

COSC 111: Computer Programming I

Dr. Bowen Hui
University of British Columbia
Okanagan

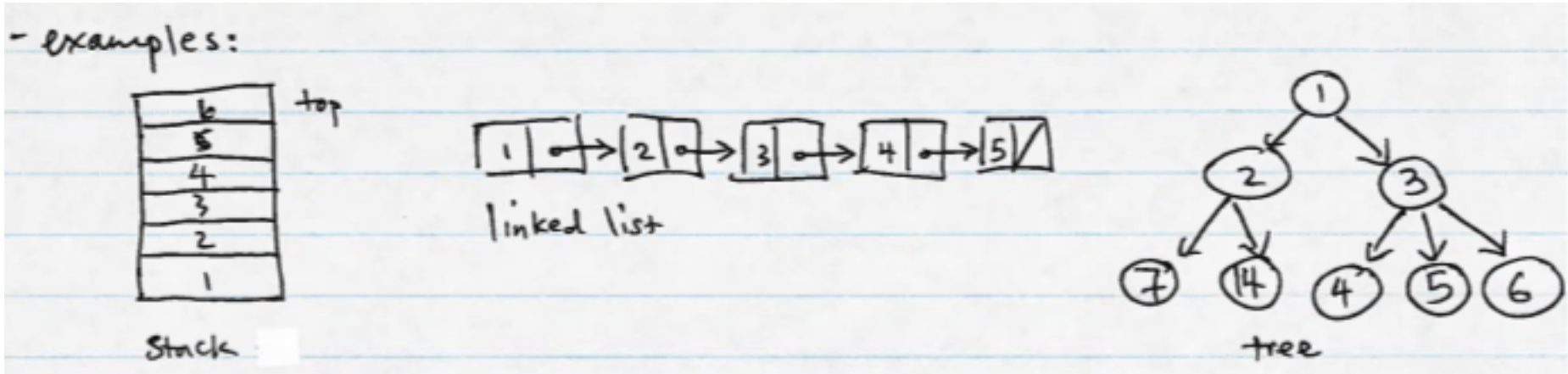
A2 feedback

- Solution online
- Statistics:
 - Average: 28/29
 - Max: 33/29
- Common mistakes:
 - Didn't initialize all attributes
 - Methods not called by another class need to be private
 - Forgetting to reset answers before playing again
 - Minor: didn't print instructions to user, didn't print back what user typed in as confirmation

Modeling Groups of Info

- Once upon a time, we talked about a sequence of objects
 - E.g., `String[]` , `Question[]`
- Notation: square brackets
- One way to organize information is to use **data structure**
 - An abstract representation of info organization only
 - Doesn't say how the structure is implemented or what programming language is used

Examples



- Java also provides a set of **collections** defined as classes that come with their own methods, such as:
 - ArrayList (see this in Lab, also Ch5)
 - Set
 - HashMap

Arrays

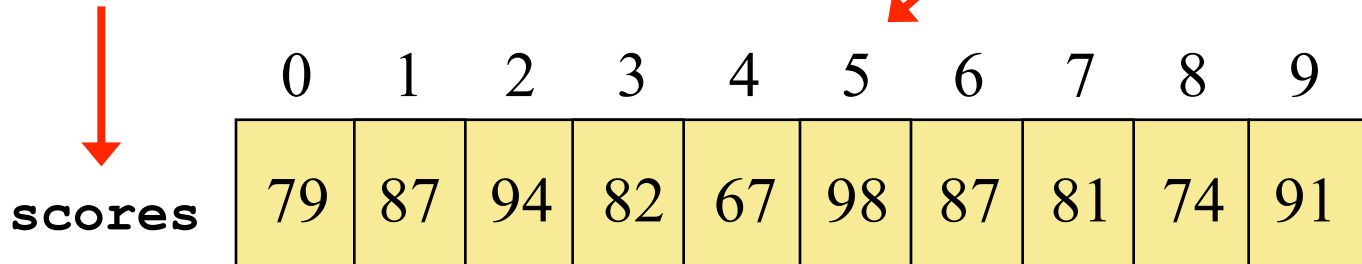
- A programming construct in Java to help us organize group of info
- Manages sequence of the same kind of info (can be primitive data types or objects)
- Examples:
 - `int[]` numbers;
 - `String[]` names;
 - `Player[]` gamers;

