

BACS2063 Data Structures and Algorithms

**ASSIGNMENT 202305**

| **University system using Entity-Control-Boundary (ECB) architectural and ADT** |
| --- |

| **Declaration**   * I confirm that I have read and complied with all the terms and conditions of Tunku Abdul Rahman University of Management and Technology’s plagiarism policy. * I declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my own properly derived work. |
| --- |

| Student Name | Student ID | Prog / Tut.Grp | Signature |
| --- | --- | --- | --- |
| Chan Li Yang | 22WMR09009 | RDS Y2G2 | cly |
| Ng Wei Khang | 22WMR08982 | RDS Y2G2 | Ng |
| Soh Wei Shen | 22WMR01084 | RDS Y2G2 | Soh |

# Part A: TEAM REPORT

# 

## **1.0 Abstract Data Type (ADT)**

### **1.1 ADT Specification**

| **public void** put(K key, V value); | | | | |
| --- | --- | --- | --- | --- |
|  | Description | | : Adds a new key-value pair to the hash map or updates the value if the key already exists. | |
|  | Postcondition | | ： A new entry has been added to the table. | |
| **public V** remove(K key); | | | | |
|  | Description | | : Removes a key-value pair from the hash map based on the provided key. | |
|  | Precondition | | : The key must exist in the hash map.. | |
|  | Postcondition | | : The key-value pair is removed from the hash map. | |
|  | Return | | : The value associated with the key or null if the key is not found. | |
| **public** V getValue(K key); | | | | |
|  | Description | | : Retrieves the value associated with a given key from the hash map. | |
|  | Precondition | | : The key must exist in the hash map. | |
|  | Postcondition | | : The dictionary remains unchanged. | |
|  | Return | | : The value associated with the key or null if the key is not found. | |
| **public boolean** isFull(); | | | | |
|  | Description | | : Checks if the hash map is full (reached its maximum capacity). | |
|  | Postcondition | | : The dictionary remains unchanged. | |
|  | Return | | : true if the hash map is full; false otherwise. | |
| **public boolean** isEmpty(); | | | | |
|  | | Description | | : Checks if the hash map is empty (contains no key-value pairs). |
|  | | Postcondition | | : The dictionary remains unchanged. |
|  |  |  |  |  |

|  | Return | : true if the hash map is empty; false otherwise. |
| --- | --- | --- |
| **public int** size(); | | |
|  | Description | : Gets the size of the dictionary. |
|  | Postcondition | : The dictionary remains unchanged. |
|  | Return | : the number of entries (key-value pairs) currently in the dictionary. |
| **Public int** capacity**();** | | |
|  | Description | : Gets the capacity of the bucket in the dictionary. |
|  | Postcondition | : The dictionary remains unchanged. |
|  | Return | : Returns the current bucket capacity |
| **public void** clear(); | | |
|  | Description | : Removes all entries from the dictionary. |
|  | Postcondition | : The dictionary remains unchanged. |
| **public** K getKey(**int** index); | | |
|  | Description | : Retrieves the key at a specified index in the hash map. |
|  | Postcondition | : The index is within the valid range of indices (0 to size-1). |
|  | Return | : The key at the specified index or null if the index is out of range. |
| **public int** getHashIndex**(K** key**);** | | |
|  | Description | : Calculates the index in the hash table where a key should be stored or looked up. |
|  | Postcondition | : The dictionary remains unchanged. |
|  | Return | : The index in the hash table where the key is located or should be located. |

### 

### 

### **1.2 ADT Implementation**

#### 1.2.1 **ADT Interface**

/\*\*

\*

\* @author Chan Li Yang

\*

\*/

package adt;

public interface HashMapInterface<K, V> {

public void put(K key, V value);

public K getKey(int index);

public V getValue(K key);

public V remove(K key);

public boolean containsKey(K key);

public int size();

public int capacity();

public boolean isEmpty();

public boolean isFull();

public void clear();

}

#### 1.2.2 **ADT Implementor**

/\*\*

\*

\* @author Chan Li Yang

\*

\*/

package adt;

public class HashMapImplementer<K, V> implements HashMapInterface<K, V> {

private static final int DEFAULT\_CAPACITY = 16;

private static final double LOAD\_FACTOR = 0.75;

private Entry<K, V>[] buckets;

private int size;

public HashMapImplementer() {

this(DEFAULT\_CAPACITY);

}

public HashMapImplementer(int capacity) {

buckets = new Entry[capacity];

size = 0;

}

private int hash(K key) {

return Math.abs(key.hashCode() % buckets.length);

}

@Override

public void put(K key, V value) {

int index = hash(key);

if (buckets[index] == null) {

buckets[index] = new Entry<>(key, value);

size++;

} else {

Entry<K, V> current = buckets[index];

while (current != null) {

if (current.getKey().equals(key)) {

current.setValue(value);

return;

}

current = current.getNext();

}

// Add new entry at the end of the chain

Entry<K, V> newEntry = new Entry<>(key, value);

newEntry.setNext(buckets[index]);

buckets[index] = newEntry;

size++;

}

if ((double) size / buckets.length > LOAD\_FACTOR) {

resize();

}

}

@Override

public K getKey(int index) {

if (index < 0 || index >= buckets.length) {

throw new IndexOutOfBoundsException("Index is out of bounds.");

}

for (Entry<K, V> entry : buckets) {

if (entry != null && hash(entry.getKey()) == index) {

return entry.getKey();

}

}

return null;

}

@Override

public V getValue(K key) {

int index = hash(key);

Entry<K, V> current = buckets[index];

while (current != null) {

if (current.getKey().equals(key)) {

return current.getValue();

}

current = current.getNext();

}

return null;

}

@Override

public V remove(K key) {

int index = hash(key);

Entry<K, V> current = buckets[index];

Entry<K, V> prev = null;

while (current != null) {

if (current.getKey().equals(key)) {

if (prev != null) {

prev.setNext(current.getNext());

} else {

buckets[index] = current.getNext();

}

size--;

return current.getValue();

}

prev = current;

current = current.getNext();

}

return null;

}

@Override

public boolean containsKey(K key) {

int index = hash(key);

Entry<K, V> current = buckets[index];

while (current != null) {

if (current.getKey().equals(key)) {

return true;

}

current = current.getNext();

}

return false;

}

@Override

public int size() {

return size;

}

@Override

public boolean isEmpty() {

return size == 0;

}

@Override

public boolean isFull() {

return (double) size / buckets.length >= LOAD\_FACTOR;

}

@Override

public void clear() {

buckets = new Entry[buckets.length];

size = 0;

}

@Override

public int capacity() {

return buckets.length;

}

//utility function

private void resize() {

Entry<K, V>[] oldBuckets = buckets;

buckets = new Entry[oldBuckets.length \* 2];

size = 0; // Reset size, we'll re-add all entries

for (Entry<K, V> entry : oldBuckets) {

while (entry != null) {

put(entry.getKey(), entry.getValue());

entry = entry.getNext();

}

}

}

private static class Entry<K, V> {

private K key;

private V value;

private Entry<K, V> next;

public Entry(K key, V value) {

this.key = key;

this.value = value;

}

public K getKey() {

return key;

}

public V getValue() {

return value;

}

public void setValue(V value) {

this.value = value;

}

public Entry<K, V> getNext() {

return next;

}

public void setNext(Entry<K, V> next) {

this.next = next;

}

}

}

### **1.3 Reason for Choosing Hash Map ADT**

First and foremost, a HashMap offers effective data retrieval and storage, which is why I'm using it in the tutor management system. Quick access to tutor data based on distinctive identifiers, such as tutor IDs, is essential in a tutor management system. Because a HashMap supports constant-time lookup, the amount of time it takes to obtain specific tutor information is constant regardless of the quantity of the dataset. This is especially helpful when working with lots of tutors.

Second, HashMaps can assist with data integrity and duplication prevention. You can use tutor IDs as keys to ensure that each tutor is individually identifiable because HashMaps do not permit duplicate keys. Inadvertently saving numerous teachers with the same ID could cause confusion or system issues, thus this prevents it from happening.

Last but not least, a HashMap can increase the effectiveness of activities involving updating and searching for tutor data. Finding and changing a specific tutor's information becomes a constant-time procedure with a properly selected key (such tutor ID). When working with a large dataset, this can considerably improve the Tutor Management System's overall performance and responsiveness.

# Part B: INDIVIDUAL REPORTS

## **2.1 Tutor Management Subsystem**

| Student Name | Student ID | Prog / Tut.Grp | Signature |
| --- | --- | --- | --- |
| Chan Li Yang | 2209009 | RDS2G2 | cly |

Sub-system : Tutor Management Subsystem

### 2.1.1 Source code for Control class (Tutor)

/\*\*

\*

\* @author Chan Li Yang

\*

\*/

package control;

import adt.\*;

import boundary.CourseUI;

import entity.\*;

import java.util.ArrayList;

import java.util.InputMismatchException;

import java.util.Iterator;

import java.util.List;

import java.util.Scanner;

public class TutorManager{

private HashMapInterface<Integer, Tutor> tutorHashMap;

private SortedArrayList<Course> courseList;

private CourseManager CourseManager;

private int tutorIdCounter = 1;

public TutorManager(CourseManager CourseManager) {

this.CourseManager = CourseManager;

tutorHashMap = new HashMapImplementer<>();

courseList = CourseManager.getCourseList();

// Add initial tutors with their associated courses

addInitialTutors();

}

private void addInitialTutors() {

SortedArrayList<String> course1 = new SortedArrayList<>();

course1.add("BAMS2614");

course1.add("BACS2063");

Tutor tutor1 = new Tutor(101, "Heng Ong Huat", "heng101@tarc.edu.my", "012-2345789", course1, 3, "Master");

SortedArrayList<String> course2 = new SortedArrayList<>();

course2.add("BACS2063");

course2.add("BACS3074");

