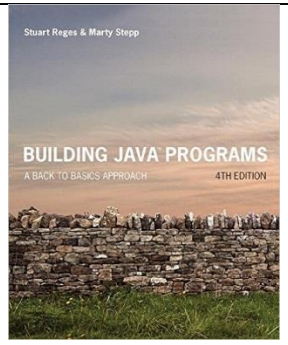
Java Arrays & Strings

Reading Assignment: Read Chapters 3.3 and 7.

Building Java Programs (Stuart Reges and Marty Stepp)



Overview

- What is an Array
- Declaring arrays
- Creating arrays
- Get Length and initializing arrays
- Array Literals
- Arrays cloning
- Arrays Equality Check
- Multidimensional arrays
- Vectors (**resizable array of objects)**
- Strings

What is an Array?

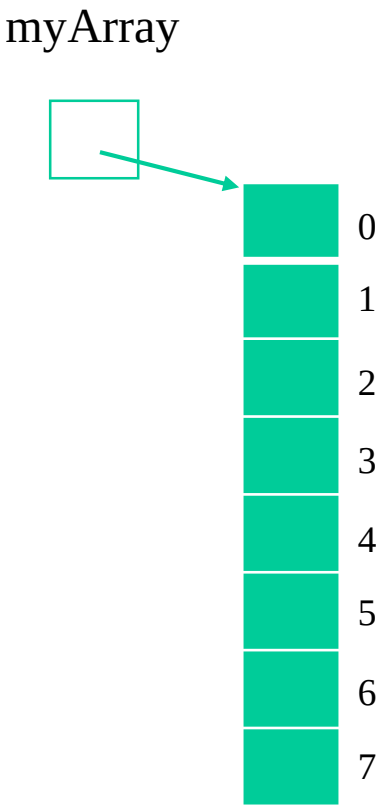
Array is a data structure that holds a *collection* of data of the same type.

Java treats arrays as *objects*. This means array variables are *reference* variables, not value variables.

Individual data items in arrays are called *elements*.

Elements are stored consecutively (contiguously) in memory.

Each element of an array is indexed. Indexing begins at 0.



Declaring Arrays

- **datatype[] arrayname1, arrayname2;**

  Example:

  int[] myArray1, myArray2;

- **datatype arrayname[];**

  Example:

  int myArray1[], x, myArray2[];

NOTE: *Declaring* an array does not *create* it! No memory is allocated for individual array elements. This requires a separate creation step.

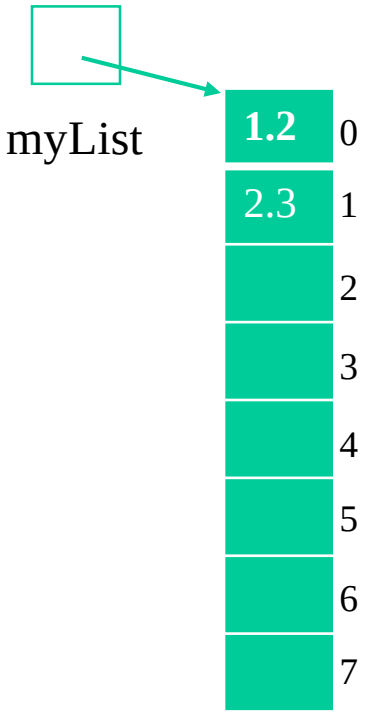
Creating Arrays

```
arrayName = new datatype[arraySize];
```

Integer value indicates the number of elements

Example:

```
myList = new double[8];
```



NOTE: the ***new*** keyword creates an object or array. The object or array is created in a location of memory called the **heap**. A reference (pointer) to the array is assigned to the variable.