Representation

- A sequence of information
- Example: scores

The entire array
has a single name

Each value has a numeric *index*



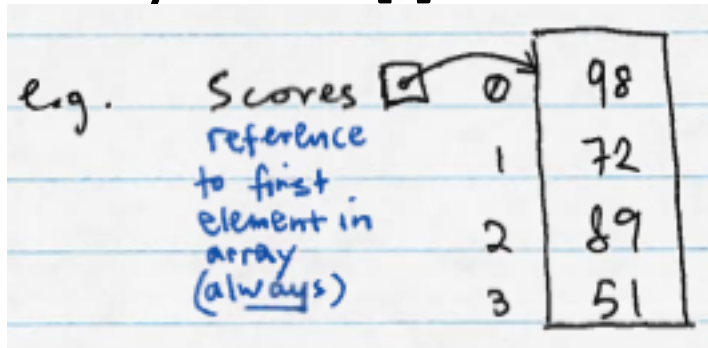
	0	1	2	3	4	5	6	7	8	9
scores	79	87	94	82	67	98	87	81	74	91

An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Accessing Individual Elements

- To access the i^{th} element in an array, write: `arrayName[i]` where $0 \leq i \leq N-1$



The name of the array is an **object reference** variable (i.e. pointer)

- # elements in scores?
- `scores[0]` is ?
- `scores[3]` is ?
- `scores[4]` is ?

Comparison to Memory

- Array representation is modeled after computer memory
 - Array index is like memory address
 - When you create an array of size N, you ask for a contiguous block of memory to be put away for use
 - Indices are a more convenient way to refer to specific locations than using hexadecimal addresses (0xA129)

How to Use Array Elements

- Can be assigned a value, printed, or used in calculation just like before:

```
scores[2] = 89;
```

How to Use Array Elements

- Can be assigned a value, printed, or used in calculation just like before:

```
scores[2] = 89;
```

```
int first = 0;
```

```
scores[first] = scores[first] + 2;
```

How to Use Array Elements

- Can be assigned a value, printed, or used in calculation just like before:

```
scores[2] = 89;
```

```
int first = 0;
```

```
scores[first] = scores[first] + 2;
```

```
double mean = (scores[0] + scores[1]) / 2;
```

```
System.out.println( "Best = " + scores[2] );
```

How to Use Array Elements

- Can be assigned a value, printed, or used in calculation just like before:

```
scores[2] = 89;
```

```
int first = 0;
```

```
scores[first] = scores[first] + 2;
```

```
double mean = (scores[0] + scores[1]) / 2;
```

```
System.out.println( "Best = " + scores[2] );
```

```
Random generator = new Random();
```

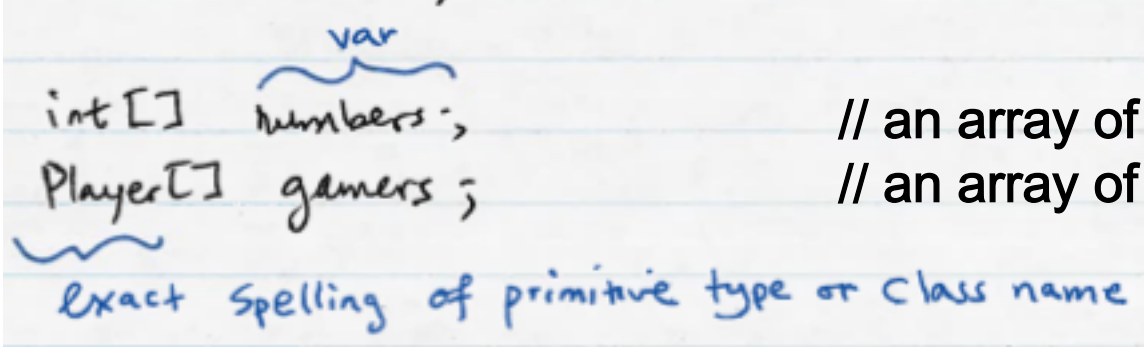
```
int pick = scores[ generator.nextInt(4) ];
```

Declaring Arrays

- (Partial) Template:

type[] varName;

- Examples:



int[] numbers; // an array of integers

Player[] gamers; // an array of Player objects

Exact Spelling of primitive type or class name

The image shows handwritten code on lined paper. The first line is 'int[] numbers;' with a blue bracket above 'numbers' labeled 'var'. The second line is 'Player[] gamers;' with a blue bracket above 'gamers'. To the right of each line is a comment: '// an array of integers' and '// an array of Player objects'. Below the code, a blue note says 'Exact Spelling of primitive type or class name'.

