Programming in Java

4. Arrays and Collections

Introduction

- Arrays and Strings are special Java classes
  - Special in the sense that the allow for a syntax that is more like regular programming languages
  - For example, we have been treating Strings like primitive variables
  - Java provides that syntax as “syntactic sugar” to make it easier to code in a natural stype

Arrays

- Arrays is Java look just like arrays in other programming languages
  - An array name with an indexed set of entries like a[0] etc.
  - We cannot reference an array entry that is out of range
  - Arrays know how long they are

```
public class Runner {

    public static void main(String[] args) {
        // defining an array by providing initial values
        int [] arr = {11,11,12,13};
        System.out.println("arr[] is a" + arr + "with length " + arr.length);
        for (int index = 0 ; index < arr.length; index++) {
            System.out.println("Entry " + index + " is " + arr[index]);
        }
    }
}
```

```
arr[] is a[I@251a69d7 with length 4
Entry 0 is 11
Entry 1 is 11
Entry 2 is 12
Entry 3 is 13
```

Arrays

- Arrays are allocated heap objects

```
public static void main(String[] args) {
    // define an array object that will reference a block of memory containing
    // 4 ints
    int [] arr = null;
    // allocate the memory
    arr = new int[4];
    // initialize
    arr[0] = 10;
    arr[1] = 11;
    arr[2] = 12;
    arr[3] = 13;
    System.out.println("arr[] is a" + arr + " with length " + arr.length);

    for (int index = 0 ; index < arr.length; index++) {
        System.out.println("Entry " + index + " is " + arr[index]);
    }
}
```

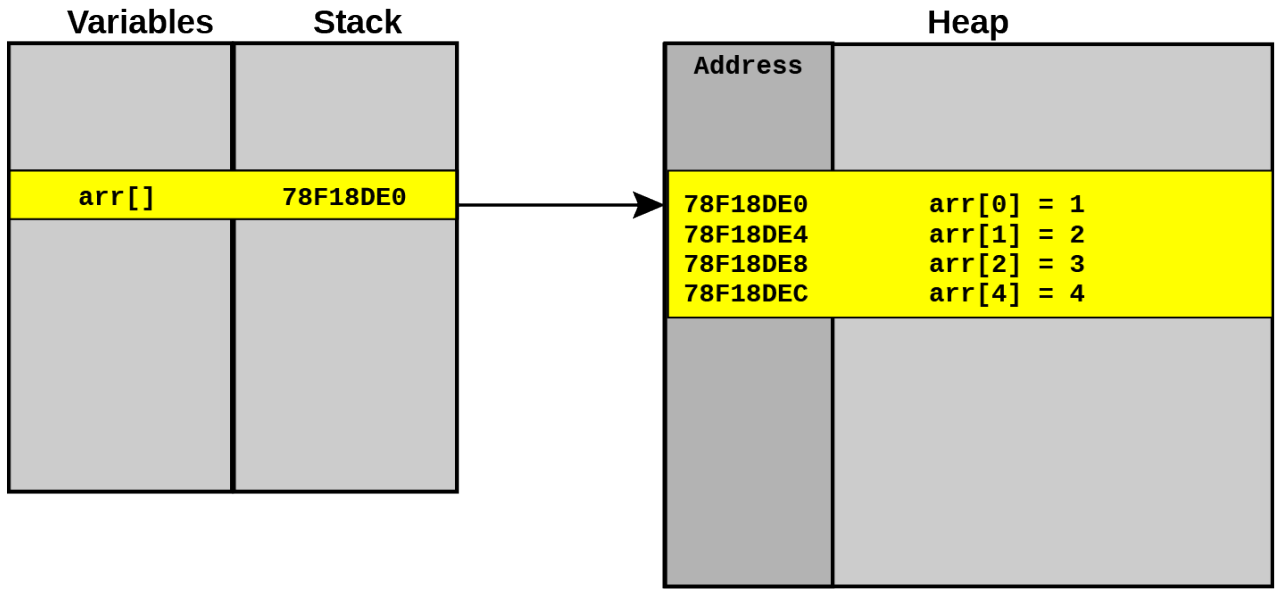
```
arr[] is a[I@251a69d7 with length 4
Entry 0 is 10
Entry 1 is 11
Entry 2 is 12
Entry 3 is 13
```

