

King Saud University
College of Computer & Information Science
CSC111 – Tutorial13
Arrays – II –
All Sections

Objectives:

This tutorial focuses on the following concepts:

- Passing an array element as method arguments/parameters.
- To know how to access array elements.
- To know how to iterate over arrays using loops
- To know how to manipulate arrays
- To know how to add elements to array and delete them
- To know how to search arrays
- Showing an example of **ArrayIndexOutOfBoundsException** Exception.
- Showing an example of **NullPointerException** Exception.
- Declaring an instance variable as an array.
- Creating an Array of Objects.
- Creating an Array of a primitive type.
- Sending an array of a primitive type as an argument of a method.
- Sending an array of a class type as an argument of a method.
- Methods that return an array.
- Stepping through array elements.
- Swapping array elements of both primitive and Class types (**reversing** arrays).

Exercise 1

1. Write a method **add** that receives an array of integers **arr**, the number of the elements in the array **arr** and an integer **n**. It then adds the integer **n** to the array **arr** if the number of elements in the array is less than its size. Method **add** uses another method **find** that checks if the integer **n** is in the array or not. Method **add** returns **false** if **n** can not be added or is already in **arr**.
2. Write a method **flipCoin** that receives an array of boolean **flips** and the number of coin flips so far. The method randomly flips a coin by calling method **nextBoolean** of class **java.util.Random** and stores the new flip in array **flips** if array is not full.
3. Write a method **deleteTweet** that receives your **tweets**, their number and a tweet that you would like to remove. The method then searches for the tweet and delete it from your twitter history. If tweet was not found, an error message is reported.
4. Write a method **findMove** that receives the history of moves made by a robot, the number of moves so far and a move. A move consists of two parts **dx** and **dy** which represent the amount of units traveled on x and y axis. The history is stored in two arrays one for each axis. The method looks up the move and returns its index in the two arrays otherwise it returns -1.

Solution

1)

```
public int find(int[] arr, int num, int n){
    for (int i = 0; i < num; i++) {
        if (arr[i] == n) {
            return i;
        }
    }
    return -1;
}

public boolean add (int[] arr, int num, int n) {
    if (num < arr.length) {
        if (find(arr, num, n) == -1){
            arr[num] = n;
            num++;
            return true;
        }
        else
            System.out.println("ERROR: ELEMENT ALREADY"
                + " ADDED.");
    } else
        System.out.println("ERROR: ARRAY IS FULL");
    return false;
}
```

2)

```
public void flipCoin (boolean[] flips, int num) {
    if (num < flips.length) {
        java.util.Random r = new java.util.Random();
        boolean newFlip = r.nextBoolean();
        flips[num] = newFlip;
        num++;
    } else
        System.out.println("ERROR: CAN NOT FLIP COIN");
}
```

3)

```
public void deleteTweet (String[] tweets, int numOfTweets,
    String tweet) {
    boolean found = false;
    for (int i = 0; i < numOfTweets && !found; i++) {
        if (tweets[i].equalsIgnoreCase(tweet)) {
            tweets[i] = tweets[numOfTweets];
            found = true;
        }
    }
    if (!found)
        System.out.println("ERROR: TWEET IS "
            + "ALREADY DELETED");
}
```

4)

```
public int findMove(double[] xMoves, double[] yMoves,  
    double dx, double dy, int numMoves) {  
    for (int i = 0; i < numMoves; i++)  
        if ((xMoves[i] == dx) && (yMoves[i] == dy))  
            return i;  
    return -1;  
}
```

Exercise 2

Suppose we have the following class **Customer**:

```
public class Customer {  
  
    private int id;  
    private String name;  
    private double totalSales;  
  
    <Constructors, Setters, and Getters are here>  
  
    public void addSales(double price)  
    {  
        totalSales = totalSales + price;  
    }  
  
    public boolean equalsC(Customer c)  
    {  
        return (this.id == c.id &&  
                this.name.equalsIgnoreCase(c.name) &&  
                this.totalSales == c.totalSales);  
    }  
}
```

Part A – passing an array element as an argument:

In a different class, suppose you created an array of objects of type **Customer** and an array of type **double** to store prices as follows:

```
Customer[] cmr = new Customer[3];  
double[] prices = new double[3];  
  
// Create objects for 1st and 2nd elements of cmr:  
cmr[0] = new Customer(1, "Ahmad", 0);  
cmr[1] = new Customer(2, "Saleh", 0);  
  
prices[0] = 10.0; prices[1] = 20.0; prices[2] = 30.0;
```

Q1: Write a code to call the method `equalsC` to compare the 1st element and the 2nd element of the array `cmr`.

