# Presentation 7: Design Patterns

Java Programming:
Program Design Including Data Structures

# Chapter Objectives

- Learn about Creational Patterns.
- Learn about Structural Patterns.
- Learn about  Behavioral Patterns

# Why use Design Patterns?

- *Design patterns help us to design a flexible system that is reusable and extensible.*

**Principles for using design patterns**

- *Program to an Interface, not an implementation*
- *Favor object composition over class inheritance*

**Causes of redesign**

- *Creating an object by specifying a class explicitly: create an object indirectly.*
- *Dependency on specific operations: get the requests (when calling a method) to be satisfied at compile time and run time.*
- *Dependence on object representations or implementations*
- *Algorithmic dependencies*
- *Tight coupling*

# Creational Patterns

- *Creational class patterns defer some part of object creation to subclasses, while creational object patterns defer it to another object.*

- Singleton Pattern.

# Singleton Pattern

A Singleton pattern provides a global access point.

Purpose:

- Ensure a class has only one instance, and provides a global point of access to it.

Scenario:

- Where only one object is created from a class such as one printer spooler.
- One file system. One window manager. single DB connection.

# Singleton Pattern

A Singleton pattern provides a global access point.

## Advantage:

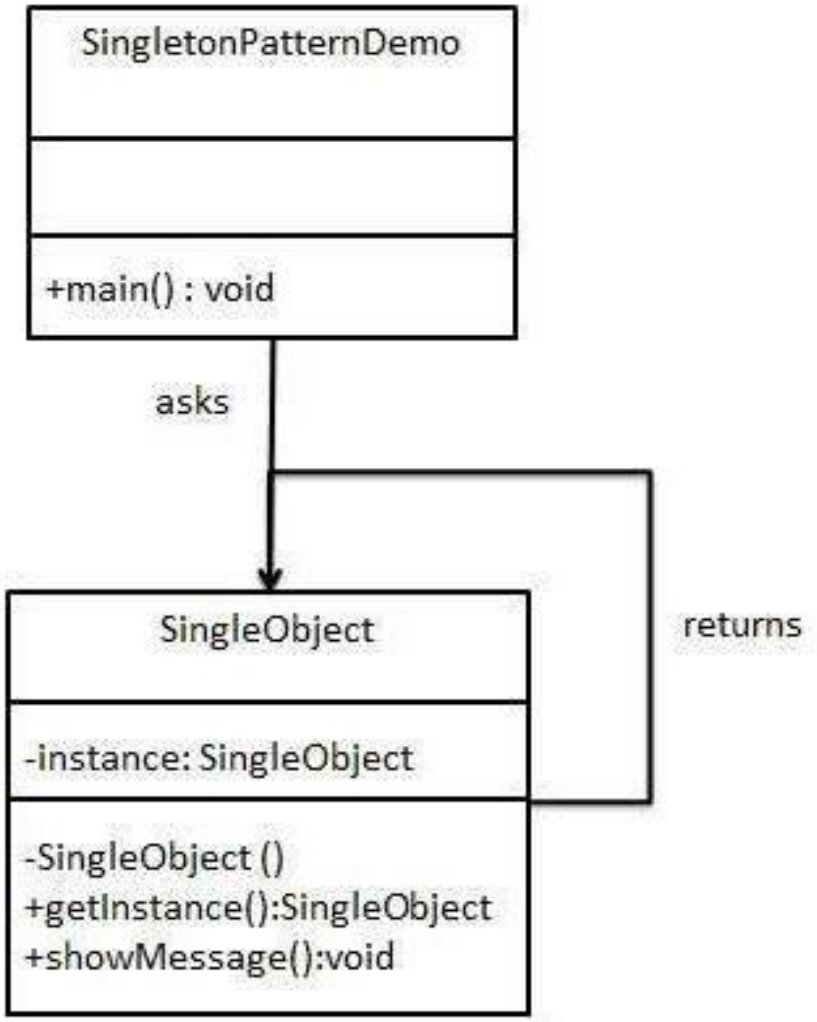
- make the constructor private, so that only one instance is ever created.
- Controlled access to sole instance.
- Permits to create a variable number of instances

# Singleton Pattern Steps

Steps:

- Create a Singleton Class.
- Get the only object from the singleton class.
- Verify the output.

# Singleton Pattern



# Behavioral Patterns

- *The behavioral class patterns use inheritance to describe algorithms and flow of control, whereas the behavioral object patterns describe how a group of objects cooperate to perform a task that no single object can carry out alone.*

- Iterator Pattern.

# Iterator Pattern

Iterator is applied for collections.

The key idea is to take the responsibility for access and traversal out of the aggregate object and put it into an Iterator object that defines a standard traversal protocol.

Purpose:

- Accessing structured object representations without having to know the internal data structure.

