King Saud University
College of Computer & Information Science
CSC111 – Lab11
Arrays – II –
All Sections
--

Objectives:

- Describe objects and classes, and use classes to model objects.
- To know how to define and create an array.
- To know how to pass array to method and return array from method.
- To know how to create objects with arrays as attributes.
- To know how to add elements to arrays.
- To know how to search arrays.
- To know how to find max element in an array.

Lab Exercise 1

Part1 (read/write array values)

Write a program **CourseManager1** that reads and prints scores of students in a course. The scores are double numbers between 0 and 100.

- Your program should start by reading the number of the students taking the course.
- Then it reads and stores the scores in an array. If a score is invalid then your program should store 0.
- After that it prints the scores

Sample run:

```
Enter number of students: 4 ↵
Please enter students' scores: 100 -30 75 90 ↵
The score -30.0 you entered is wrong. Program will store score 0.
The scores are: 100.0 0.0 75.0 90.0
```

Complete following pseudo code:

```java
import java.util.Scanner;
public class CourseManager1 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter number of students: ");
    /* read the number of students ( array size) */
    while (/* array size less than 1*/ numOfStudents < 1){
     System.out.print("Number of students is invalid."
                  + " Enter number of students: ");
     // read array size again
    }
    /* define and declare the array ( double) */
    System.out.print("Please enter students' scores: ");
    //read the array score
    for (int i = 0; i < scores.length; i++){
      /*read the array score*/
      if (score >= 0 && score <= 100){
        scores[i] = score;
      } else {
        System.out.println("The score " + score
                  + " you entered is wrong. Program will store score 0.");
      }
    }
    System.out.print("The scores are: ");
    /* write a code to output the contents of the array (print the array score) */
    System.out.println();
  }// end of main method
}// end of CourseManager1 class
```

Part 2 (copy array, pass array as parameter, return array as result)

Modify the previous program such that after reading the scores, your program computes the letter grades and stores them in an array of type **char**.

- Write a static method scoreToGrade that
    - takes the **scores** array as parameter
    - creates the **grades** array

    - fill **grades** array up with letter grades (A: 90-100, B: 80-89, C: 70-79, D: 60-69, F: 0-59) then return it. Use the rules of KSU to convert a score into a letter grade.
- Prints each score along with the letter grade using format *score/letter_grade*.

Name your new program **CourseManager2**.

Sample run:

```
Enter number of students: 5 ↵
Please enter students' scores: 100 40 79 89 90 ↵
The scores/grades are: 100.0/A 40.0/F 79.0/C 89.0/B 90.0/A
```

Complete following pseudo code:

```
import java.util.Scanner;
public class CourseManager1 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter number of students: ");
    /* read the number of students ( array size) */
    while (/* array size less than 1*/){
     System.out.print("Number of students is invalid."
                 + " Enter number of students: ");
     // read array size again
    }
    /* define and declare the array ( double) */
    System.out.print("Please enter students' scores: ");
    //read the array score
    for (int i = 0; i < scores.length; i++){
      /*read score and store it in the array*/
      if (score >= 0 && score <= 100){
        scores[i] = score;
      } else {
        System.out.println("The score " + score
                 + " you entered is wrong. Program will store score 0.");
      }
    }
    // define and declare the grades array (char)
    char[] grades = scoreToGrade(scores);
    System.out.print("The scores/grades are: ");
    /* prints each score/grade using format scores[i] + "/" + grades[i] + " " */
    System.out.println();
  }// end of main method
  // Precondition: all scores in the array are between 0 and 100
  // create scoreToGrade method
  public static char[] scoreToGrade(double[] scores){
    char[] grades = new char[scores.length];
    for (int i = 0; i < scores.length; i++){
      if (scores[i] >= 90)

      //if score >=90 store A in grade

      //if score > 80 store B in grade

      //if score > 70 store C in grade

      else if (scores[i] >= 60)
        grades[i] = 'D'; //if score > 60 stores D in grade
      else grades[i] = 'F'; // store F
    }
    return grades;
  }
}// end of CourseManager1 class
```

Part 3

Since we are doing object oriented programming, a better design is to declare and use arrays as attributes of the class **CourseManagerE1**. This means that we will avoid passing/return arrays to/from methods.

Rewrite previous program using object oriented programming methodology as following:

- Define the two arrays **scores** and **grades** as attributes of the class **CourseManagerE1**.
- Change the method **scoreToGrade** such that:
  - It becomes an instance method.
  - It does not receive or return anything.
- Add a method **readScores** to create the **scores** array and read its values.
- Add a method **printGrades** to print the **scores** and **grades** arrays as done in the previous program.