Sol:

```
if(cmr[0].equalsC(cmr[1]))  
    System.out.println("They are Equal!");  
  
OR  
  
if(cmr[1].equalsC(cmr[0]))  
    System.out.println("They are Equal!");
```

Note that we are sending a single element, which is sending a single object of type `Customer` to the method.

Q2: Write a code to call the method `addSales` from the 1st customer. We want to add (send) the 2nd element from the array `prices`.

Sol:

```
22    cmr[0].addSales(prices[1]);
```

Part B – Dealing with runtime errors:

Q3: Suppose we run this code fragment:

```
24    int id = cmr[2].getId();
```

What will happen?

- Nothing, it will return the ID of the 3rd customer to be assigned to the variable `id`.
- There is a compilation error.
- There is a runtime error.

Sol:

```
Exception in thread "main" java.lang.NullPointerException  
at Tutorial13E1.main(Tutorial13E1.java:24)
```

- The answer is C, we got a runtime error which is a Null Pointer Exception.
- This happened because we tried to retrieve a value of an instance variable (from within an instance method) for an object that hasn't been created!
- In other words, we created an array of objects, but we did NOT create each object of the array.
- That means we need to write the following statement before line 24 above:

```
23    cmr[2] = new Customer(3, "Any Name", 0);
```

Q4: Suppose we run this code fragment:

```
28     for (int i = 0; i <= cmr.length; i++)
29         System.out.print("Name " + i + " = " + cmr[i].getName());
```

What will be the output, if any?


- Name0 = Ahmad Name1 = Saleh Name2 = Any Name
- There is a compilation error.
- There is a runtime error.

Sol:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
    at Tutorial13E1.main(Tutorial13E1.java:29)
```

- The answer is C, we got a runtime error which is an Array Index Out Of Bound Exception.
- This happened because we tried to access the element `cmr [3]` which is not part of the array since the array has only 3 elements (indexed from 0 to 2).
- In other words, the error was caused from this operator (it should be "<"):

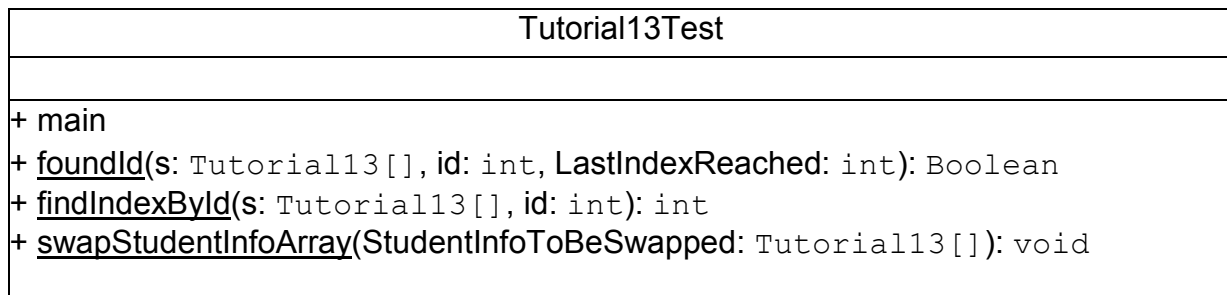
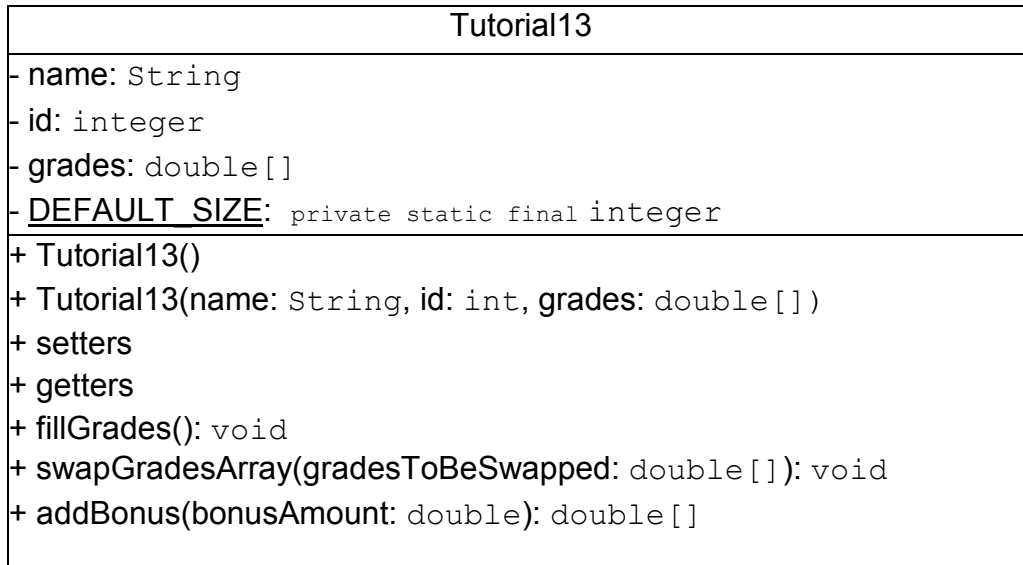
```
28     for (int i = 0; i <= cmr.length; i++)
29         System.out.print("Name " + i + " = " + cmr[i].getName());
```



Exercise 3

Part A:

Suppose we have the UML diagram for these two classes:



- Note that all methods of the class **Tutorial13Test** are static.
- Try to write the methods using their description, before you look at the code.

Now, here is the class **Tutorial13** with constructors, setters and getters code:

