# Refactor 1

From

|  |
| --- |
| readTimeslots  printSchedule  from Schedule.java to IO.java |

Schedule.java

From

|  |
| --- |
| readTimeslots(timeslots, inputFile); |

To

|  |
| --- |
| IO.readTimeslots(timeslots, inputFile); |

Schedule.java

From

|  |
| --- |
| if (rc.isFulfilled()) {printSchedule(validPermutatedUniqueCourseTimeslotsList.get(i)); break; }//System.out.println(); count++;} |

To

|  |
| --- |
| if (rc.isFulfilled()) {IO.printSchedule(validPermutatedUniqueCourseTimeslotsList.get(i)); break; }//System.out.println(); count++;} |

TestSchedule.java

Add

|  |
| --- |
| Import schedule.IO; |

testCS2205Fri/ testCS2201Tue/testExtractMonday

From

|  |
| --- |
| Schedule.readTimeslots(timeslots, "CS3343\_data.txt"); |

To

|  |
| --- |
| IO.readTimeslots(timeslots, "CS3343\_data.txt"); |

Remove

|  |
| --- |
| printScheduleHeader()  testPrintScheduleHeader() |

testPrintSchedule

From

|  |
| --- |
| Schedule.printSchedule(timeslots); |

To

|  |
| --- |
| IO.printSchedule(timeslots); |

# Add Class Timetable

Before

Schedule.java

|  |
| --- |
| ArrayList<ArrayList<ArrayList<Timeslot>>> allPerm = GeneratePermutations(permutatedUniqueCourseTimeslotsList);  ArrayList<ArrayList<Timeslot>> validPermutatedUniqueCourseTimeslotsList = **new** ArrayList<ArrayList<Timeslot>>();  **for** (ArrayList<Timeslot> i : allPerm.get(0)) {  **boolean** overlap = **false**;  **for** (Timeslot j : i) {  **for** (Timeslot k : i) {  **if** (j.equals(k)) {  **break**;  }  **if** (j.overlap(k)) {  overlap = **true**;  **break**;  }  }  }  **if** (!overlap)  validPermutatedUniqueCourseTimeslotsList.add(i);  }  **int** numValidCombinations = validPermutatedUniqueCourseTimeslotsList.size();  main:  //int count = 0;  **for** (**int** i=0; i < validPermutatedUniqueCourseTimeslotsList.size(); i++) {  ArrayList<Timeslot> l = validPermutatedUniqueCourseTimeslotsList.get(i);  RequiredConstraint rc = **new** RequiredConstraint(l, listOfCrns);  //System.out.println("slot 0 fulfilled: " + rc.isFulfilled());  **if** (rc.isFulfilled())  {  IO.*printSchedule*(validPermutatedUniqueCourseTimeslotsList.get(i));  **break**;  }//System.out.println(); count++;}  }  Sort:  **public** **static** **void** sortByStartTime(ArrayList<Timeslot> timetable, ArrayList<Timeslot> result) { |
| RequiredConstraint  **public** RequiredConstraint(ArrayList<Timeslot> l, ArrayList<String> listOfCrns) {  **boolean** found = **true**;  **for** (String i : listOfCrns) {  **boolean** foundi = **false**;  **for** (Timeslot s : l)  **if** (i.equals(s.getCrn())) {  foundi = **true**;  }  found &= foundi;  }    **this**.fulfilled = found;  }  IO.java  Before  public static void printSchedule(ArrayList<String> timetable) |
| BuildingConstraint  **public** BuildingConstraint(Timetable timeslots, ArrayList<String> listOfBldgs) {  **for** (String i : listOfBldgs) {  **for** (Timeslot s : timeslots {  **if** (i.equals(s.getBuilding())) {  System.*out*.println("Found " + i);  **this**.fulfilled = **false**;  **break**;  }  }  }  } |

TimeConstraint

|  |
| --- |
| **public** TimeConstraint(ArrayList<Timeslot> timetable, HashMap<Integer,ArrayList<Double>> daytimeExcluded) {  **for** (Timeslot i : timetable) {  **if** (daytimeExcluded.containsKey(i.getDay())) {  **for** (**double** j : daytimeExcluded.get(i.getDay())) {  **if** (j < i.getFinishTime() && j >= i.getStartTime()) {  **this**.fulfilled = **false**;  **break**;  }  }  }  }  } |