Tutor tutor2 = new Tutor(102, "Kwan Wei Kien", "kwan102@tarc.edu.my", "017-9876543", course2, 3, "Degree");

SortedArrayList<String> course3 = new SortedArrayList<>();

course3.add("BACS3074");

course3.add("BMMS2074");

course3.add("BACS3183");

Tutor tutor3 = new Tutor(103, "Chin Zi Quan", "chin103@tarc.edu.my", "016-3272949", course3, 1, "Master");

SortedArrayList<String> course4 = new SortedArrayList<>();

course4.add("BMMS2074");

Tutor tutor4 = new Tutor(104, "Hoo Sye Qi", "hoo104@tarc.edu.my", "010-8936816", course4, 2, "Phd");

SortedArrayList<String> course5 = new SortedArrayList<>();

course5.add("BACS3183");

Tutor tutor5 = new Tutor(105, "Goh Ge Jing", "goh105@tarc.edu.my", "014-5250428", course5, 1, "Phd");

tutorHashMap.put(tutor1.getId(), tutor1);

tutorHashMap.put(tutor2.getId(), tutor2);

tutorHashMap.put(tutor3.getId(), tutor3);

tutorHashMap.put(tutor4.getId(), tutor4);

tutorHashMap.put(tutor5.getId(), tutor5);

}

public void addTutor() {

char confirmation = 'y';

do {

System.out.println("===============================");

System.out.println("+ Add a New Tutor: +");

System.out.println("===============================");

// Gather tutor details from user input

try {

int tutorId;

// Validate tutor ID

do {

tutorId = inputTutorId();

if (tutorHashMap.containsKey(tutorId)) {

System.out.println("Tutor with ID " + tutorId + " already exists. Please choose a different ID (tutorID after)"+tutorId+".");

}

} while (tutorHashMap.containsKey(tutorId));

String name = inputName();

String email = inputEmail();

String phoneNumber = inputPhoneNumber();

SortedArrayList<String> courseTaught =addCoursesToTutor();

int semester = inputSemester();

String degreeTitle = inputDegreeTitle();

// Create a new tutor

Tutor newTutor = new Tutor(tutorId, name, email, phoneNumber, courseTaught, semester, degreeTitle);

// Ask for confirmation

do {

System.out.println("");

System.out.print("Are you sure you want to add this tutor? (Y/N): ");

confirmation = new Scanner(System.in).next().charAt(0);

} while (confirmation != 'N' && confirmation != 'Y' && confirmation != 'n' && confirmation != 'y');

if (confirmation == 'Y' || confirmation == 'y') {

// Add the tutor to the hashmap

tutorHashMap.put(newTutor.getId(), newTutor);

System.out.println("New tutor has been successfully added!\n");

} else {

System.out.println("Tutor not added. Thank you!\n");

}

// Ask if the user wants to add another tutor

do {

System.out.print("Do you want to add another tutor? (Y/N): ");

confirmation = new Scanner(System.in).next().charAt(0);

} while (confirmation != 'N' && confirmation != 'Y' && confirmation != 'n' && confirmation != 'y');

} catch (InputMismatchException e) {

System.out.println("Invalid input. Please enter valid data.");

} catch (Exception e) {

System.out.println("An error occurred: " + e.getMessage());

}

} while (confirmation == 'Y' || confirmation == 'y');

}

public void displayTutor() {

boolean hasRecords = false; // Flag to check if there are records

for (int i = 0; i < tutorHashMap.capacity(); i++) {

Integer tutorId = tutorHashMap.getKey(i);

if (tutorId != null) {

Tutor tutor = tutorHashMap.getValue(tutorId);

System.out.println(tutor.toString());

hasRecords = true; // Set the flag to true as there are records

}

}

if (!hasRecords) {

System.out.println("No records found.");

}

}

public void deleteTutor() {

//show the tutor records to user

displayTutor();

System.out.println("\nEnter Tutor ID to delete: ");

int tutorIdToDelete = new Scanner(System.in).nextInt();

if (tutorHashMap.containsKey(tutorIdToDelete)) {

System.out.println("\n"+tutorHashMap.getValue(tutorIdToDelete));

char confirmation;

do {

System.out.println("Are you sure you want to delete this tutor? (Y/N): ");

confirmation = new Scanner(System.in).next().charAt(0);

} while (confirmation != 'N' && confirmation != 'Y' && confirmation != 'n' && confirmation != 'y');

if (confirmation == 'Y' || confirmation == 'y') {

tutorHashMap.remove(tutorIdToDelete);

System.out.println("Tutor has been successfully removed!");

//display the tutor records again to check whether the tutor has been successfully deleted

displayTutor();

}

} else {

System.out.println("Tutor not found.");

}

}

public void editTutor() {

System.out.println("\nEditing tutor details:");

Scanner scanner = new Scanner(System.in);

displayTutor();

System.out.println("");

System.out.print("Enter tutor ID to edit: ");

int tutorIdToEdit = scanner.nextInt();

scanner.nextLine(); // Consume newline

if (tutorHashMap.containsKey(tutorIdToEdit)) {

Tutor tutorToEdit = tutorHashMap.getValue(tutorIdToEdit);

String name = inputName();

String email = inputEmail();

String phoneNumber = inputPhoneNumber();

SortedArrayList<String> courseTaught = editCoursesToTutor(tutorToEdit); // Pass the tutor object

int semester = inputSemester();

String degreeTitle = inputDegreeTitle();

Tutor updatedTutor = new Tutor(tutorIdToEdit, name, email, phoneNumber, courseTaught, semester, degreeTitle);

tutorHashMap.put(tutorIdToEdit, updatedTutor);

System.out.println("Tutor details updated successfully.");

} else {

System.out.println("Tutor with ID " + tutorIdToEdit + " not found.");

}

}

public void search() {

Scanner scanner = new Scanner(System.in);

System.out.println("");

System.out.println("==================================");

System.out.println("+ Search for a Tutor +");

System.out.println("==================================");

System.out.println("+ Search Options: +");

System.out.println("+ 1. Search by Tutor ID +");

System.out.println("+ 2. Search by Semester +");

System.out.println("+ 3. Search by Course +");

System.out.println("+ 4. Back to Main Menu +");

System.out.println("==================================");

System.out.println("Enter your choice (1/2/3/4): ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter Tutor ID to search: ");

int tutorId = scanner.nextInt();

searchByTutorId(tutorId);

break;

case 2:

System.out.print("Enter Semester to search: ");

int semester = scanner.nextInt();

searchBySemester(semester);

break;

case 3:

displayCourses(); // Display the course list

System.out.print("Enter the number of the course you want to search: ");

int courseChoice = scanner.nextInt();

if (courseChoice >= 1 && courseChoice <= courseList.getNumberOfEntries()) {

String course = courseList.getEntry(courseChoice - 1).getCourseCode();

searchByCourse(course);

} else {

System.out.println("Invalid choice!");

}

break;

case 4:

return; // Exit the report menu and return to the main menu

default:

System.out.println("Invalid choice!");

break;

}

}

public void displayCourses() {

System.out.println("Available Courses:");

for (int i = 0; i < courseList.getNumberOfEntries(); i++) {

System.out.println(i + 1 + ". " + courseList.getEntry(i).getCourseCode() + " - " + courseList.getEntry(i).getCourseName());

}

}

public void searchByTutorId(int tutorId) {

// Search for tutors by tutorId and display the results

Tutor foundTutor = tutorHashMap.getValue(tutorId);

if (foundTutor != null) {

System.out.println("");

System.out.println("Tutor found by Tutor ID:");

System.out.println(foundTutor.toString());

} else {

System.out.println("No tutors found by Tutor ID.");

}

}

public void searchBySemester(int semester) {

// Search for tutors by semester and display the results

SortedArrayList<Tutor> foundTutors = new SortedArrayList<>();

for (int i = 0; i < tutorHashMap.capacity(); i++) {

Integer tutorId = tutorHashMap.getKey(i);

if (tutorId != null) {

Tutor tutor = tutorHashMap.getValue(tutorId);

if (tutor.getSemester() == semester) {

foundTutors.add(tutor);

}

}

}

if (foundTutors.getNumberOfEntries() > 0) {

System.out.println("");

System.out.println("Tutors found by Tutor Semester:");

for (int i = 0; i < foundTutors.getNumberOfEntries(); i++) {

Tutor tutor = foundTutors.getEntry(i);

System.out.println(tutor.toString());

}

} else {

System.out.println("No tutors found by Semester.");

}

}

public void searchByCourse(String course) {

System.out.println("");

System.out.println("Searching for tutors teaching course: " + course);

// Search for tutors by subject and display the results

SortedArrayList<Tutor> foundTutors = new SortedArrayList<>();

for (int i = 0; i < tutorHashMap.capacity(); i++) {

Integer tutorId = tutorHashMap.getKey(i);

if (tutorId != null) {

Tutor tutor = tutorHashMap.getValue(tutorId);

if (tutor.teachesCourse(course)) {

foundTutors.add(tutor);

}

}

}

if (!foundTutors.isEmpty()) {

System.out.println("");

System.out.println("Tutors found by Tutor Course:");

for (int i = 0; i < foundTutors.getNumberOfEntries(); i++) {

Tutor tutor = foundTutors.getEntry(i);

System.out.println(tutor.toString());

}

} else {

System.out.println("No tutors found by course.");

}

}

public void clearAll() {

System.out.println("\nClearing all tutors:");

tutorHashMap.clear();

tutorIdCounter = 1;

System.out.println("All tutors cleared.");

}

public void displayHeader() {

System.out.println("\n=======================================================================================================================");

System.out.println("+ Displaying all tutors: +");

System.out.println("=======================================================================================================================");

System.out.printf("%-8s %-22s %-30s %-15s %-20s %-10s %-20s%n", "ID", "Name", "Email", "Phone", "course", "Semester", "Degree");

}

public void getReport() {

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("==========================================");