```
import java.util.Scanner;

public class Tutorial13 {

    private String name;
    private int id;
    private double[] grades;

    private static final int DEFAULT_SIZE = 10;

    public Tutorial13()
    {
        name = "no name";
        id = -1;
        grades = new double[DEFAULT_SIZE];
    }

    public Tutorial13(String newName, int newId, double[] newGrades)
    {
        name = newName;
        id = newId;
        grades = newGrades;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public double[] getGrades() {
        return grades;
    }

    public void setGrades(double[] grades) {
        this.grades = grades;
    }

    //The rest of the methods will be here, and they are
    discussed in the next pages.

}
```

In the class **Tutorial13**:

- The method **fillGrades ()** steps through the grades array of a certain student. The values are read from the user. Here is the method's code:

```
// This method will fill the grades of the current student:
public void fillGrades()
{
    Scanner kb = new Scanner(System.in);

    for(int i = 0; i < grades.length; i++)
    {
        System.out.println("Enter grade #" + (i+1) + " of the student " + name + ":");
        grades[i] = kb.nextDouble();
    }
}
```

- The method **addBonus (double bonusAmount)** will add a double value to all the elements of the **grades** array of the current student. The value added is received using the parameter **bonusAmount**. After it finished adding the bonus to all the grades, the method returns the whole array to the invoker. Here is the method's code:

```
// This method will receive a bonus amount to be added to all
// the current student's grades.
// Then, it returns the array grades to the invoker.
public double[] addBonus(double bonusAmount)
{
    // For readability, lets store the size of the array in the variable n:
    int n = grades.length;

    for(int i = 0; i < n; i++)
    {
        grades[i] = grades[i] + bonusAmount;
    }

    return grades;
}
```

- The method `swapGradesArray(double[] gradesToBeSwapped)` will swap the elements of the `grades` array. The swapping process is done as following (`grades` array has n elements):
 - o The 1st element is swapped with the last element ($n-1$).
 - o The 2nd element is swapped with the element ($n-2$).
 - o The 3rd element is swapped with the element ($n-3$).
 - o And so on ...

```
// This method will swap the elements of the grades array of size n,  
// we will swap the 1st element (0) with the last element (n-1),  
// the 2nd element (1) with the element (n-2) and so on...  
// Note that we swap values here.  
public void swapGradesArray(double[] gradesToBeSwapped)  
{  
    // First, declare a temporary variable to store the value to be swapped.  
    double temp;  
  
    // For readability, lets store the size of the array in the variable n:  
    int n = gradesToBeSwapped.length;  
  