What are Arrays

- An array is a contiguous block of memory on the heap
  - The length of an array is fixed – they can't grow or shrink
- All of the elements in an array must be the same type
  - This means they are all the same size
  - This allow for very fast look ups
  - We don't have to loop through an array to find an element
- If we want to find element at index 3 in an array of ints
  - If the array starts at memory location '4577' and int is 4 bytes wide
  - Then the memory location of the element we want is '4577 +(3 + sizeOf(int))'
  - This allows for constant access time to any element no matter how big the array is

Arrays in Memory

```
int[] arr = {1,2,3,4}
```



Copying Arrays

- An array is an object
- Assigning an array variable to another array variable does not copy the array
  - It copies the address of the array – this is called a shallow copy
  - We have to copy the array element by element – this is called a deep copy
- Two arrays are “equal” the point to the same memory location
- Two arrays are “equivalent” if corresponding elements are equal
- There are utility methods in the collections class that can make the task easier

Array Operations

- In loops earlier, we had to provide an index or iterator to keep track of the iteration
  - Arrays have an internal iterator, specifically the index of the elements
  - We can use a simpler version of the for loop
- The java.util.Arrays library has a number of functions that can be used to copy, sort and perform other operations on arrays.

Arrays of Objects

- We can also have arrays of object
  - In this case, the elements of the arrays are the memory locations of the object
  - This meets the constant size requirement since all address are the same width