Create a program **TestCourseManager3** that does exactly the same as the previous program but using an instance of class **CourseManagerE1**.

Sample run:

```
Enter number of students: 5 ↵
Please enter students' scores: 100 40 79 89 90 ↵
The scores/grades are: 100.0/A 40.0/F 79.0/C 89.0/B 90.0/A
```

Complete following pseudo code: (`CourseManagerE1.java`)

```
class CourseManagerE1 {
  //declare an array scores ( double)
  //declare an array score (char)
  //create a method readScores()
  public void readScores() {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter number of students: ");
    int numOfStudents = input.nextInt();
    while (/* array size less than 1*/) {
     System.out.print("Number of students is invalid. Enter number of students: ");
     //READ numOfStudents AGAIN
    }
    scores = new double[numOfStudents];
    System.out.print("Please enter students' scores: ");
    for (int i = 0; i < scores.length; i++) {
      double score = input.nextDouble();
      // print a message ""The score " + score + you entered is wrong.
      //Program will store score 0." if you entered a wrong score.
    }
  }
  // Precondition: all scores in the array are between 0 and 100
  public void scoreToGrade() {
    //create a method scoreToGrade with return type void
    // define array grades with type char
    for (int i = 0; i < scores.length; i++) {
      //if score >=90 store A in grade
      else if (scores[i] >= 80)
        grades[i] = 'B';
      //if score > 80 store B in grade
      //if score > 70 store C in grade
      //if score > 60 store D in grade
      else grades[i] = 'F'; // store F
    }
  }
  public void printGrades(){
    //create a method printGrades with return type void
    System.out.println();
  }
}
```

Complete following pseudo code: (`TestCourseManager3.java`)

```
//create a class TestCourseManager3
public class TestCourseManager3 {
  public static void main(String[] args) {
      CourseManagerE1 cm = new CourseManagerE1();
      //create an object cm of Class CourseManagerE1
      // and call readScores, scoreToGrade and printGrades
  }
}
```

Part 4

Modify the previous program by adding two methods to class **CourseManagerE1** that will compute the average score of the course. Methods are:

- **sumScores** which computes and returns the sum of scores in scores array. This method is an *internal helper method* and it should be ***private***. It will be used by the next method **average**.
- **average** which computes and returns the average of scores in **scores** array using **sumScores** as an *internal helper method*.

Now write the main program to do the same as the previous one in addition to printing the scores average. Name your program **TestCourseManager4**.

Sample run:

```
Enter number of students: 5 ↵
Please enter students' scores: 100 40 79 89 90 ↵
The scores/grades are: 100.0/A 40.0/F 79.0/C 89.0/B 90.0/A
Average = 79.6
```

Complete following pseudo code: (`CourseManagerE1.java`)

```
public class CourseManagerE1 {
  ...

  //create a method sum() with return type double which should
  // return sum of all the elements of an array scores[]
  private double sum() {
    ...
  }

  // Precondition: scores is not null
  //create a method average which has return type double
  // ( formula sum() / scores.length)
  public double average(){
    ...
  }

  ...
}
```

Part 5

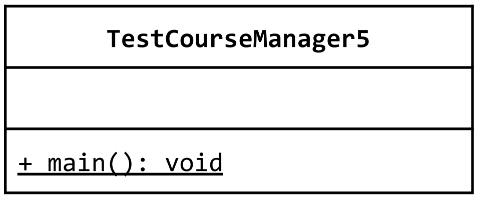
Common mistakes. If you have noticed, many methods in the class **CourseManager** have a comment `//precondition`. Why? Why does **sumScores** not have such a comment? (Hint: think what will happen if the user of class **CourseManager** calls **printGrades** before **readScores**).

Lab Exercise 2

In this exercise, we will make major changes to the previous program to make it an interactive course manager.

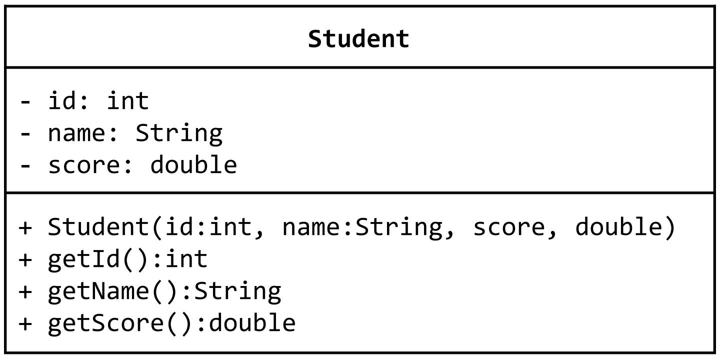
Part 1 (add elements to array)

