**EXPERIMENT NO. 1**

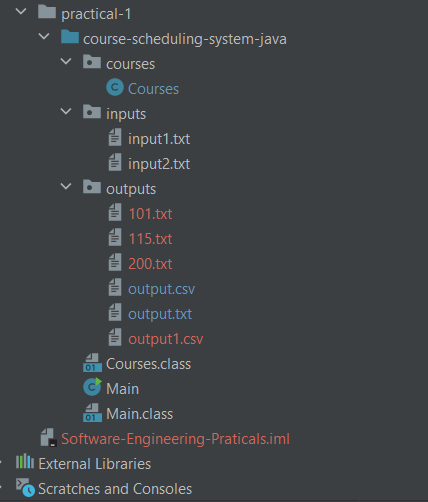
**Implementation of Course Scheduling System using Water Fall Model**

**Aim**

To implement a **Course Scheduling System** using Waterfall Model.

**Code:**

*FileStructure:*

****

*Courses.java*

package courses;  
  
import java.io.\*;  
import java.util.\*;  
import java.util.Map.\*;  
  
public class Courses {  
 protected String[] courses;  
  
 protected HashSet<String> timings = new HashSet<String>();  
  
 // [Course, Enrollment, Preferences]  
 protected Vector<Vector<String>> preferences = new Vector<Vector<String>>();  
  
 // Room -> Capacity  
  
  
 protected HashMap<String, Integer> rooms = new HashMap<String, Integer>();  
  
 //weekday -> course  
// used for each room no  
// protected HashMap<String, String[]> dayCourse = new HashMap<String, String[]>();  
  
 // Time -> Course  
 protected HashMap<String, String> courseTimings = new HashMap<String, String>();  
  
 // Course -> Room  
 protected HashMap<String, Integer> courseRooms = new HashMap<String, Integer>();  
  
 // Courses Constructor : requires 2 input files path  
 public Courses(String file1, String file2) {  
 String section;  
  
 try {  
 // File containing rooms, courses and time  
 Scanner scan = new Scanner(new FileReader(file1));  
  
 while (scan.hasNextLine()) {  
 // Read section  
 section = scan.nextLine();  
  
 // Read Rooms  
 if (section.equals("rooms"))  
 this.readRooms(scan);  
  
 // Read Courses  
 else if (section.equals("courses"))  
 this.readCourses(scan);  
  
 // Read Times  
 else if (section.equals("times"))  
 this.readTimes(scan);  
 }  
  
 // File containing preferences  
 scan = new Scanner(new FileReader(file2));  
  
 // Read Preferences  
 this.readPreferences(scan);  
  
 // Schedule Courses  
 this.schedule();  
  
 scan.close();  
 } catch (IOException e) {  
 // Error Handling  
 System.out.println(e);  
// System.out.println("lol");  
 }  
 }  
  
  
 // Output Result into File  
 public void outputFile() {  
 try {  
 FileWriter outputFile = new FileWriter("practical-1/course-scheduling-system-java/outputs/output.txt");  
  
 outputFile.write("Course\t\tRoom\t\t\tTiming\n");  
 for (Entry<String, String> entry : this.courseTimings.entrySet()) {  
 String course = entry.getValue();  
 String timing = entry.getKey();  
 Integer room = (this.courseRooms.get(course));  
 outputFile.write(course + "\t\t\t" + room + "\t\t\t" + timing + "\n");  
 }  
  
 outputFile.close();  
 } catch (IOException io) {  
 System.out.println(io);  
 }  
 }  
  
 // Output Result into File  
 public void outputFileCsv() {  
 try {  
 FileWriter outputFileCsv = new FileWriter("practical-1/course-scheduling-system-java/outputs/output.csv");  
  
 outputFileCsv.write("Course,Room,Timing\n");  
 for (Entry<String, String> entry : this.courseTimings.entrySet()) {  
 String course = entry.getValue();  
 String timing = entry.getKey();  
 Integer room = (this.courseRooms.get(course));  
 outputFileCsv.write(course + "," + room + "," + timing + "\n");  
 }  
  
 outputFileCsv.close();  
 } catch (IOException io) {  
 System.out.println("IOException occured");  
 }  
 }  
  