System.out.println("+ Select a report to generate +");

System.out.println("==========================================");

System.out.println(" 1. Tutor by Semester Report +");

System.out.println(" 2. Tutor by Course Report +");

System.out.println(" 3. Tutor by Degree Report +");

System.out.println(" 4. Back to Main Menu +");

System.out.println("==========================================");

System.out.print("\nEnter your choice (1/2/3/4): ");

int choice;

try {

choice = scanner.nextInt();

} catch (InputMismatchException e) {

System.out.println("Invalid input! Please enter a valid number (1/2/3/4).");

scanner.nextLine(); // Consume the invalid input

continue; // Skip the rest of the loop and restart

}

switch (choice) {

case 1:

TutorBySemesterReport();

break;

case 2:

generateTutorBySubjectReport();

break;

case 3:

generateTutorByDegreeReport();

break;

case 4:

return; // Exit the report menu and return to the main menu

default:

System.out.println("Invalid choice! Please try again.");

System.out.println("");

break;

}

}

}

public void TutorBySemesterReport() {

System.out.println("=============================================");

System.out.println("+ Tutor by Semester Report +");

System.out.println("=============================================");

for (int semester = 1; semester <= 3; semester++) {

System.out.println("Semester " + semester + " Tutors:");

for (int i = 0; i < tutorHashMap.capacity(); i++) {

Integer tutorId = tutorHashMap.getKey(i);

if (tutorId != null) {

Tutor tutor = tutorHashMap.getValue(tutorId);

if (tutor.getSemester() == semester) {

System.out.println(tutor.toString());

}

}

}

System.out.println("=============================================");

}

}

public void generateTutorBySubjectReport() {

System.out.println("=============================================");

System.out.println("+ Tutor by Course Report +");

System.out.println("=============================================");

SortedArrayList<String> allCourses = new SortedArrayList<>();

// Collect all unique courses taught

for (int i = 0; i < tutorHashMap.capacity(); i++) {

Integer tutorId = tutorHashMap.getKey(i);

if (tutorId != null) {

Tutor tutor = tutorHashMap.getValue(tutorId);

SortedArrayList<String> coursesTaught = tutor.getcoursesTaught();

// Manually add each course to allCourses

for (int j = 0; j < coursesTaught.getNumberOfEntries(); j++) {

String course = coursesTaught.getEntry(j);

if (allCourses.contains(course) == -1) {

allCourses.add(course);

}

}

}

}

// Remove duplicates

SortedArrayList<String> uniqueCourses = new SortedArrayList<>();

for (int i = 0; i < allCourses.getNumberOfEntries(); i++) {

String course = allCourses.getEntry(i);

if (uniqueCourses.contains(course) == -1) {

uniqueCourses.add(course);

}

}

// Generate report for each course

for (int i = 0; i < uniqueCourses.getNumberOfEntries(); i++) {

String course = uniqueCourses.getEntry(i);

System.out.println("Tutors teaching " + course + ":");

for (int j = 0; j < tutorHashMap.capacity(); j++) {

Integer tutorId = tutorHashMap.getKey(j);

if (tutorId != null) {

Tutor tutor = tutorHashMap.getValue(tutorId);

if (tutor.teachesCourse(course)) {

System.out.println(tutor.toString());

}

}

}

System.out.println("---------------------------------------------");

}

}

public void generateTutorByDegreeReport() {

System.out.println("=============================================");

System.out.println("+ Tutor by Degree Report +");

System.out.println("=============================================");

SortedArrayList<String> allDegrees = new SortedArrayList<>();

// Collect all unique degrees

for (int i = 0; i < tutorHashMap.capacity(); i++) {

Integer tutorId = tutorHashMap.getKey(i);

if (tutorId != null) {

Tutor tutor = tutorHashMap.getValue(tutorId);

allDegrees.add(tutor.getDegreeTitle());

}

}

// Remove duplicates

SortedArrayList<String> uniqueDegrees = new SortedArrayList<>();

for (int i = 0; i < allDegrees.getNumberOfEntries(); i++) {

String degree = allDegrees.getEntry(i);

if (uniqueDegrees.contains(degree) == -1) {

uniqueDegrees.add(degree);

}

}

for (int i = 0; i < uniqueDegrees.getNumberOfEntries(); i++) {

String degree = uniqueDegrees.getEntry(i);

System.out.println("Tutors with " + degree + " degree:");

for (int j = 0; j < tutorHashMap.capacity(); j++) {

Integer tutorId = tutorHashMap.getKey(j);

if (tutorId != null) {

Tutor tutor = tutorHashMap.getValue(tutorId);

if (tutor.getDegreeTitle().equals(degree)) {

System.out.println(tutor.toString());

}

}

}

System.out.println("---------------------------------------------");

}

}

public int inputTutorId() {

int tutorId = 0;

boolean validInput = false;

Scanner scanner = new Scanner(System.in);

do {

System.out.print("Enter Tutor ID (3 digits): ");

String input = scanner.nextLine();

if (input.matches("\\d{3}")) {

tutorId = Integer.parseInt(input);

validInput = true;

} else {

System.out.println("Invalid input. Please enter a 3-digit integer.");

}

} while (!validInput);

return tutorId;

}

public String inputName() {

String name;

boolean validInput = false;

Scanner scanner = new Scanner(System.in);

do {

System.out.print("Enter Name: ");

name = scanner.nextLine().trim(); // Trim leading and trailing spaces

name = name.toLowerCase(); // Convert the entire name to lowercase

// Split the name by spaces and capitalize the first letter of each word

String[] words = name.split("\\s+");

StringBuilder formattedName = new StringBuilder();

for (String word : words) {

if (!word.isEmpty()) {

formattedName.append(word.substring(0, 1).toUpperCase()); // Uppercase the first letter

if (word.length() > 1) {

formattedName.append(word.substring(1).toLowerCase()); // Lowercase the rest

}

formattedName.append(" "); // Add space between words

}

}

name = formattedName.toString().trim(); // Remove trailing space

if (name.matches("^[A-Z][a-z]\*( [A-Z][a-z]\*)\*$")) {

validInput = true;

} else {

System.out.println("Invalid input. Please enter a valid name with proper capitalization (e.g., John Doe).");

}

} while (!validInput);

return name;

}

public String inputEmail() {

String email;

boolean validInput = false;

Scanner scanner = new Scanner(System.in);

do {

System.out.print("Enter Email: ");

email = scanner.nextLine();

if (email.matches("^.+@tarc\\.edu\\.my$")) {

validInput = true;

} else {

System.out.println("Invalid input. Please enter a valid email in the format 'example@tarc.edu.my'.");

}

} while (!validInput);

return email;

}

public String inputPhoneNumber() {

String phoneNumber;

boolean validInput = false;

Scanner scanner = new Scanner(System.in);

do {

System.out.print("Enter Phone Number (e.g., 011-12345678 or 010-1234567): ");

phoneNumber = scanner.nextLine();

if (phoneNumber.matches("^(011|010|012|013|014|015|016|017|018|019)-\\d{7,8}$")) {

validInput = true;

} else {

System.out.println("Invalid input. Please enter a valid phone number in the format '011-12345678' or '010-1234567'.");

}

} while (!validInput);

return phoneNumber;

}

public SortedArrayList<String> addCoursesToTutor() {

SortedArrayList<String> selectedCourses = new SortedArrayList<>();

System.out.println("");

displayCourses();

int choice;

do {

System.out.println("");

System.out.print("Enter the number of the course you teach (1-" + courseList.getNumberOfEntries() + "), or 0 to finish: ");

choice = new Scanner(System.in).nextInt();

if (choice >= 1 && choice <= courseList.getNumberOfEntries()) {

String courseCode = courseList.getEntry(choice - 1).getCourseCode();

if (selectedCourses.contains(courseCode) == -1) {

selectedCourses.add(courseCode);

System.out.println(courseCode + " added to your taught courses.");

} else {

System.out.println("You have already selected this course.");

}

} else if (choice < 0 || choice > courseList.getNumberOfEntries()) {

System.out.println("Invalid input. Please enter a number between 1 and " + courseList.getNumberOfEntries() + " or 0 to finish.");

}

} while (choice != 0);

System.out.println("");

return selectedCourses;

}

public SortedArrayList<String> editCoursesToTutor(Tutor tutor) {

SortedArrayList<String> selectedCourses = new SortedArrayList<>();

// Add the current courses taught by the tutor to selectedCourses

System.out.println("\nCurrent Courses Taken by Tutor:");

SortedArrayList<String> coursesTaught = tutor.getcoursesTaught();

for (int i = 0; i < coursesTaught.getNumberOfEntries(); i++) {

selectedCourses.add(coursesTaught.getEntry(i));

System.out.println(coursesTaught.getEntry(i));

}

System.out.println("\nAvailable Courses:");

for (int i = 0; i < courseList.getNumberOfEntries(); i++) {

System.out.println(i + 1 + ". " + courseList.getEntry(i).getCourseCode() + " - " + courseList.getEntry(i).getCourseName());

}

int choice;

do {

System.out.println("");

System.out.print("Enter the number of the course you teach (1-" + courseList.getNumberOfEntries() + "), or 0 to finish: ");

choice = new Scanner(System.in).nextInt();

if (choice >= 1 && choice <= courseList.getNumberOfEntries()) {

String courseCode = courseList.getEntry(choice - 1).getCourseCode();

if (selectedCourses.contains(courseCode) == -1) {

selectedCourses.add(courseCode);

System.out.println(courseCode + " added to your taught courses.");

} else {

System.out.println("You have already selected this course.");

}

} else if (choice < 0 || choice > courseList.getNumberOfEntries()) {

System.out.println("Invalid input. Please enter a number between 1 and " + courseList.getNumberOfEntries() + " or 0 to finish.");

}

} while (choice != 0);

System.out.println("");

return selectedCourses;