TimeGapConstraint

|  |
| --- |
| **public** TimeGapConstraint(ArrayList<Timeslot> t, **double** timeDifference) {  ArrayList<Timeslot> r = **new** ArrayList<Timeslot>();  Schedule.*sortByStartTime*(t, r);    **for** (**int** i=0; i<r.size()-1; i++) {  **if** ((r.get(i).getDay() == r.get(i+1).getDay()) && (r.get(i+1).getStartTime() - r.get(i).getFinishTime() > timeDifference)) {  **this**.fulfilled = **false**;  **break**;  }  }  } |

# After

Schedule

|  |
| --- |
| Main  ArrayList<Timetable> validPermutatedUniqueCourseTimeslotsList = **new** ArrayList<Timetable>();  **for** (ArrayList<Timeslot> i : allPerm.get(0)) {  **boolean** overlap = **false**;  **for** (Timeslot j : i) {  **for** (Timeslot k : i) {  **if** (j.equals(k)) {  **break**;  }  **if** (j.overlap(k)) {  overlap = **true**;  **break**;  }  }  }  **if** (!overlap){  Timetable timetable = **new** Timetable(i);  validPermutatedUniqueCourseTimeslotsList.add(timetable);  }    }  Sort:  **public** **static** **void** sortByStartTime(Timetable timetable, ArrayList<Timeslot> result) { |

RequiredConstraint

|  |
| --- |
| **public** RequiredConstraint(Timetable l, ArrayList<String> listOfCrns) {  **boolean** found = **true**;  **for** (String i : listOfCrns) {  **boolean** foundi = **false**;  **for** (Timeslot s : l.getTimeslots())  **if** (i.equals(s.getCrn())) {  foundi = **true**;  }  found &= foundi;  }    **this**.fulfilled = found;  } |

printSchedule

|  |
| --- |
| **public** **static** **void** printSchedule(Timetable timetable) |

BuildingConstraint

|  |
| --- |
| **public** BuildingConstraint(Timetable timeslots, ArrayList<String> listOfBldgs) {  **for** (String i : listOfBldgs) {  **for** (Timeslot s : timeslots.getTimeslots()) {  **if** (i.equals(s.getBuilding())) {  System.*out*.println("Found " + i);  **this**.fulfilled = **false**;  **break**;  }  }  }  } |

TimeConstraint

|  |
| --- |
| **public** TimeConstraint(Timetable timetable, HashMap<Integer,ArrayList<Double>> daytimeExcluded) {  **for** (Timeslot i : timetable.getTimeslots()) {  **if** (daytimeExcluded.containsKey(i.getDay())) {  **for** (**double** j : daytimeExcluded.get(i.getDay())) {  **if** (j < i.getFinishTime() && j >= i.getStartTime()) {  **this**.fulfilled = **false**;  **break**;  }  }  }  }  } |

TimeGapConstraint

|  |
| --- |
| **public** TimeGapConstraint(Timetable timetable, **double** timeDifference) {  ArrayList<Timeslot> r = **new** ArrayList<Timeslot>();  Schedule.*sortByStartTime*(timetable, r);    **for** (**int** i=0; i<r.size()-1; i++) {  **if** ((r.get(i).getDay() == r.get(i+1).getDay()) && (r.get(i+1).getStartTime() - r.get(i).getFinishTime() > timeDifference)) {  **this**.fulfilled = **false**;  **break**;  }  }  } |

sortByStartTime Move from Schedule to Timetable

before

|  |
| --- |
| Schedule:  public void sortByStartTime(Timetable timetable, Timetable result) {  if (timetable.size() == 1) {  result.add(timetable.get(0));  return;  }  double min = Double.MAX\_VALUE;  int minIdx = 0;  for (int i=0; i<timetable.size(); i++) {  if (timetable.get(i).getStartTime()+(timetable.get(i).getDay()-1)\*24 < min) {  min = timetable.get(i).getStartTime()+(timetable.get(i).getDay()-1)\*24;  minIdx = i;  }  }  result.add(timetable.get(minIdx));  timetable.remove(minIdx);  sortByStartTime(timetable, result);  } |

