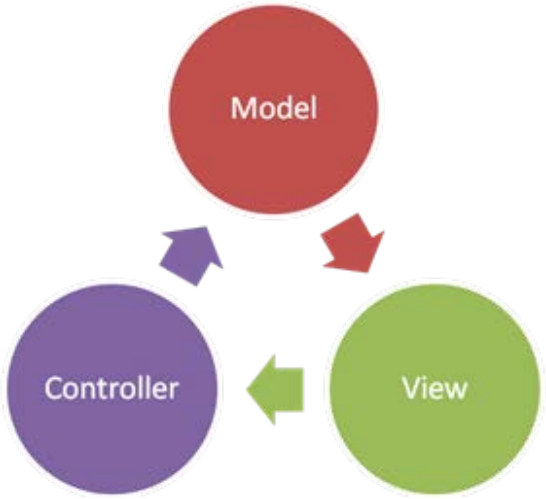
# Tutorial 9 - MVC

Week of 13-17, March, 2017

Prepared by: Gurankash Singh

# Description

- Software architecture pattern that separates the model, the user interface and control logic of an application in three distinct components:




- Separates representation of information from user interaction

# Model

- Implements the logic for the application's data domain
- Logic ensures the integrity of data and allows to derive it
- Example:
  a Product object might retrieve information from a database, operate on it, and then write updated information back to a Products table in a SQL Server database

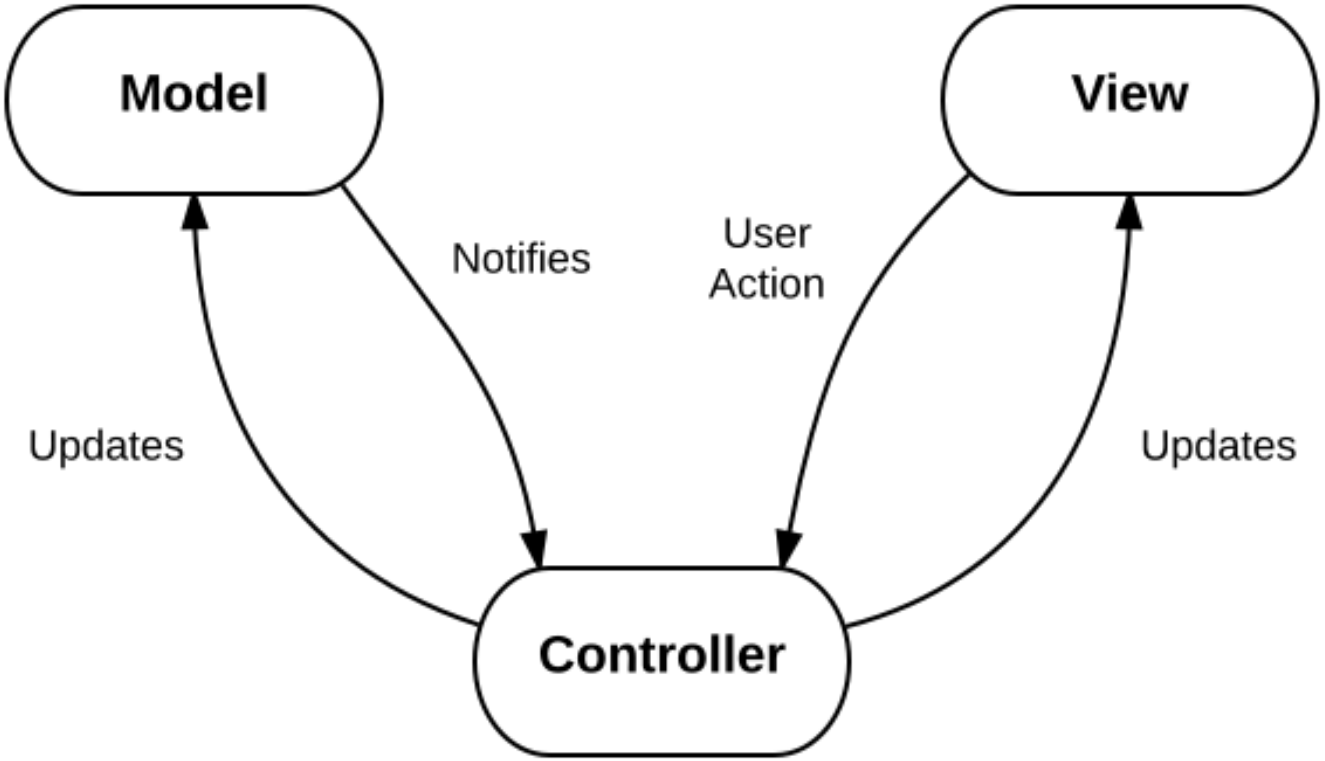
# View

- Represents the model
- Components that display the application's user interface (UI)
- Used to interact and access the data
- Example:
  an edit view of a Products table that displays text boxes, drop-down lists, and check boxes based on the current state of a Product object