}

public int inputSemester() {

int semester=0;

boolean validInput = false;

Scanner scanner = new Scanner(System.in);

do {

System.out.print("Enter Semester (1-3): ");

if (scanner.hasNextInt()) {

semester = scanner.nextInt();

if (semester >= 1 && semester <= 3) {

validInput = true;

} else {

System.out.println("Invalid input. Please enter a semester between 1 and 3.");

}

} else {

System.out.println("Invalid input. Please enter a valid integer between 1 and 3.");

scanner.next(); // Consume invalid input

}

} while (!validInput);

return semester;

}

public String inputDegreeTitle() {

String degreeTitle;

boolean validInput = false;

Scanner scanner = new Scanner(System.in);

do {

System.out.print("Enter Degree Title (Master/PhD/Degree): ");

degreeTitle = scanner.nextLine().toUpperCase(); // Convert to uppercase

if (degreeTitle.equals("MASTER") || degreeTitle.equals("PHD") || degreeTitle.equals("DEGREE")) {

validInput = true;

} else {

System.out.println("Invalid input. Please enter a valid degree title (Master/PhD/Degree).");

}

} while (!validInput);

return degreeTitle;

}

public SortedArrayList<Course> getCourseList() {

return (SortedArrayList<Course>) courseList;

}

}

### 2.1.2 **Source code for Entity class (Tutor)**

/\*\*

\*

\* @author Chan Li Yang

\*

\*/

package entity;

import adt.SortedArrayList;

public class Tutor implements Comparable<Tutor>{

private int id; // Unique identifier for each tutor

private String name;

private String email;

private String phoneNumber;

private SortedArrayList<String> coursesTaught;

private int semester;

private String degreeTitle;

public Tutor(int id, String name, String email, String phoneNumber, SortedArrayList<String> coursesTaught, int semester, String degreeTitle) {

this.id = id;

this.name = name;

this.email = email;

this.phoneNumber = phoneNumber;

this.coursesTaught = coursesTaught;

this.semester = semester;

this.degreeTitle = degreeTitle;

}

// Getters and setters for all attributes

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

public SortedArrayList<String> getcoursesTaught() {

return coursesTaught;

}

public void setcoursesTaught(SortedArrayList<String> coursesTaught) {

this.coursesTaught = coursesTaught;

}

public int getSemester() {

return semester;

}

public void setSemester(int semester) {

this.semester = semester;

}

public String getDegreeTitle() {

return degreeTitle;

}

public void setDegreeTitle(String degreeTitle) {

this.degreeTitle = degreeTitle;

}

// Add a subject to the list of subjects taught

public void addCourse(String course) {

coursesTaught.add(course);

}

// Remove a subject from the list of subjects taught

public void removeCourse(String course) {

coursesTaught.remove(course);

}

// Check if a subject is taught by the tutor

public boolean teachesCourse(String course) {

return coursesTaught.contains(course) != -1;

}

@Override

public String toString() {

return "============================================="+"\n" +

"Tutor ID: " + getId() + "\n" +

"Name: " + getName() + "\n" +

"Email: " + getEmail() + "\n" +

"Phone Number: " + getPhoneNumber() + "\n" +

"Courses Taught: " + getcoursesTaught() + "\n" +

"Semester: " + getSemester() + "\n" +

"Degree Title: " + getDegreeTitle() + "\n"+

"============================================="+"\n" ;

}

@Override

public int compareTo(Tutor other) {

// Compare tutors based on their ID

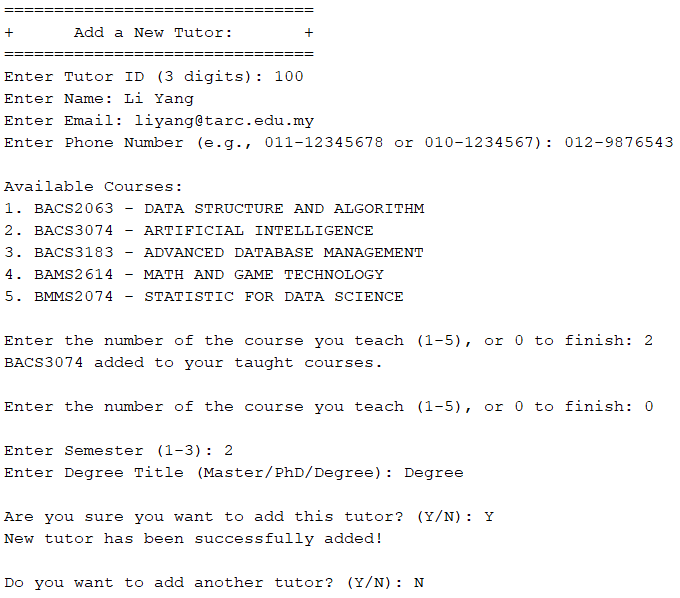
return Integer.compare(this.id, other.id);

}

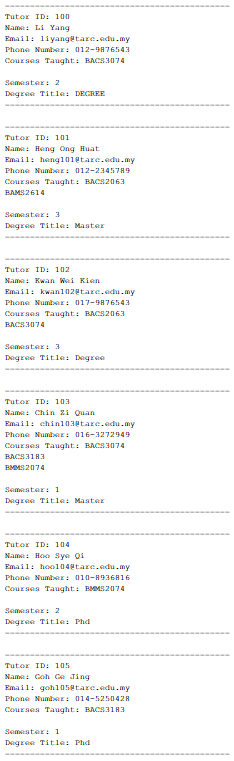
}

### 2.1.3 **Screenshot for Output (Tutor)**

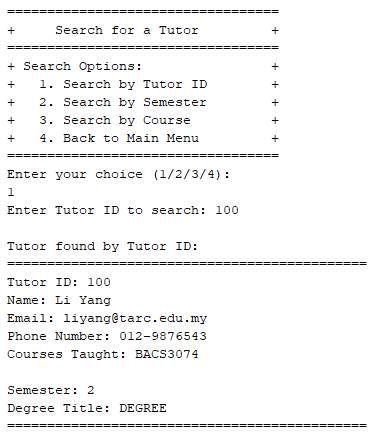
#### **Add Tutor**



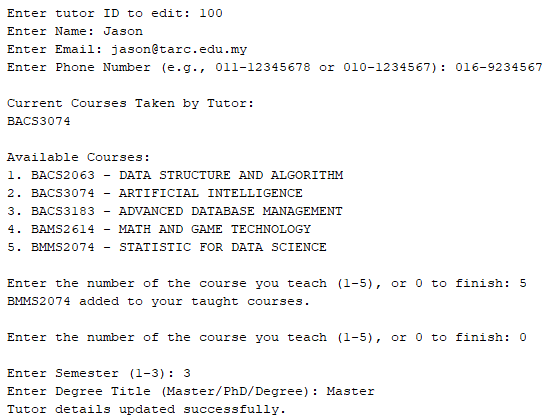
#### **List All Tutor**



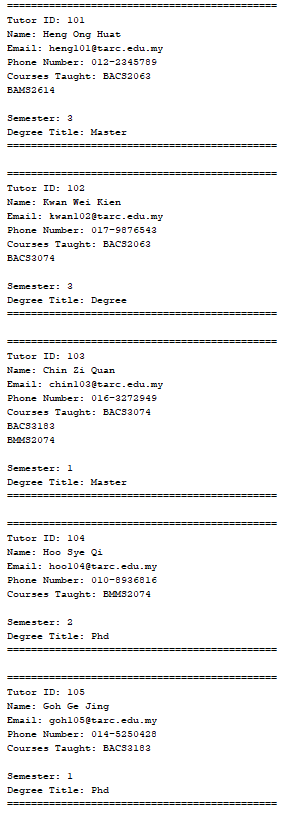
#### **Search Tutor**

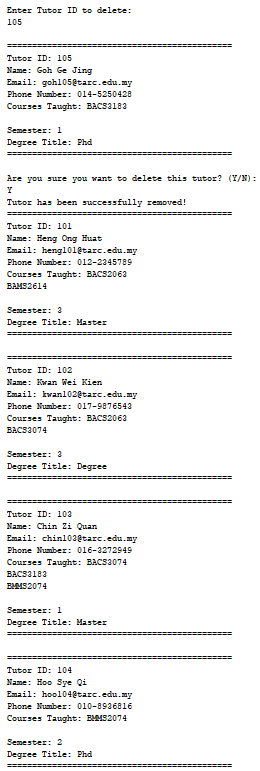


#### **Amend Tutor**

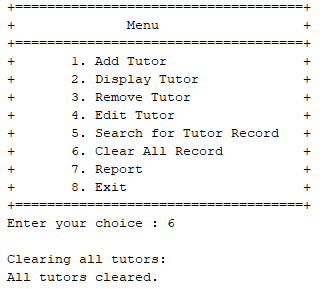


#### **Remove Tutor**

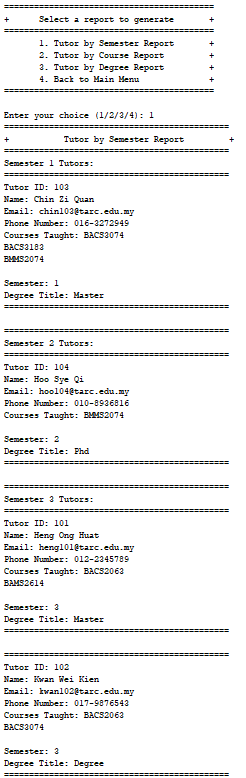




#### **Clear All Tutor**



#### **Tutor Report**



### 2.1.4 **Reason of choosing HashMap Adt (Tutor Management System)**

First and foremost, a HashMap offers effective data retrieval and storage, which is why I'm using it in the tutor management system. Quick access to tutor data based on distinctive identifiers, such as tutor IDs, is essential in a tutor management system. Because a HashMap supports constant-time lookup, the amount of time it takes to obtain specific tutor information is constant regardless of the quantity of the dataset. This is especially helpful when working with lots of tutors.