Declaring and Creating in One Step

- datatype[] arrayname = new datatype[arraySize];

  double[] myList = new double[10];

- datatype arrayname[] = new datatype[arraySize];

  double myList[] = new double[10];

The Length of an Array

Once an array is created, its size is fixed. **It cannot be changed.** You can find its size using

arrayReferenceVar**.length**

For example,

```
X = myList.length;
```

Initializing Arrays

- Using a loop:

```
for (int i = 0; i < myList.length; i++)
    myList[i] = someValue;
```

- Declaring, creating, initializing in one step:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

Array literals

- You can create an array literal with the following syntax:

  ***type*[ ] { *value1*, *value2*, …, *valueN* }**

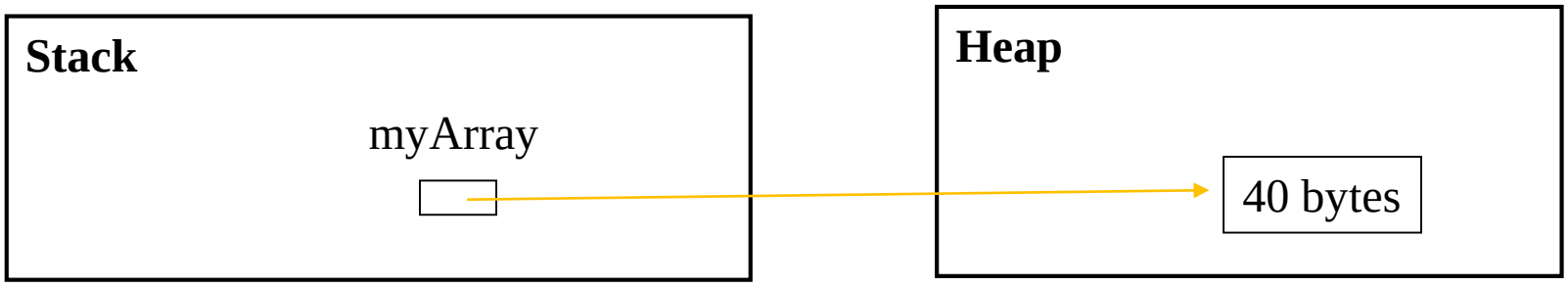
- Examples:

```
printArray( new int[] {2, 3, 5, 7, 11} );


int[ ] foo;
foo = new int[]{42, 83};
```

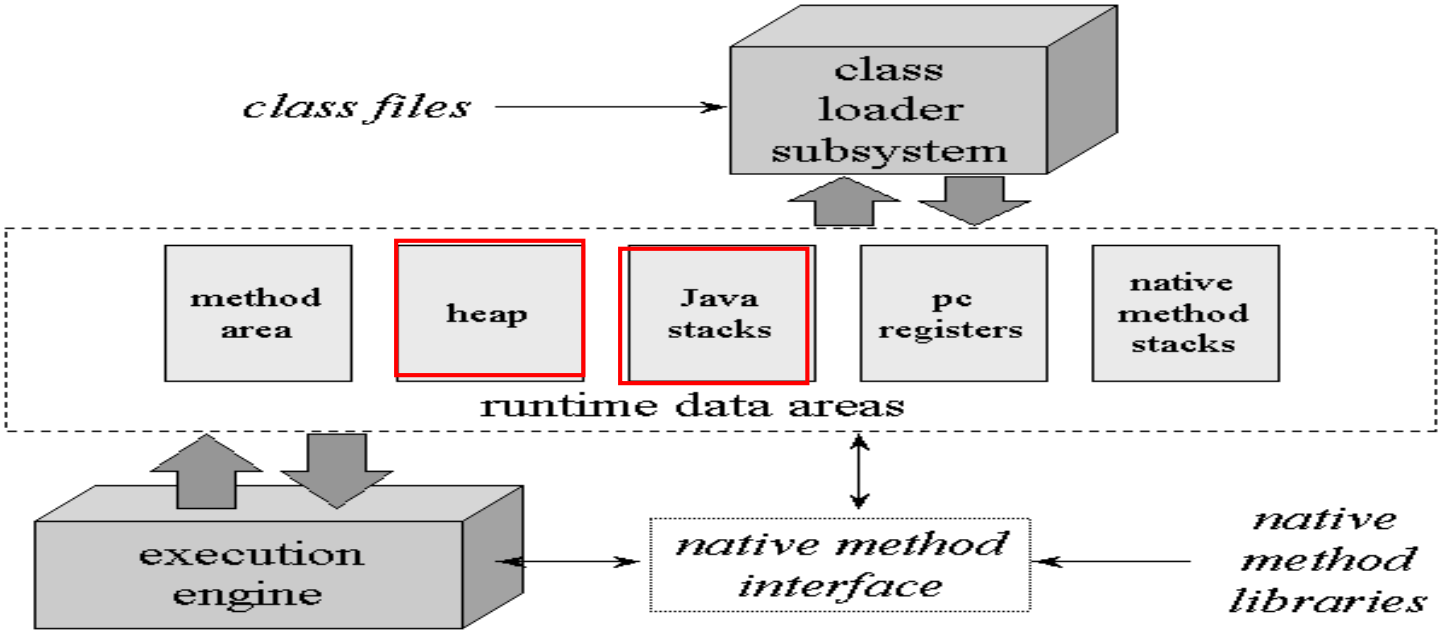
Example

```
int main() {
  int [] myArray = new int[10];
  ...
```



