# 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**;  }  }  } |