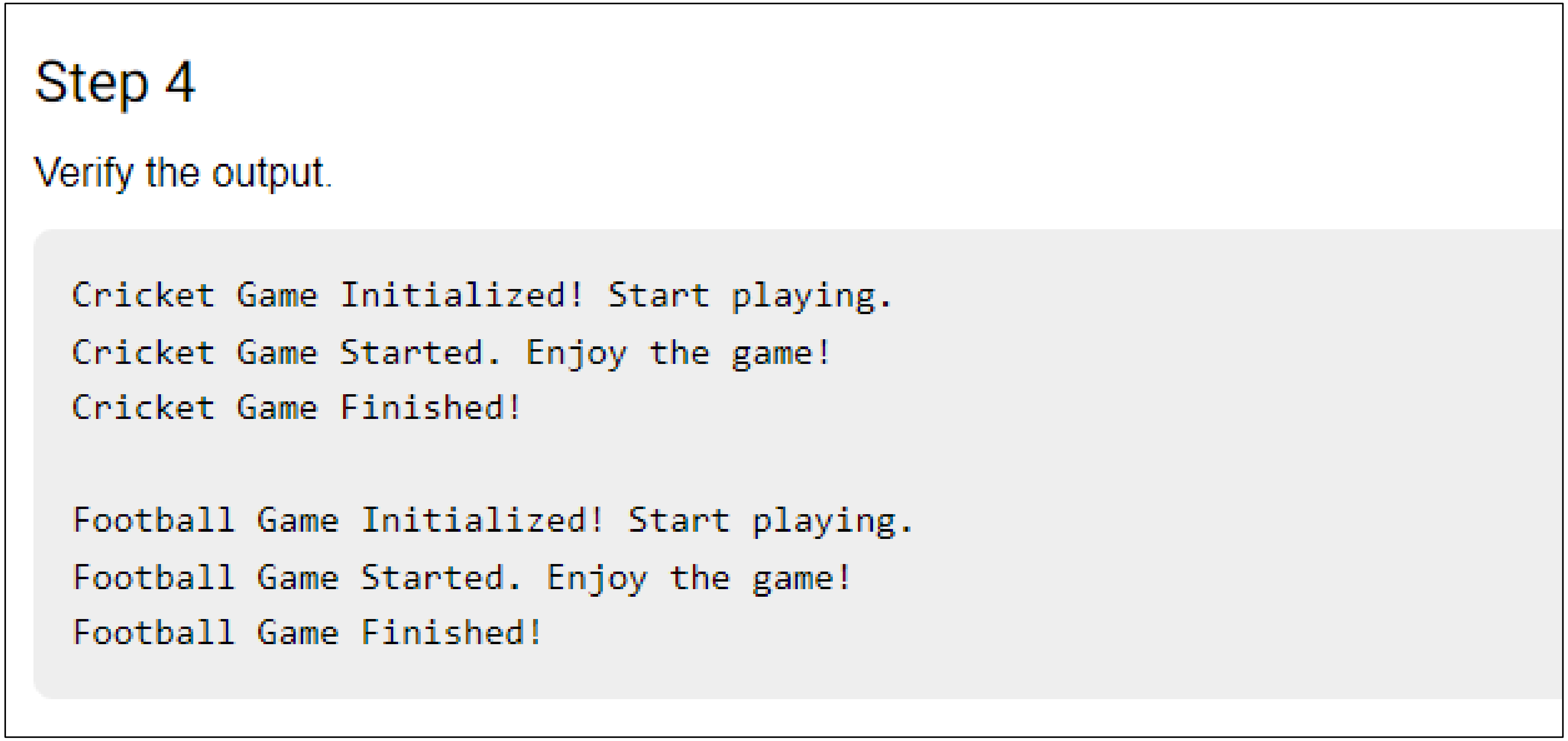
Step 3

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

TemplatePatternDemo.java

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

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

Implementation of (Template Pattern)



