```java
/**
 * The Class Algorithm, containing methods to run either the RGS algorithm
 * for HR or Kiraly's algorithm for HRT, to print the matching and to check
 * the matching for stability
 */
public class Algorithm {

        /** The HR/HRT instance */
        private Instance instance;

        /**
         * Instantiates a new Algorithm object
         * @param instance the HR/HRT instance
         */
        public Algorithm (Instance instance) {
                this.instance = instance;
        }

        /**
         * Executes RGS algorithm for HR / Kiraly's algorithm for HRT
         */
        public void run() {
                // to be completed for Tasks 2/3
        }

        /**
         * Prints the matching to the console
         */
        public void printMatching() {
                // to be completed for Task 1
        }

        /**
         * Checks the matching for stability
         */
        public void checkStability() {
                // to be completed for Task 4
        }

        /**
         * Determines whether we have a valid matching
         * @return true if we have a valid matching, false otherwise
         */
        public boolean checkMatching() {
                // get all doctors and hospitals
                Doctor [] doctors = instance.getAllDoctors();
                Hospital [] hospitals = instance.getAllHospitals();
                // reset number of assignees of each hospital to 0
                for (Hospital h : hospitals)
                        h.resetNumAssignees();
                // iterate over each doctor in turn
                for (Doctor d : doctors) {
                        // check if d is assigned
                        if (d.getAssignment() != null) {
                                // get hospital h that d is assigned to
                                Hospital h = d.getAssignment();
                                // determine whether h finds d acceptable
                                if (h.getRank(d) < 0) {
                                        // h finds d unacceptable, matching invalid
                                        System.out.println("Hospital "+h.getId()+" does not find doctor "+d.getId()+" acceptable!");
                                        return false;
                                }
                                // d is a legal assignee of h
                                h.incrementNumAssignees();
                        }
                }
                // check whether a hospital is oversubscribed
                for (Hospital h : hospitals)
                        if (h.isOverSubscribed()) {
                                System.out.println("Hospital "+h.getId()+" is oversubscribed!");
                                return false;
                        }
                // we have a valid matching
                return true;
        }
}
```

```java
import java.util.ArrayList;
import java.util.Iterator;

/**
 * The Class Doctor, to represent a single doctor
 */
public class Doctor {

        /** The doctor’s id, counting from 1 */
        private int id;

        /** The doctor’s preference list, in preference order */
        private ArrayList<Hospital> preferenceList;

        /** A list iterator over the doctor’s preference list */
        private Iterator<Hospital> iterator;

        /** The doctor’s assigned hospital, or null if unassigned */
        private Hospital assignment;

        /**
         * Instantiates a new Doctor
         * @param i the Doctor’s id
         */
        public Doctor(int i) {
                id = i;
                preferenceList = new ArrayList<Hospital>();
                assignment = null;
        }

        /**
         * Gets the doctor’s id
         * @return the id
         */
        public int getId() {
                return id;
        }

        /**
         * Gets the doctor’s preference list
         * @return the preference list
         */
        public ArrayList<Hospital> getPreferenceList() {
                return preferenceList;
        }

        /**
         * Adds a hospital to the end of the doctor’s preference list
         * @param hospital the hospital to be added
         */
        public void addPref(Hospital hospital) {
                preferenceList.add(hospital);
        }

        /**
```

```java
	 * Sets the iterator to the start of the doctor’s preference list
	 */
	public void setIterator() {
		iterator = preferenceList.iterator();
	}

	/**
	 * Gets the doctor’s assigned hospital, or null if unassigned
	 * @return the assignment
	 */
	public Hospital getAssignment() {
		return assignment;
	}

	/**
	 * Assigns the doctor to the given hospital
	 * @param hospital the hospital
	 */
	public void assignTo(Hospital hospital) {
		this.assignment = hospital;
	}

	/**
	 * return doctor’s id as String representation
	 */
	public String toString(){
		return Integer.toString(id);
	}
}
```

```java
import java.util.ArrayList;

/**
 * The Class Hospital, to represent a single hospital
 */
public class Hospital {

        /** The hospital’s id, counting from 1 */
        private int id;

        /** The hospital’s preference list, in preference order */
        private ArrayList<Doctor> preferenceList;

        /** The hospital’s ranking list
         *  Given a doctor with id i, rankList[i-1] gives the hospital’s
         *  ranking for that doctor
         */
        private int [] rankList;

        /** The hospital’s capacity */
        private int capacity;

        /** The number of doctors assigned to the hospital */
        private int numAssignees;

        /** The rank of the hospital’s worst assignee
         */
        private int rankOfWorstAssignee;

        /** The number of doctors in the instance */
        public static int numDoctors;

        /**
         * Instantiates a new Hospital
         * @param i the Hospital’s id
         */