After

|  |
| --- |
| Timetable:  **public** **void** sortByStartTime(Timetable result) {  **if** (**this**.size() == 1) {  result.add(**this**.get(0));  **return**;  }  **double** min = Double.*MAX\_VALUE*;  **int** minIdx = 0;  **for** (**int** i=0; i<**this**.size(); i++) {  **if** (**this**.get(i).getStartTime()+(**this**.get(i).getDay()-1)\*24 < min) {  min = **this**.get(i).getStartTime()+(**this**.get(i).getDay()-1)\*24;  minIdx = i;  }  }  result.add(**this**.get(minIdx));  **this**.remove(minIdx);  **this**.sortByStartTime(result);  }  TimeGapConstraint:  **public** TimeGapConstraint(Timetable timetable, **double** timeDifference) {  Timetable sorted = **new** Timetable();  timetable.sortByStartTime(sorted);    **for** (**int** i=0; i<sorted.size()-1; i++) {  **if** ((sorted.get(i).getDay() == sorted.get(i+1).getDay()) && (sorted.get(i+1).getStartTime() - sorted.get(i).getFinishTime() > timeDifference)) {  **this**.fulfilled = **false**;  **break**;  }  }  } |

# Move all utilities from Schedule to Utilities

Before (omitting functions which moved to Utilities)

|  |
| --- |
| **public** **static** **void** main(String[] args) {  ArrayList<Timeslot> timeslots = **new** ArrayList<Timeslot>();    //String inputFile = "CS3343\_data2.txt";    String inputFile = args[0];    IO.*readTimeslots*(timeslots, inputFile); //Extract method  //System.out.println(timeslots.size());    // validate input  //validateInput(ArrayList<Timeslot> timeslots)    // input and validate constraints  //parseInputFile(inputFile, timeConstraint, timeGapConstraint, requiredConstraint, buildingConstraint);    ArrayList<String> uniqueCourses = allCourses(timeslots);  HashMap<String,HashMap<String,ArrayList<Timeslot>>> uniqueCourseTimeslots = **new** HashMap<String,HashMap<String,ArrayList<Timeslot>>>();  **for** (String i : uniqueCourses) {  ArrayList<Timeslot> t = extractTimeslotsByCode(timeslots, i);  ArrayList<Timeslot> allLectures = extractTimeslotsByType(t, "Lecture");  ArrayList<Timeslot> allTutorials = extractTimeslotsByType(t, "Tutorial");  // filter Timeslot according to constraints  // System.out.println(allLectures.size());  // System.out.println(allTutorials.size());  HashMap<String,ArrayList<Timeslot>> slot = **new** HashMap<String,ArrayList<Timeslot>>();  **if** (allLectures.size()>0)  slot.put("Lecture", allLectures);  **if** (allTutorials.size()>0)  slot.put("Tutorial", allTutorials);  uniqueCourseTimeslots.put(i, slot);  }  //System.out.println(uniqueCourseTimeslots);  HashMap<String,ArrayList<ArrayList<Timeslot>>> permutatedUniqueCourseTimeslots = **new** HashMap<String,ArrayList<ArrayList<Timeslot>>>();  