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!
```

Topic Objective

Topic : Visitor Pattern.

- In this topic, the students will gain , The idea of a Behavioral design pattern, In these design patterns This pattern comes under behavior pattern category. As per the pattern, element object has to accept the visitor object so that visitor object handles the operation on the element object.

Visitor Pattern

Visitor Pattern:-

- In Visitor pattern, we use a visitor class which changes the executing algorithm of an element class. By this way, execution algorithm of element can vary as and when visitor varies.

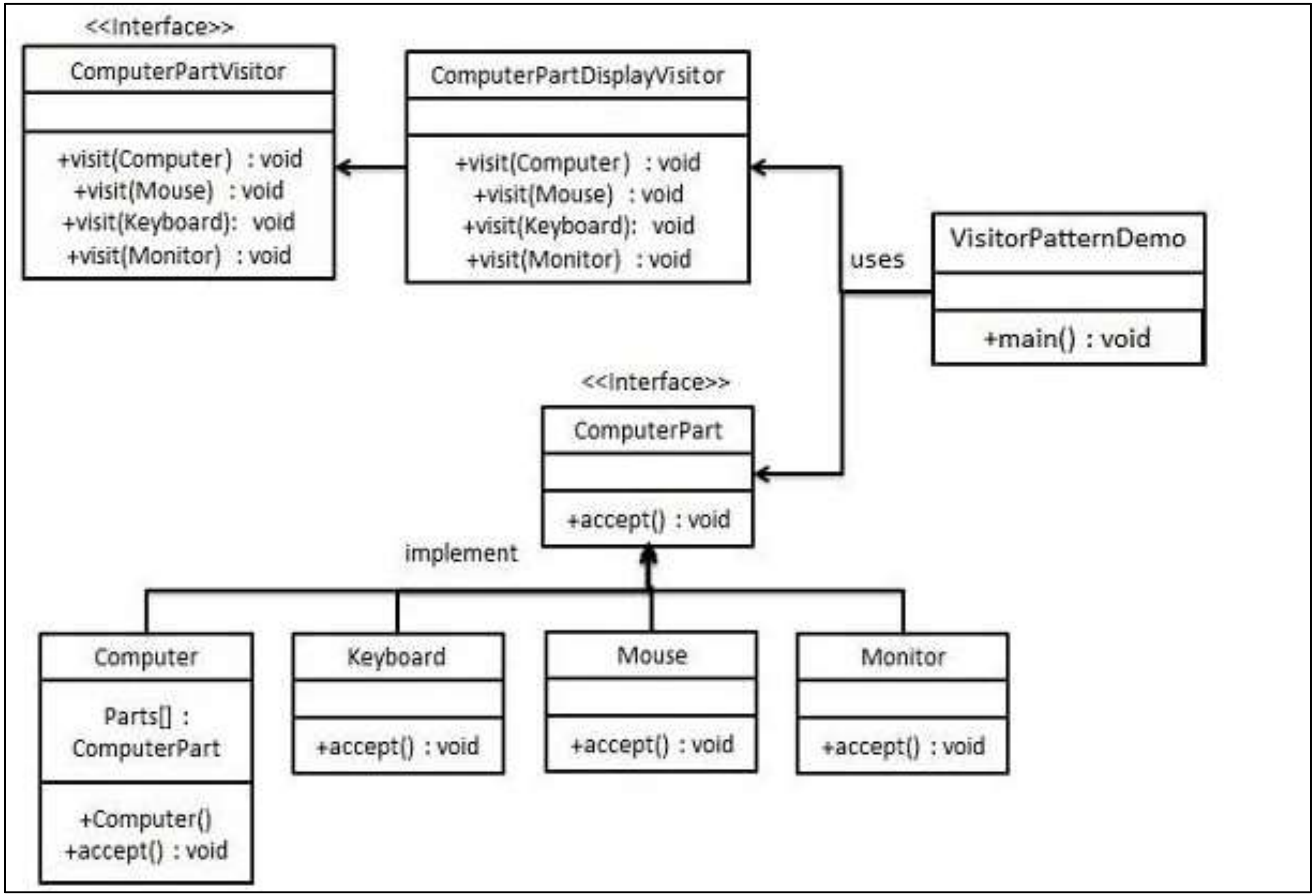
- pattern comes under behavior pattern category. As per the pattern, element object has to accept the visitor object so that visitor object handles the operation on the element object.