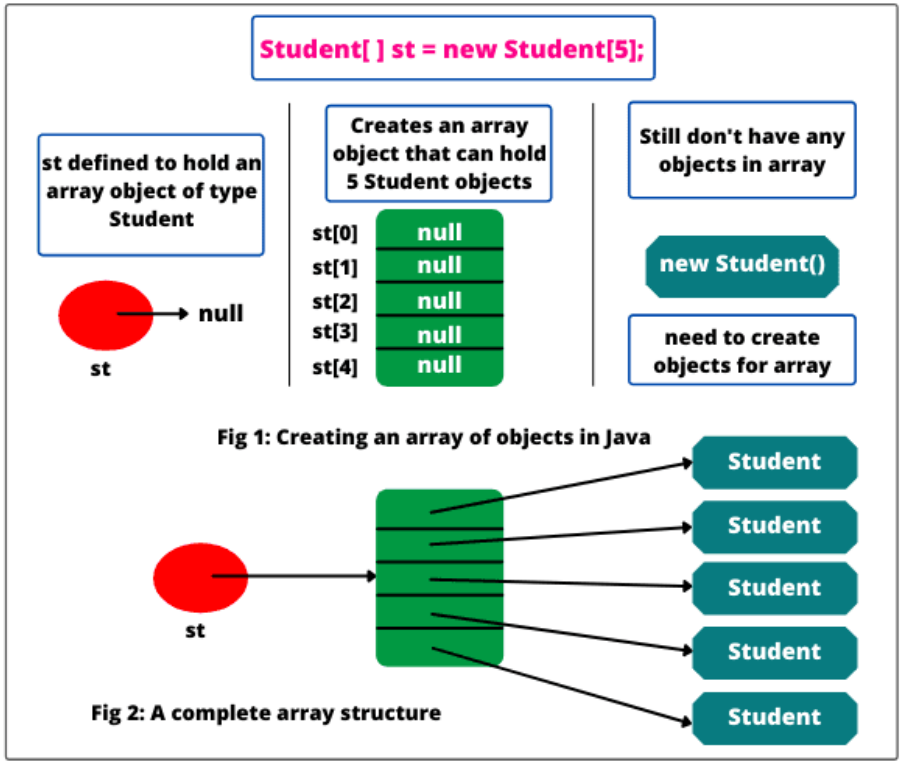


Fig 1: Creating an array of objects in Java

Fig 2: A complete array structure

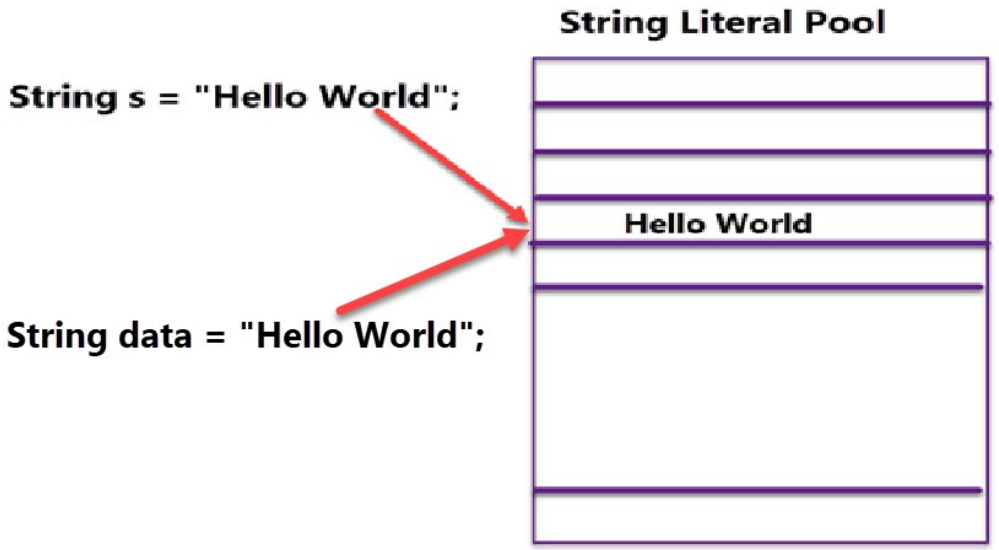

Demo
Arrays

Lab 4-1

Arrays

Strings

- Strings are objects just like other objects
- Java allows syntactic sugar so that we can write them as if they were primitive types
- Strings are immutable
  - Once created, the value of a string can not be changed
  - This is because there in only one copy of String literal, like “Hello World” that is shared by all String objects that have that literal as a value
  - These unique literals are said to be interned in a special constant area called the string pool



String Builder

- A far more efficient way to build a string is to convert it to an array of chars
  - A StringBuilder is a buffer that allows us work with a String in this way.