Second, HashMaps can assist with data integrity and duplication prevention. You can use tutor IDs as keys to ensure that each tutor is individually identifiable because HashMaps do not permit duplicate keys. Inadvertently saving numerous teachers with the same ID could cause confusion or system issues, thus this prevents it from happening.

Last but not least, a HashMap can increase the effectiveness of activities involving updating and searching for tutor data. Finding and changing a specific tutor's information becomes a constant-time procedure with a properly selected key (such tutor ID). When working with a large dataset, this can considerably improve the Tutor Management System's overall performance and responsiveness.

## **2.2 Course Management Subsystem**

| Student Name | Student ID | Prog / Tut.Grp | Signature |
| --- | --- | --- | --- |
| Ng Wei Khang | 2208982 | RDS2G2 | Ng |

Sub-system : Course Management Subsystem

### 2.2.1 **Source code for ADT (Course)**

package adt;

/\*\*

\*

\* @author Ng Wei Khang

\*/

public class SortedArrayList<T extends Comparable<T>> implements SortedArrayListInterface<T> {

private T[] array;

private int numberOfEntries;

private static final int DEFAULT\_CAPACITY = 25;

public SortedArrayList() {

this(DEFAULT\_CAPACITY);

}

public SortedArrayList(int initialCapacity) {

numberOfEntries = 0;

array = (T[]) new Comparable[initialCapacity];

}

public boolean add(T newEntry) {

int i = 0;

while (i < numberOfEntries && newEntry.compareTo(array[i]) > 0) {

i++;

}

makeRoom(i + 1);

array[i] = newEntry;

numberOfEntries++;

return true;

}

public boolean remove(T anEntry) {

int i = 0;

while (i < numberOfEntries && anEntry.compareTo(array[i]) > 0) {

i++;

}

if (i < numberOfEntries) {

removeGap(i);

}

numberOfEntries--;//just ignore the behind one

return true;

}

public void clear() {

numberOfEntries = 0;

}

@Override

public T getEntry(int givenPosition) {

T result = null;

if ((givenPosition >= 0) && (givenPosition <= numberOfEntries)) {

result = array[givenPosition];

}

return result;

}

public void replace(T oldEntry,T newEntry){

int i = 0;

while (i < numberOfEntries && oldEntry.compareTo(array[i]) > 0) {

i++;

}

array[i] = newEntry ;

}

public int contains(T anEntry) {

if(isEmpty())

return -1;

return binarysearch(anEntry);

}

public int getNumberOfEntries() {

return numberOfEntries;

}

public boolean isEmpty() {

return numberOfEntries == 0;

}

public String toString() {

String outputStr = "";

for (int index = 0; index < numberOfEntries; ++index) {

outputStr += array[index] + "\n";

}

return outputStr;

}

private boolean isArrayFull() {

return numberOfEntries == array.length;

}

private void doubleArray() {

T[] oldList = array;

int oldSize = oldList.length;

array = (T[]) new Object[2 \* oldSize];

for (int index = 0; index < oldSize; index++) {

array[index] = oldList[index];

}

}

private void makeRoom(int newPosition) {

int newIndex = newPosition - 1;

int lastIndex = numberOfEntries - 1;

for (int index = lastIndex; index >= newIndex; --index) {

array[index + 1] = array[index];

}

}

private void removeGap(int givenPosition) {

int removedIndex = givenPosition;

int lastIndex = numberOfEntries - 1;

for (int index = removedIndex; index < lastIndex; index++) {

array[index] = array[index + 1];

}

}

private int binarysearch(T search){

int lower = 0;

int higher = numberOfEntries - 1;

while(lower<=higher){

int middle=(lower+higher)/2;

int campresult=array[middle].compareTo(search);

if (campresult == 0) {

return middle; // element found

} else if (campresult < 0) {

lower = middle + 1;

} else {

higher = middle - 1;

}

}

return -1;//element not found

}

}

### 2.2.2 **Interface Source code for ADT (Course)**

package adt;

/\*\*

\*

\* @author Ng Wei Khang

\*/

public interface SortedArrayListInterface <T extends Comparable<T>>{

public boolean add(T newEntry);

public boolean remove(T anEntry);

public T getEntry(int givenPosition);

public int contains(T anEntry);

public void replace(T oldEntry,T newEntry);

public void clear();

public int getNumberOfEntries();

public boolean isEmpty();

}

### 2.2.3 **Source code for control class (Course)**

package control;

import java.util.Scanner;

import adt.\*;

import entity.\*;

import boundary.CourseUI;

/\*\*

\*

\* @author Ng Wei Khang

\*/

public class CourseManager {

private SortedArrayListInterface<Course> courseList = new SortedArrayList<>();

private CourseUI courseUi = new CourseUI();

public void predefinedProgramme(){

//courseCode, courseName, courseCreditHours, courseSession

Course newCourse1 = new Course("BAMS2614","MATH AND GAME TECHNOLOGY", 3 , 1);

Course newCourse2 = new Course("BACS2063","DATA STRUCTURE AND ALGORITHM", 4 , 1);

Course newCourse3 = new Course("BACS3074","ARTIFICIAL INTELLIGENCE", 3 , 2);

Course newCourse4 = new Course("BMMS2074","STATISTIC FOR DATA SCIENCE", 4 , 2);

Course newCourse5 = new Course("BACS3183","ADVANCED DATABASE MANAGEMENT", 4 , 3);

courseList.add(newCourse1);

courseList.add(newCourse2);

courseList.add(newCourse3);

courseList.add(newCourse4);

courseList.add(newCourse5);

}

public void addNewCourse() {

Course newCourse = ADDnewCourseDetails();

courseList.add(newCourse);

}

public void deleteCourse() {

listCourse();

String code = courseCode();

Course del = new Course(code, null, 0, 0);

if(courseList.contains(del) != -1){

courseList.remove(del);

courseUi.successfulDeleteMessage();

courseUi.pressEnter();

}else{

courseUi.invalidCourseCode();

courseUi.pressEnter();

deleteCourse();

}

}

public void findCourse() {

String code = courseCode();

Course find = new Course(code, null, 0, 0);

if(courseList.contains(find) != -1){

courseUi.successfulSearchMessage();

courseUi.courseDetails(courseList.getEntry(courseList.contains(find)).getCourseCode(),

courseList.getEntry(courseList.contains(find)).getCourseName(),

courseList.getEntry(courseList.contains(find)).getCourseCreditHours(),

courseList.getEntry(courseList.contains(find)).getCourseSession());

courseUi.pressEnter();

}else{

courseUi.invalidCourseCode();

findCourse();

}

}

public void listCourse() {

courseUi.listHeader();

for (int i = 0; i < courseList.getNumberOfEntries(); i++) {

System.out.println(courseList.getEntry(i));

}

}

public void amendCourse() {

String code = courseCode();

Course oldAmend = new Course(code, null, 0, 0);

if(courseList.contains(oldAmend) != -1){

// Get the old course from the course list

Course oldCourse = courseList.getEntry(courseList.contains(oldAmend));

courseUi.courseDetails(oldCourse.getCourseCode(),

oldCourse.getCourseName(),

oldCourse.getCourseCreditHours(),

oldCourse.getCourseSession());

courseUi.newAmendNewCourse(code);

String name = courseName();

int creditHours = courseCreditHours();

int session = courseSession();

Course newAmend = new Course(code, name, creditHours, session);

// Update the course in the course list

courseList.replace(oldAmend, newAmend);

courseUi.successfulAmendMessage();

}else{

courseUi.invalidCourseCode();

courseUi.pressEnter();

amendCourse();

}

}

public SortedArrayList<Course> getCourseList() {

return (SortedArrayList<Course>) courseList;

}

public String courseCode() {

String code;

boolean validInput = false;

Scanner scanner = new Scanner(System.in);

do {

courseUi.enterCourseCode();

code = scanner.nextLine().toUpperCase();

if (code.length() <= 8) {

// Check if the first 4 characters are letters and the last 4 are digits

boolean validFormat = true;

for (int i = 0; i < code.length(); i++) {

char ch = code.charAt(i);

if ((i < 4 && !Character.isLetter(ch)) || (i >= 4 && !Character.isDigit(ch))) {

validFormat = false;

break;

}

}

if (validFormat) {

validInput = true; // Input is valid, exit the loop

} else {

courseUi.invalidFormat();

}

} else {

courseUi.invalidLength();

}

} while (!validInput);

return code;

}

public String courseName() {

String name;

Scanner scanner = new Scanner(System.in);

do {

courseUi.enterCourseName();

name = scanner.nextLine().toUpperCase();

} while (name.isEmpty()); // Validate that name is not empty

return name;

}

public int courseCreditHours() {

int creditHours;

boolean validInput = false;

Scanner scanner = new Scanner(System.in);

do {

courseUi.enterCourseCreditHour();

// Check if the input is an integer

if (scanner.hasNextInt()) {

creditHours = scanner.nextInt();

scanner.nextLine(); // Consume the newline character

if (creditHours >= 2 && creditHours <= 4) {

validInput = true; // Input is valid, exit the loop

} else {

courseUi.invalidCourseCreditHour();

}

} else {

scanner.nextLine(); // Consume the invalid input

courseUi.invalidInput();

creditHours = -1; // Set creditHours to an invalid value

}

} while (!validInput);

return creditHours;

}

public int courseSession() {

int session;

boolean n = true;

Scanner scanner = new Scanner(System.in);

do{

courseUi.enterCourseSession();

session = scanner.nextInt();

if(session<0){

n=false;

courseUi.invalidCourseSession();

}else{

n=true;

}

}while(!n);

return session;

}

public Course ADDnewCourseDetails() {

String courseCode = courseCode();

Course newCode = new Course(courseCode, null, 0, 0);

if(courseList.contains(newCode) != -1){

courseUi.existCourseCode();