- In object-oriented programming , the visitor design pattern is a way of separating an algorithm from an object structure on which it operates. A practical result of this separation is the ability to add new operations to existing object structures without modifying the structures.

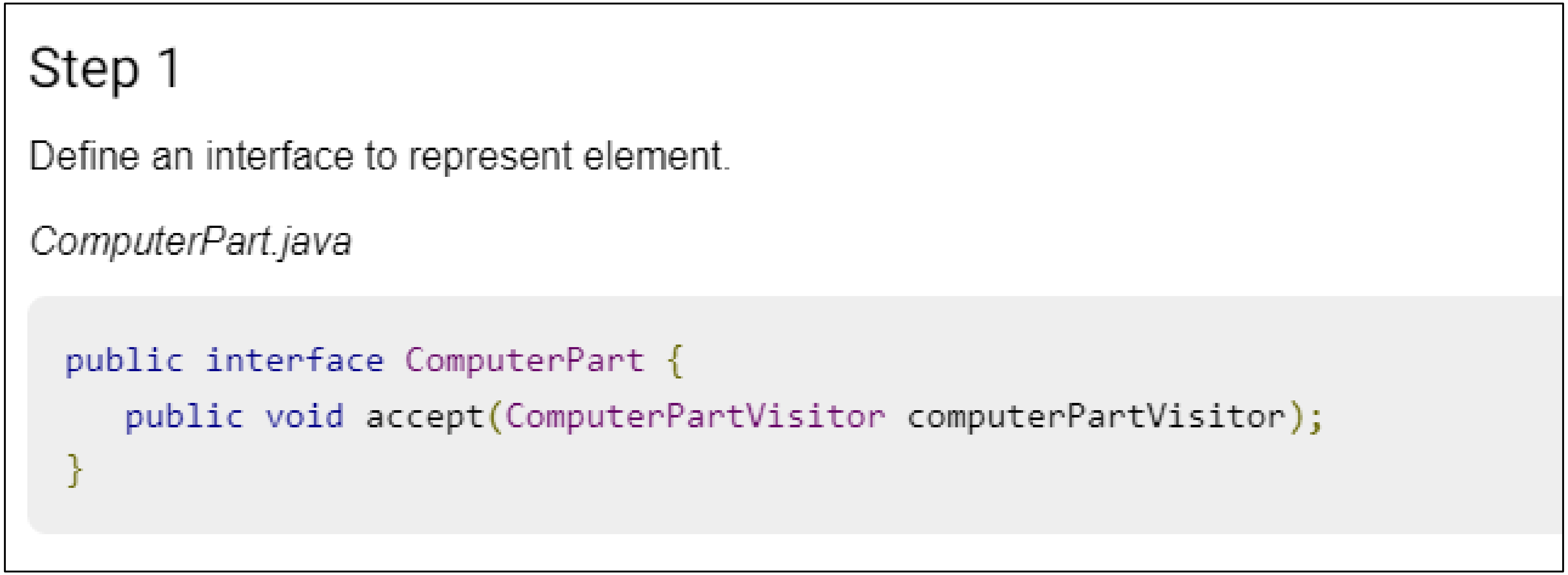
Implementation of (Visitor Pattern)

- We are going to create a ComputerPart interface defining accept opearation.Keyboard, Mouse, Monitor and Computer are concrete classes implementing ComputerPart interface.
- We will define another interface ComputerPartVisitor which will define a visitor class operations. Computer uses concrete visitor to do corresponding action.
- VisitorPatternDemo, our demo class, will use Computer and ComputerPartVisitor classes to demonstrate use of visitor pattern.

UML\Structure for Visitor Pattern



Implementation of (Visitor Pattern)



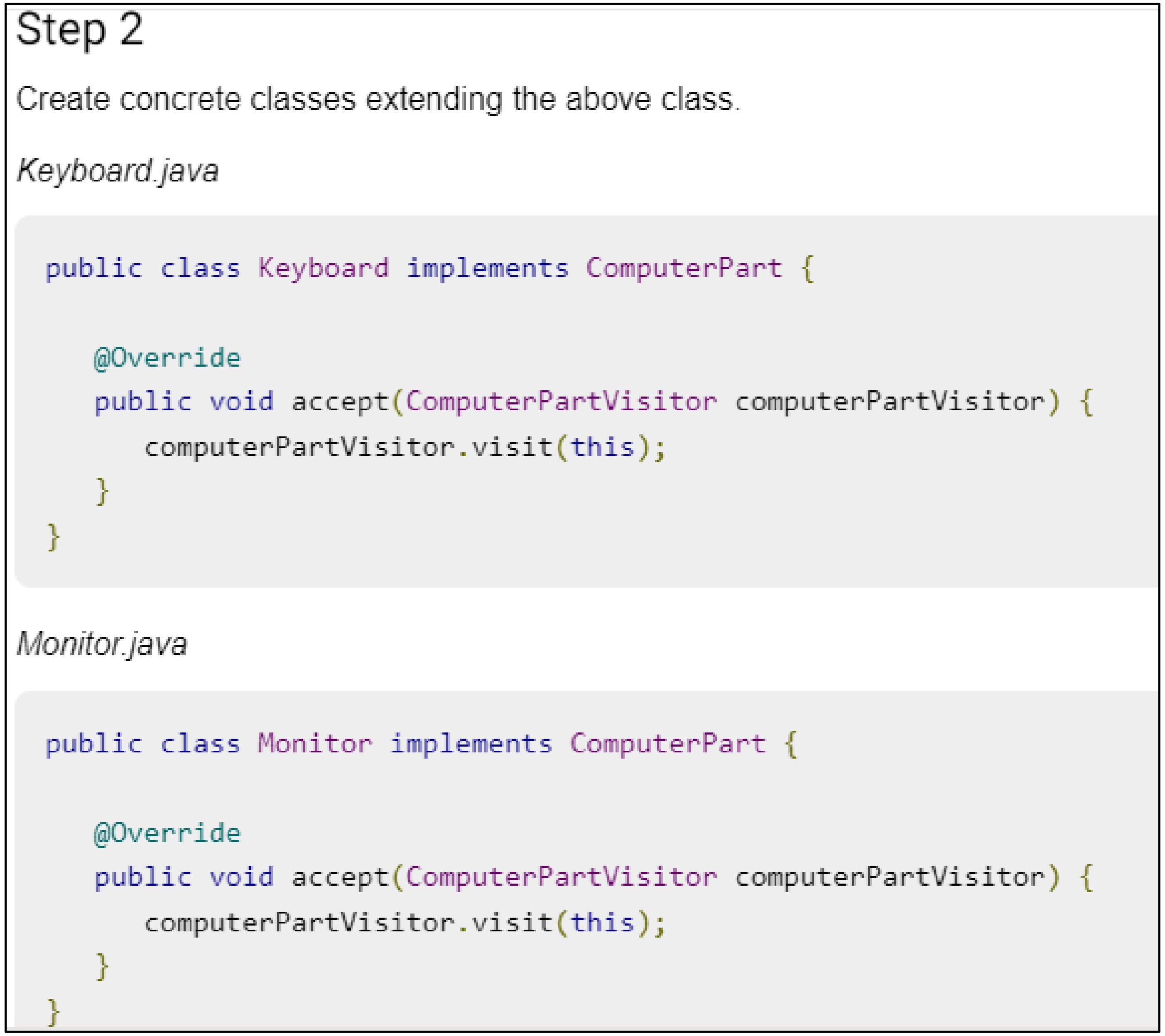
Step 1

Define an interface to represent element.

ComputerPart.java

