Information Processing Techniques

WEEK 01

Abeeha Sattar

Let’s Introduce Ourselves

Name: Abeeha Sattar

Qualifications: MS(CS) from IBA (2020), BS(CS) from FAST-NUCES (2016)

Experience: Worked in Systems Ltd. for about 1.5 years.

Tell me… your name, hobbies , your field of interest, and your expectations from this course.

Office Location

Basement - II, Room # 16

Just check the time table before visiting!

Pre- Requisites

The course assumes that you have prior knowledge on the following:

- Software Engineering
-
- Programming Basics

Course Outline

- You may find the course outline on Google Classroom.

- Assignments:
  - Assignment # 1 – Week 3
  - Assignment # 2 – Week

- Quizzes

- Project:
  - Project Proposal – Week 2
  - Project SRS – Week 5
  - Project SDS, UI – Week 10
  - Project Presentations and Demos – Week 16

Grading Scheme

Assignments	5 %
Quizzes	5 %
Project	10 %
Midterms (2)	30 %
Final	50 %

Assignments Policies

- No late submissions
- Plagiarism on 1st assignment: 0 for that question
- Plagiarism on 2nd assignment: 0 for that assignment

Reference Books

- Clean Code: A Handbook of Agile Software Craftsmanship 1st Edition by Robert C.
- Martin
- Head First Design Patterns by Eric Freeman, Elisabeth Robson
- Refactoring - Improving the Design of Existing Code by Martin Fowler, with Kent Beck
- Software Architecture Patterns by Mark Richards, O'Reilly Media, Inc., 2015

Introduction to Java

History of Java

- Java was originally developed by Sun Microsystems starting in 1991
  - James Gosling
  - Patrick Naughton
  - Chris Warth
  - Ed Frank
  - Mike Sheridan
- This language was initially called “Oak”
- Renamed **Java** in 1995

What is Java?

A simple, **object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded**, and **dynamic** language

Object-Oriented

– No free functions

– All code belong to some class

– Classes are in turn arranged in a hierarchy or package structure

Distributed

– Fully supports IPv4, with structures to support IPv6

– Includes support for Applets: small programs embedded in HTML documents

Interpreted

– The program are compiled into Java Virtual Machine (JVM) code called bytecode

– Each bytecode instruction is translated into machine code at the time of execution

Robust

– Java is simple

– no pointers/stack concerns

– Exception handling

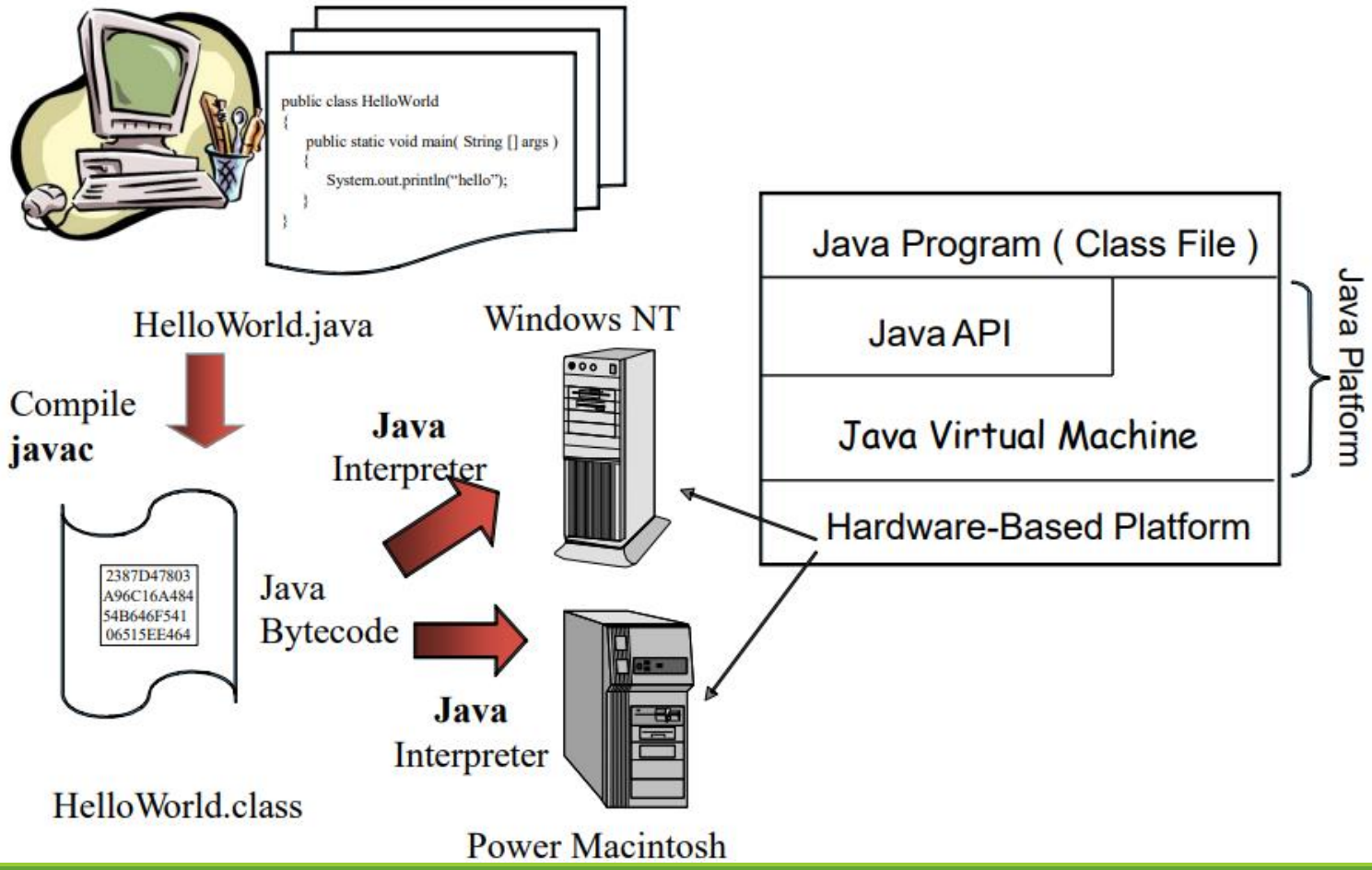
– try/catch/finally series allows for simplified error recovery