How variables are kept in memory

- Two different locations of memory in JVM
  - Stack – contains local variables and parameters of methods
  - Heap – contains objects and arrays

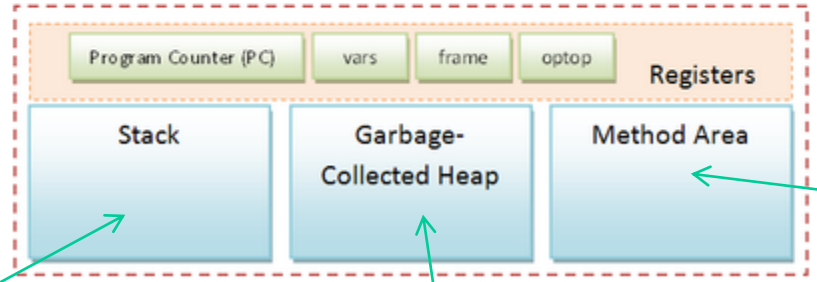


Reminder - JVM

Java Virtual Machine (JVM)

Java Virtual Machine **executes** the program and generates output. To execute any code, JVM utilizes different components.

JVM is divided into several components like the **stack, the garbage-collected heap, the registers and the method area.**



Stack in JVM stores various method **arguments** as well as the **local variables** of any method.

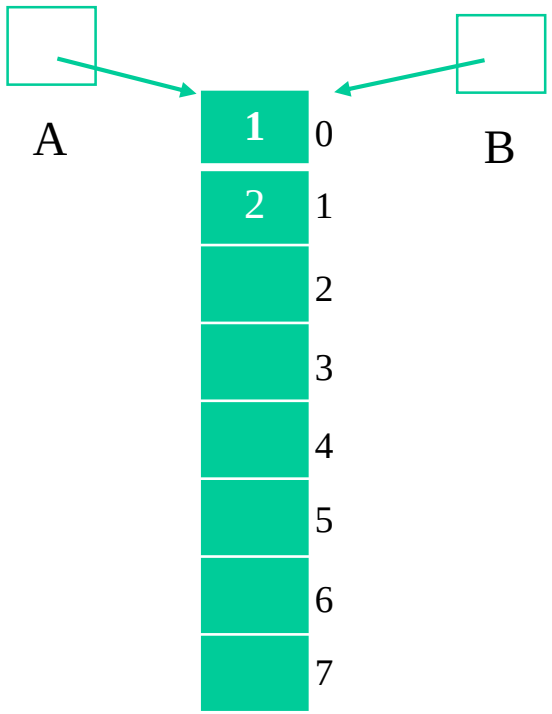
The Garbage-collected Heap is where the **objects** in Java programs are **stored**. Whenever we allocate an object using new operator, the heap comes into picture and memory is allocated from there.

This is the area where byte codes reside. The program counter points to some byte in the method area. It always keeps tracks of the current instruction which is being executed (interpreted).