```
public interface ComputerPart {
   public void accept(ComputerPartVisitor computerPartVisitor);
}
```

Implementation of (Visitor Pattern)



Step 2

Create concrete classes extending the above class.

Keyboard.java

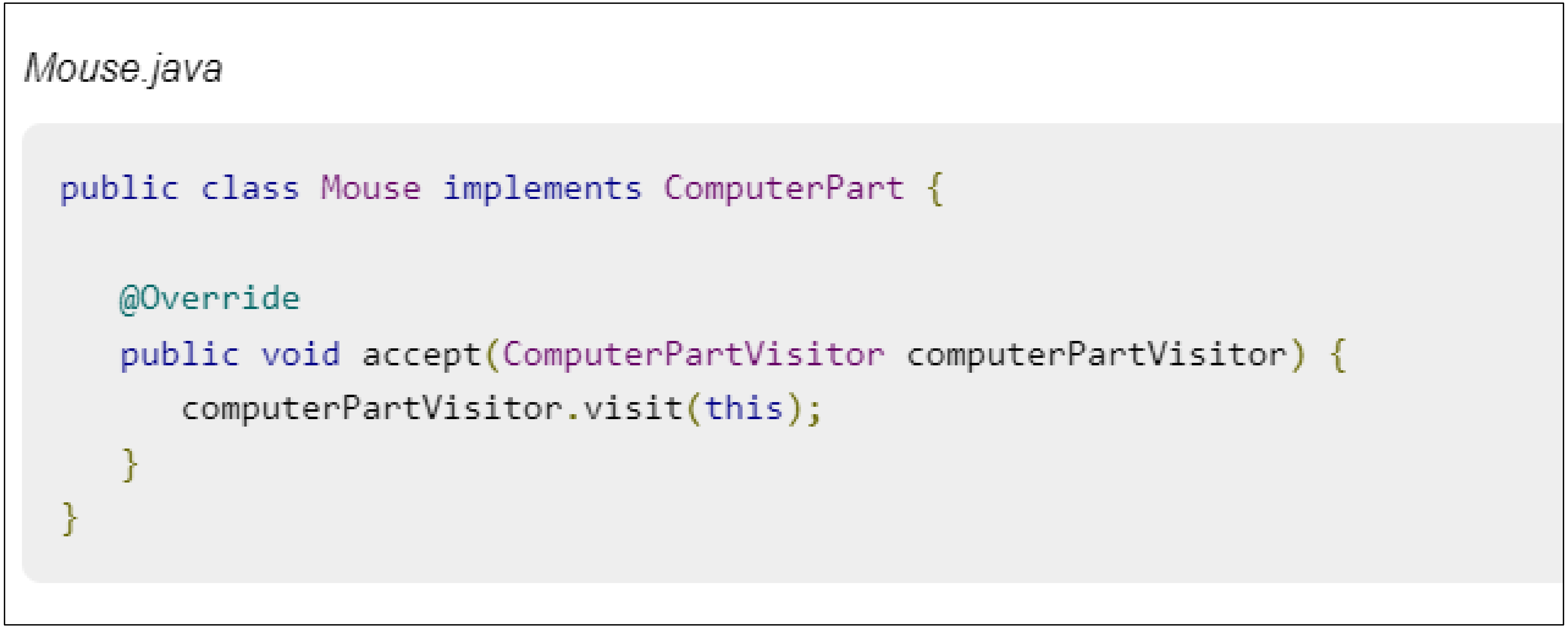
```
public class Keyboard implements ComputerPart {