Scenario:

- Adding iterator class and subclassing it to represent any data structure used in developing text editor application (ArrayIterator or LinkedListIterator representing row of character text).

# Iterator Pattern

The iterator has 4 main parts: an iterator, an aggregator Concrete iterator, a concrete aggregator.

## Advantage:

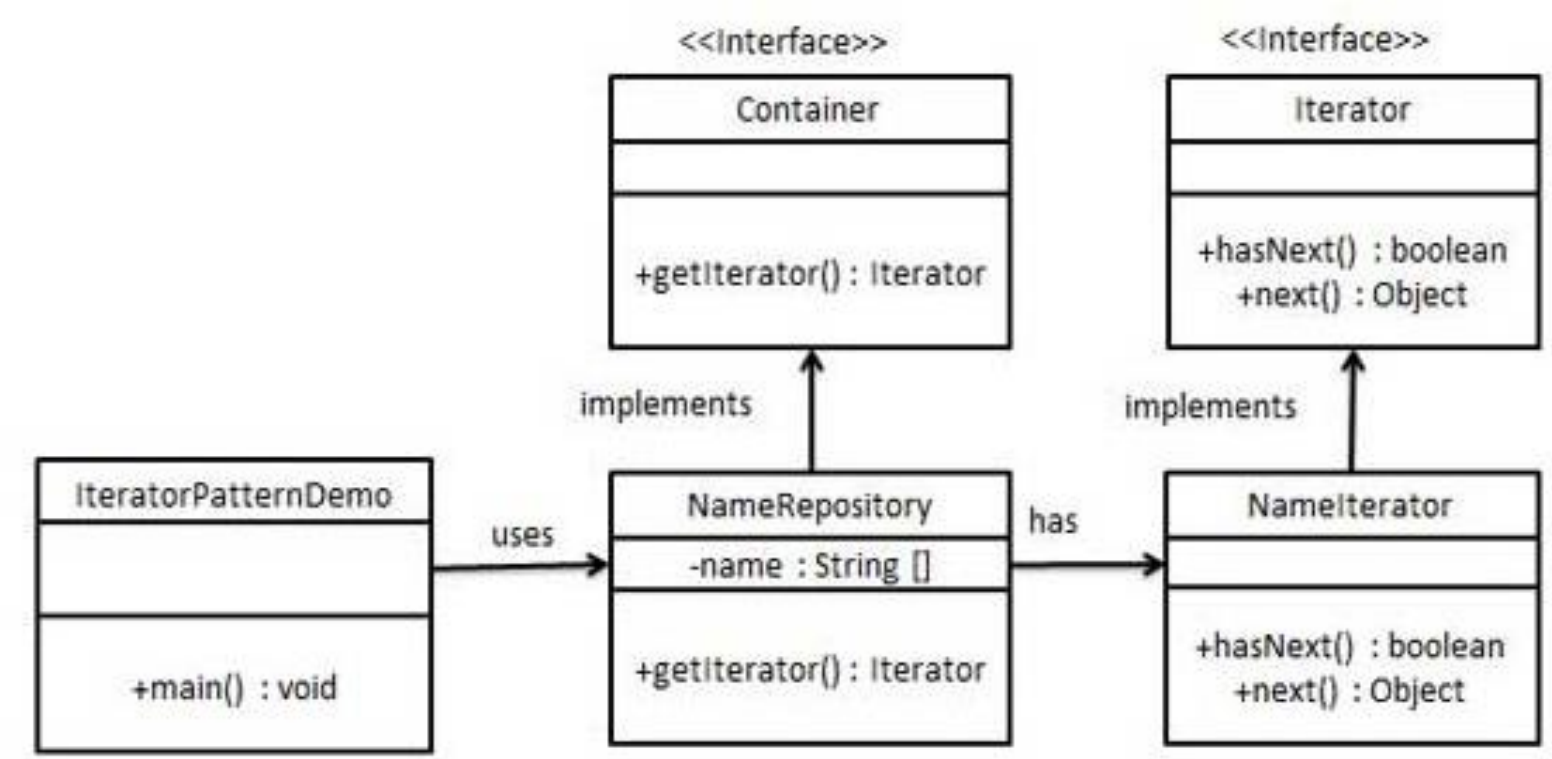
- The possibility to add new dataIterator (for example traversal only text element in the tree) without having to alter the client.
- Write a client code that is independent from the concrete Iterator sub classes. It would be better if we can change the aggregate concrete sub class (NameIterator in our ex) without changing client code.
- You can define NameIterator subclasses to support new traversals.

# Iterator Pattern Steps

Steps:

- Create interfaces..
- Create concrete classes.

# Iterator Pattern



# Architectural Pattern

- MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns
- MVC Pattern.

# MCV Pattern

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

## Purpose:

- Separate internal representations of information from the ways information is presented to and accepted from the user.

## Scenario:

- Commonly used for developing user interfaces.