    // Now, loop to the half of the array,  
    // we do not need to reswap the rest of the elements!  
    for(int i = 0; i < n/2; i++)  
    {  
        temp = gradesToBeSwapped[i];  
        gradesToBeSwapped[i] = gradesToBeSwapped[(n-1)-i]; // note that the last element  
is (n-1).  
        gradesToBeSwapped[(n-1)-i] = temp;  
    }  
}
```

- Note that this is a swapping of **double** elements, so the values are exchanged.
- In the next page we show an illustration of how the swapping is done.

- Assume that we have this **grades** array that contains 5 grades:

→ $n = 5, (n/2) = 2$

Indices:

20	15	30	10	25
0	1	2	3	4

- At the beginning, the current grade will be the value of **grades** [0] which is 20:

↓

20	15	30	10	25
----	----	----	----	----

- The value of **grades** [i] will be swapped with the value of **grades** [(n-1) - i].
When $i = 0$, then $(n-1) - i = (5-1) - 0 = 4$.

→ **grades** [0] will be swapped with **grades** [4] .

- The array will look like this, and the current grade will be the value of **grades** [1] which is 15:

↓

25	15	30	10	20
----	----	----	----	----

- Again, The value of **grades** [i] will be swapped with the value of **grades** [(n-1) - i].
When $i = 1$, then $(n-1) - i = (5-1) - 1 = 3$.

→ **grades** [1] will be swapped with **grades** [3] .

- The array will look like this, and the current grade will be the value of **grades** [2] which is 30:

↓

25	10	30	15	20
----	----	----	----	----

- However, now that $i = 2$, the loop will end since i is not $< (n/2)$.

→ $2 < 2$ evaluates to **false**.

- Now the swapping is done.

- End of the Class **Tutorial13**.
-
-

In the class **Tutorial13Test**:

- We have the **main** method, and three other **static** helping methods. We will discuss the helping methods first. Then, we will discuss the **main** method later.
- The method **foundId(Tutorial13[] s, int id, int lastIndexReached)** will look for a student ID and return **true** if the student was found. This method will help us to avoid adding an existing ID, since IDs must be unique. It will receive a student information array, which is an array of class **Tutorial13** type (array of objects). In addition, it will receive the ID of the student we are looking for, and the last index reached so we look for previous elements only! (The rest of the elements will be **null!**).

```
// This method will look for an ID, and returns true if the ID was found in the
// studentInfo array. Note that we will loop over the previous elements of
// the studentInfo array (hence the parameter lastIndexReached), since we
// haven't filled the rest of the elements yet!

public static boolean foundId(Tutorial13[] s, int id, int lastIndexReached)
{
    boolean found = false;

    for(int i = 0; i < lastIndexReached; i++)
    {
        if(s[i].getId() == id)
            found = true;
    }

    return found;
}
```

- If you did not understand the method, you can look at how we used it in the **main** method and it should clarify this method.

- The method `findIndexById(Tutorial13[] s, int id)` will look for the location of the student who's ID is `id` in the student information array. It will return the location (index) if ID is found; otherwise, it will return `-1` if not found.

```
// This method will look for the location of the student who's ID is id,  
// and returns the location (index) if found, or -1 if not found.
```

```
public static int findIndexById(Tutorial13[] s, int id)  
{  
    for(int i = 0; i < s.length; i++)  
    {  
        if(s[i].getId() == id)  
            return i;  
    }  
  
    return -1;  
}
```

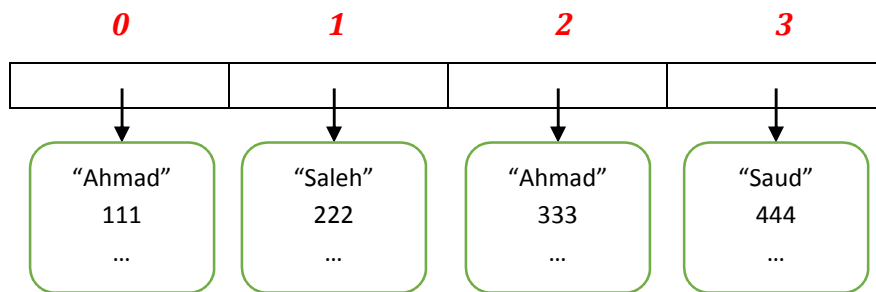
- The method `swapStudentInfoArray(Tutorial13[] StudentInfoToBeSwapped)` will swap the elements of the `studentInfo` array. The swapping process is done as following (`studentInfo` array has `n` elements):
 - o The 1st element is swapped with the last element (`n-1`).
 - o The 2nd element is swapped with the element (`n-2`).
 - o The 3rd element is swapped with the element (`n-3`).
 - o And so on ...

