COSC 123
Computer Creativity

Java Lists and Arrays

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Objectives

1) Create and use arrays of base types and objects.

2) Create and use `ArrayList`.

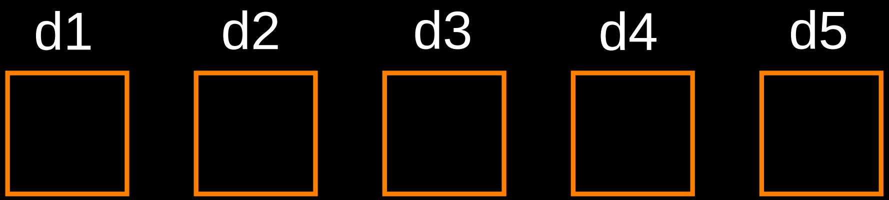
3) Understand the role of generic types to catch and prevent errors.

Arrays Overview

Suppose you need many variables in your program.

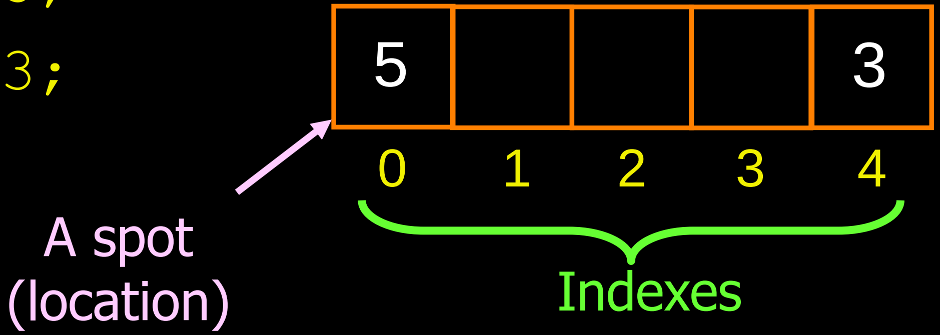
You could either create a separate name for each variable:

- `double d1, d2, d3, d4, d5;`



Or you could create an array that has multiple spots (indexes):

- `double[] myArray = new double[5];`
- `myArray[0] = 5;`
- `myArray[4] = 3;`



Arrays

An ***array*** is a collection of data items of the same type.

An array reference is denoted using the open and close square brackets "`[]`" during declaration.

⇨ You can have an array of any data type including the base types (`int`, `double`, `String`) and object-types (`BankAccount`).

- Examples:

```
int[] myArray;
String[] strings;
BankAccount[] accounts;
```

Similar to an object, when you declare an array you are creating a ***reference*** to an array. Until you actually create the space for the array using **`new`**, no array exists in memory.

- `String[] strings = new String[10];`

Array Indexing

When creating an array using `new`, the number in square brackets is the number of elements in the array:

- `double[] values = new double[20]; // 20 items`

Note the first element of the array has index 0 instead of 1.

- In the previous example, the first index is 0 and the last is 19.

When an array is created, its values are initialized to defaults:

- 0 for numbers, `false` for boolean, `null` for object references

To access or set a value in an array, use its subscript:

- `values[0] = 10;` // Sets first element to 10
- `values[19] = values[0];` // Sets last element same as 1st

Array Details

Java performs automatic ***bound checking*** whenever an array element is referenced.

- If the index is in the valid range, the reference is carried out.
- If the index is not valid, an exception, `ArrayIndexOutOfBoundsException`, is thrown.

To get the length of an array in your program:

- `int[] numbers = new int[25];`
- `int size = numbers.length;` // Returns 25

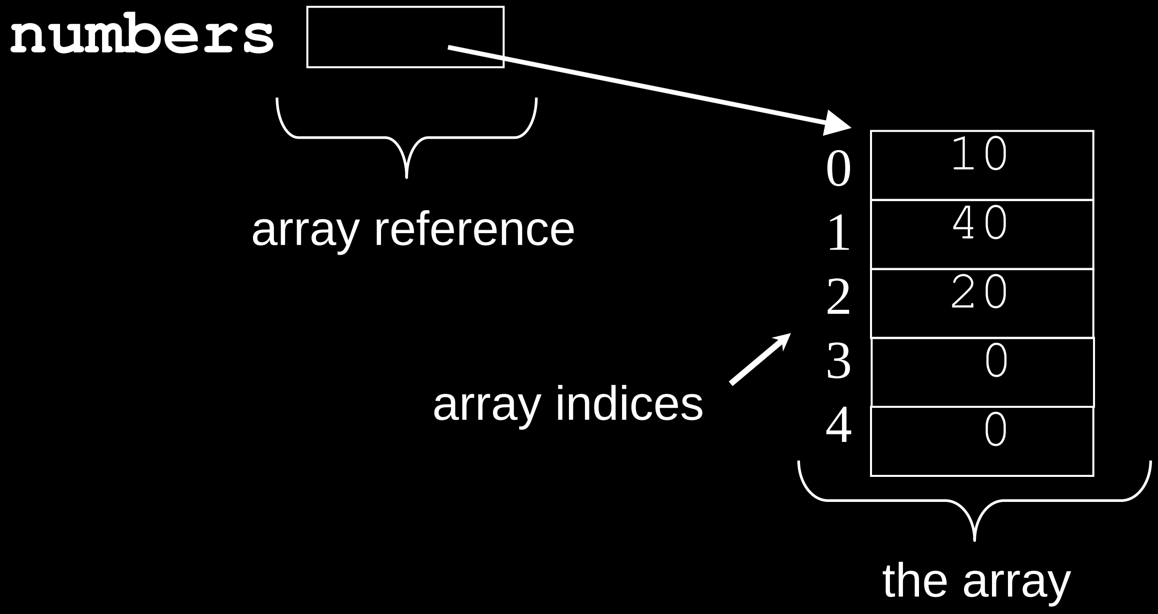
You can initialize an array with values when you first declare it:

- `int[] primes = {2, 3, 5, 7, 11};`
  - ⇨ `new` is not used with an initializer list. Initializers can only be used during declaration.

Arrays in Memory Diagram Base Types

An array of base types stores the values in the slots in the array. Example:

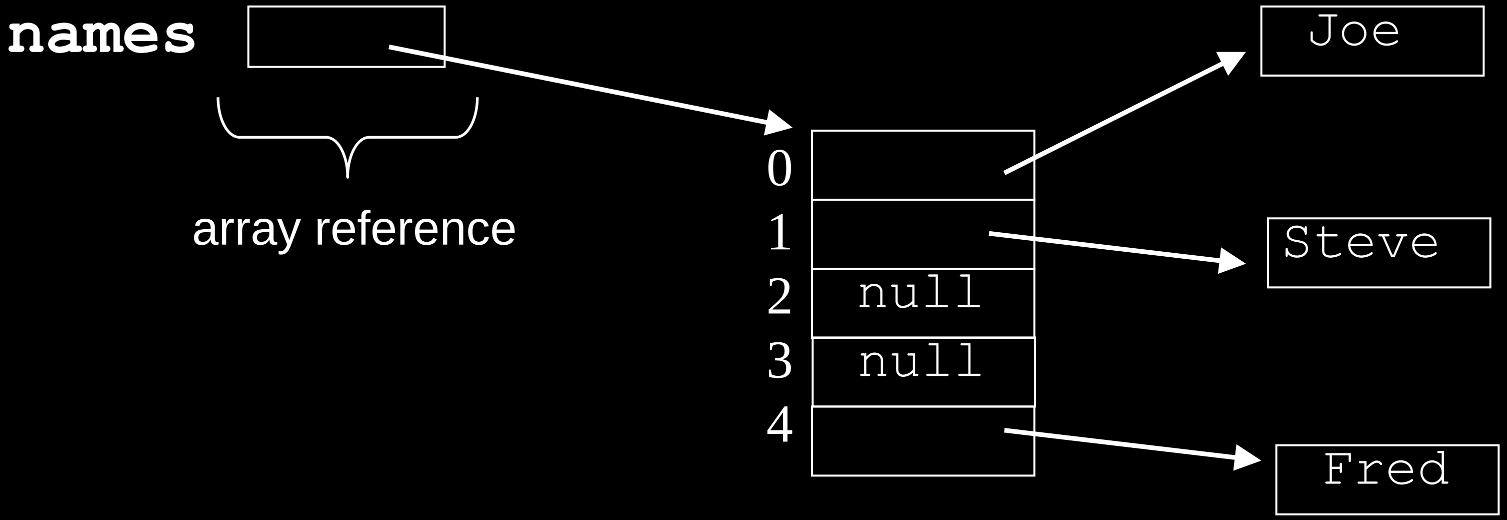
```
int[] numbers = new int[5];
numbers[0] = 10;
numbers[1] = 40;
numbers[2] = numbers[0]+10;
```



Arrays of Objects

An object array is an array of object references. Example:

```
String[] names = new String[5];
names[0] = "Joe";
names[1] = "Steve";
names[4] = "Fred";
```



When allocating an object array, Java does not also allocate space for each object in the array. Each object reference is initialized to null.
You must create a new object for each object reference.

Arrays as Parameters and References

An array can be passed as a parameter to a method and returned from a method.

- The values of the array can be changed but not the array reference itself. This is similar to how objects work.

Since an array is just a reference, it is possible to change which array a reference points to using assignment:

- `int[] array1 = new int[10];`
- `int[] array2 = new int[20];`
- `array2 = array1;` // array2 now references array1

Practice Questions

1) Create an `int` array with name `myArray` with 20 elements.

2) Set the value of the 1st element to 10.

3) Set the value of the last element to 1.

4) Create an array that has 10 elements. Put the numbers from 1 to 10 in the array.