– Strongly typed language – many errors caught during compilation

Java Development Environment

- Edit – Create/edit the source code
- Compile – Compile the source code
- Load – Load the compiled code
- Verify – Check against security restrictions
- Execute – Execute the compiled

Java Platform



Bytecode

- They are not machine language binary code
- They are independent of any particular microprocessor or hardware platform
- They are platform-independent instructions
- Another entity (interpreter) is required to convert the bytecodes into machine codes that the underlying microprocessor understands

- This is the job of the JVM (Java Virtual Machine)

JVM (Java Virtual Machine)

- It is a part of the JDK and the foundation of the Java platform
- It can be installed separately or with JDK

- A virtual machine (VM) is a software application that simulates a computer, but hides the underlying operating system and hardware from the programs that interact with the VM

- It is the JVM that makes Java a portable language

JVM (Java Virtual Machine)

- The same bytecodes can be executed on any platform containing a compatible JVM
- The JVM is invoked by the java command – **java Welcome**
- It searches the class Welcome in the current directory and executes the main method of **class Welcome**
- It issues an error if it cannot find the class Welcome or if class Welcome does not contain a method called main with proper signature

JIT Compiler

- When a JIT compiler is part of the JVM, selected portions of bytecode are compiled into executable code in real time, on a piece-by-piece, demand basis.
- An entire Java program is not compiled into executable code all at once, in this case.

- Not all sequences of bytecode are compiled—only those that will benefit from compilation.
- The remaining code is simply interpreted.

Commands for Reference:

- Use **javac <filename>.java** to compile a java file using the command prompt
- Use **java <classname>** to execute the class file that was generated by the javac command.

Fin.

Software Construction & Development

WEEK 03

Abeeha Sattar

Layered Architecture

Layered Application Architecture

- Presentation layer
  - Concerned with presenting the results of a computation to system users and with collecting user inputs
- Application processing layer
  - Concerned with providing application specific functionality e.g., in a banking system, banking functions such as open account, close account, etc.
- Data management layer
  - Concerned with managing the system databases