 // Output Room Result  
 // output format data structure  
 public HashMap<String, String> makeRoomWiseCourse(Integer room) {  
  
 String[] dayTime = new String[2];  
 String[] courseTime = new String[2];  
 String tempTime;  
  
 // Day -> [Course, Time]  
 HashMap<String, String> dayCourse = new HashMap<String, String>();  
 Vector<String> course = new Vector<>();  
// System.out.println(this.courseRooms);  
// System.out.println(this.courseTimings);  
 for (Entry<String, Integer> entry : this.courseRooms.entrySet()) {  
 if (entry.getValue().equals(room)) {  
 course.add(entry.getKey());  
 }  
 }  
// System.out.println(course);  
// System.out.println(courseTimings);  
 for (String row : course) {  
 for (Entry<String, String> entry : this.courseTimings.entrySet()) {  
  
 if (entry.getValue() == row) {  
 tempTime = entry.getKey();  
 dayTime = tempTime.split("(?<=\\D)(?=\\d)|(?<=\\d)(?=\\D)", 2);  
  
 courseTime = new String[]{row, dayTime[1]};  
 String[] days = dayTime[0].split("");  
// System.out.println(Arrays.toString(days));  
 for (int i = 0; i < days.length; i++) {  
 dayCourse.put(days[i], "\t" + courseTime[0] + "\t\t" + courseTime[1]);  
// System.out.println(Arrays.toString(courseTime));  
 }  
 }  
 }  
 }  
// System.out.println(dayCourse);  
 return dayCourse;  
 }  
  
 public void roomOutput() {  
// System.out.println(rooms);  
// System.out.println("``\_\_\_```\_\_\_```\_\_\_");  
 System.out.println("Roomwise Reports: \n");  
 for (Map.Entry<String, Integer> rElement : rooms.entrySet()) {  
  
 System.out.println("Room - " + rElement.getKey().trim());  
  
 HashMap<String, String> roomData = this.makeRoomWiseCourse(Integer.valueOf(rElement.getKey().trim()));  
  
 System.out.println("Day\t\tCourse\t\tTime");  
  
 for (Map.Entry<String, String> rd : roomData.entrySet()) {  
 System.out.print(rd.getKey() + "\t");  
 System.out.println(rd.getValue());  
// System.out.println("-------------------------------------------------------------------------");  
 }  
 System.out.println("-------------------------------------------------------------------------");  
 System.out.println("");  
 }  
 }  
  
 public void roomOutputFile() {  
 try {  
 for (Map.Entry<String, Integer> rElement : rooms.entrySet()) {  
 String room = rElement.getKey().trim();  
  
 FileWriter fw = new FileWriter("practical-1/course-scheduling-system-java/outputs/" + room + ".txt");  
// System.out.println("Room - " + room);  
  
 fw.write("Day\t\tCourse\t\tTime\n");  
  
 HashMap<String, String> roomData = this.makeRoomWiseCourse(Integer.valueOf(room));  
  
 for (Map.Entry<String, String> rd : roomData.entrySet()) {  
 fw.write(rd.getKey() + "\t\n");  
 fw.write(rd.getValue() + "\n");  
// System.out.println("-------------------------------------------------------------------------");  
 }  
 }  
 } catch (IOException e) {  
 System.out.println(e);  
 }  
 }  
  
// coursewise data  
 public void courseOutputData(){  
 System.out.println("Course Wise Reports: \n");  
// System.out.println(courseRooms);  
 for (Map.Entry<String, Integer> rElement : courseRooms.entrySet()) {  
 System.out.println("Course = " + rElement.getKey()+" room - time");  
 HashMap<String, String> courseData = this.makeCourseWiseRooms(rElement.getKey().trim());  
 for (Map.Entry<String, String> rd : courseData.entrySet()) {  
 System.out.print("Day = " + rd.getKey() + "\t");  
 System.out.println("\t"+rd.getValue());  
// System.out.println("-------------------------------------------------------------------------");  
 }  
 System.out.println("-------------------------------------------------------------------------");  
 System.out.println("");  
 }  
  
  
 }  
  
  
 // output format data structure  
 public HashMap<String, String> makeCourseWiseRooms(String course) {  
  