ADDnewCourseDetails();

}

String courseName = courseName();

int creditHours = courseCreditHours();

int session = courseSession();

return new Course(courseCode, courseName, creditHours,session);

}

public void generateReport() {

courseUi.topReport();

// Check if there are any courses to generate a report

if (courseList.isEmpty()) {

courseUi.emptyReport();

return;

}

courseUi.headerReport();

for (int i = 0; i < courseList.getNumberOfEntries(); i++) {

Course course = courseList.getEntry(i);

courseUi.listReport(course.getCourseCode(), course.getCourseName(),

course.getCourseCreditHours(), course.getCourseSession());

}

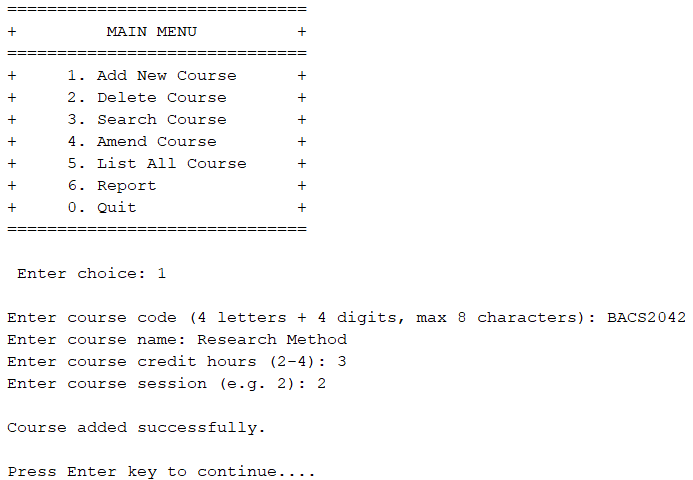
courseUi.footerReport();

}

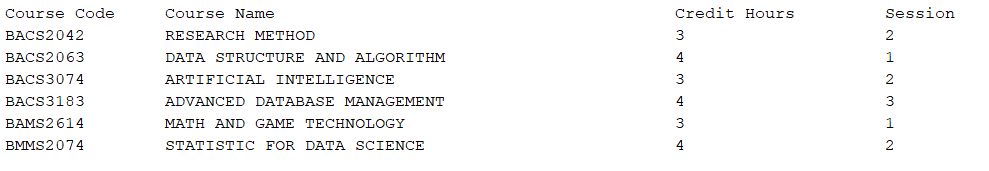
}

### 2.2.4 **Screenshot for Output (Course)**

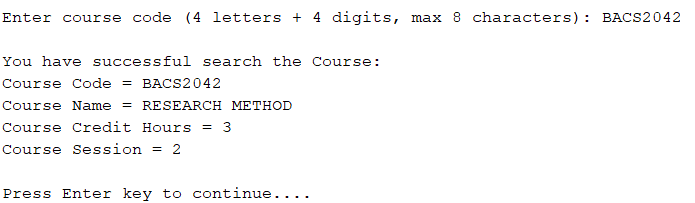
#### **Add Course**



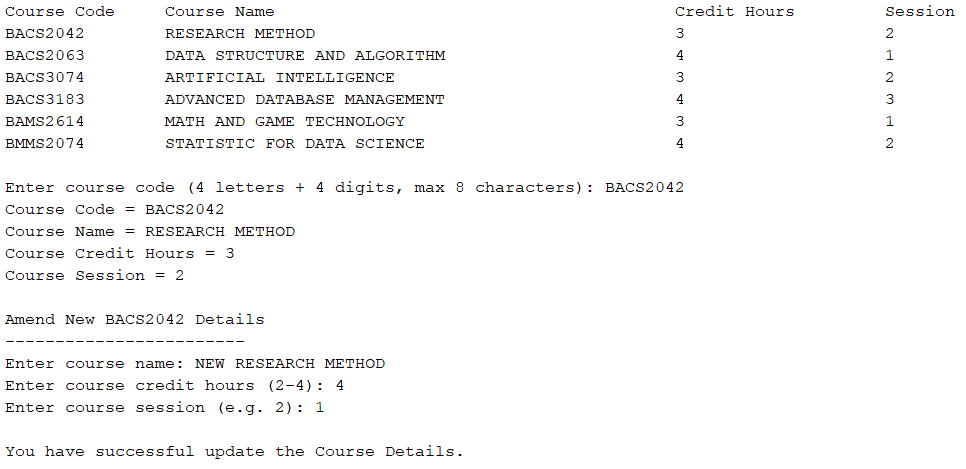
#### **List All Course**



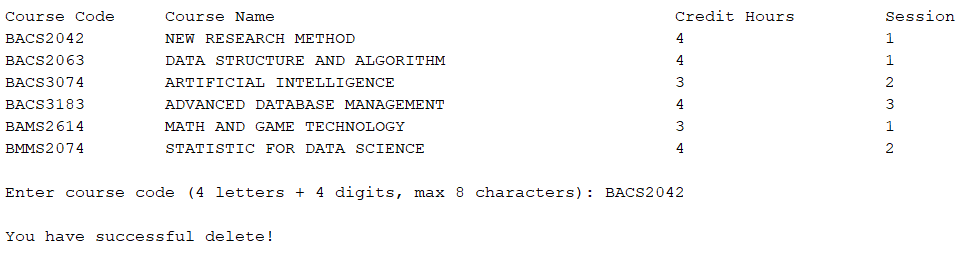
#### **Search Course**



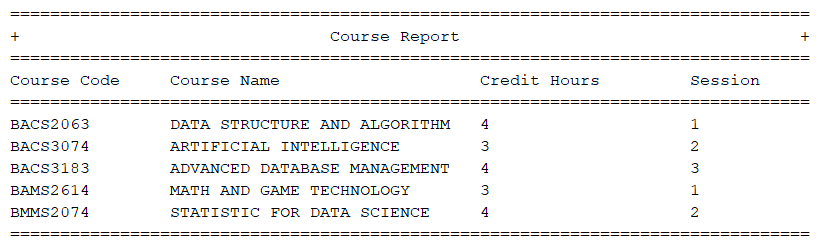
#### **Amend Course**



#### **Remove Course**



#### **Course Report**



### 

### 2.2.5 **Reason of Choosing Sorted array lists ADT**

Firstly, a sorted array list enables efficient searching through techniques like binary search, making it a compelling choice for applications requiring rapid element retrieval and this has made me choose for adt. Secondly, sorted array lists simplifies the orderly iteration and eliminates the need to sort the data repeatedly every time. This efficiency is especially important when processing large data sets or sorting data is the main requirement. In addition, they are very suitable for range queries, allowing quick identification of elements in a specific ranges, and they provide a definite order, ensuring predictability in various operations. Sorted array lists remain a robust option for applications where ordered data, efficient searching, and predictable performance are critical factors in achieving optimal functionality.

## **2.3 Programme Management Subsystem**

| Student Name | Student ID | Prog / Tut.Grp | Signature |
| --- | --- | --- | --- |
| Soh Wei Shen | 22WMR01084 | RDS Y2G2 | Soh |

Sub-system : Programme Management Subsystem

### 2.3.1 **Source code for ADT (Programme)**

/\*\*

\*

\* @author Soh

\*/

package adt;

public class SetArrayList <T extends Comparable<T>> implements SetArrayListInterface<T>{

T[] setArray;

int numberOfElements;

private static final int DEFAULT\_CAPACITY = 50;

public SetArrayList(){

setArray = (T[]) new Comparable[DEFAULT\_CAPACITY];

numberOfElements = 0;

}

@Override

public boolean add(T newElement){

for (int i = 0; i < numberOfElements; i++) {

if (this.setArray[i].equals(newElement)) {

return false;

}

}

if(isFull()){

doubleArray();

}

setArray[numberOfElements] = newElement;

numberOfElements++;

return true;

}

public void doubleArray(){

T[] oldArray = setArray;

setArray = (T[])new Object[2 \* oldArray.length];

System.arraycopy(oldArray, 0, setArray, 0, oldArray.length);

}

@Override

public boolean remove(T anElement) {

for (int i = 0; i < numberOfElements; i++) {

if (setArray[i].equals(anElement)) {

// perform the remove

removeGap(i);

numberOfElements--;

return true;

}

}

return false;

}

public void removeGap(int index) {

for (int i = index; i < numberOfElements - 1; i++) {

setArray[i] = setArray[i + 1];

}

}

private void makeRoom(int newPosition) {

int newIndex = newPosition - 1;

int lastIndex = numberOfElements - 1;

for (int index = lastIndex; index >= newIndex; index--) {

setArray[index + 1] = setArray[index];

}

}

@Override

public void clear(){

setArray = null;

numberOfElements = 0;

}

@Override

public T getEntry(int givenPosition) {

T result = null;

if ((givenPosition >= 1) && (givenPosition <= numberOfElements)) {

result = setArray[givenPosition - 1];

}

return result;

}

@Override

public boolean contains(T anEntry) {

boolean found = false;

for (int index = 0; !found && (index < numberOfElements); index++) {

if (anEntry.equals(setArray[index])) {

found = true;

}

}

return found;

}

@Override

public int getNumberOfElements(){

return numberOfElements;

}

@Override

public boolean isEmpty() {

return numberOfElements == 0;

}

@Override

public boolean isFull(){

return numberOfElements == setArray.length;

}

}

### 2.3.2 **Interface Source code for ADT (Programme)**

package adt;

/\*\*

\*

\* @author Soh

\*/

public interface SetArrayListInterface<T> {

public boolean add(T newElement);

public boolean remove(T anElement);

public void clear();

public T getEntry(int givenPosition);

public boolean contains(T anEntry);

public int getNumberOfElements();

public boolean isEmpty();

public boolean isFull();

