Problem

- Create a program that inputs 3 numbers from the user and computes the average and standard deviation.
public class AverageStdDev {
    public static void main(String[] args) {
        double d1, d2, d3;
        double avg, varianceSum, variance, stdDev;

        d1 = Double.parseDouble(JOptionPane.showInputDialog(null, "Enter number"));
        d2 = Double.parseDouble(JOptionPane.showInputDialog(null, "Enter number"));
        d3 = Double.parseDouble(JOptionPane.showInputDialog(null, "Enter number"));

        avg = (d1 + d2 + d3) / 3.0;

        varianceSum = Math.pow((d1-avg),2) + Math.pow((d2-avg),2) + Math.pow((d3-avg),2);
        variance = varianceSum/3;

        stdDev = Math.sqrt(variance);

        System.out.println("Average is:" + avg + ", Deviation is:" + stdDev);
    }
}
Problem

- Create a program that inputs 100 numbers from the user and computes the average and standard deviation.
Arrays

- Having a separate variable for value is cumbersome
- Each value is similar, so can we treat them as part of a collection of numbers?
- Arrays are an important type of collection
- The collection has a name, and each member is referenced by a number.
- In Java, an array is an indexed collection of data values of the same type (primitive or object).
int i;
double d[];
double sum, avg, variance, varianceSum, stdDev;

d = new double[100];

for(i=0; i<100; i++)
    d[i] = Double.parseDouble(JOptionPane.showInputDialog(null, "Enter number " + i));

sum = 0;
for(i=0; i<100; i++)
    sum += d[i];
avg = sum/100;

varianceSum = 0;
for(i=0; i<100; i++)
    varianceSum += Math.pow((d[i]-avg),2));
variance = varianceSum/100;

stdDev = Math.sqrt(variance);

System.out.println("Average is:" + avg + ", Deviation is:" + stdDev);
Arrays are objects

double d[];

d = new double[10];

d[2] = 3;

j = 2 + d[2];

j = 5
double[] wages = new double[7];
String[] dayOfWeek = new String[7];
dayOfWeek[0] = "Monday";
dayOfWeek[1] = "Tuesday";
...
double averageWage, sum = 0.0;

for (int i = 0; i < 7; i++) {
    wages[i] = Double.parseDouble(
        JOptionPane.showInputDialog(null,
            "Wages for " + dayOfWeek[i] ));
    sum += wages[i];
}
averageWage = sum / 7;

The same pattern for the remaining five days.
The actual day name instead of a number.
Index out of bounds

Whenever an array member is accessed, the index must be a valid value between 0 and length of array - 1.

If it is not, then the program will terminate with an error:

- A run time exception called `ArrayIndexOutOfBoundsException`
- How to handle this situation will be dealt with later in the course.
- For now -- it should be avoided.
In Java, we are not required to declare the size at compile time.
The following code prompts the user for the size of an array and declares an array of the designated size:

```java
int size;
int[] number;
size = Integer.parseInt(JOptionPane.showInputDialog(null, "Size of the array:"));
number = new int[size];
```
Length of an array

- Each array has a special data member that records the number of members of the array: `length`
  - Note: not a method as in String.length()

```java
double d[];
...
System.out.println("Array d has "+ d.length + " elements");
```
Array Initialization

- Like other data types, it is possible to declare and initialize an array at the same time.
- The size of the array is equal to the number of items in the initialization.

```java
int[] primes = { 2, 3, 5, 7, 11, 13, 17, 19};
double[] measurements = { 45, 3.42, 2.66 };;
String[] daysOfWeek = {"Monday", "Tuesday", "Wednesday", 
                        "Thursday", "Friday", "Saturday", 
                        "Sunday" };;
```
Initializing arrays

- If we do not initialize values at creation time, then the elements are initialized to the default value of the corresponding type.
- It is also common to initialize an array using a for loop.

```java
int[] odd, even;
odd = new int[100000];
even = new int[100000];

for(int i = 0; i < 100000; i++){
   odd[i] = 2*i+1;
   even[i] = 2*i;
}
```
Problem

Create a program that manages all students for CS180

- each student object will have:
  - an ID (string)
  - Last Name
  - GPA
Arrays of Objects