   @Override
   public void accept(ComputerPartVisitor computerPartVisitor) {
      computerPartVisitor.visit(this);
   }
}
```

Monitor.java

```
public class Monitor implements ComputerPart {

   @Override
   public void accept(ComputerPartVisitor computerPartVisitor) {
      computerPartVisitor.visit(this);
   }
}
```

Implementation of (Visitor Pattern)

Step -2 Cont.........

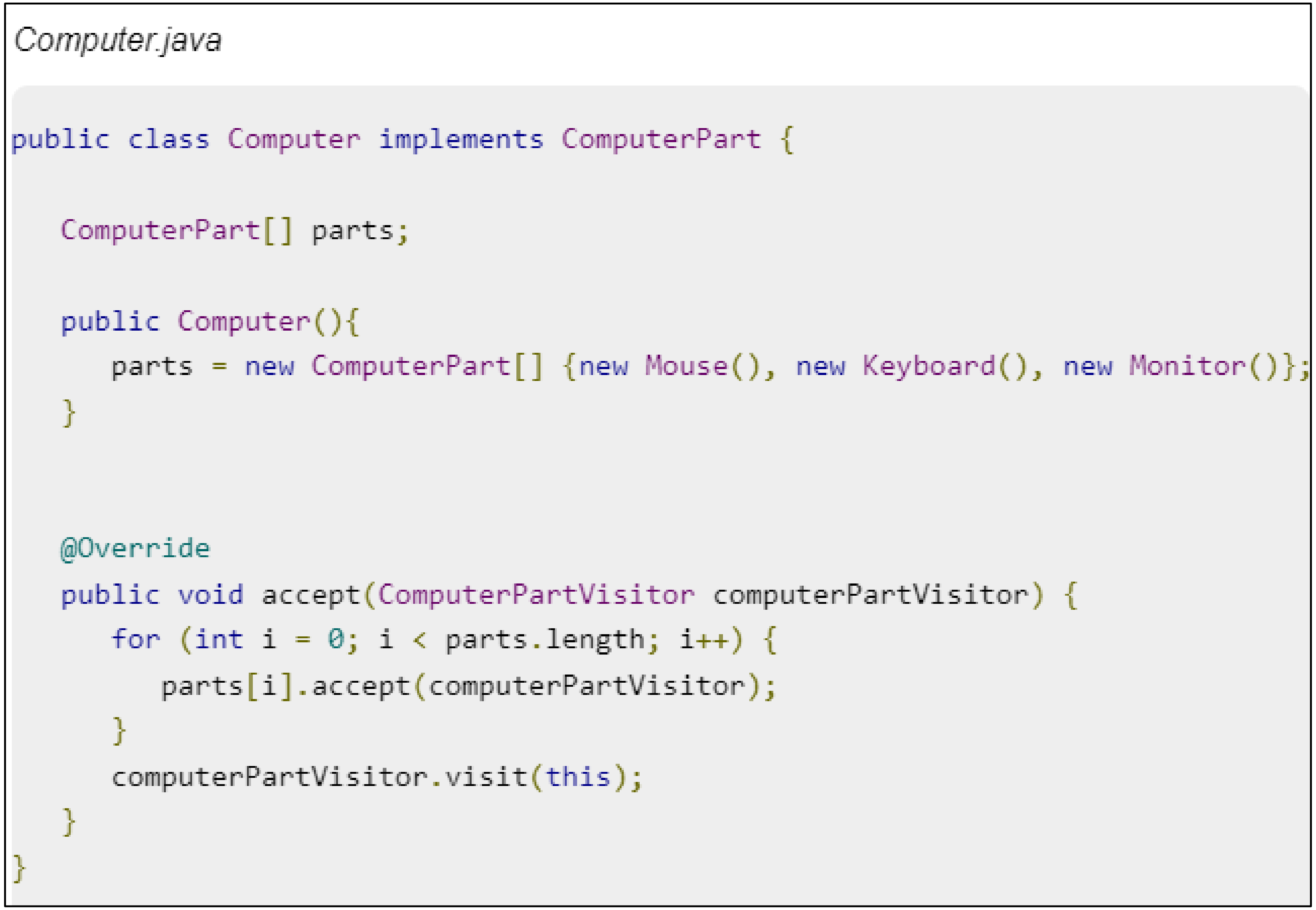


Mouse.java