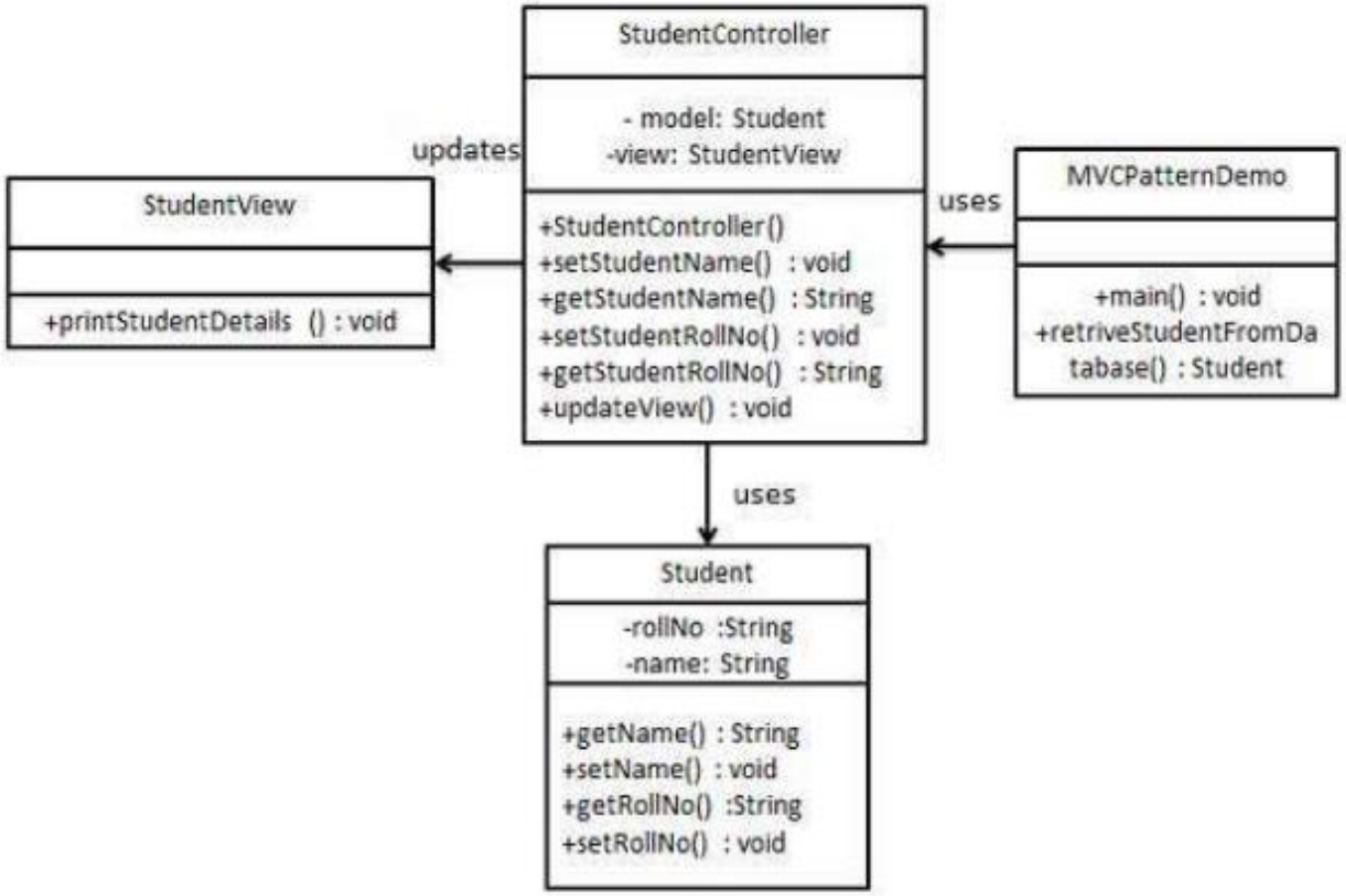
Tiering

- A two-tier architecture is one where a client talks directly to a server, with no intervening server
  - This type of architecture is typically used in small environments with less than 50 users
- A three-tier architecture introduces another server (or an "agent") between the client and the server
  - The role of the middle-tier agent is many-fold - it can provide translation services as in adapting a legacy application on a mainframe to a client/server environnent
  - A plethora of software technologies have evolved to fill the middle tier - middleware

MVC Pattern

- Model
  - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
- View
  - View represents the visualization of the data that model contains.
- Controller
  - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

Example


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

Student.java

```
public class Student {
   private String rollNo;
   private String name;

   public String getRollNo() {
      return rollNo;
   }

   public void setRollNo(String rollNo) {
      this.rollNo = rollNo;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

StudentView.java

```
public class StudentView {
   public void printStudentDetails(String studentName, String studentRollNo){
      System.out.println("Student: ");
      System.out.println("Name: " + studentName);
      System.out.println("Roll No: " + studentRollNo);
   }
}
```

StudentController.java

```
public class StudentController {
   private Student model;
   private StudentView view;

   public StudentController(Student model, StudentView view){
      this.model = model;
      this.view = view;
   }

   public void setStudentName(String name){
      model.setName(name);
   }

   public String getStudentName(){
      return model.getName();
   }
```

StudentController.java