}

### 2.3.3 **Source code for control class (Programme)**

/\*\*

\*

\* @author Soh

\*/

package control;

import java.util.Scanner;

import adt.ListInterface;

import adt.SetArrayList;

import adt.SetArrayListInterface;

import adt.SortedArrayList;

import entity.Course;

import entity.Programme;

public class ProgrammeManager {

private SetArrayListInterface<Programme> programmeList = new SetArrayList<>();

private ListInterface<Course> courseList = new SortedArrayList<>();

private CourseManager CourseManager;

Scanner scan = new Scanner(System.in);

public ProgrammeManager(CourseManager CourseManager) {

initializeRecords();

this.CourseManager = CourseManager; // Initialize CourseManager

courseList = CourseManager.getCourseList();

}

public void initializeRecords() {

Programme programme1 = new Programme("RDS", 1, 1, "Bachelor of Data Science");

Programme programme2 = new Programme("RIT", 2, 1, "Bachelor of Internet Technology");

Programme programme3 = new Programme("RMM", 3, 1, "Bachelor of Mathematical Computing");

Programme programme4 = new Programme("RCS", 1, 2, "Bachelor of Computer Science");

Programme programme5 = new Programme("RIS", 2, 2, "Bachelor of Information Security");

Course course1 = new Course("BAMS2614","MATH AND GAME TECHNOLOGY", 3 , 1);

Course course2 = new Course("BACS2063","DATA STRUCTURE AND ALGORITHM", 4 , 1);

Course course3 = new Course("BACS3074","ARTIFICIAL INTELLIGENCE", 3 , 2);

Course course4 = new Course("BMMS2074","STATISTIC FOR DATA SCIENCE", 4 , 2);

Course course5 = new Course("BACS3183","ADVANCED DATBASE MANAGEMENT", 4 , 3);

programme1.addCourse(course1);

programme1.addCourse(course2);

programme1.addCourse(course3);

programme2.addCourse(course4);

programme2.addCourse(course5);

programme3.addCourse(course2);

programme3.addCourse(course5);

programme4.addCourse(course1);

programme4.addCourse(course4);

programme5.addCourse(course3);

programme5.addCourse(course5);

programmeList.add(programme1);

programmeList.add(programme2);

programmeList.add(programme3);

programmeList.add(programme4);

programmeList.add(programme5);

}

public void addProgramme() {

String programmeCode = getProgrammeCode();

int programmeYear = getProgrammeYear();

int programmeSemester = getProgrammeSemester();

String description = getProgrammeDescription();

Programme newProgramme = new Programme(programmeCode, programmeYear, programmeSemester, description);

// Get the list of predefined courses from CourseManager

SortedArrayList<Course> predefinedCourses = CourseManager.getCourseList();

// Display the list of predefined courses

System.out.println("Available Courses:");

for (int i = 0; i < predefinedCourses.getNumberOfEntries(); i++) {

System.out.println((i + 1) + ". " + predefinedCourses.getEntry(i).getCourseCode() + " - " + predefinedCourses.getEntry(i).getCourseName());

}

// Ask the user to choose courses

System.out.print("Enter the indexes of courses to add (comma-separated): ");

String[] chosenIndexes = scan.nextLine().split(",");

for (String indexStr : chosenIndexes) {

int index = Integer.parseInt(indexStr.trim()) - 1;

if (index >= 0 && index < predefinedCourses.getNumberOfEntries()) {

Course chosenCourse = predefinedCourses.getEntry(index);

newProgramme.addCourse(chosenCourse);

} else {

System.out.println("Invalid index: " + (index + 1));

}

}

programmeList.add(newProgramme);

System.out.println("Programme Added!");

}

public void removeProgramme() {

listProgrammes();

System.out.print("Enter the programme to remove (e.g. 1): ");

int choice = scan.nextInt();

scan.nextLine();

programmeList.remove(programmeList.getEntry(choice));

System.out.println("The chosen programme has been successfully removed!");

}

public void findProgramme() {

System.out.print("Enter programme code to find: ");

String programCodeToFind = scan.nextLine().toUpperCase();

int position = 0;

String outputStr = "";

for (int i = 1; i <= programmeList.getNumberOfElements(); i++) {

if (programmeList.getEntry(i).getName().equals(programCodeToFind)) {

position = i;

break;

}

}

outputStr += programmeList.getEntry(position);

System.out.println(outputStr);

}

public void generateReport() {

System.out.println("================================================================================");

System.out.println("+ Programme Report +");

System.out.println("================================================================================");

// Check if there are any programs to generate a report

if (programmeList.isEmpty()) {

System.out.println("No programs to report.");

return;

}

System.out.printf("%-15s %-10s %-10s %-30s%n", "Program Code", "Year", "Semester", "Description");

System.out.println("================================================================================");

for (int i = 1; i <= programmeList.getNumberOfElements(); i++) {

Programme programme = programmeList.getEntry(i);

if (programme != null) {

System.out.printf("%-15s %-10d %-10d %-30s%n",

programme.getName(), programme.getYear(),

programme.getSemester(), programme.getDescription());

} else {

System.out.println("Programme object is null.");

}

}

System.out.println("================================================================================");

}

public void amendProgrammeDetails() {

listProgrammes(); // Display all programmes

do {

System.out.print("Enter programme code to amend: ");

String programCodeToAmend = scan.nextLine().toUpperCase();

int position = -1;

// Find the index of the programme to amend

for (int i = 1; i <= programmeList.getNumberOfElements(); i++) {

if (programmeList.getEntry(i).getName().equals(programCodeToAmend)) {

position = i;

break;

}

}

if (position != -1) {

Programme selectedProgramme = programmeList.getEntry(position);

// Display available courses

SortedArrayList<Course> availableCourses = new SortedArrayList<>();

for (int i = 0; i < courseList.getNumberOfEntries(); i++) {

if (selectedProgramme.getCourses().contains(courseList.getEntry(i)) == -1) {

availableCourses.add(courseList.getEntry(i));

}

}

if (availableCourses.isEmpty()) {

System.out.println("No available courses to add.");

return;

}

System.out.println("Available Courses:");

for (int i = 0; i < availableCourses.getNumberOfEntries(); i++) {

System.out.println((i + 1) + ". " + availableCourses.getEntry(i));

}

// Ask the user to choose courses

System.out.print("Enter the indexes of courses to add (comma-separated): ");

String[] chosenIndexes = scan.nextLine().split(",");

for (String indexStr : chosenIndexes) {

int index = Integer.parseInt(indexStr.trim()) - 1;

if (index >= 0 && index < availableCourses.getNumberOfEntries()) {

Course chosenCourse = availableCourses.getEntry(index);

// Check if the course is already in the programme

if (selectedProgramme.getCourses().contains(chosenCourse) == -1) {

selectedProgramme.addCourse(chosenCourse);

System.out.println("Course " + chosenCourse.getCourseCode() + " added to programme.");

} else {

System.out.println("Course " + chosenCourse.getCourseCode() + " is already in the programme.");

}

} else {

System.out.println("Invalid index: " + (index + 1));

}

}

String name = getProgrammeCode();

int year = getProgrammeYear();

int semester = getProgrammeSemester();

String ProgramDesc = getProgrammeDescription();

// Update programme details

selectedProgramme.setName(name);

selectedProgramme.setYear(year);

selectedProgramme.setSemester(semester);

selectedProgramme.setDescription(ProgramDesc);

System.out.println("Programme details updated successfully!");

break; // Exit the loop if successful

} else {

System.out.println("Programme not found. Please try again.");

}

} while (true); // Keep looping until a valid program code is provided

}

// Function to get a valid programme code (cannot contain integers)

public String getProgrammeCode() {

String code;

boolean validInput = false;

do {

System.out.print("Enter programme code (cannot contain integers): ");

code = scan.nextLine().toUpperCase();

if (code.matches(".\*\\d.\*")) {

System.out.println("Invalid input. Programme code cannot contain integers.");

} else {

validInput = true;

}

} while (!validInput);

return code;

}

// Function to get a valid programme year (1-3)

public int getProgrammeYear() {

int year = 0;

do {

System.out.print("Enter programme year (1-3): ");

if (scan.hasNextInt()) {

year = scan.nextInt();

scan.nextLine(); // Consume the newline character

if (year < 1 || year > 3) {

System.out.println("Invalid input. Please enter a value between 1 and 3.");

}

} else {

scan.nextLine(); // Consume the invalid input

System.out.println("Invalid input. Please enter a valid integer value.");

}

} while (year < 1 || year > 3);

return year;

}

// Function to get a valid programme semester (1-3)

public int getProgrammeSemester() {

int semester = 0;

do {

System.out.print("Enter programme semester (1-3): ");

if (scan.hasNextInt()) {

semester = scan.nextInt();

scan.nextLine(); // Consume the newline character

if (semester < 1 || semester > 3) {

System.out.println("Invalid input. Please enter a value between 1 and 3.");

}

} else {

scan.nextLine(); // Consume the invalid input

System.out.println("Invalid input. Please enter a valid integer value.");

}

} while (semester < 1 || semester > 3);

return semester;

}

// Function to get programme description

public String getProgrammeDescription() {

System.out.print("Enter programme description: ");

return scan.nextLine().toUpperCase();

}

public void listProgrammes() {

String outputStr = "";

for (int i = 1; i <= programmeList.getNumberOfElements(); i++) {

String num = String.format("%d. ", i);

outputStr += num + programmeList.getEntry(i) + "\n";

}

System.out.println(outputStr);

}

public int back() {

System.out.println("Press 0 to go back..");

int back = scan.nextInt();

scan.nextLine();

return back;

}

public void exitMessage() {

System.out.println("\nYou have leave the Programme Menu!\n");

}

}

### 2.3.4 **Source code for Entity class (Programme)**

/\*\*

\*

\* @author Soh

\*/

package entity;

import adt.SortedArrayList;

public class Programme implements Comparable<Programme>{

private String programCode;

private int year;

private int semester;

private String description;

private SortedArrayList<Course> courses; // Use SortedArrayList to store courses

public Programme() {

}

public Programme(String name, int year, int semester, String description) {

this.programCode = name;

this.year = year;

this.semester = semester;

this.description = description;

this.courses = new SortedArrayList<>(); // Initialize the list of courses

}

public String getName() {

return programCode;