Write a class **CourseManagerE2** that stores a list of students' objects in a given class. Each student has an **id**, **name**, **score.** The class allows the user to add a student, and display students' data. Here is the UML diagram:

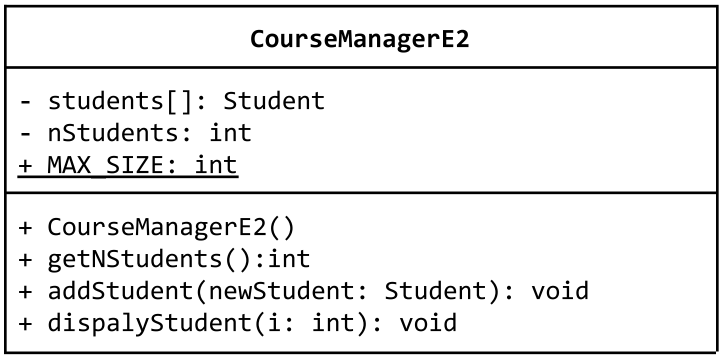


```
TestCourseManager5

+ main(): void
```



```
Student
- id: int
- name: String
- score: double
+ Student(id:int, name:String, score, double)
+ getId():int
+ getName():String
+ getScore():double
```



```
CourseManagerE2
- students[]: Student
- nStudents: int
+ MAX_SIZE: int
+ CourseManagerE2()
+ getNStudents():int
+ addStudent(newStudent: Student): void
+ dispalyStudent(i: int): void
```

As shown in the UML diagram, write the class **Student** that has the attributes: **ids**, **names**, **scores**, and a **constructor** to initialize the attributes. Then write the class **CourseManagerE2** that has an array of student objects. The attribute **nStudents** represents the current number of students in the list. ***The maximum number of students in the class is 100***.

The methods are:

- **CourseManagerE2**: a constructor that initializes the attributes and creates an array of students of size 100.
- **getNStudents** : returns the current number of students.
- **addStudent**: adds the student with the given object to the list. If the course is full, it prints the error message: "ERROR: COURSE IS FULL" .
- **displayStudent**: displays all data of the student at index i.

Write a **main** class called **TestCourseManager5** with a main method that will do the following: .

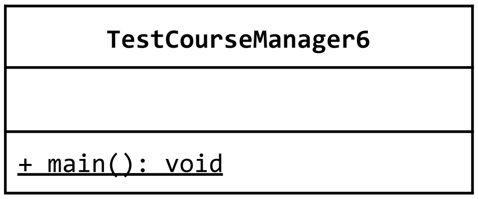
- It creates a **CourseManagerE2** object.
- Then, it adds 3 students by reading their IDs, names, and scores from the user.
- Then, it displays all students in class.

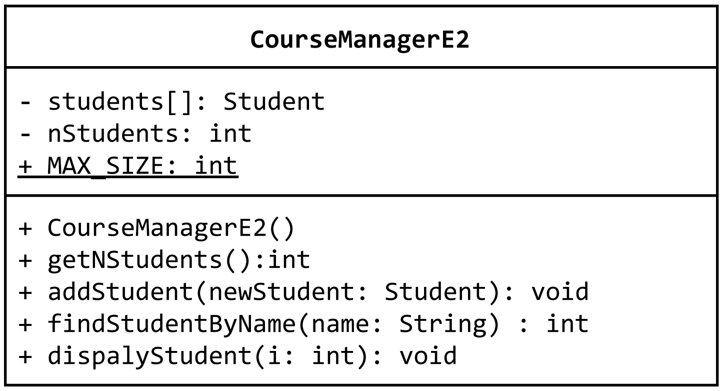
Sample run:

```
Please enter the ID, name, and score of student 0:
434000000 ↵
Ahmed ↵
95 ↵
Please enter the ID, name, and score of student 1:
433001234 ↵
Ali ↵
85 ↵
Please enter the ID, name, and score of student 2:
434005421 ↵
Fahad ↵
76 ↵
Students are:
434000000, Ahmed, 95.0
433001234, Ali, 85.0
434005421, Fahad, 76.0
```

Part 2 (find elements in an array)

Modify the previous program by adding a method to find a student by name. We will update the class **CourseManagerE2**. Modify **addStudent** such that it uses the **findStudentByName** method to make sure the student is not added twice to class. Here is the UML diagram:




CourseManagerE2
- students[]: Student
- nStudents: int
+ MAX_SIZE: int
+ CourseManagerE2()
+ getNStudents():int
+ addStudent(newStudent: Student): void
+ findStudentByName(name: String) : int
+ dispalyStudent(i: int): void

As shown in the UML diagram, the new and modified methods are:

- **addStudent**: adds the student with the given students' object to the list. If the course is full, it prints the error message: "ERROR: COURSE IS FULL". If a student is already added it prints the error message: "ERROR: STUDENT ALREADY ADDED".
- **findStudentByName**: returns the index of the student whose name is name. If it is not found, -1 is returned.