- In Java, in addition to arrays of primitive data types, we can declare arrays of objects.
- An array of primitive data is a powerful tool, but an array of objects is even more powerful.
- The use of an array of objects allows us to model the application more cleanly and logically.
The Student Class

class Student {
    private String name, id;
    private double gpa;

    public void Student(){
        name = JOptionPane.showInputDialog(null, "Enter Name:");
        id = JOptionPane.showInputDialog(null, "Enter ID:");
        gpa = 0.0;
    }

    public void printNeatly(){
        System.out.println("    "+name);
        System.out.println("    ID: "+id);
        System.out.println("    GPA: " + gpa);
    }

    public void setName(String studentName){
        name = studentName;
    }

    public String getName(){
        return name;
    }

    public String getId(){
        return id;
    }

    public double getGpa(){
        return gpa;
    }

    public void setGpa(double g){
        gpa = g;
    }
}

// CONTINUED ...
Creating the Roster object array

```java
Student[ ] student;
student = new Student[10];
student[0] = new Student();
student[1] = new Student();
```

![Diagram of Roster object array with two students: Jane and John. Jane has an ID of 2343 and a GPA of 0.0. John has an ID of 3421 and a GPA of 0.0.]}
public class Roster{
    public static void main(String[] args) {
        Student[] student;

        student = initializeRoster();
        for(i=0; i< student.length;i++)
            student[i].printNeatly();
    }

    public static Student[] initializeRoster(){
        Student[] st;
        int classSize, i;
        classSize = Integer.parseInt(
            JOptionPane.showInputDialog(null,
                "Enter number of students in class"));
        st = new Student[classSize];
        for(i=0; i<classSize;i++)
            st[i] = new Student();
        return st;
    }
}
Caution

- Creating an array of objects only creates the references.
- They are all initialized to null values -- i.e. they don't reference valid objects.
- Trying to access this reference will cause an error: a Null Pointer Exception.

```java
public class Roster{
    public static void main(String[] args) {
        Student[] student;
        student[0].printNeatly();
    }
}
```
Finding a Student

```java
public class Roster {
    public static void main(String[] args) {
        Student[] studentList;
        Student student1;

        studentList = initializeRoster();
        student1 = findStudent("2334", studentList);
        if (student1 == null)
            System.out.println("Student with id 2334 not found in class");
        else
            student1.printNeatly();
    }

    public static Student findStudent(String id, Student[] sList) {
        Student s;
        int i;
        for (i = 0; i < sList.length; i++)
            if (id.equals(sList[i].getId()))
                return sList[i];
        return null;
    }
}
```
Finding Student with Highest GPA

```java
public class Roster{
    public static void main(String[] args) {
        Student[] studentList;
        Student student1;

        studentList = initializeRoster();
        student1 = topStudent(studentList);
        student1.printNeatly();
    }

    public static Student topStudent(Student[] sList){
        Student maxStudent = sList[0];
        double maxGpa = maxStudent.getGpa();
        for (int i=1; i < sList.length; i++)
            if(sList[i].getGpa()>maxGpa){
                maxGpa = sList[i].getGpa();
                maxStudent = sList[i];
            }
        return maxStudent;
    }
}
```

Tuesday, February 21, 2012
Deleting an object from an array

```java
student[0] = null;
```

![Diagram of an array with student objects](image)

```java
: Student
  name -> Jane
  id  -> 2343
  gpa -> 0.0

: Student
  name -> John
  id  -> 3421
  gpa -> 0.0
```
With the approach of setting the deleted reference to null, we have to be careful to test each member before accessing the referenced object.

```java
if(student[0]!=null) {
    student[0].printNeatly();
}
```

Otherwise we could crash the program.
Deleting an object from an array

```java
lastEntryIndex = 1;
student[0] = student[lastEntryIndex];
student[lastEntryIndex] = null;
```
Finding a Student

- With keeping the array packed -- the first null element indicates the end of the array.

```java
public static Student findStudent(String id, Student[] sList){
    Student s;
    int i;
    while(sList[i]!=null && !(id.equals(sList[i].getId())))
        i++;
    if(sList[i]==null)
        return null;
    return sList[i];
}
```
Array data type

- An array with elements of type T is a new data type represented as T[]
  - int [] age;
  - double salary[];
  - Person student[];
  - age is of type int[]
  - salary is of type double []
  - student is of type Person []

- Each element of this array is of type T
  - age[0], salary [0] are int data types
  - student [1] is a Person object.
There are two acceptable alternatives.