```
   public void setStudentRollNo(String rollNo){
      model.setRollNo(rollNo);
   }

   public String getStudentRollNo(){
      return model.getRollNo();
   }

   public void updateView(){
      view.printStudentDetails(model.getName(), model.getRollNo());
   }
}
```

MVCPatternDemo.java

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

      //fetch student record based on his roll no from the database
      Student model  = retriveStudentFromDatabase();

      //Create a view : to write student details on console
      StudentView view = new StudentView();

      StudentController controller = new StudentController(model, view);

      controller.updateView();

      //update model data
      controller.setStudentName("John");

      controller.updateView();
   }

   private static Student retriveStudentFromDatabase(){
      Student student = new Student();
      student.setName("Robert");
      student.setRollNo("10");
      return student;
   }
}
```

DAO Pattern

- Data Access Object

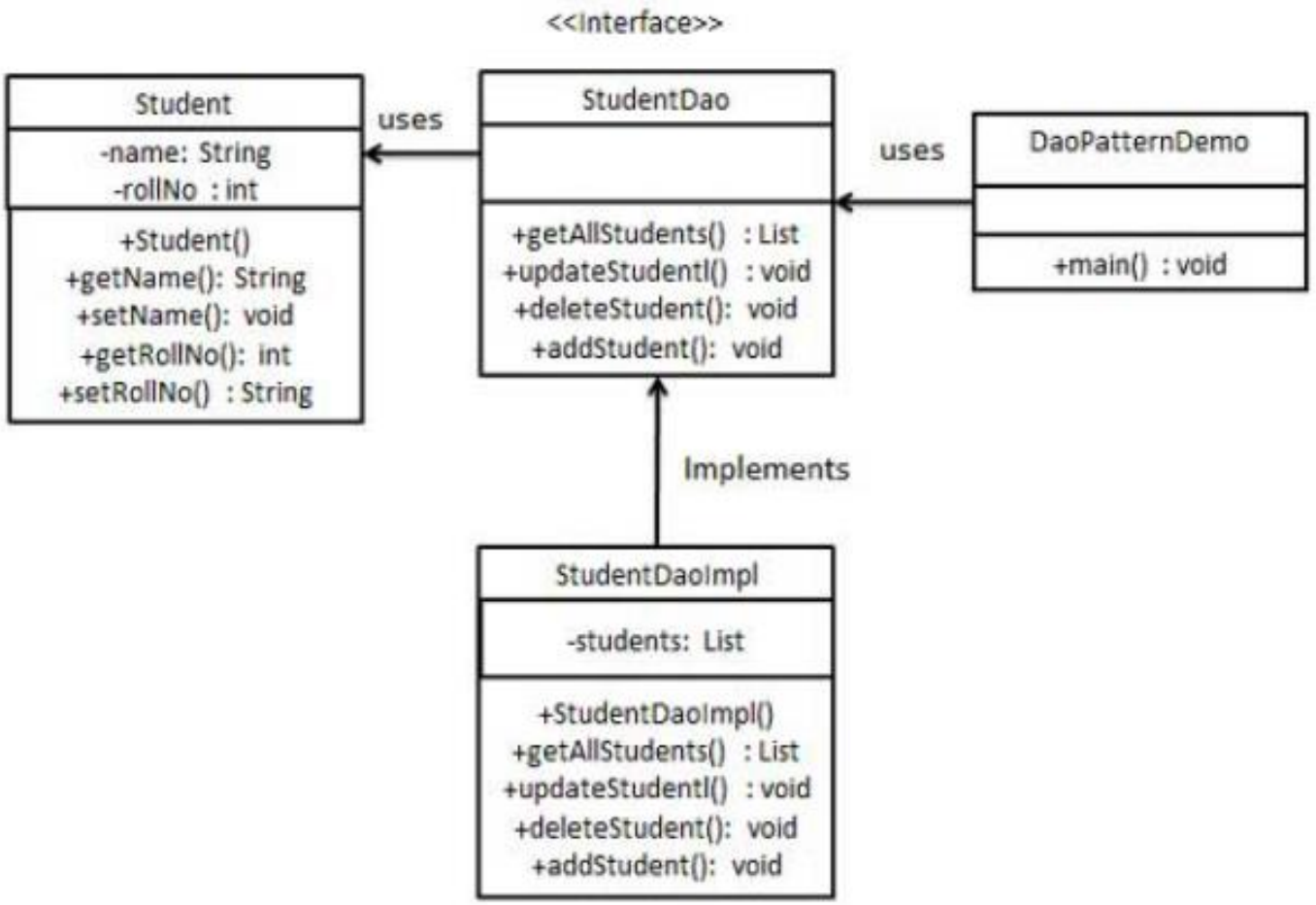
Has the following:

- **Data Access Object Interface -** This interface defines the standard operations to be performed on a model object(s).