```
// This method will swap the elements of the StudentInfo array of size n,  
// we will swap the 1st element (0) with the last element (n-1),  
// the 2nd element (1) with the element (n-2) and so on...  
// Note that we swap objects here.
```

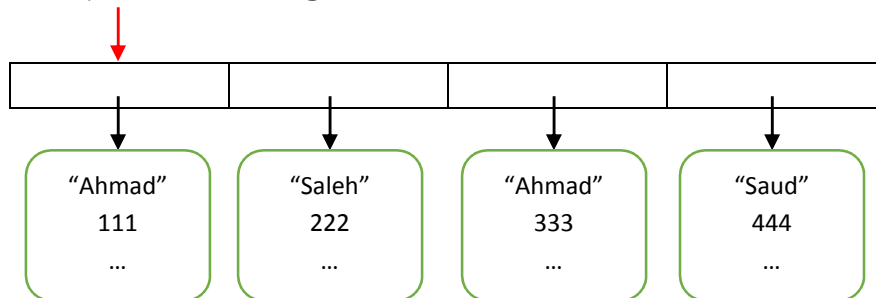
```
public static void swapStudentInfoArray(Tutorial13[] StudentInfoToBeSwapped)  
{  
    // First, declare a temporary variable to store the object to be swapped.  
    Tutorial13 temp; // Note that no need for using "new", since we do not need the  
                    // variable for creating a new object here,  
                    // but just for using it as a temporary object holder.  
  
    // For readability, lets store the size of the array in the variable n:  
    int n = StudentInfoToBeSwapped.length;  
  
    // Now, loop to the half of the array,  
    // we do not need to re-swap the rest of the elements!  
    for(int i = 0; i < n/2; i++)  
    {  
        temp = StudentInfoToBeSwapped[i];  
        StudentInfoToBeSwapped[i] = StudentInfoToBeSwapped[(n-1)-i]; // note that the  
last element is (n-1).  
        StudentInfoToBeSwapped[(n-1)-i] = temp;  
    }  
}
```

- To illustrate, assume that we have this **studentInfo** array that contains 4 students → $n = 4, (n/2) = 2$
- Remember that **studentInfo** is an array of objects. Each element contains a reference to an object of type **Tutorial113**.

Index:

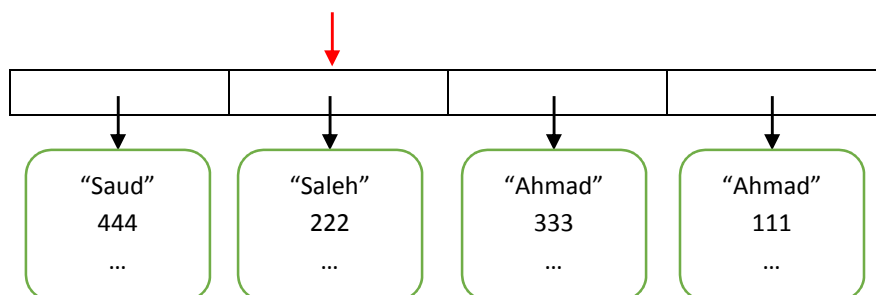


- At the beginning, the current student will be referenced at **studentInfo[0]** which is the object containing ID = 111:

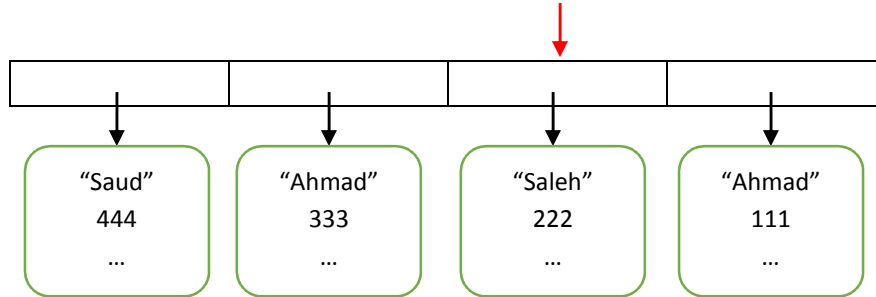


- The reference at **studentInfo[i]** will be swapped with the reference at **studentInfo[(n-1)-i]**. So, when $i = 0$, then $(n-1) - i = (4-1) - 0 = 3$.
→ **studentInfo[0]** will be swapped with **studentInfo[3]**.

- The array will look like this, and the current student will be referenced at **studentInfo[1]** which is the object containing ID = 222:



- Again, The reference at `studentInfo[i]` will be swapped with the reference at `studentInfo[(n-1)-i]`. So, when $i = 1$, then $(n-1)-i = (4-1)-1 = 2$.
 → `studentInfo[1]` will be swapped with `studentInfo[2]`.
- The array will look like this, and the current student will be referenced at `studentInfo[2]` which is the object containing ID = 222:



- However, now that $i = 2$, the loop will end since i is not $< (n/2)$.
 → $2 < 2$ evaluates to **false**.
- Now the swapping is done.

- The **main** method: For our program, we want to do the following:

1. Create an array of objects `studentInfo` that contains students' information.
2. Loop over the `studentInfo` array and create objects as elements of the array.

I.e. adding elements to the array of objects "`studentInfo`":

- a) First, we ask the user to give us the ID of the current student. Then, before we ask the user for the rest of the information, we check if the ID is unique or not.
- b) Then, we ask the user to give us the name the current student.
- c) After that, we set up the grades array of the current student.
- d) Now, we can create the object to store the current student information.

e) Now that the **studentInfo** is created, we will fill the current student's grades. I.e. fill the elements of the "**grades**" array of the current student.

3. After we finish looping over all the students, and fill their information, we want to try out our interesting methods!
 4. First, we will swap a certain student's grades after we give him a bonus of 2 to all his grades.
 - Print the **grades** array of the selected student to assure that the bonus adding and the swapping worked properly.
 5. Then, swap the whole **studentInfo** array.
 - Print the **studentInfo** array to assure that the swapping worked properly.
 6. We are done!
-
-

Part B:

- Look at the sample run in the next page and try to write the main program yourself by using all the 6 steps above and the sample run.

- Let us assume that this is a normal sample run for our program:

