Program Design

MVC Architecture

There are many different *design patterns* for programs – ways to organize your code to make it easy to read and easy to change. You will learn more about these design patterns in a later class, CIS 501. Once such design pattern is the **Model-View-Controller architecture**, which has you divide your code into three parts:

- **Model**: stores data, performs calculations
- **View**: handles user input and output
- **Controller**: controls the flow of the program

In this organization, the Model and View components never communicate. Instead, the Controller gets input from the View and gives it to the Model. Then it gets results from the Model and gives them to the View to display.

In larger programs, the Model, View, and Controller component will likely be several classes each. However, in our first examples, we will use one class for each component.

For an example, suppose we want to write a program that determines whether a number is divisible by nine. Numbers are divisible by nine if the sum of their digits are divisible by nine. We will write this program using MVC architecture. We will also allow the user to check whether 10 different numbers are divisible by 9.

Below is a list of the classes we'll need, and which methods and instance variables will go there. It is a very good idea to make a list like this before you start writing any code.

`IO` (View component)

`Scanner` instance variable/constructor to initialize

```
public int getNum()
public void displayDivisible(int num, boolean divisible)
```

`Calculator` (Model component)

```
public boolean getDivisible(int num)
```

`Proj` (Controller component)

Single `main` method – call back and forth between `IO` and `Calculator`

Now, here's the implementation of each class:

```
//In IO.cs
import java.util.*;
public class IO {
      private Scanner s;

      public IO() {
            s = new Scanner(System.in);
```

```
	}

	public int getNum() {
		System.out.print("Enter a positive integer: ");
		return Integer.parseInt(s.nextLine());
	}

	public void displayDivisible(int num, boolean divisible) {
		System.out.print(num + " is ");
		if (!divisible) System.out.print("not ");
		System.out.println("divisible by 9.");
	}
}

//In Calculator.cs
public class Calculator {
	public boolean getDivisible(int num) {
		int sum = 0;
		while (num != 0) {
			int digit = n%10;
			num = num/10;
			sum += digit;
		}
		if (sum % 9 == 0) return true;
		else return false;
	}
}

//In Proj.cs
using System;
public class Proj {
	public static void main(String[] args) {
		//Create IO, Calculator objects
		IO io = new IO();
		Calculator calc = new Calcuator();

		//Test 10 different numbers
		for (int i = 0; i < 10; i++) {
			//Get input number
			int num = io.getNum();

			//Get divisibility
			boolean div = calc.getDivisible(num);

			//Display results
			io.displayDivisible(num, div);
		}
	}
}
```

The first thing you probably noticed on this example is that it is significantly more code than the single-class implementation. And you're absolutely right – for this example, it makes much more sense to just use a single class. However, as your programs get bigger, you will need to divide them up like this to be able to keep track of what's going on. This example illustrates how to break things into several classes so that you'll know how when you do get a big project.

Example: Connect Four

In this example, we will use the Model-View-Controller architecture to implement a Connect Four game. Again, we will first divide this game into model, view, and controller components, and then implement each piece:

`IO` (View component)

`Scanner` instance variable/constructor to initialize

```
public void printBoard(String board)
public int getMove(char piece)
public void printResults(String msg)
```

`Board` (Model component)

`char[][]` instance variable/constructor to initialize

```
public boolean move(int column, char piece)
public boolean full()
public String toString()
public boolean winner(char piece)
```

`ConnectFour` (Controller component)

Single `main` method – call back and forth between `IO` and `Board`.

Now, here's the implementation of each class:

```
import java.util.*;
public class IO {
     private Scanner s;

     public IO() {
          s = new Scanner(System.in);
     }

     public void printBoard(String board) {
          System.out.println("\nCurrent board:\n");
          System.out.println(board);
          System.out.println();
     }

     public int getMove(char piece) {
          System.out.print("User "+piece+", enter a column: ");
```

```
          return Integer.parseInt(s.nextLine());
     }

     public void printResults(String msg) {
          System.out.println(msg+"\n");
     }
}


public class Board {
     private char[][] board;

     public Board() {
          board = new char[6][7];
          for (int i = 0; i < 6; i++) {
               for (int j = 0; j < 7; j++) {
                    board[i][j] = '_';
               }
          }
     }

     public boolean move(int column, char piece) {
          if (column < 0 || column > 6) {
               return false;
          }
          else if (board[0,column] != '_') {
               return false;
          }
          else {
               int i;
               for (i = 5; i>=0 && board[i,column] != '_'; i--);

               board[i][column] = piece;
          }
     }

     public boolean full() {
          for (int i = 0; i < 6; i++) {
               for (int j = 0; j < 7; j++) {
                    if (board[i][j] == '_') return false;
               }
          }

          return true;
     }

     public String toString() {
```

```
        String result = "";
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 7; j++) {
                result += board[i][j] + " ";
            }
            result += "\n";
        }

        return result;
    }

    public boolean winner(char piece) {
        //check row win
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 3; j++) {
                int x;
                for(x = 0; x < 4; x++) {
                    if (board[i][j+x] != piece) break;
                }
                if (x == 4) return true;
            }
        }

        //check column win
        for (int j = 0; j < 7; j++) {
            for (int i = 0; i < 2; i++) {
                int x;
                for (x = 0; x < 4; x++) {
                    if (board[i+x][j] != piece) break;
                }
                if (x == 4) return true;
            }
        }

        //check / diagonals
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 4; j++) {
                int x;
                for (x = 0; x < 4; x++) {
                    if (board[i+x][j+3-x] != piece) break;
                }
                if (x == 4) return true;
            }
        }

        //check \ diagonals
        for (int i = 0; i < 3; i++) {
```

```
                for (int j = 0; j < 4; j++) {
                    int x;
                    for (x = 0; x < 4; x++) {
                        if (board[i+x][j+x] != piece) break;
                    }
                    if (x == 4) return true;
                }
            }

            return false;
        }
}


public class ConnectFour {
    public static void main(String[] args) {
        IO io = new IO();
        Board b = new Board();

        char piece = 'X';
        io.printBoard(b.toString());

        while (b.full() == false) {
            int col = io.getMove(piece);
            boolean worked = b.move(col, piece);
            if (worked) {
                io.printBoard(b.toString());
                if (b.winner(piece) == true) {
                    io.printResults(piece + " wins!");
                    return;
                }
                if (piece == 'X') piece = 'O';
                else piece = 'X';
            }
            else {
                io.printResults("Invalid move");
            }
        }

        //we would have quit the program if someone had won
        io.printResults("Tie game");
    }
}
```