- **Data Access Object concrete class -** This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.

- **Model Object or Value Object -** This object is simple POJO containing get/set methods to store data retrieved using DAO class.

Example


<<Interface>>
Student
-name: String
-rollNo : int
+Student()
+getName(): String
+setName(): void
+getRollNo(): int
+setRollNo() : String
uses
StudentDao
+getAllStudents() : List
+updateStudentl() : void
+deleteStudent(): void
+addStudent(): void
uses
DaoPatternDemo
+main() : void
Implements
StudentDaoImpl
-students: List
+StudentDaoImpl()
+getAllStudents() : List
+updateStudentl() : void
+deleteStudent(): void
+addStudent(): void

Student.java

```
public class Student {
   private String name;
   private int rollNo;

   Student(String name, int rollNo){
      this.name = name;
      this.rollNo = rollNo;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public int getRollNo() {
      return rollNo;
   }

   public void setRollNo(int rollNo) {
      this.rollNo = rollNo;
   }
}
```

StudentDao.java

```
import java.util.List;

public interface StudentDao {
   public List<Student> getAllStudents();
   public Student getStudent(int rollNo);
   public void updateStudent(Student student);
   public void deleteStudent(Student student);
}
```

StudentDaoImpl.java

```
import java.util.ArrayList;
import java.util.List;

public class StudentDaoImpl implements StudentDao {

   //list is working as a database
   List<Student> students;

   public StudentDaoImpl(){
      students = new ArrayList<Student>();
      Student student1 = new Student("Robert",0);
      Student student2 = new Student("John",1);
      students.add(student1);
      students.add(student2);
   }
   @Override
   public void deleteStudent(Student student) {
      students.remove(student.getRollNo());
      System.out.println("Student: Roll No " + student.getRollNo() + ", deleted from database");
   }

   //retrive list of students from the database
   @Override
   public List<Student> getAllStudents() {
      return students;
   }
```

StudentDaoImpl.java

```
   @Override
   public Student getStudent(int rollNo) {
      return students.get(rollNo);
   }

   @Override
   public void updateStudent(Student student) {
      students.get(student.getRollNo()).setName(student.getName());
      System.out.println("Student: Roll No " + student.getRollNo() + ", updated in the database");
   }
}
```

DaoPatternDemo.java

```
public class DaoPatternDemo {
   public static void main(String[] args) {
      StudentDao studentDao = new StudentDaoImpl();

      //print all students
      for (Student student : studentDao.getAllStudents()) {
         System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : " + student.getName() + " ]");
      }


      //update student
      Student student =studentDao.getAllStudents().get(0);
      student.setName("Michael");
      studentDao.updateStudent(student);

      //get the student
      studentDao.getStudent(0);
      System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : " + student.getName() + " ]");
   }
}
```

Fin.

Software Construction & Development

WEEK 03

Abeeha Sattar

Event-Driven Programming

What is Event-Driven Programming?

It is a programming paradigm in which the flow of a program is driven by events.

Event Handling

CHAPTER # 24

The Delegation Event Model

It defines standard and consistent mechanisms to generate and process events.

The Concept:

A **source** generates an **event** and sends it to one or more **listeners**

Event

An event is an object that describes a state change in a source.

An event can be generated as a consequence of a person interacting with the elements in a graphical user interface.

- Pressing a button, entering a character via the keyboard, etc.

An event may also be generated when:

- A timer expires
- A counter exceeds a value
- A software or hardware failure occurs
- An operation is completed

Event Sources

A source is an object that generates an event.

- Sources may generate more than one type of event.
- A source must register listeners (why…?)

A listener register method looks like:

```
public void addTypeListener (TypeListener el )
```

Type = name of event

el = reference of the event listener

Event Sources

A source must also provide a method that allows a listener to unregister an interest in a specific type of event.

A listener unregister method looks like:

```
public void removeTypeListener(TypeListener el )
```

Type = name of event

el = reference of the event listener

Multicasting & Unicasting an Event

Multicasting:

When an event occurs, **all** registered listeners are notified and receive a copy of the event object.