# Controller

- Link between view and model
- Components that handle user interaction, invoke changes on the model, and selects a view
- Responsible for receiving and responding to user actions

# Visual Representation

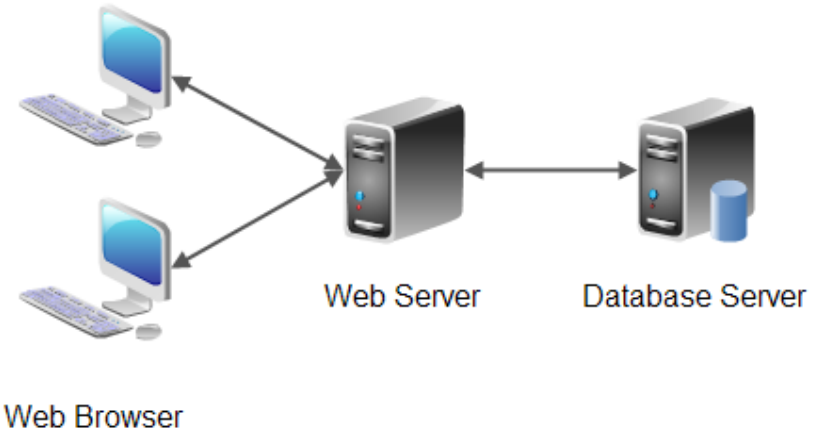


# Control Flow

1. The user performs an action on the interface.
2. The controller takes the input event.
3. The controller notifies the user action to the model, which may involve a change of state of the model.
4. It generates a new view. The view takes the data model.
5. The user interface waits for another user interaction, which starts a new cycle.

# History

- Introduced in 1987 in the Smalltalk programming language
- Fit quite well with Web applications
  - With both the model and the controller on the server side
  - View on the client side