ArrayList<ArrayList<ArrayList<Timeslot>>> permutatedUniqueCourseTimeslotsList = **new** ArrayList<ArrayList<ArrayList<Timeslot>>>();  **for** (String i : uniqueCourses) {  permutatedUniqueCourseTimeslots.put(i, permutate(uniqueCourseTimeslots.get(i).get("Lecture"), uniqueCourseTimeslots.get(i).get("Tutorial")));  permutatedUniqueCourseTimeslotsList.add(permutate(uniqueCourseTimeslots.get(i).get("Lecture"), uniqueCourseTimeslots.get(i).get("Tutorial")));  }  // for (String i : uniqueCourses) {  // System.out.println(i + " - " + permutatedUniqueCourseTimeslots.get(i).size());  // }  //System.out.println(permutatedUniqueCourseTimeslots);  //System.out.println("\*\*\*\*");  //System.out.println(permutatedUniqueCourseTimeslotsList);  //System.out.println("\*\*\*\*");  //  // System.out.println("=========");  //permutate all lists  ArrayList<ArrayList<ArrayList<Timeslot>>> allPerm = GeneratePermutations(permutatedUniqueCourseTimeslotsList);  //System.out.println(allPerm.get(0).size());  //for (ArrayList<Timeslot> i : allPerm.get(0))  //System.out.println(i);  ArrayList<Timetable> validPermutatedUniqueCourseTimeslotsList = **new** ArrayList<Timetable>();  **for** (ArrayList<Timeslot> i : allPerm.get(0)) {  **boolean** overlap = **false**;  **for** (Timeslot j : i) {  **for** (Timeslot k : i) {  **if** (j.equals(k)) {  **break**;  }  **if** (j.overlap(k)) {  overlap = **true**;  **break**;  }  }  }  **if** (!overlap){  Timetable timetable = **new** Timetable(i);  validPermutatedUniqueCourseTimeslotsList.add(timetable);  }    }  **int** numValidCombinations = validPermutatedUniqueCourseTimeslotsList.size();  **if** (numValidCombinations == 0)  System.*out*.println("There is no possible combination i.e. You should remove at least 1 course.");  **else**  System.*out*.println("There are " + numValidCombinations + " possible combinations.");  // filter Timeslot according to constraints  ArrayList<String> listOfCrns = **new** ArrayList<String>();  listOfCrns.add("60002");  listOfCrns.add("50005");  //int count = 0;  **for** (**int** i=0; i < validPermutatedUniqueCourseTimeslotsList.size(); i++) {  Timetable l = validPermutatedUniqueCourseTimeslotsList.get(i);  RequiredConstraint rc = **new** RequiredConstraint(l, listOfCrns);  //System.out.println("slot 0 fulfilled: " + rc.isFulfilled());  **if** (rc.isFulfilled())  {  IO.*printSchedule*(validPermutatedUniqueCourseTimeslotsList.get(i));  **break**;  }//System.out.println(); count++;}  }  //System.out.println(count);        // Export to CSV  // https://support.google.com/calendar/answer/45656?hl=en      // Export to ICS  // http://stackoverflow.com/questions/10551776/time-format-used-in-ics-file    } |