5) How do you know how many elements are in an array?

Arrays

Question: What is the size of this array?

```
int[] myArray = new int[10];
```

A) error

B) 10

C) 9

D) 11

Arrays

Question: What are the contents of this array?

```
int[] myArray = new int[4];
myArray[3] = 1;
myArray[2] = 2;
myArray[1] = 3;
myArray[0] = 4;
```

A) error

B) 0, 1, 2, 3

C) 1, 2, 3, 4

D) 4, 3, 2, 1

Java Collections

A ***collection*** is an object that serves as a repository for other objects. A collection provides methods to add, remove, and manage the elements it contains.

The underlying data structure used to implement the collection is independent of the operations the collection provides.

Java Collections API classes defines collection interfaces such as Set, List, SortedSet, Queue, and BlockingQueue.

List collection has two linear data structure implementations:

- **ArrayList** - resizable-array implementation of the List interface.
- **LinkedList** - linked list implementation of the List interface.

ArrayLists

An `ArrayList` implements a *resizable* array of objects.

⇨ Base types such as `int` are not objects. Use wrapper class `Integer`.

Create an `ArrayList` by:

```
ArrayList names = new ArrayList(); // Size 10 (default)
ArrayList accounts = new ArrayList(5); // Size of 5
```

Add element to an `ArrayList` by:

```
names.add("Joe");           // Add to end of list
names.add(2,"Steve");       // Add at index 2 and shift up
```

Remove element from an `ArrayList` by:

```
names.remove(2);            // Remove index 2 and shift down
```

ArrayLists (2)

Get number of items in list by:

```
 int count = names.size();
```

Get element at an index from an `ArrayList` by:

```
String n = names.get(2);   // Get item at index 2
```

Set element at an index in an `ArrayList` by:

```
names.set(2,"Fred");       // Put Fred at index 2
```

Traversing an ArrayList

A simple way to traverse an `ArrayList` is using a for loop:

```
for (int i=0; i < names.size(); i++)
{  String s = (String) names.get(i);
   System.out.println(s);
}
```

Traversing an ArrayList Iterators

All collections also have iterators which are special classes designed to allow you to traverse through the collection.

Using an iterator with an `ArrayList`:

```
Iterator it = names.iterator();
while (it.hasNext())
{   String s = (String) it.next();
    System.out.println(s);
}
```

Generic Types

Collections store any type of object as all objects are a subclass of `Object`. It is better to precisely specify what objects are in a collection so that the compiler can check for errors.

All collections support ***generic*** (or parameterized) ***types*** to indicate what type is stored in the collection.

Examples:

```
// ArrayList can ONLY store strings

ArrayList<String> myNames = new ArrayList<String>(5);

// This ArrayList can only store BankAccount objects
ArrayList<BankAccount> accounts
                  = new ArrayList<BankAccount>();
```

ArrayList Example

```
import java.util.ArrayList;

public class TestArrayList
{  public static void main(String[] args)
   {  ArrayList a = new ArrayList();
      BankAccount b1, b = new BankAccount(100);
      SavingsAccount s1, s = new SavingsAccount(5,50);

      a.add(b);   // Add bank account b to list
      a.add(0,s); // Add s to front of list
      b1 = (BankAccount) a.get(1);
      s1 = (SavingsAccount) a.get(0);
      System.out.println(b1.getBalance());
      System.out.println(s1.getBalance());

      a.remove(0); // Remove s from list
      System.out.println(a.size()); // Prints 1
   }
}
```

ArrayList

Question: What is the value of `st`?

```
ArrayList a = new ArrayList();
a.add("Fred");
a.add(0,"Joe");
a.add("Steve");
a.remove(1);
String st = (String) a.get(1);
```

A) Fred

B) Joe

C) Steve

D) error

Practice Questions

1) Write a method **`reverse`** that returns a new array that contains the reverse sequence of numbers.

- Example:
  - ⇨ 1 4 9 19 9 7 4 9 11 becomes 11 9 4 7 9 19 9 4 1

2) Write a method that reads in strings using Scanner and stores them in an `ArrayList` until "STOP" is entered. Print out the list after you finish reading.

Conclusion

Arrays are a data structure for storing multiple items using the same name.

- An array has a fixed size and is indexed from 0 to size-1.
- An array can store both base types or object references.

A collection is an object that stores other objects and provides methods for adding, removing, and retrieving objects.

An `ArrayList` is a linear collection.

- `ArrayList` has methods for adding, removing, getting, and setting values.
- `ArrayList` can be traversed using a loop or an iterator.
- A generic type ensures the collection only stores the proper objects.

Objectives

Java skills:

- Creating an array
- Array indexing and bounds checking
- Arrays of base types and objects
- Arrays as parameters
- Copying arrays and `System.arraycopy`
- Two-dimensional arrays
- `ArrayList` – create, add, remove, get, set, traversing
- Iterators