```
public class Mouse implements ComputerPart {

   @Override
   public void accept(ComputerPartVisitor computerPartVisitor) {
      computerPartVisitor.visit(this);
   }
}
```

Implementation of (Visitor Pattern)

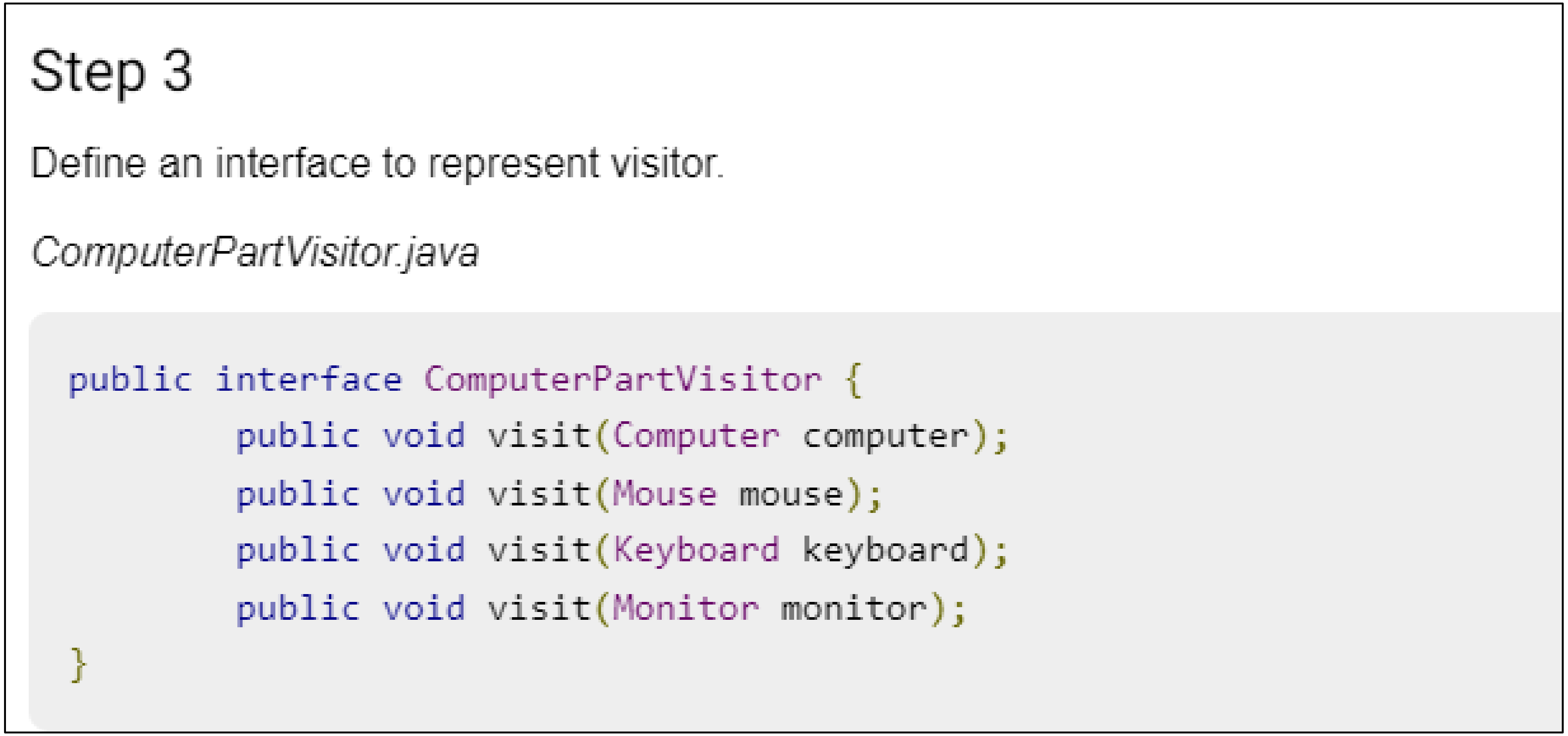
Step -2 Cont.........



Computer.java

```
public class Computer implements ComputerPart {

   ComputerPart[] parts;

   public Computer(){
      parts = new ComputerPart[] {new Mouse(), new Keyboard(), new Monitor()};
   }


   @Override
   public void accept(ComputerPartVisitor computerPartVisitor) {
      for (int i = 0; i < parts.length; i++) {
         parts[i].accept(computerPartVisitor);
      }
      computerPartVisitor.visit(this);
   }
}
```

Implementation of (Visitor Pattern)



Step 3

Define an interface to represent visitor.

ComputerPartVisitor.java