After (Schedule)

|  |
| --- |
| ArrayList<Timeslot> timeslots = **new** ArrayList<Timeslot>();    //String inputFile = "CS3343\_data2.txt";    String inputFile = args[0];    IO.*readTimeslots*(timeslots, inputFile); //Extract method  //System.out.println(timeslots.size());    // validate input  //validateInput(ArrayList<Timeslot> timeslots)    // input and validate constraints  //parseInputFile(inputFile, timeConstraint, timeGapConstraint, requiredConstraint, buildingConstraint);    ArrayList<String> uniqueCourses = Utilities.*allCourses*(timeslots);  HashMap<String,HashMap<String,ArrayList<Timeslot>>> uniqueCourseTimeslots = **new** HashMap<String,HashMap<String,ArrayList<Timeslot>>>();  **for** (String i : uniqueCourses) {  ArrayList<Timeslot> t = Utilities.*extractTimeslotsByCode*(timeslots, i);  ArrayList<Timeslot> allLectures = Utilities.*extractTimeslotsByType*(t, "Lecture");  ArrayList<Timeslot> allTutorials = Utilities.*extractTimeslotsByType*(t, "Tutorial");  // filter Timeslot according to constraints  // System.out.println(allLectures.size());  // System.out.println(allTutorials.size());  HashMap<String,ArrayList<Timeslot>> slot = **new** HashMap<String,ArrayList<Timeslot>>();  **if** (allLectures.size()>0)  slot.put("Lecture", allLectures);  **if** (allTutorials.size()>0)  slot.put("Tutorial", allTutorials);  uniqueCourseTimeslots.put(i, slot);  }  //System.out.println(uniqueCourseTimeslots);  HashMap<String,ArrayList<ArrayList<Timeslot>>> permutatedUniqueCourseTimeslots = **new** HashMap<String,ArrayList<ArrayList<Timeslot>>>();  ArrayList<ArrayList<ArrayList<Timeslot>>> permutatedUniqueCourseTimeslotsList = **new** ArrayList<ArrayList<ArrayList<Timeslot>>>();  **for** (String i : uniqueCourses) {  permutatedUniqueCourseTimeslots.put(i, Utilities.*permutate*(uniqueCourseTimeslots.get(i).get("Lecture"), uniqueCourseTimeslots.get(i).get("Tutorial")));  permutatedUniqueCourseTimeslotsList.add(Utilities.*permutate*(uniqueCourseTimeslots.get(i).get("Lecture"), uniqueCourseTimeslots.get(i).get("Tutorial")));  }  // for (String i : uniqueCourses) {  // System.out.println(i + " - " + permutatedUniqueCourseTimeslots.get(i).size());  // }  //System.out.println(permutatedUniqueCourseTimeslots);  //System.out.println("\*\*\*\*");  //System.out.println(permutatedUniqueCourseTimeslotsList);  //System.out.println("\*\*\*\*");  //  // System.out.println("=========");  //permutate all lists  ArrayList<ArrayList<ArrayList<Timeslot>>> allPerm = Utilities.*GeneratePermutations*(permutatedUniqueCourseTimeslotsList);  //System.out.println(allPerm.get(0).size());  //for (ArrayList<Timeslot> i : allPerm.get(0))  //System.out.println(i);  ArrayList<Timetable> validPermutatedUniqueCourseTimeslotsList = **new** ArrayList<Timetable>();  **for** (ArrayList<Timeslot> i : allPerm.get(0)) {  **boolean** overlap = **false**;  **for** (Timeslot j : i) {  **for** (Timeslot k : i) {  **if** (j.equals(k)) {  **break**;  }  **if** (j.overlap(k)) {  overlap = **true**;  **break**;  }  }  }  **if** (!overlap){  Timetable timetable = **new** Timetable(i);  validPermutatedUniqueCourseTimeslotsList.add(timetable);  }    }  **int** numValidCombinations = validPermutatedUniqueCourseTimeslotsList.size();  **if** (numValidCombinations == 0)  System.*out*.println("There is no possible combination i.e. You should remove at least 1 course.");  **else**  System.*out*.println("There are " + numValidCombinations + " possible combinations.");  // filter Timeslot according to constraints  ArrayList<String> listOfCrns = **new** ArrayList<String>();  listOfCrns.add("60002");  listOfCrns.add("50005");  **for** (**int** i=0; i < validPermutatedUniqueCourseTimeslotsList.size(); i++) {  Timetable l = validPermutatedUniqueCourseTimeslotsList.get(i);  RequiredConstraint rc = **new** RequiredConstraint(l, listOfCrns);  **if** (rc.isFulfilled())  {  IO.*printSchedule*(validPermutatedUniqueCourseTimeslotsList.get(i));  **break**;  }  } |

After (Utilities)