Unicasting:

Some sources may allow **only one** listener to register. When an event occurs, the registered listener is notified.

These methods throw an exception when more than one listener try to connect to it.

Event Listeners

A listener is an object that is notified when an event occurs.

It has two requirements:

- Must be registered with one or more listeners
- Must implement methods to receive and process

An event handler must return quickly, and must not maintain control for an extended period. (Why?)

Event Classes

The classes that represent events are at the core of Java's event handling mechanism.

- **EventObject** is a superclass of all events.
- **AWTEvent** is a superclass of all AWT events that are handled by the delegation event model.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 24-1 Commonly Used Event Classes in **java.awt.event**

Event Classes

The ActionEvent Class

An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected.

The AdjustmentEvent Class

An AdjustmentEvent is generated by a scroll bar.

The ComponentEvent Class

A ComponentEvent is generated when the size, position, or visibility of a component is changed.

The ContainerEvent Class

A ContainerEvent is generated when a component is added to or removed from a container.

Event Classes

The FocusEvent Class

A FocusEvent is generated when a component gains or loses input focus.

The InputEvent Class

The abstract class InputEvent is a subclass of **ComponentEvent** and is the superclass for component input events. Its subclasses are **KeyEvent** and **MouseEvent**.

The ItemEvent Class

An ItemEvent is generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected.

Event Classes

The KeyEvent Class

A KeyEvent is generated when keyboard input occurs. (KEY_PRESSED, KEY_RELEASED, KEY_TYPED)

The MouseEvent Class

MOUSE_CLICKED	The user clicked the mouse.
MOUSE_DRAGGED	The user dragged the mouse.
MOUSE_ENTERED	The mouse entered a component.
MOUSE_EXITED	The mouse exited from a component.
MOUSE_MOVED	The mouse moved.
MOUSE_PRESSED	The mouse was pressed.
MOUSE_RELEASED	The mouse was released.
MOUSE_WHEEL	The mouse wheel was moved.

Event Classes

The MouseWheelEvent Class

The MouseWheelEvent class encapsulates a mouse wheel event. It is a subclass of **MouseEvent**.

The TextEvent Class

Instances of this class describe text events. These are generated by text fields and text areas when characters are entered by a user or program.

Event Classes

The WindowEvent Class

WINDOW_ACTIVATED	The window was activated.
WINDOW_CLOSED	The window has been closed.
WINDOW_CLOSING	The user requested that the window be closed.
WINDOW_DEACTIVATED	The window was deactivated.
WINDOW_DEICONIFIED	The window was deiconified.
WINDOW_GAINED_FOCUS	The window gained input focus.
WINDOW_ICONIFIED	The window was iconified.
WINDOW_LOST_FOCUS	The window lost input focus.
WINDOW_OPENED	The window was opened.
WINDOW_STATE_CHANGED	The state of the window changed.

Sources of Events

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 24-2 Event Source Examples

Event Listener Interfaces

Listeners are created by implementing one or more of the interfaces defined by the `java.awt.event` package.

When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 24-3 Commonly Used Event Listener Interfaces

Examples

MOUSE EVENTS & KEYBOARD EVENTS

Adapter Classes

An adapter class provides an empty implementation of all methods in an event listener interface.

Why would we want Adapter Classes?

Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.

Adapter Classes

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener, MouseMotionListener, and MouseWheelListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener, WindowFocusListener, and WindowStateListener

Table 24-4 Commonly Used Listener Interfaces Implemented by Adapter Classes

Example

ADAPTER CLASSES

Inner Classes

A class which is defined within another class, or within an expression.

Anonymous Inner Classses

An anonymous inner class is an inner class that is not assigned a name

```
public AnonymousInnerClassDemo() {
  // Anonymous inner class to handle mouse pressed events.
  addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent me) {
      msg = "Mouse Pressed.";
      repaint();
    }
  });

  // Anonymous inner class to handle window close events.
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent we) {
      System.exit(0);
    }
  });
}
```

This is a constructor

Fin.