Using the assignment operator on arrays

Consider:

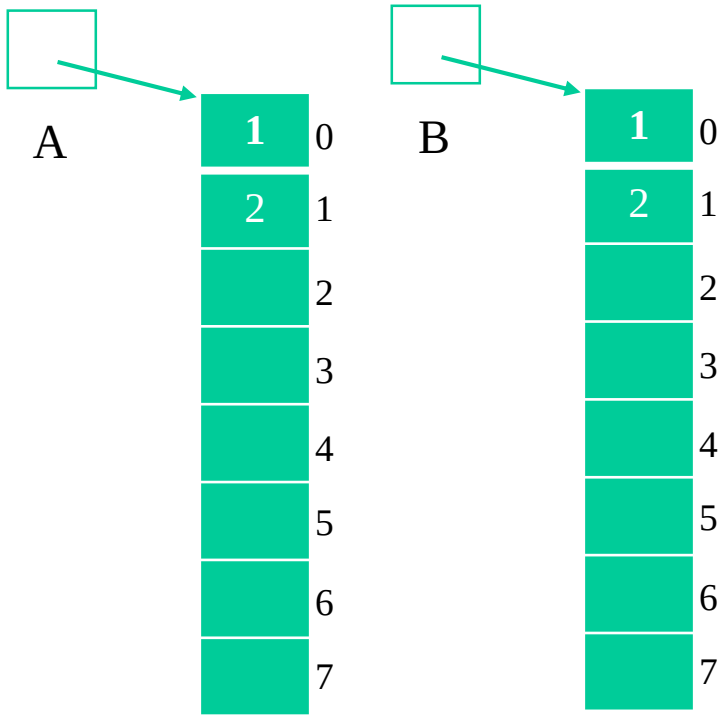
```
int A[] = {1, 2, 3, …};
int B[];
B = A;
```



Array Cloning

To actually create another array with its own values, Java provides the **clone()** method.

```
int[] A = {1,2,3,…};
int[] B;
B = A.clone();
```



Write your own array cloning method

A method returns an array

array parameter

```
int[] clone(int[] original) {
   int [] result = new int[original.length];
   for (int i = 0; i < original.length; i++)
      result[i] = original[i];
   return result;
}
```

Write your own array cloning method

```
// Use System.arraycopy(source_a, pos, dest_a, pos, length);
int[] clone(int[] original) {
    int [] result = new int[original.length];
    System.arraycopy(original, 0, result, 0, original.length);
    return result;
}
```

Array Equality

- Java has an **`equals()`** method for classes
  `if (C1.equals(C2))` …
- If **`A1`** and **`A2`** are arrays, the **`equals()`** method just compares the references.
- Similar the equal operator == only compares references.
- If (A1==A2)
- To compare contents of two arrays, need to use a loop.

Declaring Multidimensional Arrays

- datatype[][] arrayname;

  Example:

  int[][] myTable;

- datatype arrayname[][];

  Example:

  int myTable[][];