Write a main class called **TestCourseManager6** with a **main** method that will do the following:

- It creates a **CourseManagerE2** object.
- Then, it adds a student by reading its ID, name, and score from the user.

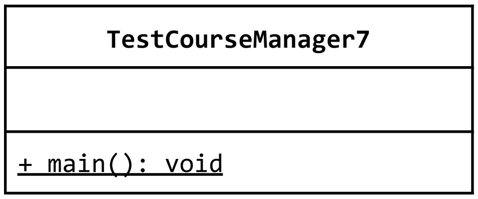
- Then, it tries to add the same student again and prints a failure message.
- Then, it displays the students.

Sample run:

```
Please enter the ID, name, and score of student 0:
434000000 ↵
Ahmed ↵
95 ↵
Please enter the ID, name, and score of student 1:
434000000 ↵
Ahmed ↵
95 ↵
ERROR: STUDENT ALREADY THERE
Students are:
434000000, Ahmed, 95.0
```

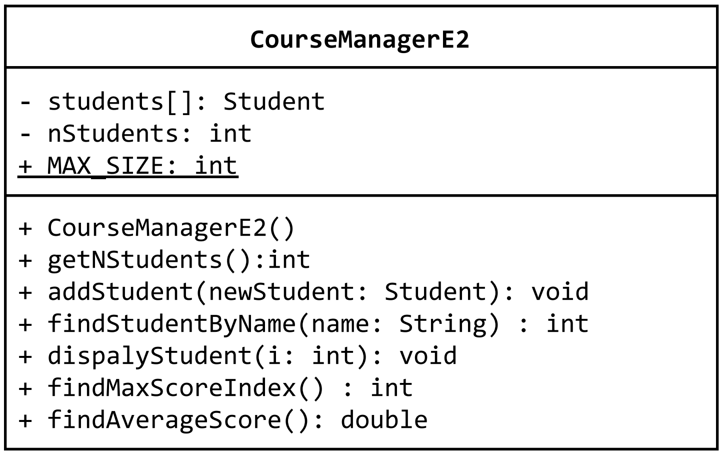
Part 3 (find max element in an array)

Modify the previous program by adding two methods to find the student with maximum score and compute the average score. Update the class **CourseManagerE2** and add the methods **findMaxScoreIndex** and **findAverageScore** to the class. Here is the UML diagram:



```
TestCourseManager7

+ main(): void
```



```
CourseManagerE2

- students[]: Student
- nStudents: int
+ MAX_SIZE: int

+ CourseManagerE2()
+ getNStudents():int
+ addStudent(newStudent: Student): void
+ findStudentByName(name: String) : int
+ dispalyStudent(i: int): void
+ findMaxScoreIndex() : int
+ findAverageScore(): double
```

As shown in the UML diagram, the new and modified methods are:

- **findMaxScoreIndex**: returns the index of a student whose score is the highest in the class.
- **findAverageScore**: returns the average score of the class.

Write a main class called **TestCourseManager7** with a **main** method that will do the following:

- It creates a **CourseManagerE2** object.
- Then, it adds 3 students by reading their IDs, names, and scores from the user.
- Then, it displays the average class scores.
- Then, it displays the student with the maximum score.

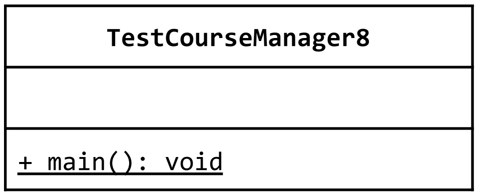
Sample run:

```
Please enter the ID, name, and score of student 0:
434000000 ↵
Ahmed ↵
60.0 ↵
Please enter the ID, name, and score of student 1:
433001234 ↵
Ali ↵
100.0 ↵
Please enter the ID, name, and score of student 2:
434005421 ↵
Fahad ↵
50.0 ↵
The class average = 70.0
The student with the highest score:
433001234, Ali, 100.0
```

Part 4 (Delete an element from an array)