- Just like typical variable declaration, except we add []

Alternate Syntax

- When declaring arrays, there are two options:
 `type[] var;`
 `type var[];`
- Both are possible in Java
- First is more readable and consistent

Alternate Syntax

- When declaring arrays, there are two options:

`type[] var;`

~~`type var[];`~~

- Both are possible in Java
- First is more readable and consistent
- Should avoid second one

Declaring Arrays Part 2

- Must also tell Java how many elements are needed
 - Unlike primitive data types
 - Similar to objects
- Full template 1:
`type[] varName;`
`varName = new type[N];`
- Full template 2:
`type[] varName = new type[N];`

Examples

- Template 1: `type[] varName;`
 `varName = new type[N];`
- Examples:
 - `int[] numbers;`
 `numbers = new int[200];`
 `Player[] gamers;`
 `gamers = new Player[5];`
- Template 2: `type[] varName = new type[N];`
- Examples:
 - `int[] numbers = new int[200];`
 `Player[] gamers = new Player[5];`
- Not yet initialized!

If you use before initialize ...

- Example:

```
String[] verbs = new String[ 4 ];  
System.out.println( verbs[0] );
```

If you use before initialize ...

- Example:

```
String[] verbs = new String[ 4 ];
```

```
System.out.println( verbs[0] );
```

- This will immediately crash the program with a `NullPointerException`

Initializing Arrays

- Option 1:
- “Manually” give entire array values
- Template:
`type[] var = { e0, ... eN-1 };`
- Examples:
`int[] units = { 147, 93, 251, 79, 578 };`
`char[] grades = { 'A', 'B', 'C', 'D', 'F' };`
`Player[] gamers = { bob, ann, cam, eva };` Player objects
defined earlier
- Must declare and initialize at the same time

Initializing Arrays

- Option 2:
- “Manually” assign array values one at a time
- Template (after declaration):

`var[i] = ei;`

- Examples:

`units[0] = 147;`

`grades[1] = 'B';`

`gamers[3] = eva;`

**eva was defined as a Player
object earlier**

Initializing Arrays

- Option 3:
- Use a loop to assign values based on a pattern
 - Initialize all the elements to 0
 - Initialize the elements to multiples of the loop counter
- Examples:
 - `for(int count=0; count < scores.length; count++)
 scores[count] = 0;`
 - `for(int count=0; count < scores.length; count++)
 scores[count] = count*count;`

Exercises

- Write an array declaration to represent the ages of 100 children
- Write code that prints each value in an array of integers named `values`

Exercises

- Write an array declaration to represent the ages of 100 children

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

- Write code that prints each value in an array of integers named `values`

Exercises

- Write an array declaration to represent the ages of 100 children

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

- Write code that prints each value in an array of integers named `values`

```
for( int i=0; i<values.length; i++ )  
    System.out.println( values[i] );
```

Array Bounds

- Once an array is created, it has a fixed size
- When you access the i^{th} element of an array:
 - must remember array ranges from 0 to $N-1$
 - so index must be $0 \leq \text{index} \leq N-1$
- Otherwise, array index is **out of bounds**
 - Java throws an `ArrayIndexOutOfBoundsException` error when you try to run the program
 - In effect, program crashes
 - No syntax error!

Another Example

- What does this (partial) program do?

```
Scanner sysin = new Scanner( System.in );  
System.out.println( "Type in a word" );  
String word = sysin.nextLine();  
for( int ch=word.length()-1; ch>=0; ch-- )  
    System.out.print( word.charAt( ch ) );  
System.out.println();
```

- If the user types in “flower”, what will be displayed from the loop?
- Be careful: the length of a string is a method!

Class Activity

- Define a class called `MidtermStats`
- It has an attribute `scores` array
 - For simplicity, say test scores are integers
- It has a method called `enterScores ()`
 - We don't know how to read from a file yet
 - For now, just randomly generate the test scores
- It has a method called `calcAverage ()`
 - Goes through `scores` and returns average
- Test class:

```
MidtermStats mt2 = new MidtermStats( 153 );  
mt2.enterScores();  
System.out.println( "Avg = " + mt2.calcAverage() );
```

Arrays as Parameters

- Like variables (primitive type or objects), arrays can be passed as an input parameter to a method
- Example:

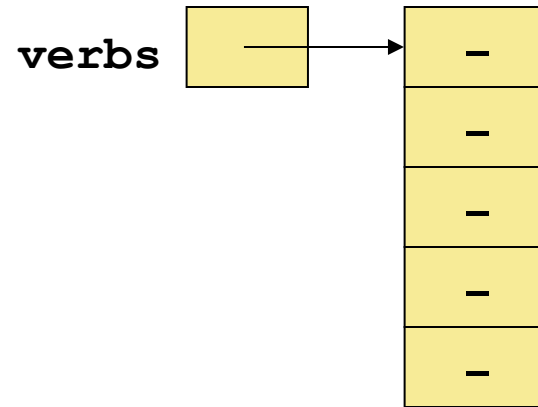
```
public double sumUp( int[] numbers ) { ... }  
...  
sumUp( scores );
```
- If you change an array value inside the method, the array will remember that change even after we exit the method
 - Pass by reference (as opposed to pass by value)

Array of Objects

- We showed some examples earlier
- Example:
`String[] verbs = new String[5];`
 - We've only allocated the space to hold 5 Strings
 - Doesn't create String objects themselves
- After declaration, an array of objects holds `null` references
- Each object must be initialized ("newed") separately

Representation

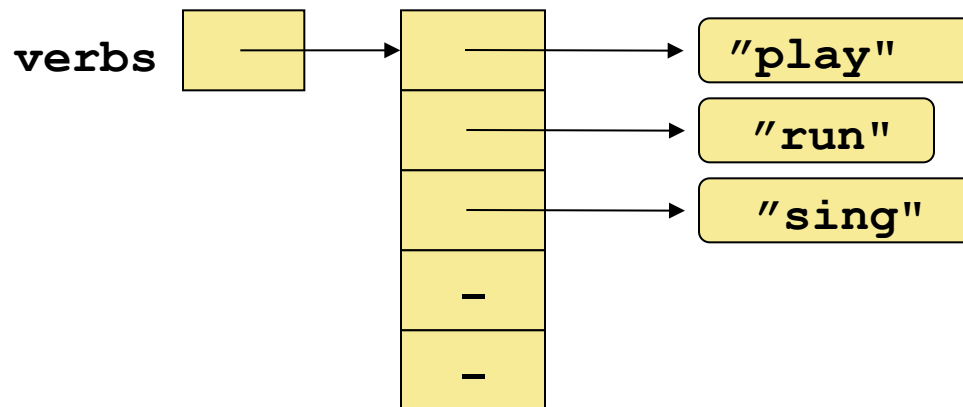
- After declaration:



- If you ran:
`System.out.println(verbs[0]);`
- You'd get a `NullPointerException` error
 - Syntax is okay, but program still crashes

After Initialization

- Suppose you have:
 `verbs[0] = "play";`
 `verbs[1] = "run";`
 `verbs[2] = "sing";`
- Your array will change to:



Alternative Initializations

- Can also use a loop like before
- Can also manually initialize all the elements at once

```
String[] verbs = { "play", "run", "sing",  
                  "sleep", "eat" };
```

- Use array as usual, e.g.:
 `verbs[0].equals(verb[1])`
 `verbs[2] = verbs[3].toLowerCase()`

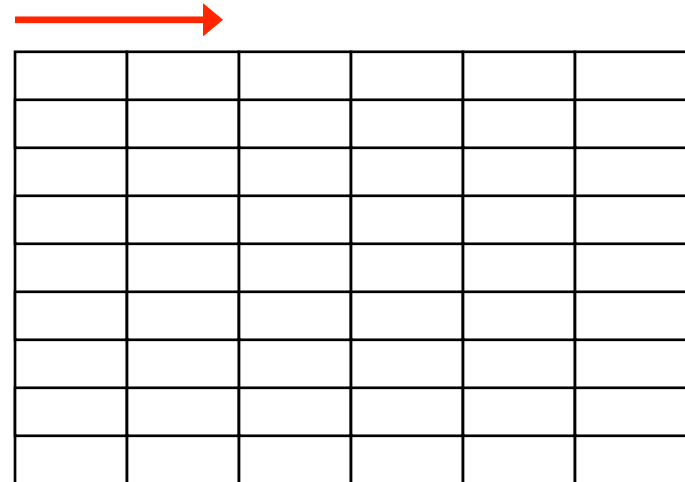
2D Arrays

- A **one-dimensional** array stores a sequence of elements
- A **two-dimensional** (2D) array stores a table of elements

one
dimension



two
dimensions



- Next class