```
public interface ComputerPartVisitor {
	public void visit(Computer computer);
	public void visit(Mouse mouse);
	public void visit(Keyboard keyboard);
	public void visit(Monitor monitor);
}
```

Implementation of (Visitor Pattern)

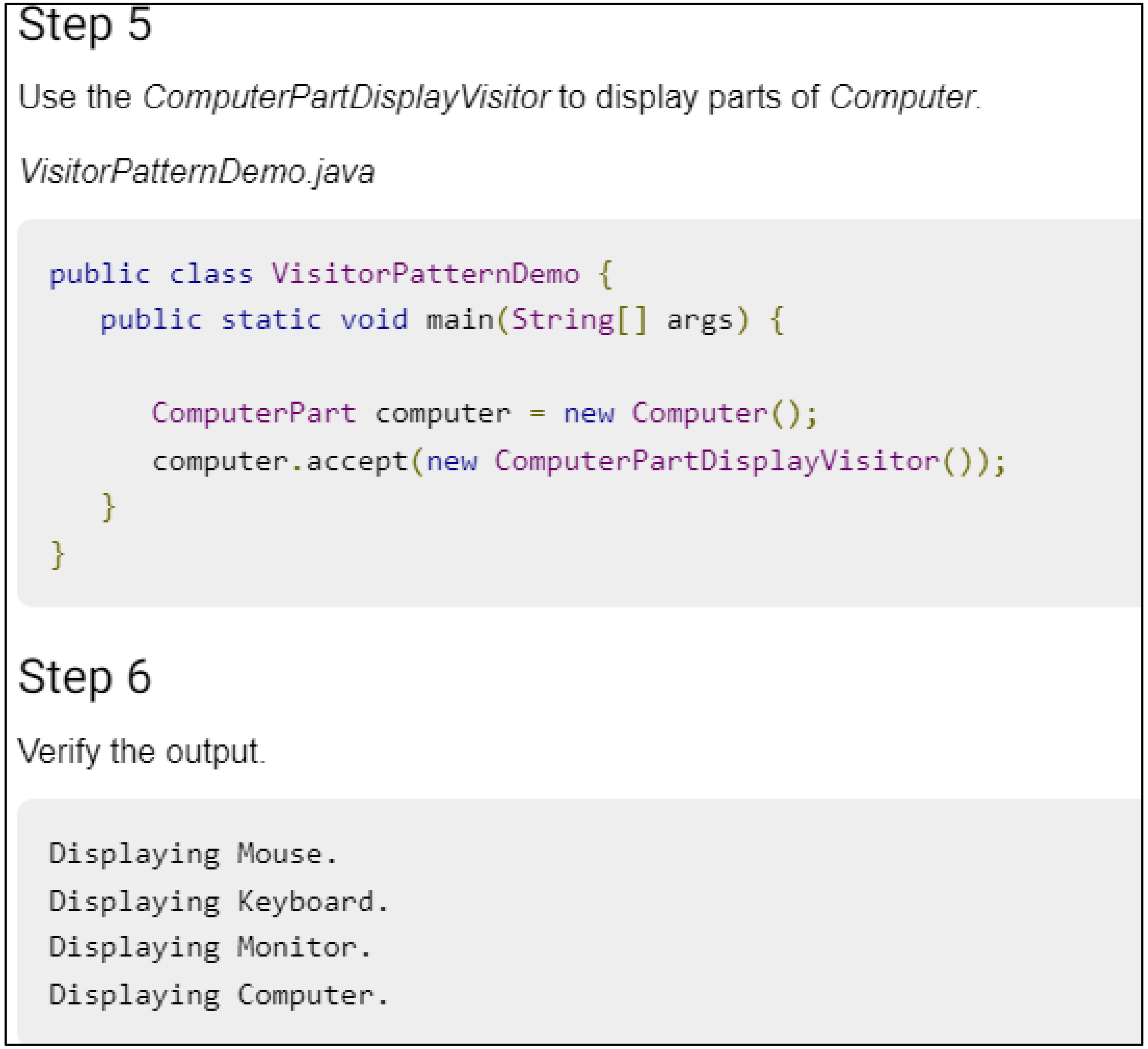
Step 4

Create concrete visitor implementing the above class.

ComputerPartDisplayVisitor.java

```java
public class ComputerPartDisplayVisitor implements ComputerPartVisitor {

   @Override
   public void visit(Computer computer) {
      System.out.println("Displaying Computer.");
   }

   @Override
   public void visit(Mouse mouse) {
      System.out.println("Displaying Mouse.");
   }

   @Override
   public void visit(Keyboard keyboard) {
      System.out.println("Displaying Keyboard.");
   }

   @Override
   public void visit(Monitor monitor) {
      System.out.println("Displaying Monitor.");
   }
}
```

Implementation of (Visitor Pattern)



Step 5

Use the *ComputerPartDisplayVisitor* to display parts of *Computer*.

VisitorPatternDemo.java

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

      ComputerPart computer = new Computer();
      computer.accept(new ComputerPartDisplayVisitor());
   }
}
```

Step 6

Verify the output.

```
Displaying Mouse.
Displaying Keyboard.
Displaying Monitor.
Displaying Computer.
```

Daily Quiz

Which one pattern creating duplicate object?