 String[] dayTime = new String[2];  
  
 String[] roomTime = new String[2];  
 String tempTime;  
 HashMap<String, String> dayCourse = new HashMap<String, String>();  
 Vector<Integer> rooms = new Vector<>();  
// System.out.println(this.courseRooms);  
// System.out.println(course);  
 for (Entry<String, Integer> entry : this.courseRooms.entrySet()) {  
// System.out.println(entry.getKey());  
 if (Objects.equals(entry.getKey(), course)) {  
// System.out.println("here");  
 rooms.add(entry.getValue());  
 }  
 }  
// System.out.println(rooms);  
// System.out.println(course);  
// System.out.println(courseTimings);  
  
 for (Entry<String, String> entry : this.courseTimings.entrySet()) {  
 if (Objects.equals(entry.getValue(), course)) {  
// System.out.println("here again");  
 tempTime = entry.getKey();  
 dayTime = tempTime.split("(?<=\\D)(?=\\d)|(?<=\\d)(?=\\D)", 2);  
 roomTime = new String[]{String.valueOf(rooms.get(0)), dayTime[1]};  
 String[] days = dayTime[0].split("");  
// System.out.println(Arrays.toString(days));  
 for (int i = 0; i < days.length; i++) {  
 dayCourse.put(days[i], Arrays.toString(roomTime));  
// System.out.println(Arrays.toString(roomTime));  
 }  
 }  
 }  
  
// System.out.println(dayCourse);  
 return dayCourse;  
 }  
  
 // Schedule the Courses  
 protected void schedule() {  
 String[] prefsList;  
 String course, availableTiming;  
 boolean foundRoom = false;  
 int enrollmentDiff = Integer.MAX\_VALUE;  
 HashSet<String> tempTimes = (HashSet<String>) timings.clone();  
  
 for (Vector<String> row : this.preferences) {  
 // Get course name  
 course = row.get(0);  
  
 // Find Suitable Room for Course  
 for (Entry<String, Integer> entry : this.rooms.entrySet()) {  
 int enrollment = Integer.parseInt(row.get(1));  
  
 if (entry.getValue() >= enrollment && enrollmentDiff > entry.getValue() - enrollment) {  
 foundRoom = true;  
 this.courseRooms.put(course, Integer.parseInt(entry.getKey().trim()));  
 }  
 }  
  
 // Room not found  
 if (!foundRoom) {  
 System.out.println("Couldn't find room for course");  
 return;  
 }  
  
 // Check if row has preference  
 if (row.size() > 2) {  
 // Split preference timings  
 prefsList = row.get(2).split(",");  
  
 // Check for available time according to preference  
 availableTiming = getAvailable(prefsList);  
  
 // Check if available course in found  
 if (availableTiming.equals("")) {  
 System.out.println("Couldn't find suicourseTimings allotment according to preferred timing for " + availableTiming);  
 return;  
 }  
  
 // Allot timing for available course  
 tempTimes.remove(availableTiming);  
 this.courseTimings.put(availableTiming, course);  
 } else {  
  
 // Check if sufficient timings are available  
 if (tempTimes.size() < 1) {  
 System.out.println("Insufficient timings for alloted courses");  
 return;  
 }  
  
 // Get first available timing  
 availableTiming = tempTimes.iterator().next();  
  
 // Assign timings  
 tempTimes.remove(availableTiming);  
 this.courseTimings.put(availableTiming, course);  
 }  
 }  
 }  
  
 // Get available timings  
 protected String getAvailable(String[] prefsList) {  
 // Search for available timing  
 for (int i = 0; i < prefsList.length; i++) {  
 // Found available timing  
 if (!this.courseTimings.containsKey(prefsList[i]))  
  
 // Cour  
 return prefsList[i];  
 }  
  
 // Couldn't find preferred timing  
 return "";  
 }  
  
 // Read Rooms from File  
 protected void readRooms(Scanner scan) {  
 String str;  
 boolean exit = false;  
  
 do {  
 // Read line  
 str = scan.nextLine();  
  
 // Check if last line  
 if (str.contains(";")) {  
 // Replace ;  
 str = str.replace(" ;", "");  
 exit = true;  
 }  
  