```
Enter how many students you want to enter:
3
Enter the ID of student #1 :
111
Enter the name of student #1 :
Ahmad
Enter how many grades you want to enter for student #1 :
4
Enter grade #1 of the student Ahmad:
10
Enter grade #2 of the student Ahmad:
20
Enter grade #3 of the student Ahmad:
30
Enter grade #4 of the student Ahmad:
40
Enter the ID of student #2 :
222
Enter the name of student #2 :
Saleh
Enter how many grades you want to enter for student #2 :
5
Enter grade #1 of the student Saleh:
15
Enter grade #2 of the student Saleh:
10
Enter grade #3 of the student Saleh:
22
Enter grade #4 of the student Saleh:
31
Enter grade #5 of the student Saleh:
9
Enter the ID of student #3 :
333
Enter the name of student #3 :
Saud
Enter how many grades you want to enter for student #3 :
2
Enter grade #1 of the student Saud:
50
Enter grade #2 of the student Saud:
20

==== Done Adding ====

Now, enter the Id of the student that you want to give a bonus and swap his grades:
111
student 111 grades in order:
42.0, 32.0, 22.0, 12.0,

Now, we will swap the whole studentInfo array!
StudentInfo IDs in order:
333, 222, 111,
=====
Tutorial 13 is Finished!
=====
```

- After knowing our program's steps and looking at a sample run, let us write the **main** method together based on the 6 steps above:

1. Create an array of objects **studentInfo** that contains students' information. Here is the code fragment for that:

```
import java.util.Scanner;
public class Tutorial13Test {
    public static void main(String[] args) {

        int studentsNum, gradesNum; // To store arrays sizes
        int id;
        Scanner kb = new Scanner(System.in);
        double[] studentGrades; // a temporary array to store the grades
                                // of a current student.

        // Now let us create an array of objects to contain students' information:
        System.out.println("Enter how many students you want to enter:");
        studentsNum = kb.nextInt(); // studentInfo's array size.
        Tutorial13[] studentInfo = new Tutorial13[studentsNum]; // ← Array of Objects.
        . . .
    }
}
```

Program header + local variables declaration.

1

2. Loop over the **studentInfo** array and create objects as elements of the array:

- a) First, we ask the user to give us the ID the current student. Then, before we ask the user for the rest of the information, we check if the ID is unique or not.

