Array, ArrayList

Cheng, Wei  COMP110-001   June 5, 2014
More Complicated Issues

• Constructor is used to create instance of class
  – Can constructor be private?

  – Can a constructor call another constructor?
The Random class

• Why methods in random are not static?
• Don’t we just want a method that returns a random number whenever called?

Random rand = new Random();
int newNum = rand.nextInt();
Think back to Lab 4

• You wrote a program to read in a list of basketball scores from the user and output a bunch of statistics
Let’s get rid of extra stuff for now

System.out.println("Enter the list of basketball scores " +
    "(enter a negative number to end your list): ");

while ( (score = keyboard.nextInt()) >= 0)
{
    totalGames++;
    scoreSum += score;
}

if (totalGames > 0)
{
    double average = (double) scoreSum / (double) totalGames;
    System.out.println("Average score: "+ average);
}
What if...

• ...we wanted to know which of the scores entered were
  – above average?
  – below average?

• How would we do it?
System.out.println("Enter 5 basketball scores:");

int score1 = keyboard.nextInt();
int score2 = keyboard.nextInt();
int score3 = keyboard.nextInt();
int score4 = keyboard.nextInt();
int score5 = keyboard.nextInt();

double average =
    (double) (score1 + score2 + score3 + score4 + score5) / 5.0;

System.out.println("Average score: " + average);

// repeat this for each of the 5 scores
if (score1 > average)
    System.out.println(score1 + ": above average");
else if (score1 < average)
    System.out.println(score1 + ": below average");
else
    System.out.println(score1 + ": equal to the average");
What if we had 80 scores?

System.out.println("Enter 80 basketball scores:");

int score1 = keyboard.nextInt();
int score2 = keyboard.nextInt();
int score3 = keyboard.nextInt();
// ...are we done yet?
int score23 = keyboard.nextInt();
int score24 = keyboard.nextInt();
int score25 = keyboard.nextInt();
// ...how about now?
int score67 = keyboard.nextInt();
int score68 = keyboard.nextInt();
// ...all typing and no play makes Homer...go crazy?
int score80 = keyboard.nextInt();
// ...whew!

double average = (double) (score1 + score2 + score3 + score4 + ... score23 + score24 + score25 + ... ) / 80.0;
System.out.println("Average score: " + average);
// now do below/above average check for all 80 scores
Well, *that* was a pain

- Arrays to the rescue!
- An **array** is a collection of items of the same type
- Like a list of variables, but with a nice, compact way to name them
- A special kind of object in Java
Creating an array

```java
int[] scores = new int[5];
```

- This is like declaring 5 strangely named variables of type `int`:
  - scores[0]
  - scores[1]
  - scores[2]
  - scores[3]
  - scores[4]
Indexing

• Variables such as scores[0] and scores[1] that have an integer expression in square brackets are known as:
  – *indexed variables, subscripted variables, array elements*, or simply *elements*

• An *index* or *subscript* is an integer expression inside the square brackets that indicates an array element
Indexing

• Where have we seen the word index before?
  – String’s indexOf method

• Index numbers start with 0. They do NOT start with 1 or any other number.
Indexing

• The number inside square brackets can be any integer expression
  – An integer: scores[3]
  – Variable of type int: scores[index]
  – Expression that evaluates to int: scores[index*3]

• Can use these strangely named variables just like any other variables:
  – scores[3] = 68;
  – System.out.println(scores[1]);
Array

- The array itself is referred to by the name `scores` (in this particular case)
System.out.println("Enter 5 basketball scores:");

int[] scores = new int[5];
int scoreSum = 0;
for (int i = 0; i < 5; i++)
{
    scores[i] = keyboard.nextInt();
    scoreSum += scores[i];
}
double average = (double) scoreSum / 5;
System.out.println("Average score: "+ average);

for (int i = 0; i < 5; i++)
{
    if (scores[i] > average)
        System.out.println(scores[i] + ": above average");
    else if (scores[i] < average)
        System.out.println(scores[i] + ": below average");
    else
        System.out.println(scores[i] + ": equal to the average");
}
Array details

• Syntax for creating an array:

```
Base_Type[] Array_Name = new Base_Type[Length]
```

• Example:

```
int[] pressure = new int[100];
```

• Alternatively:

```
int[] pressure;
pressure = new int[100];
```
Array details

• The base type can be any type
  ```java
double[] temperature = new double[7];
Student[] students = new Student[35];
  ```