|  |
| --- |
| /\*\* The first session's start time. \*/  **public** **static** **double** *firstTime* = 8.0;    /\*\* The last session's finish time. \*/  **public** **static** **double** *lastTime* = 23.0;  /\*\*  \* All courses.  \*  \* @param timeslots the timeslots  \* @return the array list of all courses  \*/  public static ArrayList<String> allCourses(ArrayList<Timeslot> timeslots) {  ArrayList<String> courses = new ArrayList<String>();  for (Timeslot j : timeslots) {  boolean contains = false;  for(String i: courses){  if(j.getCode().equals(i)){  contains = true;  break;  }  }  if (!contains)  courses.add(j.getCode());  }  return courses;  }  /\*\*  \* Extract timeslots by day.  \*  \* @param timeslots the timeslots  \* @param day the day  \* @return the array list of courses of a particular day  \*/  public static ArrayList<Timeslot> extractTimeslotsByDay(ArrayList<Timeslot> timeslots, Weekday day) {  ArrayList<Timeslot> t = new ArrayList<Timeslot>();  for (Timeslot i : timeslots)  if (i.getDay() == day.getDay())  t.add(i);  return t;  }  /\*\*  \* Extract timeslots by course code.  \*  \* @param timeslots the timeslots  \* @param code the course code  \* @return the array list of courses of a particular course  \*/  public static ArrayList<Timeslot> extractTimeslotsByCode(ArrayList<Timeslot> timeslots, String code) {  ArrayList<Timeslot> t = new ArrayList<Timeslot>();  for (Timeslot i : timeslots)  if (i.getCode().equals(code))  t.add(i);  return t;  }  /\*\*  \* Extract timeslots by session type (lecture/tutorial).  \*  \* @param timeslots the timeslots  \* @param type the type of session  \* @return the array list of courses of a particular type of session  \*/  public static ArrayList<Timeslot> extractTimeslotsByType(ArrayList<Timeslot> timeslots, String type) {  ArrayList<Timeslot> t = new ArrayList<Timeslot>();  for (Timeslot i : timeslots)  if (i.getType().equals(type))  t.add(i);  return t;  }  /\*\*  \* Permutate.  \*  \* @param list1 the first array list to permutate  \* @param list2 the second array list to permutate  \* @return the array list of permutating list1 with list2  \*/  public static ArrayList<ArrayList<Timeslot>> permutate(ArrayList<Timeslot> list1, ArrayList<Timeslot> list2) {  ArrayList<ArrayList<Timeslot>> res = new ArrayList<ArrayList<Timeslot>>();  for (Timeslot i: list1) {  for (Timeslot j: list2) {  ArrayList<Timeslot> t = new ArrayList<Timeslot>();  t.add(i);  t.add(j);  res.add(t);  }  }  return res;  }  /\*\*  \* Permutate array list of array list.  \*  \* @param list1 the first array list of array list to permutate  \* @param list2 the second array list of array list to permutate  \* @return the array list of permutating list1 with list2  \*/  public static ArrayList<ArrayList<Timeslot>> permutateArrayList(ArrayList<ArrayList<Timeslot>> list1, ArrayList<ArrayList<Timeslot>> list2) {  ArrayList<ArrayList<Timeslot>> res = new ArrayList<ArrayList<Timeslot>>();  for (ArrayList<Timeslot> i: list1) {  for (ArrayList<Timeslot> j: list2) {  ArrayList<Timeslot> t = new ArrayList<Timeslot>();  t.addAll(i);  t.addAll(j);  res.add(t);  }  }  return res;  }  /\*\*  \* Generate permutations by splitting an array list by course code and session type recursively.  \*  \* @param list the list storing all sessions  \* @return the array list of all permutated sessions  \*/  public static ArrayList<ArrayList<ArrayList<Timeslot>>> GeneratePermutations(ArrayList<ArrayList<ArrayList<Timeslot>>> list)  {  if (list.size() == 1)  return list;  ArrayList<ArrayList<Timeslot>> t = permutateArrayList(list.get(0), list.get(1));  list.add(t);  list.remove(1);  list.remove(0);  return GeneratePermutations(list);  }  /\*\*  \* To generate an array list of time before a given time t.  \*  \* @param t the given time t  \* @return the array list of time NOT before time t  \*/  public static ArrayList<Double> beforeTime(double t) {  ArrayList<Double> listOfExcludedTime = new ArrayList<Double>();  for (double i = firstTime; i < t; i++) {  listOfExcludedTime.add(i);  }  return listOfExcludedTime;  }  /\*\*  \* To generate an array list of time after a given time t.  \*  \* @param t the given time t  \* @return the array list of time NOT after time t  \*/  public static ArrayList<Double> afterTime(double t) {  ArrayList<Double> listOfExcludedTime = new ArrayList<Double>();  for (double i = t; i < lastTime; i++) {  listOfExcludedTime.add(i);  }  return listOfExcludedTime;  }  /\*\*  \* To generate an array list of time between 2 given times t1 and t2. (t1, t2]  \*  \* @param t1, t2 the given times  \* @return the array list of time NOT between t1 and t2  \*/  public static ArrayList<Double> betweenTime(double t1, double t2) {  ArrayList<Double> listOfExcludedTime = new ArrayList<Double>();  for (double i = t1; i < t2; i++) {  listOfExcludedTime.add(i);  }  return listOfExcludedTime;  } |