        public Hospital(int i) {
                id = i;
                // create empty preference list initially
                preferenceList = new ArrayList<Doctor>();
                // capacity and number of assignees are 0 initially
                capacity = 0;
                resetNumAssignees();
                // instantiate ranking list
                rankList = new int[numDoctors];
                // each doctor initially is given a rank of -1 which means
                // that the hospital finds that doctor unacceptable
                for (int index = 0; index < numDoctors; index++)
                        rankList[index] = -1;
        }

        /**
         * Gets the hospital’s id
         * @return the id
         */
	public int getId() {
		return id;
	}

	/**
	 * Sets the hospital’s capacity
	 * @param capacity the new capacity
	 */
	public void setCapacity(int capacity) {
		this.capacity = capacity;
	}

	/**
	 * Gets the hospital’s preference list
	 * @return the preference list
	 */
	public ArrayList<Doctor> getPreferenceList() {
		return preferenceList;
	}

	/**
	 * Adds a doctor with a given rank to the end of the hospital’s
	 * preference list
	 * @param doctor the doctor to be added
	 * @param rank the rank of the doctor
	 */
	public void addPref(Doctor doctor, int rank) {
		preferenceList.add(doctor);
		rankList[doctor.getId()-1] = rank;
	}

	/**
	 * Finds the rank of the provided doctor in this hospital’s
	 * preference list. Returns -1 if the doctor does not appear in the list
	 * @param doctor the doctor
	 * @return the rank of the doctor in this hospital’s list
	 */
	public int getRank(Doctor doctor) {
		return rankList[doctor.getId()-1];
	}

	/**
	 * Sets the hospital’s number of assignees to 0
	 */
	public void resetNumAssignees() {
		numAssignees = 0;
	}

	/**
	 * Increments the hospital’s number of assignees
	 */
	public void incrementNumAssignees() {
		numAssignees++;
	}
```

```java
        /**
         * Determine whether hospital is oversubscribed
         * @return true if hospital is oversubscribed, false otherwise
         */
        public boolean isOverSubscribed() {
                return (numAssignees > capacity);
        }

        /**
         * return hospital’s id as String representation
         */
        public String toString() {
                return Integer.toString(id);
        }
}
```

```
/**
 * The Class Instance, to represent an HR / HRT problem instance
 */
public class Instance {

        /** The array of Doctor objects */
        private Doctor [] doctors;

        /** The array of Hospitals objects */
        private Hospital [] hospitals;

        /**
         * Instantiates a new instance
         * @param numDoctors the number of doctors
         * @param numHospitals the number of hospitals
         */
        public Instance(int numDoctors, int numHospitals) {
                // record the number of doctors in a static variable
                // of class Hospital
                Hospital.numDoctors = numDoctors;

                // instantiate Doctor and Hospital arrays
                doctors = new Doctor[numDoctors];
                hospitals = new Hospital[numHospitals];

                // instantiate Doctor and Hospital objects within arrays
                for (int index = 1; index <= numDoctors; index++)
                        doctors[index - 1] = new Doctor(index);
                for (int index = 1; index <= numHospitals; index++)
                        hospitals[index - 1] = new Hospital(index);
        }

        /**
         * Gets the array of Doctor objects
         * @return the array of Doctor objects
         */
        public Doctor [] getAllDoctors() {
                return doctors;
        }

        /**
         * Gets the array of Hospital objects
         * @return the array of Hospital objects
         */
        public Hospital [] getAllHospitals() {
                return hospitals;
        }

        /**
         * Gets the Doctor object with a given id, assumes id counts from 1
         * @param id the Doctor’s id
         * @return the Doctor object with the given id
         */
        public Doctor getDoctorById(int id) {
                return doctors[id - 1];
        }

        /**
         * Gets the Hospital object with a given id, assumes id counts from 1
         * @param id the Hospital’s id
         * @return the Hospital object with the given id
         */
        public Hospital getHospitalById(int id) {
                return hospitals[id - 1];
        }
}
```

```java
/**
 * Main class containing main method
 */
public class Main {
        /**
         * The main method
         * @param args the command-line arguments
         */
        public static void main(String[] args) {
                // parse instance from first input file
                Parser parser = new Parser();
                Instance instance = parser.parseInstance(args[0]);
                // create Algorithm object, supplying instance
                Algorithm algorithm = new Algorithm(instance);
                boolean matchingValid;
                if (args.length > 1) // matching given
                        // parse matching from second input file
                        matchingValid = parser.parseMatching(args[1]);
                else {
                        // run RGS algorithm / Kiraly's algorithm
                        algorithm.run();
                        // check constructed matching for validity
                        matchingValid = algorithm.checkMatching();
                }
                if (matchingValid) {
                        // print matching to console
                        algorithm.printMatching();
                        // check matching for stability
                        algorithm.checkStability();
                } else
                    System.out.println("The matching is invalid!");
        }
}
```