• The number of elements in an array is its **length, size, or capacity**
  – temperature has 7 elements, temperature[0] through temperature[6]
  – students has 35 elements, students[0] through students[34]
Finding the length of an existing array

• An array is a special kind of object
  – It has one public instance variable: `length`
  – `length` is equal to the length of the array
    ```java
    Pet[] pets = new Pet[20];
    pets.length has the value 20
    ```
  – You cannot change the value of `length`
  – *Once declared, an array cannot be resized!*
    • *Why array cannot be resized?*
Returning to our example...

```java
System.out.println("Enter 5 basketball scores:");

int[] scores = new int[5];
int scoreSum = 0;
for (int i = 0; i < scores.length; i++)
{
    scores[i] = keyboard.nextInt();
    scoreSum += scores[i];
}
double average = (double) scoreSum / 5;
System.out.println("Average score: " + average);

for (int i = 0; i < scores.length; i++)
{
    if (scores[i] > average)
    {
        System.out.println(scores[i] + ": above average");
    }
    else if (scores[i] < average)
    {
        System.out.println(scores[i] + ": below average");
    }
    else
    {
        System.out.println(scores[i] + ": equal to the average");
    }
}
```
Be careful with your indices

• Indices MUST be in bounds
  
  ```java
  double[] entries = new double[5];
  entries[5] = 3.7;  // ERROR! Index out of bounds
  ```

• Your code will compile if you are using an index that is out of bounds, but it will give you an error when you run your program
A typical problem in programming

• We do not know the size of input data
• E.g., a program reads a list of numbers from user or a file.
  
• The program does not know how many numbers are there beforehand.
• But we have to create an array beforehand to store input and the array cannot be resized!
How to solve this problem?

• Fixed array size VS unknown data size

A naïve solution:

Declare a very large array.
But how large is large enough?
Also it is a waste of memory.
A more practical solution

• Replace the old array with a new bigger array when it gets full

  – Initialize an array
  • Fill in data.
  • If the array is full,
    – we create a new array of twice the size
    – Copy all data from the old array to the new array
    – Make the new array the “current” array

• How many copy operations do we need in the worst case?
ArrayList

• “Dynamic Array”
• This is a common problem and the solution is quite complicated
• Java has several built-in classes that implements the copy-once-get-full strategy and can serve as a “dynamic array” – array that can be resized

• A popular one is ArrayList
ArrayList

- Internally, it maintains an array of specified type.
- You can view it as a list of data.
- To initialize a list of particular type:

```
ArrayList<Data_type> var = new ArrayList<Data_type>();
```

e.g.: `ArrayList<Student> myList = new ArrayList<Student>();`

or

```
ArrayList<Data_type> var =
    new ArrayList<Data_type>(initial_capacity);
```
ArrayList

- You cannot access ArrayList elements with direct indices: [..]
- But you can use many methods provided:

  - `add(Type element)`, \( \leftarrow \) element must be of the same type
  - `get(int index)`, \( \leftarrow \) get the element at the index
  - `remove(int index)`,
  - `indexOf(Type element)`,
  - `set(int index, Type element)`,
  - `size()`
**ArrayList**

- **Pay attention here. This is related to your assignment 4**
- ArrayList can only store objects (class type). It cannot store primitive types (int, double ...)

  ```java
  ArrayList<int> numbers = new ArrayList<int>();
  This will not compile
  ```

- What if we want to store a list of integer numbers?
- **Use wrapper class!**
Wrapper classes

- All primitive types have an associated wrapper class
- Start with upper case letters
  - Byte
  - Short
  - Integer
  - Long
  - Float
  - Double
  - Character
  - Boolean
Use ArrayList to store primitive values

ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add( 1 ); // a shortcut to numbers.add( new Integer( 1 ) );
numbers.add( 2 );
numbers.add( 3 );

System.out.println( numbers.size() ); // this prints out 3

for( int i = 0; i< numbers.size(); i++ ) {
    System.out.println( numbers.get(i) ); // print out all elements
}

numbers.remove(0); // we have 2 & 3 left
numbers.set(1, 15); // we have 2 & 15 now
Summary

• Array – fixed size. Good if the size is known and fixed
  – Arr[ index ] : use as variable
  – Arr.length : this is a public instance variable. Not method

• ArrayList – dynamic size. Use methods to manipulate data.
  – add, get, set, size, remove ..... Check documentation
  – Only stores objects. Need wrapper class for primitive values