- all four are arrays of double values.

- array5 is an array of doubles, but d1 is a single double variable

A new array is created each time. Reference to old array is lost.
Arguments and return values

- An array can be returned by a method.
- The return type must be an array in this case.

```java
public int[] doubleValues(int[] inArray)
```

- An element can be passed to any method that accepts an argument of the base type of the array.

```java
double x[] = new double[5];
y = Math.exp(x[2]);
```
The main method

- Recall the only argument to main:
  ```java
  public static void main(String[] args)
  ```
- The argument is an array of strings. Each element of this array is set to the words that follow the program name when executing:
  ```bash
  %java Test one two three
  ```
- In main: `args[0]` is “one” `args[1]` is “two” and `args[2]` is three.
- Also, `args.length` will be 3 for this case.
Multi-Dimensional Arrays

- Multi-dimensional arrays are useful for representing multi-dimensional data. E.g.,
  - Grid cells in a checkers game.
  - A distance table between cities.
  - A list of coordinates (2D or 3D) of polygon.
Declaring and Creating a 2-D Array

```java
int[][] ticTacToeCells;
ticTacToeCells = new int[3][3];
```

```java
int ticTacToeCells [][];
ticTacToeCells = new int[3][3];
```

```java
int [][] ticTacToeCells = new int[3][3];
```

```java
int ticTacToeCells [][] = new int[3][3];
```
```java
char[][] ticTacToeCells;
ticTacToeCells = new char[3][3];
ticTacToeCells[0][2] = 'X';
ticTacToeCells[1][1] = 'O';
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>O</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Problem

Create a program to input the pay hours worked by 5 employees over 10 days. The program should output the total hours per employee and the total hours per day.

- Each employee is identified as a number (0-4)
- Each day is identified as a number (0-9)
public class HoursWorked{
    public static void main(String[] args) {
        int[][] hours = new int[5][10];
        int emp, day, total;
        for (emp = 0; emp < 5; emp++)
            for (day = 0; day < 10; day++)
                hours[emp][day] = Integer.parseInt(
                    JOptionPane.showInputDialog(null,
                        "Enter hours for Employee " + emp + " day " + day));
    }
}
public class HoursWorked{
    public static void main(String[] args) {
        int[][][] hours = new int[5][10];
        int emp, day, total;
        // input hourly data
        for(emp=0; emp<5; emp++){
            total = 0;
            for(day=0; day < 10; day++)
                total += hours[emp][day];
            System.out.println("Employee "+ emp + " worked "+ total + " hours");
        }
    }
}
public class HoursWorked{
    public static void main(String[] args) {
        int[][] hours = new int[5][10];
        int emp, day, total;
        // input hourly data
        for(day=0; day < 10; day++) {
            total = 0;
            for(emp=0; emp<5; emp++)
                total += hours[emp][day];
            System.out.println(total + " hours worked on day"+ day);
        }
    }
}
Java Implementation of 2-D Arrays

- The sample array creation

```java
hours = new double[4][5];
```

is really a shorthand for

```java
hours = new double [4][ ];
```

```java
hours[0] = new double [5];
hours[1] = new double [5];
hours[2] = new double [5];
hours[3] = new double [5];
```
Java Implementation

```java
hours = new double[4][5];
hours = new double [4][];
hours[0] = new double [5];
```

double[][]

double[]
hours[1]
hours[1][2]

double

0 1 2 3 4

0 1 2 3 4
Java Implementation

```java
hours.length  →  4
hours[1].length  →  5
hours[1][2].length  →  ERROR!
```
In Java, subarrays may have different lengths.

```java
double distanceArray[][] = new double[4][];
for (int i = 0; i < 4; i++)
    distanceArray[i] = new double[i + 1];
```

results in an array that looks like:

```
distanceArray

distanceArray[1]

distanceArray[2][1]
```
Limitation of Arrays

- Once an array object is created, its size is fixed -- it cannot be changed.
- If we need to store more elements than the size with which an array was created, we have to
  - Create a new larger array
  - Copy all elements from current to new array
  - Change the reference to the new array
- Alternatively, we can use Java Collections: more later in course.