```
// Now we loop over the studentInfo array and create objects as elements of the array:
// i.e. Adding elements to the array of objects "studentInfo".
for(int i = 0; i < studentInfo.length; i++)
{
    // First, we ask the user to give us the id the current student:
    System.out.println("Enter the ID of student #" + (i+1) + " :");
    id = kb.nextInt();

    // Before creating the current student object,
    // we need to check if the ID is already entered (unique ID):
    while(foundId(studentInfo, id, i)) //while the ID is not unique.
    { // i here is the last index reached,
        // saves searching in empty elements (null elements).
        System.out.println("The ID you entered is not unique!");
        System.out.println("Enter the ID of student #" + (i+1) + " again:");
        id = kb.nextInt();
    }

    . . . <The Rest of the for-Loop comes next>
}
```

Loop over all the array!

While the entered ID already exist in the array!

b) Then, we ask the user to give us the name the current student.

```
    . . .  
  
    // Then, we ask the user to give us the name the current student:  
    System.out.println("Enter the name of student #" + (i+1) + " :");  
    String name = kb.next();  
  
    . . .
```

c) After that, we set up the grades array of the current student.

```
    . . .  
  
    // After that, we set up the grades array of the current student:  
    System.out.println("Enter how many grades you want to enter for student #" + (i+1) + " :");  
    gradesNum = kb.nextInt();  
    studentGrades = new double[gradesNum];  
  
    . . .
```

← Create an empty array who's size is entered from the user!

d) Now, we can create the object to store the current student information.

```
    . . .  
  
    // Now, we can create the object to store the current student information:  
    studentInfo[i] = new Tutorial13(name, id, studentGrades); // we used the 2nd constructor  
    here.  
  
    . . .
```

↑ Note how we send a whole array as an argument.

e) Now that the **studentInfo** is created, we will fill the current student's grades. I.e. fill the elements of the "**grades**" array of the current student.

```
    . . .  
  
    // Now that the studentInfo is created, let us fill their grades:  
    // i.e. fill the elements of the "grades" array of the current student.  
    studentInfo[i].fillGrades();  
    }  
    System.out.println("\n==== Done Adding ==== \n");  
  
    . . .
```

← Simply invoke the fillGrades() method for the current object!

← This marks the end of the for-Loop.

3. After we finish looping over all the students, and fill their information, we want to try out our interesting methods!
4. First, we will swap a certain student's grades after we give him a bonus of 2 to all his grades.

```
    . . .
// Now, let us swap a certain student's grades after we give him a bonus of 2 to all his
grades:
System.out.println("Now, enter the Id of the student that you want to give a bonus and swap
his grades:");
id = kb.nextInt();
int index = findIndexById(studentInfo, id);
if(index != -1)
{
    double [] g = studentInfo[index].addBonus(2); // add 2 to all this student grades.
    studentInfo[index].swapGradesArray(g);
}
else
    System.out.println("ID not Found!");
    . . .
```

Get the index of the entered student ID, then use this index to access the actual object!

- Print the **grades** array of the selected student to assure that the bonus adding and the swapping worked properly.

```
    . . .
// To make sure that the swapping was done correctly,
// we will print the grades of the student selected:
System.out.println("student " + studentInfo[index].getId() + " grades in order:");
studentGrades = studentInfo[index].getGrades();
// Note that we used the array "studentGrades" above,
// since we were done using it.
for(int i = 0; i < studentGrades.length ; i++)
{
    System.out.print(studentGrades[i] + ", ");
}
System.out.println();
    . . .
```

GetGrades() is a method that returns a whole array!

5. Then, swap the whole **studentInfo** array.

```
    . . .  
  
    // Now, let us swap the whole studentInfo array:  
    System.out.println("\nNow, we will swap the whole studentInfo array!");  
    Tutorial13Test.swapStudentInfoArray(studentInfo);  
    // we called the static method above using the class name.  
    . . .
```

- Print the **studentInfo** array to assure that the swapping worked properly.

```
    . . .  
  
    // To make sure that the swapping was done correctly,  
    // we will print the ID'd of the students in the array:  
    System.out.println("StudentInfo IDs in order:");  
    for(int i = 0; i < studentInfo.length; i++)  
    {  
        System.out.print(studentInfo[i].getId() + ", ");  
    }  
    System.out.println();  
  
    . . .
```

7. We are done!

```
    . . .  
  
    System.out.println("=====");  
    System.out.println("Tutorial 13 is Finished!");  
    System.out.println("=====");  
}
```