Demo
Working with Strings

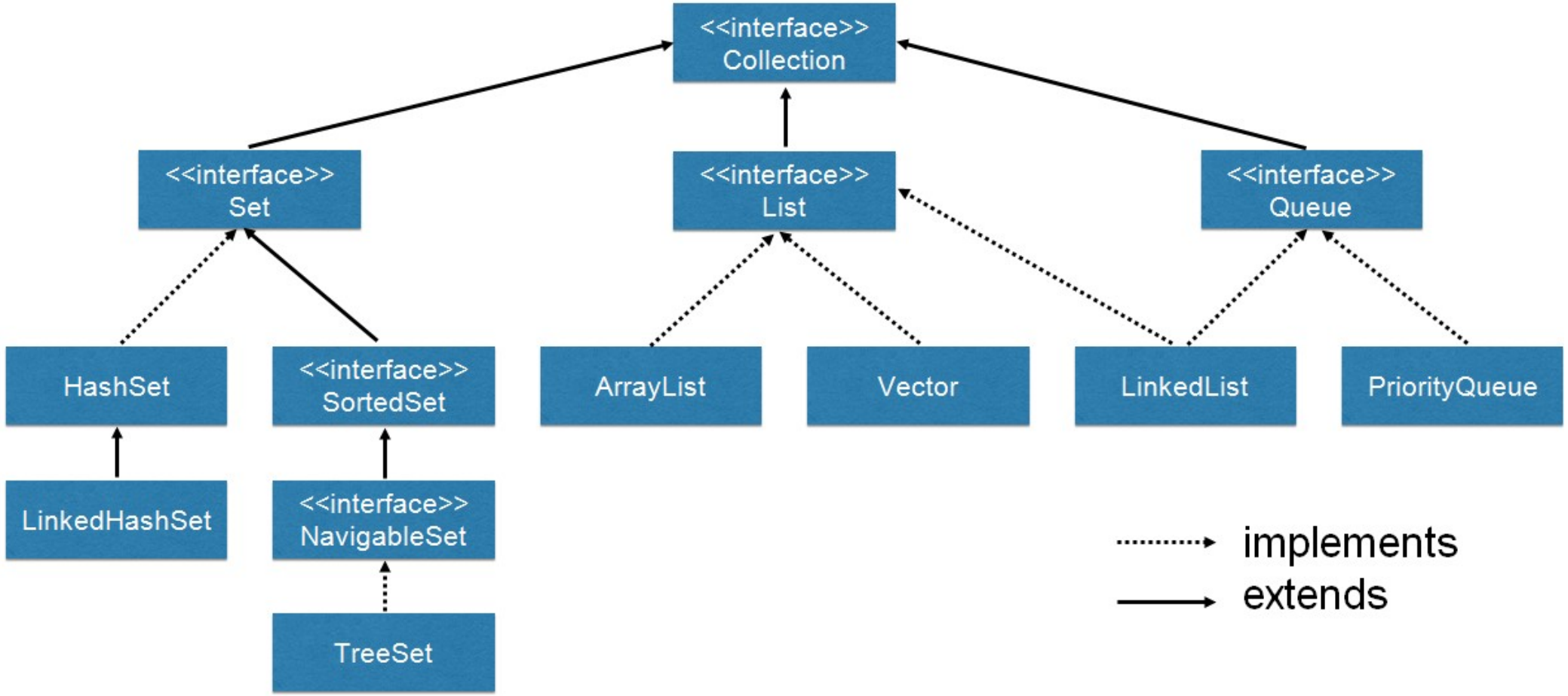
Java Collections

- Unified architecture for representing and manipulating collection
  - Library of standard computer science data structures and algorithms
  - Inspired by the C++ Standard Template Library (STL)
- Structures and algorithms are generic
  - How we add, read, modify and delete elements depends only on the type of structure not on the elements in the structure
  - A linked list works the same way for integers as for Customer objects
  - Sorting, reversing and other operations also don’t depend on what we are sorting or reversing
  - However, there are a restriction that we can only sort structures where the elements can be compared to each other
    - We can’t sort people for example, unless we define a way to compare them: by age or by weight for example

Why Use Collections

- Avoids us writing the same boilerplate code over and over again
- The library implementations are usually more efficient that a hand coded solution
- Lets us work with data structures through interfaces without having to know the details or anything about the underlying implementation
- Since we work through interfaces, we can easily switch implementations without having to rewrite code
  - Each implementation will have different performance characteristics
- The collections framework consists of
  - **Interfaces**: abstract data types that represent collections
  - **Implementations**: these are the concrete classes that implement the interfaces
  - **Algorithms**: methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces

Interfaces


<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
<<interface>>
Queue
HashSet
<<interface>>
SortedSet
ArrayList
Vector
LinkedList
PriorityQueue
LinkedHashSet
<<interface>>
NavigableSet
TreeSet
implements
extends

Collection Interfaces

- **Collection**: the abstract root of the collection hierarchy
  - A collection represents a group of objects known as its elements
  - The Collection interface is the least common denominator that all collections implement and is used to pass collections around and to manipulate them when maximum generality is desired
  - Some types of collections allow duplicate elements, and others do not. Some are ordered and others are unordered
- **Set**: a collection that cannot contain duplicate elements.
- **List**: an ordered collection that can contain duplicate elements with precise control over where the position of the elements in the list
- **Queue** and **Deque**: specialized lists that provides additional insertion, extraction, and inspection operations often use a FIFO or LIFO ordering
- **Map**: an associative array of keys and values.

Collection Implementations

- An “implementation” is a Java class that implements one of the Collections interfaces
- For example, the “List” interface is implemented by both *ArrayList* and *LinkedList*
- Each implementation has different performance characteristics
  - The choice of which implementation to use depends on your non-functional requirements
- We always write code to the interface, not the implementation
  - This allows us to swap implementations without changing our code

Collection Algorithms

- There are a number of standard cs algorithms that are part of the collections class
  - Saves programmers from having to write this low-level code over and over
  - The library implementation is optimized for execution in a JRE
- Algorithms are organized into categories
  - Sorting
  - Shuffling
  - Routine Data Manipulation
  - Searching
  - Composition
  - Finding Extreme Values
- Within each category are different implementations that have different performance characteristics


Demo
Using the Collections Framework


Java™