A. Filter Pattern

B. Prototype Pattern

C. Bridge Pattern

D. Builder Pattern

Which of the following describes the Builder pattern correctly?

A. This pattern builds a complex object using simple objects and using a step by step approach.

B. This pattern refers to creating duplicate object while keeping performance in mind.

C. This pattern is used when creation of object directly is costly.

D. This pattern is used when we need to decouple an abstraction from its implementation so that the two can vary independently.

Daily Quiz

In which of the following pattern a class represents functionality of another class?

A. Proxy Pattern
B. Chain of Responsibility Pattern
C. Command Pattern
D. Interpreter Pattern

Which of the following describes the MVC pattern correctly?

A. In this pattern, a visitor class is used which changes the executing algorithm of an element class.
B. This pattern is used to separate application's concerns.
C. This pattern is used to decouple presentation tier and business tier.
D. This pattern is used in EJB persistence mechanism

Daily Quiz

The use of design patterns for the development of object-oriented software has important implications for

A. Component-based software engineering
B. Reusability in general
C. All of the above
D. None of the above

Attach additional responsibilities to an object dynamically. It provides a flexible alternative to sub classing for extending functionality.

A. Chain of responsibility
B. Adapter
C. Decorator
D. Composite

Weekly Assignment

1. Which design pattern is used to get a way to access the elements of a collection object in sequential manner?
2. When service locator pattern is used?
3. Mention in how many ways can you create singleton pattern?
4. Mention how one should describe a design pattern?
5. Explain why access to the non-static variable is not allowed from static method in Java?

Topic Link (YouTube & NPTEL Video Links)

YouTube /other Video Links

- https://youtu.be/rI4kdGLaUiQ?list=PL6n9fhu94yhUbctIoxoVTrklN3LMwTCmd
- https://youtu.be/v9ejT8FO-7I?list=PLrhzvIcii6GNjpARdnO4ueTUAVR9eMBpc
- https://youtu.be/VGLjQuEQgkI?list=PLt4nG7RVVk1h9lxOYSOGI9pcP3I5oblbx

MCQ (End of Unit)

Which of the following pattern is primarily used to reduce the number of objects created and to decrease memory footprint and increase performance?

- ❑ Composite Pattern
- ❑ Facade Pattern
- ❑ Flyweight Pattern
- ❑ Decorator Pattern

In which of the following pattern, a class behavior changes based on its state?

- ❑ State Pattern
- ❑ Null Object Pattern
- ❑ Strategy Pattern
- ❑ Template Pattern

MCQ (End of Unit)

3. Which design pattern provides a single class which provides simplified methods required by client and delegates call to those methods?

- ❑ Adapter pattern
- ❑ Builder pattern
- ❑ Facade pattern
- ❑ Prototype pattern

4. Which design pattern suggests multiple classes through which request is passed and multiple but only relevant classes carry out operations on the request?

- ❑ Singleton pattern
- ❑ Chain of responsibility pattern
- ❑ State pattern
- ❑ Bridge pattern

Glossary Questions

Top 10 design pattern interview questions

1. Explain Data Access Object (DAO) pattern?
2. Mention what is the difference between VO and JDO?
3. Explain the benefits of design patterns in Java.
4. Describe the proxy pattern.
5. Differentiate ordinary and abstract factory design patterns.
6. What do you think are the advantages of builder design patterns?
7. How is the bridge pattern different from the adapter pattern?
8. What is a command pattern?
9. Describe the singleton pattern along with its advantages and disadvantages.
10. What are anti patterns?

Expected Questions for University Exam

- What Is Abstract Factory Pattern?
- How are design patterns categorized?
- Explain the benefits of design patterns in Java.
- Describe the factory pattern.
- Differentiate ordinary and abstract factory design patterns.
- What do you think are the advantages of builder design patterns?
- How is the bridge pattern different from the adapter pattern?
- What is a command pattern?
- Describe the singleton pattern along with its advantages and disadvantages.
- What are anti patterns?

Recap of Unit

- **Till Now we understand**, Behavioral design patterns are concerned with the interaction and responsibility of objects.
- A State Pattern says that "the class behavior changes based on its state. i.e Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern.
- The idea of a Behavioral design pattern, In these design patterns i.e In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods.
- In Visitor pattern, we use a visitor class which changes the executing algorithm of an element class. By this way, execution algorithm of element can vary as and when visitor varies.