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

/**
 * The Class Parser, to handle reading in an instance and a matching
 */
public class Parser {

        /** The HR/HRT instance */
        private Instance instance;

        /**
         * Parses the instance file into an Instance object
         * @param fileName the name of the instance file
         * @return the instance
         */
        public Instance parseInstance(String fileName) {
                try {
                        // open input file
                        File file = new File(fileName);
                        Scanner scanner = new Scanner(file);
                        // firstly obtain numbers of doctors and hospitals
                        String line = scanner.nextLine().trim();
                        int numDoctors  = Integer.parseInt(line);
                        line = scanner.nextLine().trim();
                        int numHospitals = Integer.parseInt(line);
                        // create Instance object
                        instance = new Instance(numDoctors, numHospitals);

                        //read in the doctors' preference lists, line by line
                        for (int index = 1; index <= numDoctors; index++) {
                                line = scanner.nextLine();
                                // get Doctor object with id index
                                Doctor doctor = instance.getDoctorById(index);
                                // split line into tokens delimited by a colon
                                String [] doctorInfo = line.trim().split(":");
                                // first token is Doctor id
                                // second token should be preference list
                                if (doctorInfo.length > 1) {
                                        // split preference list into tokens, delimited by whitespace
                                        String [] preferences = doctorInfo[1].trim().split("\\s+");
                                        // iterate over tokens
                                        for (String preference : preferences) {
                                                // get hospital id
                                                int hospId = Integer.parseInt(preference);
                                                // add corresponding Hospital object to Doctor's preference list
                                                doctor.addPref(instance.getHospitalById(hospId));
                                        }
                                }
                                // initialise Doctor's iterator to start of preference list
                                doctor.setIterator();
                        }

                        //read in the hospitals' capacities and preference lists, line by line
                for (int index = 1; index <= numHospitals; index++) {
                        line = scanner.nextLine();
                        Hospital hospital = instance.getHospitalById(index);
                        String [] hospitalInfo = line.trim().split(":");

                        // first token is hospital id
                        // second token is hospital capacity
                        hospital.setCapacity(Integer.parseInt(hospitalInfo[1].trim()));