}

public void setName(String name) {

this.programCode = name;

}

public int getYear() {

return year;

}

public void setYear(int year) {

this.year = year;

}

public int getSemester() {

return semester;

}

public void setSemester(int semester) {

this.semester = semester;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public void addCourse(Course course) {

courses.add(course); // Add a course to the list of courses

}

public boolean removeCourse(Course course) {

return courses.remove(course); // Remove a course from the list of courses

}

public SortedArrayList<Course> getCourses() {

return courses; // Get the list of courses

}

@Override

public String toString() {

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.append("Programme Code: ").append(programCode)

.append("\nDescription: ").append(description)

.append("\nYear: ").append(year)

.append("\nSemester: ").append(semester)

.append("\nCourses: \n");

for (int i = 0; i < courses.getNumberOfEntries(); i++) {

Course course = courses.getEntry(i);

stringBuilder.append(" - ").append(course.getCourseCode())

.append(": ").append(course.getCourseName())

.append("\n");

}

return stringBuilder.toString();

}

public int compareTo(Programme o){

if (o == null) {

return 0;

}

Programme o2 = (Programme)o;

for(int i=0; i< o2.programCode.length();i++){

if(this.programCode.charAt(i) > o2.programCode.charAt(i) ){

return 1;

}else if(this.programCode.charAt(i) < o2.programCode.charAt(i)){

return -1;

}

}

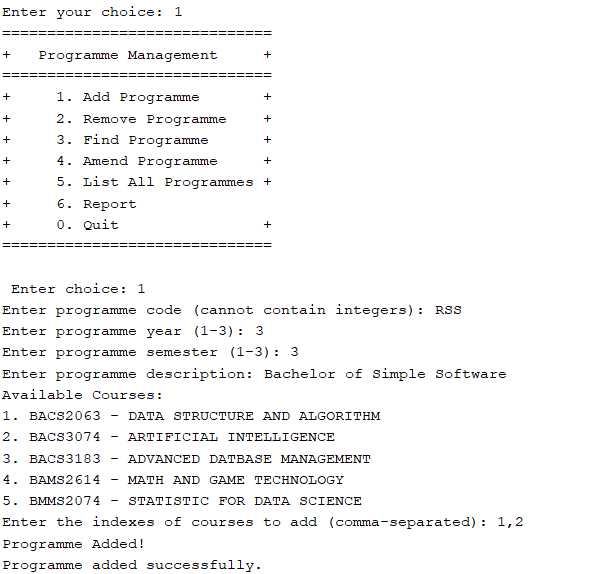
return -1;

}

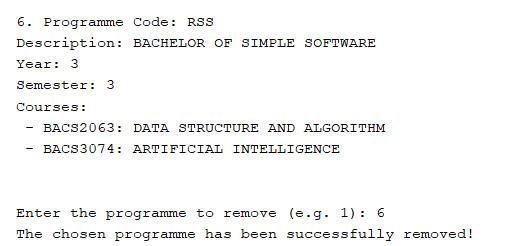
}

### **2.3.5 Screenshot for Output(Programme)**

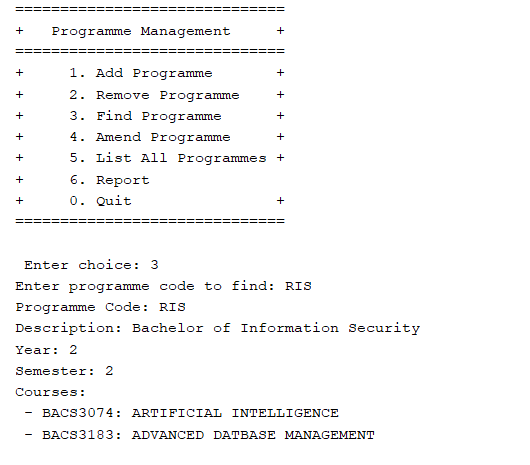
#### **Add Programme**

****

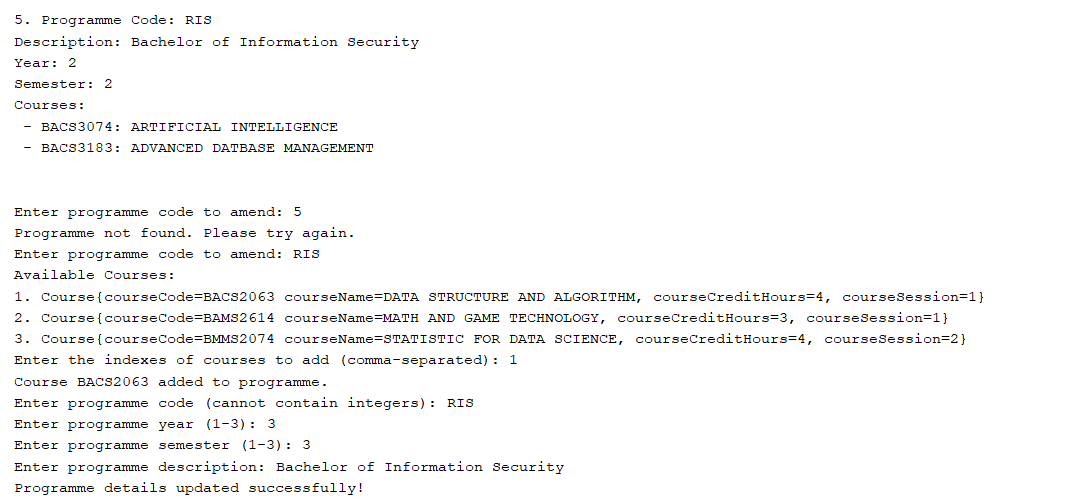
#### **Remove Programme**

****

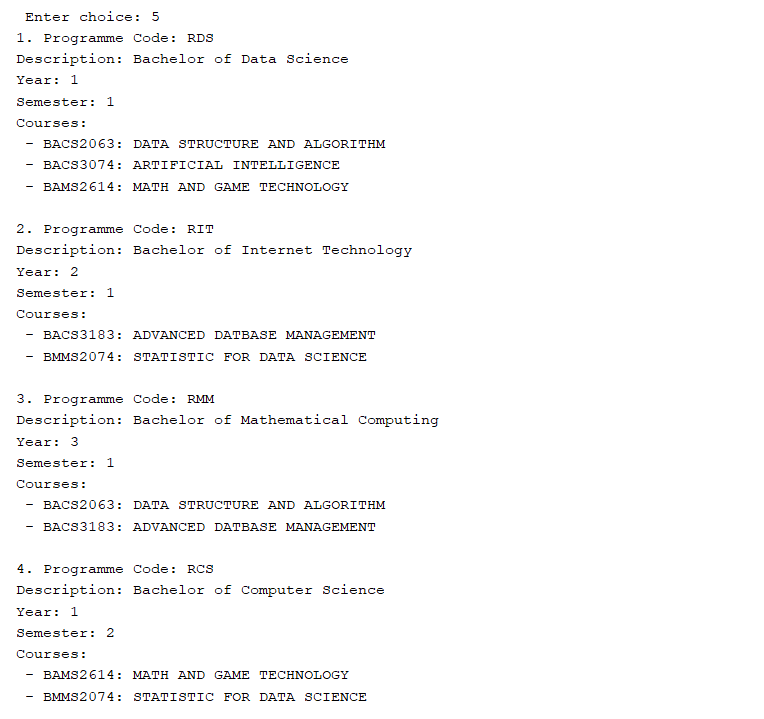
#### **Find Programme**

****

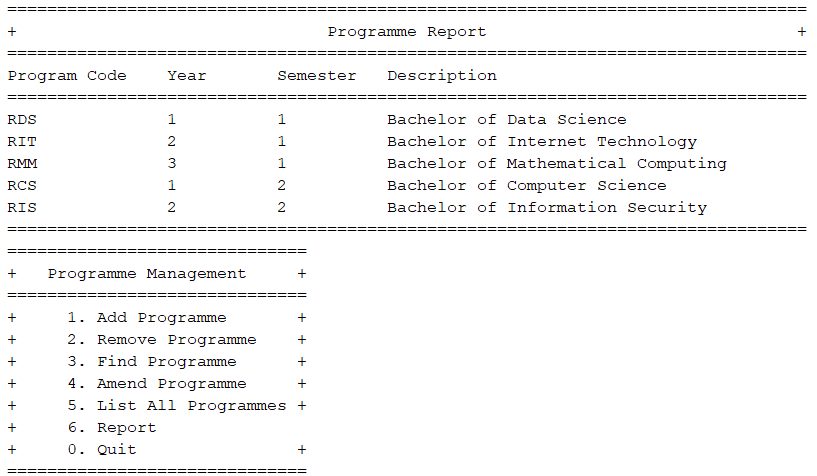
#### **Amend Programme**

****

#### **List All Programme**

****

#### **Programme Report**



**2.3.6 Reasons for Choosing Set ArrayList ADT for Programme**

The reason for using SetArrayList is because of the characteristics of Set and the benefits Array list data structure can provide. By using Set, we can make sure that each program the university provides has a unique element because there is no way a university can provide 2 different programs that share the same name. Moreover, Sets use a function called ‘efficient membership testing’ which means If users frequently need to check whether a specific program is offered by the university, using a Set is faster than searching through a list. There is also no ordering needed in Set which makes storing using Set so much simpler and efficient. By using Arraylist’s data structure, we can directly access the element we are looking for by their position in the list. In conclusion, SetArrayList provides the users with flexibility. We choose to use the Set for operations where uniqueness and efficient membership testing are critical, and the ArrayList when order preservation is important or when you need to iterate through all elements sequentially.