# MCV Pattern

The MVC has 3 main parts: a Model, View, and a Controller.

## Advantage:

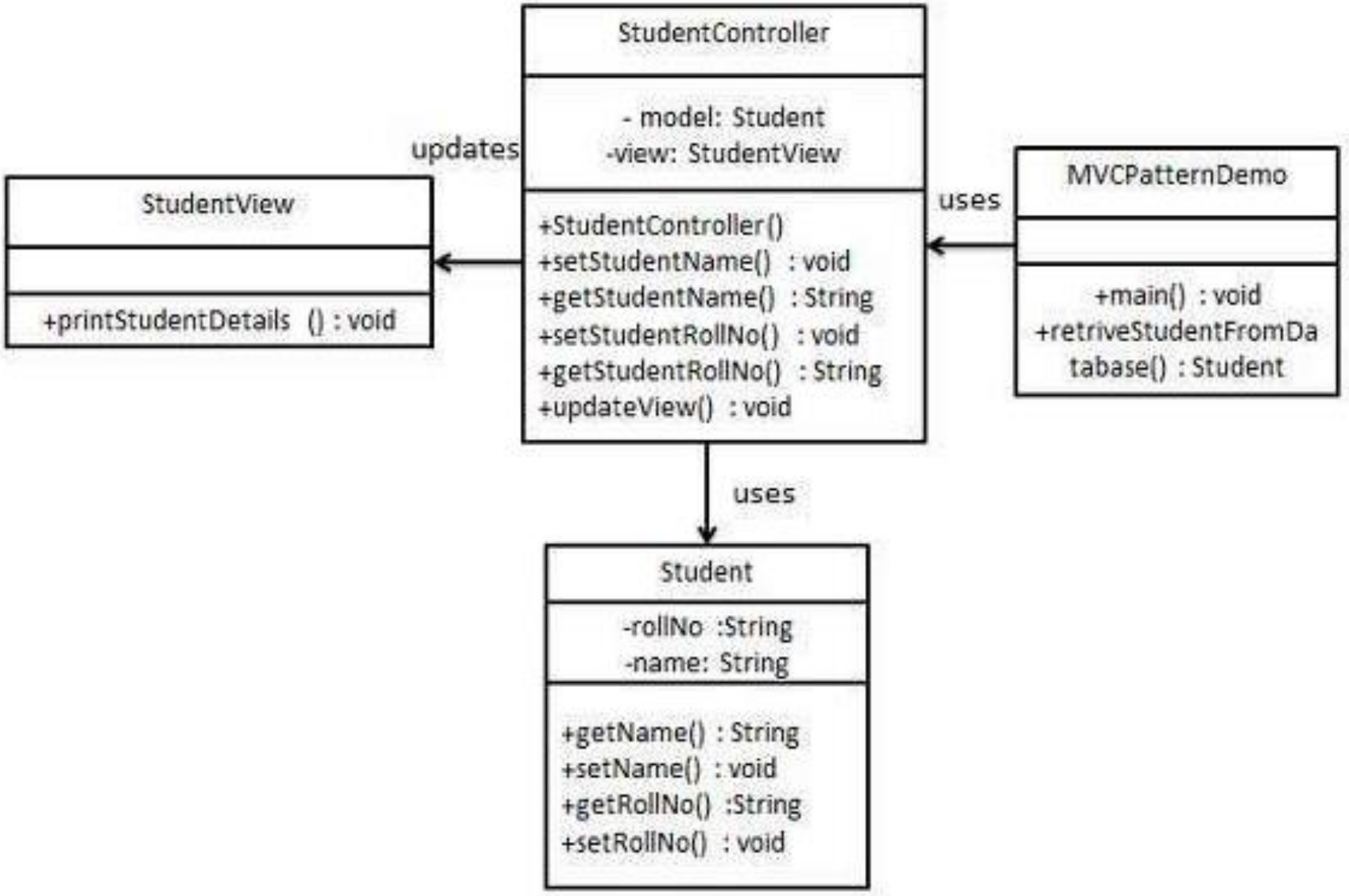
- Independence of presentation and data, e.g. multiple views on one model simultaneously.

# MCV Pattern Steps

Steps:

- Create a Model class.
- Create View class.
- Create Controller class.

# MCV Pattern


StudentController
- model: Student
-view: StudentView
+StudentController()
+setStudentName() : void
+getStudentName() : String
+setStudentRollNo() : void
+getStudentRollNo() : String
+updateView() : void
updates
StudentView
+printStudentDetails () : void
uses
MVCPatternDemo
+main() : void
+retriveStudentFromDatabase() : Student
uses
Student
-rollNo :String
-name: String
+getName() : String
+setName() : void
+getRollNo() :String
+setRollNo() : void

# Chapter Summary

- How to use Singleton Pattern.
- How to use Iteration Pattern.
- How to use MVC Pattern.