 // Split String on ':' to get key and value pair  
 String[] splitString = str.split(" : ");  
  
 // Put key and value pair in HashMap  
 rooms.put(splitString[0], Integer.parseInt(splitString[1]));  
 } while (!exit);  
  
 }  
  
 // Read Courses from File  
 protected void readCourses(Scanner scan) {  
 String str;  
  
 // Get line with list of courses  
 str = scan.nextLine();  
  
 // Remove whitespaces and ;  
 str = str.replace(";", "").replace(" ", "");  
  
 // Split elements by ,  
 courses = str.split(",");  
 }  
  
 // Read Times from File  
 protected void readTimes(Scanner scan) {  
 String str;  
  
 // Get line with list of times  
 str = scan.nextLine();  
  
 // Remove whitespaces and ;  
 str = str.replace(";", "").replace(" ", "");  
  
 // Split elements by ,  
 timings = new HashSet<String>(Arrays.asList(str.split(",")));  
 }  
  
 // Read Preferences from File  
 protected void readPreferences(Scanner scan) {  
 String str;  
  
 // Skip First Line  
 scan.nextLine();  
  
 while (scan.hasNextLine()) {  
 str = scan.nextLine();  
  
 // Remove whitespaces  
 str = str.replace(" ", "");  
  
 // Split on '|'  
 String[] prefs = str.split("\\|");  
  
 // Append String[] to Preferences  
 preferences.add(new Vector<String>(Arrays.asList(prefs)));  
 }  
 }  
  
 // Debugging  
 public void printData() {  
 System.out.println("Rooms\n" + rooms);  
  
 System.out.println("\nCourses");  
 for (String course : courses)  
 System.out.print("\"" + course + "\"" + ", ");  
  
 System.out.println("\n\nTimings");  
 for (String time : timings)  
 System.out.print("\"" + time + "\"" + ", ");  
  
 System.out.println("\n\nPreferences");  
 for (Vector<String> prefs : preferences)  
 System.out.print(prefs + ",");  
  
 System.out.println("\n\nResult");  
 System.out.println(courseTimings);  
  
 // Cour  
 }  
}

*Main.java*

import courses.Courses;  
  
import java.util.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 Courses courses = new Courses("practical-1/course-scheduling-system-java/inputs/input1.txt",  
 "practical-1/course-scheduling-system-java/inputs/input2.txt");  
 String currentDirectory = System.*getProperty*("user.dir");  
// System.out.println("The current working directory is " + currentDirectory);  
 courses.outputFile();  
 courses.outputFileCsv();  
 courses.roomOutput();  
// courses.roomOutputFile();  
 courses.courseOutputData();  
// Sys  
// tem.out.println(courses.makeRoomWiseCourse(115));  
// System.out.println(courses.makeCourseWiseRooms("cs101"));  
  
 // // courses.printData();  
 }  
}

**Input Files:**

*Input1.txt*

rooms  
 101 : 25  
 115 : 50  
 200 : 250 ;  
courses  
 cs101, cs102, cs110, cs120, cs220, cs412, cs430, cs612, cs630 ;  
times  
 MWF9, MWF10, MWF11, MWF2, TT9, TT10:30, TT2, TT3:30 ;

*Input2.txt*

course | enrollment | preferences  
cs101 | 180 | MWF9, MWF10, MWF11, TT9  
cs412 | 80 | MWF9, TT9, TT10:30  
cs612 | 35 |  
cs630 | 40 |