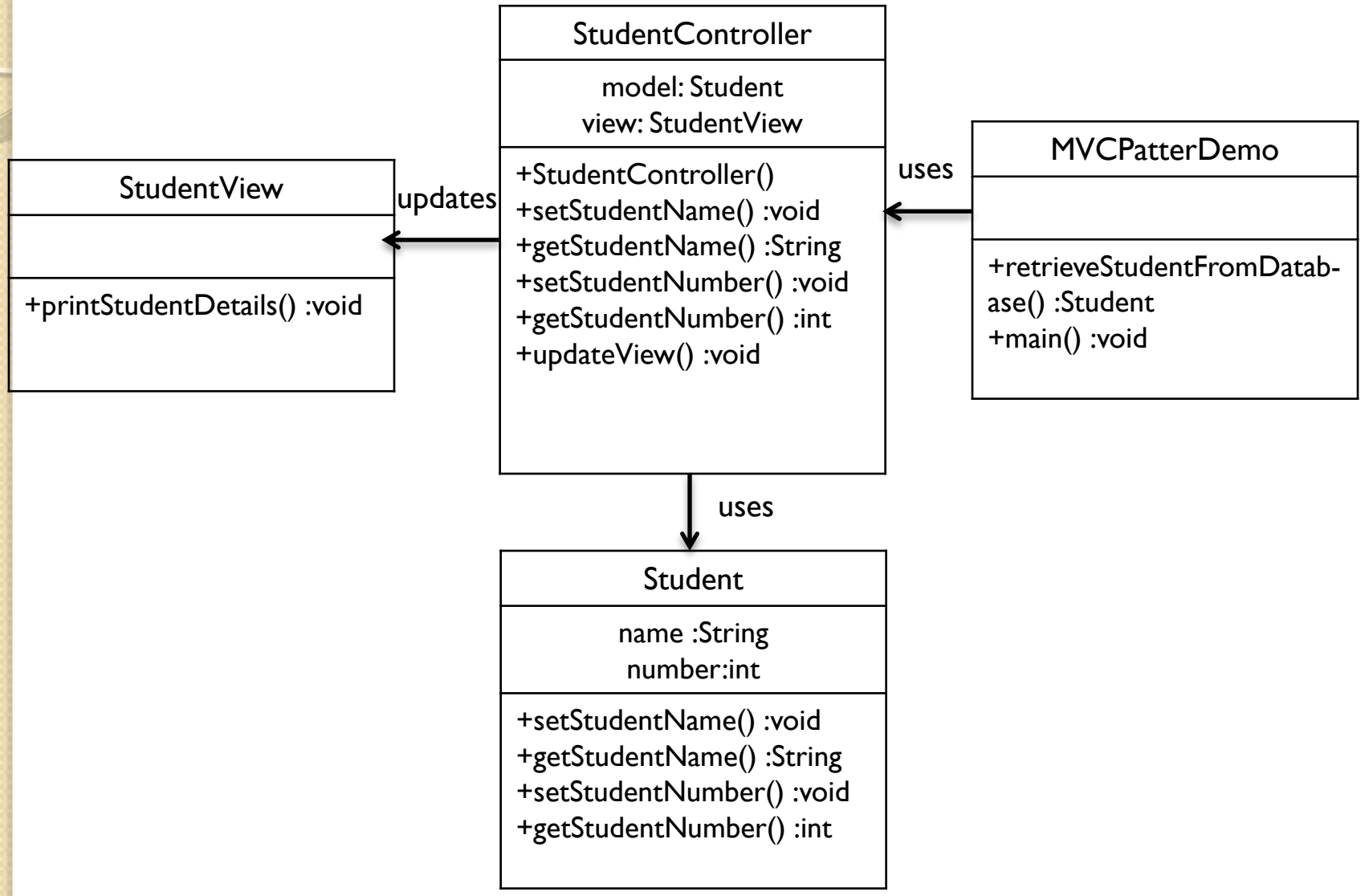
# Benefits

- More organized
- Parallel development allows for rapid application development
- Frequent UI updates can be made without slowing down the business logic process
- Due to the separation of the model from the view, the user interface can display multiple views of the same data at the same time

# Example

- Create a *Student* object
  - Name
  - Student Number
- a view class which can print student details on console and
- A controller class responsible to store data in *Student* object and update view accordingly

# Diagram


StudentController
model: Student
view: StudentView
+StudentController()
+setStudentName() :void
+getStudentName() :String
+setStudentNumber() :void
+getStudentNumber() :int
+updateView() :void
StudentView
+printStudentDetails() :void
updates
MVCPatterDemo
+retrieveStudentFromDatab-
ase() :Student
+main() :void
uses
uses
Student
name :String
number:int
+setStudentName() :void
+getStudentName() :String
+setStudentNumber() :void
+getStudentNumber() :int

# Model – Student.java

```
1  public class Student {
2     private String name;
3     private int number;
4
5
6     public int getStudentNumber() {
7        return number;
8     }
9
10    public void setStudentNumber(int number) {
11       this.number = number;
12    }
13
14    public String getName() {
15       return name;
16    }
17
18    public void setName(String name) {
19       this.name = name;
20    }
21 }
```

# View – StudentView.java

```
1  public class StudentView {
2
3      public void printStudentDetails(String studentName, int studentNumber){
4          System.out.println("---Student---");
5          System.out.println("Name: " + studentName);
6          System.out.println("Student Number: " + studentNumber);
7      }
8
9  }
```

# Controller - StudentController

```
1  public class StudentController {
2     private Student model;
3     private StudentView view;
4
5     public StudentController(Student model, StudentView view){
6        this.model = model;
7        this.view = view;
8     }
9
10    public void setStudentName(String name){
11       model.setName(name);
12    }
13
14    public String getStudentName(){
15       return model.getName();
16    }
17
18    public void setStudentNumber(int studentNumber){
19       model.setStudentNumber(studentNumber);
20    }
21
22    public int getStudentstudentNumber(){
23       return model.getStudentNumber();
24    }
25
26    public void updateView(){
27       view.printStudentDetails(model.getName(), model.getStudentNumber());
28    }
29 }
```

# Main – MVCPatternDemo.java

```java
public class MVCPatternDemo {
    private static Student retriveStudentFromDatabase(){
          Student student = new Student();
          student.setName("Robert");
          student.setStudentNumber(10);
          return student;
    }
   public static void main(String[] args) {

      //fetch student record based on his Student Number 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.setStudentNumber(11);
      controller.updateView();
   }

}
```

# Output

```
---Student---
Name: Robert
Student Number: 10
---Student---
Name: John
Student Number: 11
```

# References

- https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm
- https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

## Source files

- Tutorials\T9\src