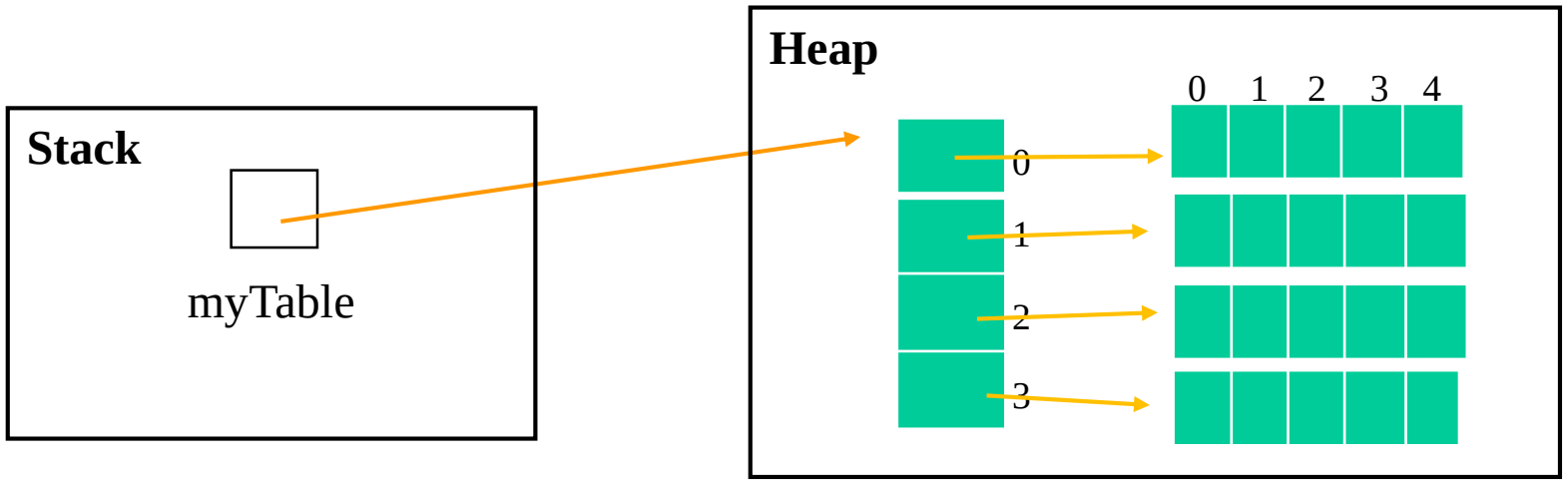
Creating Multidimensional Arrays

arrayName = new datatype[rowSize][columnSize];

Integer value indicates the number of rows

Integer value indicates the number of columns

Example: myTable = new double[4][5];



A 2-D array is an array of arrays

Declaring and Creating in One Step

- datatype[][] arrayname = new datatype[rowSize][columnsize];

  double[][] myList = new double[10][3];

- datatype arrayname[][] = new datatype[rowSize][columnSize];

  double myList[][] = new double[10][3];

Initializing 2-Dimensional Arrays

```
// myTable.length => number of rows
// myTable[r].length => number of columns for row r
for (int r =0; r<myTable.length; r++) {
    for (int c = 0; c<myTable[r].length; c++){
          myTable[r][c]=someValue;
     }
}
```

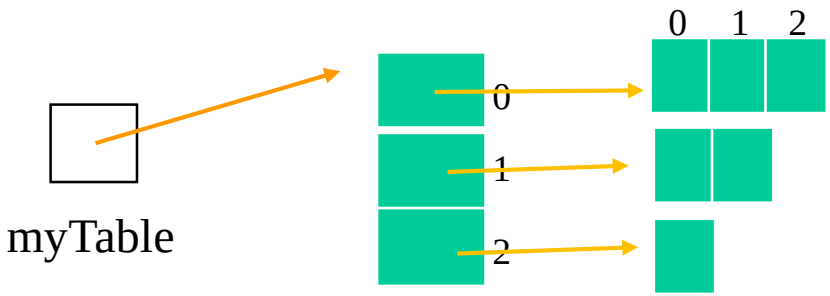
Or

```
double[][] myList = {{1.9, 2.9},{3.4, 3.5}};
```

Ragged Arrays

In a 2-d array rows can vary in size

Int [] [] myTable = {{1, 2, 3},{1, 2},{1}};



Vectors

- **General purpose resizable array of objects.**
  - User can use integer index to access items in a vector, like an array.
  - A vector can expand or shrink as needed when add or delete items.
- Import the Vector class before use it.
  - import java.util.Vector; or
  - import java.util.*;
- Declaring Vectors
  - Vector V = new Vector(); or
  - Vector V = new Vector(capacity); or
  - Vector V = new Vector(capacity,increment);

Adding, removing, and accessing vector elements

void **add**(int index, Object element)/boolean **add**(Object element)

int **capacity**()

boolean **contains**(Object elem)

void **copyInto**(Object[] anArray)/Object[]**toArray**()

Object **elementAt**(int index)

Object **get**(int index)

int **indexOf**(Object elem)/int **indexOf**(Object elem, int index)

Void **insertElementAt**(Object obj, int index)

Boolean **isEmpty**()

Object **remove**(int index)/Boolean **remove**(Object o)/void **removeAllElements**()

Boolean **removeElement**(Object obj)/Void **removeElementAt**(int index)

void **setSize**(int newSize)

Int **size**()

See: https://docs.oracle.com/javase/7/docs/api/java/util/Vector.html

Overview

- What is an Array
- Declaring arrays
- Creating arrays
- Get Length and initializing arrays
- Array Literals
- Arrays cloning
- Arrays Equality Check
- Multidimensional arrays
- Vectors (**resizable array of objects)**
- Strings

Strings

- Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects.