Software Construction & Development

WEEK 02

Abeeha Sattar

Layered Architecture

Layered Application Architecture

- Presentation layer
  - Concerned with presenting the results of a computation to system users and with collecting user inputs
- Application processing layer
  - Concerned with providing application specific functionality e.g., in a banking system, banking functions such as open account, close account, etc.
- Data management layer
  - Concerned with managing the system databases

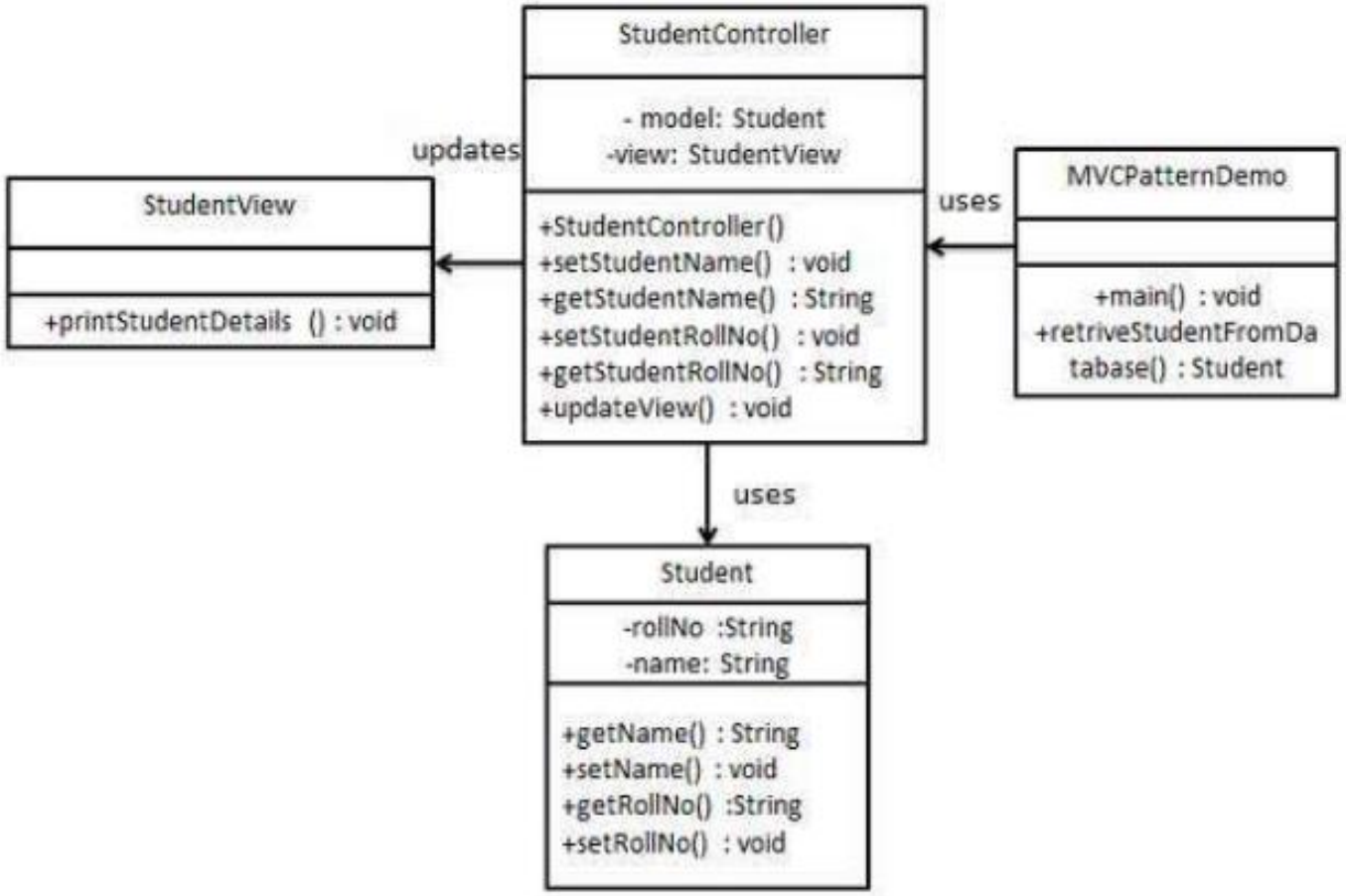
Tiering

- A two-tier architecture is one where a client talks directly to a server, with no intervening server
  - This type of architecture is typically used in small environments with less than 50 users
- A three-tier architecture introduces another server (or an "agent") between the client and the server
  - The role of the middle-tier agent is many-fold - it can provide translation services as in adapting a legacy application on a mainframe to a client/server environnent
  - A plethora of software technologies have evolved to fill the middle tier - middleware

MVC Pattern

- Model
  - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
- View
  - View represents the visualization of the data that model contains.
- Controller
  - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