                        // determine whether preference list is non-empty
                        if (hospitalInfo.length > 2) {
                                // copy preference list into String, trimming leading / trailing whitespace
                                String preferences = hospitalInfo[2].trim();
                                // create StringBuilder object from preferences String for faster processing
                                StringBuilder prefs = new StringBuilder(preferences);
                                // keep track of rank, starting from 1 initially
                                int rank = 1;
                                // maintain boolean to determine whether current pref list entry is in a tie
                                boolean inTie = false;
                                // iterate as long as prefs is non-emtpy
                                while (prefs.length() > 0) {
                                        // iterate past a space
                                        if (prefs.charAt(0)==' ')
                                                prefs.delete(0,1);
                                        // if open bracket, we are now entering a tie
                                        else if (prefs.charAt(0)=='(') {
                                                inTie = true;
                                                prefs.delete(0,1);
                                        }
                                        // if close bracket, we are now leaving a tie
                                        else if (prefs.charAt(0)==')') {
                                                inTie = false;
                                                // increment rank for next preference list entry
                                                rank++;
                                                prefs.delete(0,1);
                                        }
                                        else {
                                                // we should have an integer id representing a doctor
                                                int index2;
                                                for (index2 = 0; index2 < prefs.length(); index2++) {
                                                        // read character at position index2 of prefs
                                                        char c = prefs.charAt(index2);
                                                        // if this is not numeric, halt loop
                                                        if (c < '0' || c > '9')
                                                                break;
                                                }
                                                // all characters between 0..(index2-1) inclusive are doctor id
                                                String docIdStr = prefs.substring(0, index2);
                                                int docId = Integer.parseInt(docIdStr);
                                                // add Doctor with given id and rank to Hospital preference list
                                                hospital.addPref(instance.getDoctorById(docId), rank);
                                                // remove Doctor id from prefs ready for parsing to continue
                                                prefs.delete(0,index2);
                                                // if we are not within a tie, rank must increment
                                                if (!inTie)
                                                rank++;
                                        }
                                }
                        }
                }
                scanner.close();
        // catch blocks to deal with potential issues with the input file
        } catch (FileNotFoundException e) {
                System.out.println("File not found!");
                System.exit(0);
        }
        catch (NumberFormatException e) {
                System.out.println("Instance file not formatted correctly!");
                System.exit(0);
        }
        catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Instance file not formatted correctly!");
                System.exit(0);
        }
        return instance;
}

/**
 * Parses the matching file and populates the existing
 * instance with the matching read in
 * @return true if the matching is valid, false otherwise
 */
public boolean parseMatching(String fileName) {
        String line="";
        try {
                // open input file
                File file = new File(fileName);
                Scanner scanner = new Scanner(file);

                // read in the matching line by line
                while (scanner.hasNextLine()) {
                        // read next line
                        line = scanner.nextLine();
                        // split line into tokens delimited by brackets,
                        // commas and spaces
                        String [] tokens = line.trim().split("[(), ]+");
                        // second token should be doctor id
                        int doctorId = Integer.parseInt(tokens[1]);
                        // third token should be hospital id
                        int hospitalId = Integer.parseInt(tokens[2]);
                        // get Doctor and Hospital objects from ids
                        Doctor doctor = instance.getDoctorById(doctorId);
                        Hospital hospital = instance.getHospitalById(hospitalId);
                        // determine whether hospital finds doctor acceptable
                        if (hospital.getRank(doctor) < 0) {
                                // hospital finds doctor unacceptable, matching invalid
                                System.out.println("Hospital "+hospital.getId()+" finds doctor "+doctor.getId()+" unacceptable!");
                                return false;
                        }
                        // determine whether doctor is already assigned
```

```
					else if (doctor.getAssignment() != null) {
						System.out.println("Doctor "+doctor.getId()+" is multiply assigned!");
						return false;
					}
					else {
						// doctor is a legal assignee of hospital
						doctor.assignTo(hospital);
						hospital.incrementNumAssignees();
					}

				}
				// now get all hospitals
				Hospital [] hospitals = instance.getAllHospitals();
				// check whether a hospital is oversubscribed
				for (Hospital hospital : hospitals)
					if (hospital.isOverSubscribed()) {
						System.out.println("Hospital "+hospital.getId()+" is oversubscribed!");
						return false;
					}
				// we have a valid matching
				scanner.close();
			// catch blocks to deal with potential issues with the input file
			} catch (FileNotFoundException e) {
				System.out.println("File not found!");
				System.exit(0);
			}
			catch (NumberFormatException e) {
				System.out.println("Matching file not formatted correctly!");
				System.out.println(line);
				System.exit(0);
			}
			catch (ArrayIndexOutOfBoundsException e) {
				System.out.println("Matching not consistent with instance!");
				System.out.println(line);
				System.exit(0);
			}
			return true;
	}
}
```