**Outputs:**

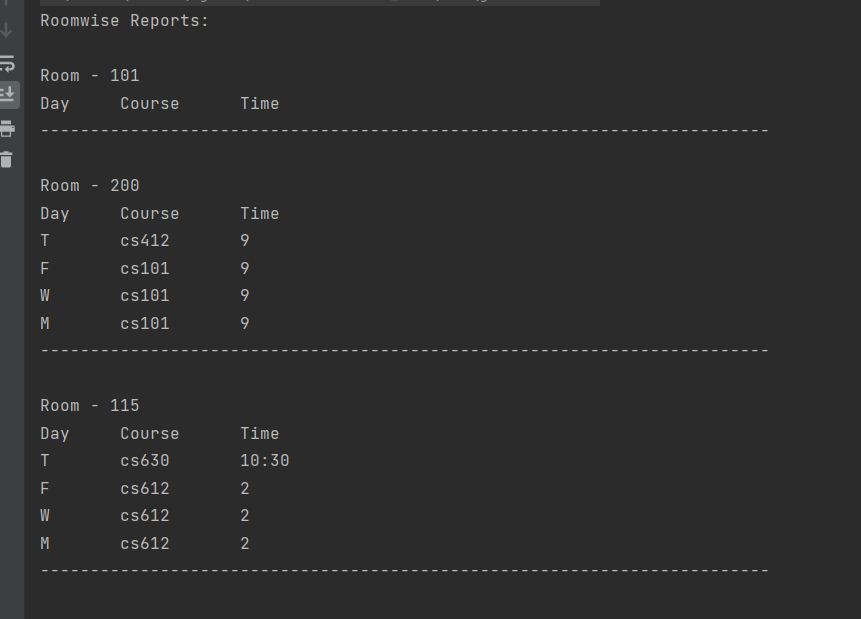
*Output.txt*

Course Room Timing  
cs612 115 MWF2  
cs630 115 TT10:30  
cs412 200 TT9  
cs101 200 MWF9

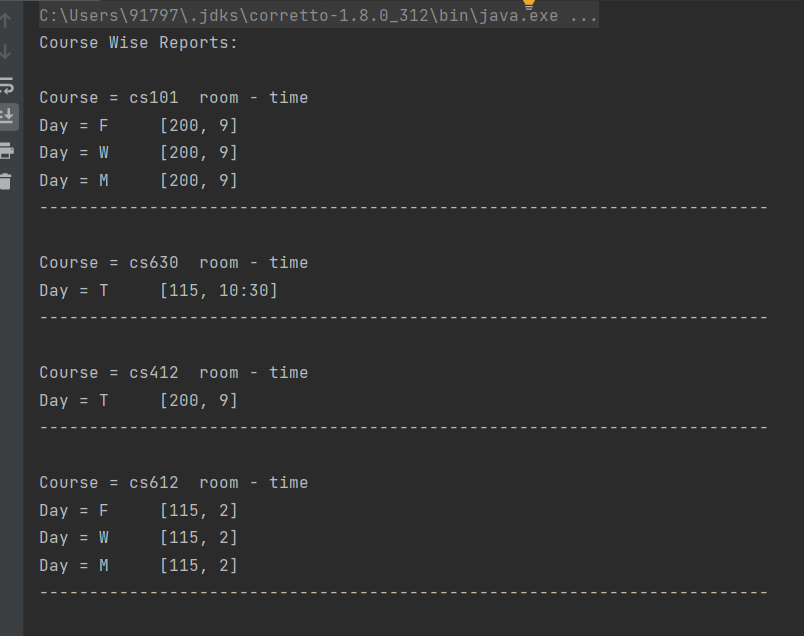
*Output.csv*

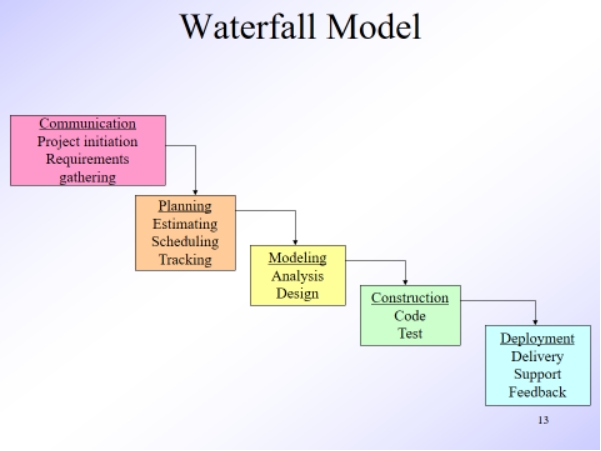
Course,Room,Timing  
cs612,115,MWF2  
cs630,115,TT10:30  
cs412,200,TT9  
cs101,200,MWF9

*Roomwise Output*

**

*Coursewise output*

****

**Conclusion : The waterfall** 

WaterFall model of software engineering is the traditional method of development , but it works well for projects of only small size , for larger projects agile methodology is preferred.