- Source: http://www.tutorialspoint.com/java/java_strings.htm

The String Class

- Although we haven't yet discussed classes and object, we will discuss the String class.
- String objects are handled specially by the compiler.
  - String is the only class which has "implicit" instantiation.
- The String class is defined in the java.lang package.
- Strings are immutable. The value of a String object can never be changed.
  - For mutable Strings, use the StringBuffer class.

Creating String Objects

- Normally, objects in Java are created with the *new* keyword.

```
String name;
  name = new String("Craig");
```

- However, String objects can be created "implicitly":

```
String name;
  name = "Craig";
```

- Strings can also be created using the + operator. The + operator, when applied to Strings means concatenation.

```
int age = 21;
String message = "Craig wishes he was " + age + " years old";
```

Commonly used String methods

- The String class has many methods. The most commonly used are:
  - length() - returns the number of characters in the String
  - charAt() - returns the character at the specified index
  - equals() - returns true if two strings have equal contents
  - compareTo() -returns 0 if equal, -# if one String is "less than" the other, +# if one String is "greater than" the the other.
  - indexOf() - returns the index of specified String or character
  - substring() -returns a portion of the String's text
  - toUpperCase(), toLowerCase() - converts the String to upper or lower case characters

String Examples

```
String name = "Craig";
String name2 = "Craig";

if (name.equals(name2))
     System.out.println("The names are the same");
```

```
String name = “John Doe";
int lastNameIndex = name.indexOf(“Doe");
```

```
String grade = "B+";
double gpa = 0.0;

if (grade.charAt(0) == 'B')
     gpa = 3.0;

if (grade.charAt(1) == '+')
     gpa = gpa + 0.3;
```

Testing Strings for Equality

- Important note: The == operator cannot be used to test String objects for equality
  - Variables of type String are references to objects (ie. memory addresses)
  - Comparing two String objects using == actually compares their memory addresses. Two separate String objects may contain the equivalent text, but reside at different memory locations.
- Use the equals method to test for equality.

The StringBuffer Class

- StringBuffer objects are similar to String objects
  - Strings are immutable
  - StringBuffers are mutable
- The StringBuffer class defines methods for modifying the String value
  - insert()
  - append()
  - setLength()
- To clear a StringBuffer, set it's length to 0

```
StringBuffer nameBuffer = new StringBuffer("Joe");
[...]
  nameBuffer.setLength(0);  // clear StringBuffer
```

StringBuffer Example

```
StringBuffer sql = new StringBuffer();

sql.setLength(0);
sql.append("Select * from Employee");
sql.append(" where Employee_ID = " + employeeId);
sql.append(" and Employee_name = '" + employeeName + "'");
```

Attendance Quiz

Problem 1:

```
int[] array = { 1, 4, 3, 6, 8, 2, 5};
int what = array[0];

// scan the array
for ( int index=0; index < array.length; index++ )
{
  if ( array[ index ] < what )
    what = array[ index ];
}
System.out.println( what );
```

What does the fragment write to the monitor?

- **a.** 1
- **b.** 5
- **c.** 1 4 3 6 8 2 5
- **d.** 8

Problem 2:

```
int[] array = { 1, 4, 3, 6 };
int what    = 0;

// scan the array
for ( int index=0; index < array.length; index++ )
{
    what = what + array[ index ] ;
}
System.out.println( what );
```

What does the fragment write to the monitor?

- **a.** 14
- **b.** 1
- **c.** 6
- **d.** 1 4 3 6