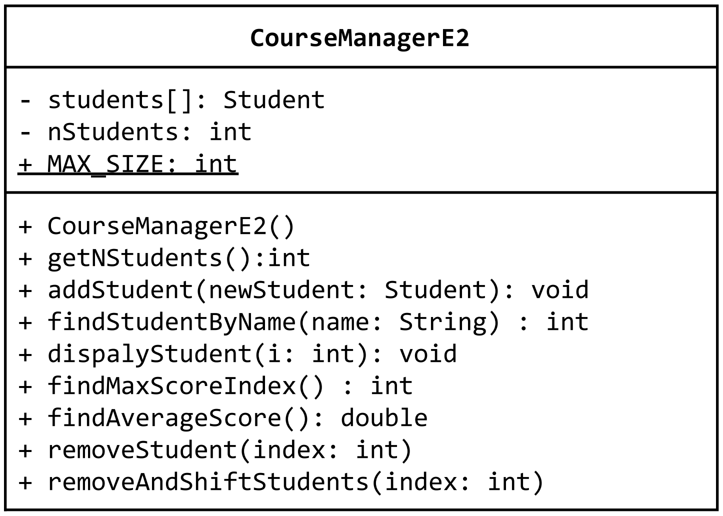
Modify the previous program by adding two methods to remove an element from an array. Update the class **CourseManagerE2** and add methods

removeStudent and **removeAndShiftStudents** to the class. Here is the UML diagram:



```
TestCourseManager8
--------------------

--------------------
+ main(): void
```



```
CourseManagerE2
--------------------
- students[]: Student
- nStudents: int
+ MAX_SIZE: int
--------------------
+ CourseManagerE2()
+ getNStudents():int
+ addStudent(newStudent: Student): void
+ findStudentByName(name: String) : int
+ dispalyStudent(i: int): void
+ findMaxScoreIndex() : int
+ findAverageScore(): double
+ removeStudent(index: int)
+ removeAndShiftStudents(index: int)
```

As shown in the UML diagram, the new and modified methods are:

- **removeStudent**: removes a Student with the given index and sets its value to NULL.
- **removeAndShiftStudents**: you shift elements with index greater than i left by one element. For example, if you want to remove element 3, you copy element 4 to element 3, element 5 to element 4 etc.

Write a **main** class called **TestCourseManager8** with a main method that will do the following:

- It creates a **CourseManagerE2** object.
- Then, it adds 3 students by reading their IDs, names, and scores from the user.
- Then, it removes an element at index 1 using **removeStudent** method.

- Then, it displays the students making sure to handle the null object before you print it.
- Creates another **CourseManagerE2** object.
- Then, it adds 3 students by reading their IDs, names, and scores from the user.
- Then, it removes an element at index 0 using **removeAndShiftStudents** method.
- Then, print the arrays' content to see the change.

Sample run:

```
Please enter the ID, name, and score of student 0:
433000111 ↵
Mohammed ↵
60.0 ↵
Please enter the ID, name, and score of student 1:
433000222 ↵
Ahmad ↵
100.0 ↵
Please enter the ID, name, and score of student 2:
433000333 ↵
Khalid ↵
50.0 ↵
The array after removing the Student at index 1:
433000111, Mohammed, 60.0
433000333, Khalid, 50.0
Please enter the ID, name, and score of student 0:
433000111 ↵
Mohammed ↵
60.0 ↵
Please enter the ID, name, and score of student 1:
433000222 ↵
Ahmad ↵
100.0 ↵
Please enter the ID, name, and score of student 2:
433000333 ↵
Khalid ↵
50.0 ↵
The array after removing the Student at index 0:
433000111, Ahmad, 100.0
433000333, Khalid, 50.0
433000333, Khalid, 50.0
```

Complete following pseudo code: (`CourseManagerE2.java`)

```
class CourseManagerE2 {
       /*
        * Declare the class data members as shown in the UML
        */
public CourseManagerE2() {
       /* part 1 */
}

public /* method type */ addStudent(/* students’ object */) {
       /* part 1 and 2 */
}

public /* method type */ findStudentName(/* parameters list */) {
       /* part 1 */
}

public /* method type */ findAverageScore(/* parameters list */) {
       /* part 3 */
}

public /* method type */ findMaxScoreIndex(/* parameters list */) {
       /* part 3 */
}

public /* method type */ displayStudent(/* parameters list */) {
       /* print the id, name , and scores of the index i passed to the method */
}

public /* method type */ getNStudents(/* parameters list */) {
       /* return nStudents */
}

public /* method type */ removeStudent(/* parameters list */) {
       /* set the value of the object at index i to null */
}

public /* method type */ removeAndShiftStudent(/* parameters list */) {
       /* set the value of the object at index i++ to i
       Continue doing this for all the elements after index i till the end of the
array*/
}
}
```