Example


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

DAO Pattern

- Data Access Object

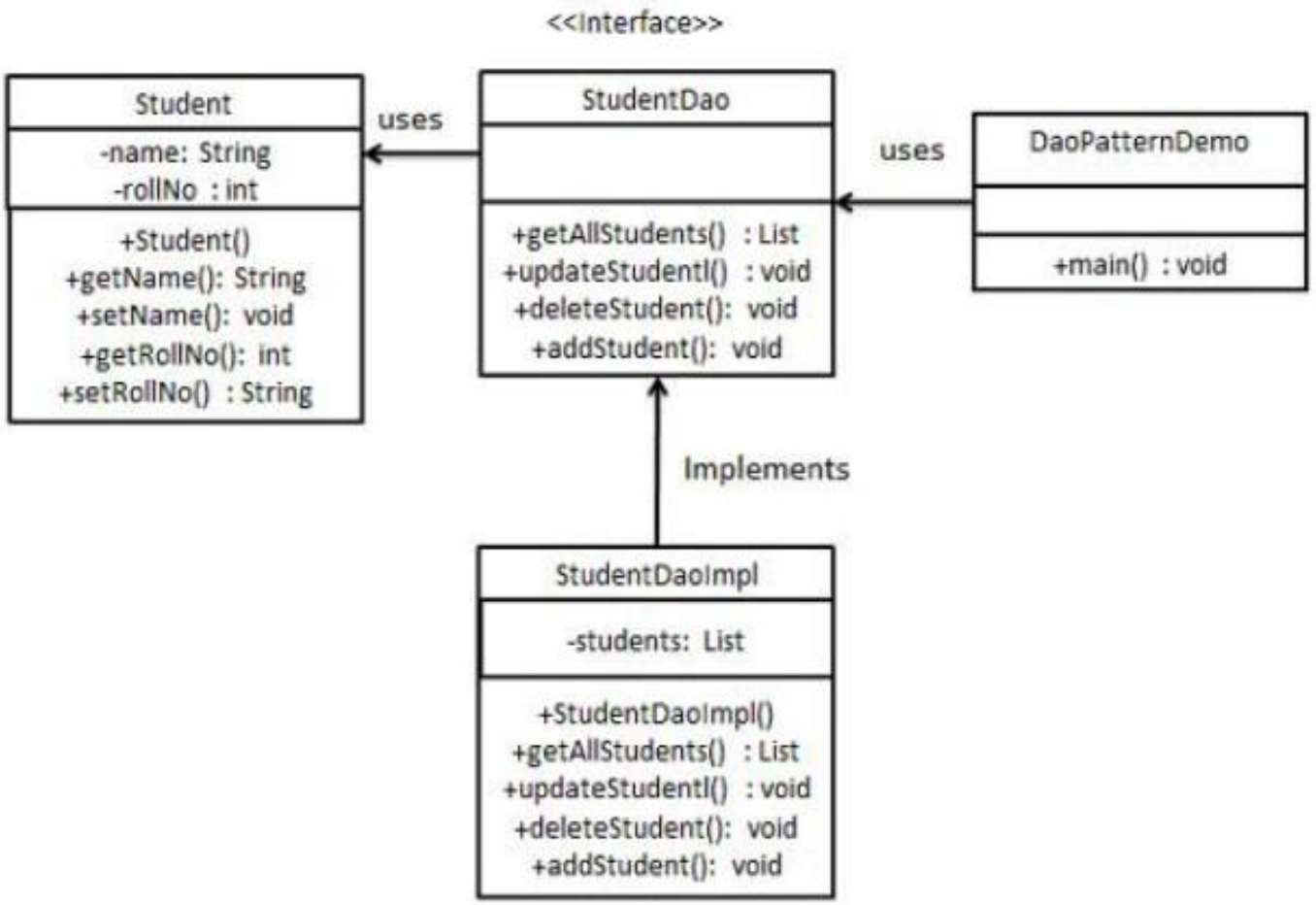
Has the following:

- **Data Access Object Interface -** This interface defines the standard operations to be performed on a model object(s).

- **Data Access Object concrete class -** This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.

- **Model Object or Value Object -** This object is simple POJO containing get/set methods to store data retrieved using DAO class.

Example


<<Interface>>
Student
-name: String
-rollNo : int
+Student()
+getName(): String
+setName(): void
+getRollNo(): int
+setRollNo() : String
uses
StudentDao
+getAllStudents() : List
+updateStudentl() : void
+deleteStudent(): void
+addStudent(): void
uses
DaoPatternDemo
+main() : void
Implements
StudentDaoImpl
-students: List
+StudentDaoImpl()
+getAllStudents() : List
+updateStudentl() : void
+deleteStudent(): void
+addStudent(): void

Fin.