Introduction to Computer Science

PTI CS 103 Lecture Notes

The Prison Teaching Initiative

This version: April 3, 2019

Acknowledgements

TK.

Contents

1

Arrays

The goal of this chapter...

1.1 Motivation

Consider this snippet of code:

```java
if (i == 3) System.out.println("foo");
else System.out.println("bar");
```

Example 1.1.1: What does this code do?

```java
System.out.println();
```

This code prints the day of the week after conditioning on the value of an integer ‘day‘. But this code is repetitive. It would be useful if we had some way of creating a list of days of the week, and then just specifying which of those days we wanted to print. Something like this:

To achieve this in Java, we need arrays.

Definition 1.1.1. An *array* is an ordered and fixed-length list of values that are of the same type. We can access data in an array by *indexing*, which means referring to specific values in the array by number. If an array has ‘n‘ values, then we think of it as being numbered from ‘0‘ to ‘n-1‘:

To *loop* or *iterate* over an array means that our program accesses every value in the array, typically in order. For example, if we looped over the array in the diagram, that would mean that we looked at the value at the 0th index, then the value at the 1st index, then the value at the 2nd index, and so on.

When we say that the array is ”ordered” is that the relationship between an index and its stored value is unchanged (unless we explicitly modify it). If we loop over an unchanged array multiple times, we will always access the same values.

Arrays are *fixed-length*, meaning that after we have created an array, we cannot change its length. We will see in the next chapter [TK: confirm] that `ArrayLists` are an array-like data structure that allows for changing lengths.

Finally, all the values in an array must be of the same type. For example, an array can hold all floating point numbers or all characters or all strings. But an array cannot hold values of different types.

1.1.1 Creating arrays

The syntax for creating an array in Java has three parts:

1. Array type
2. Array name
3. Either: array size or specific values

For example, this code creates an array of size `n = 10` and fills it with all `0.0`s

```
double[] arr;                        // Declare array
arr = new double[n];                 // Initialize the array
for (int i = 0; i < n; i++) {        // Iterate over array
    arr[i] = 0.0;                    // Initialize elements to 0.0
}
```

The key steps are: we first declare and initialize the array. We then loop over the array to initialize specific values. We can also initialize the array at compile time, for example

```
String[] DAYS_OF_WEEK = {
//  Indices:
//  0      1      2      3      4      5      6
    "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"
};
```

Notice the difference in syntax. When creating an empty array, we must specify a size. When initialize an array at compile time with specific values, the size is implicit in the number of values provided.

Finally, in Java, it is acceptable to move the brackets to directly after the type declaration to directly after the name declaration. For example, these two declarations are equivalent:

```
int arr[];
int[] arr;
```

1.1.2 Indexing

Consider the array `DAYS_OF_WEEK` from the previous section. We can *index* the array using the following syntax:

```
System.out.println(DAYS_OF_WEEK[3]);  // Prints "Thursday"
```

In Java, array's are said to use *zero-based indexing* because the first element in the array is accessed with the number `0` rather than `1`.

Example 1.1.2: What does `System.out.println(DAYS_OF_WEEK[1]);` print?

Example 1.1.3: What does this code do? What number does it print?

```
double sum = 0.0;
double[] arr = { 1, 2, 2, 3, 4, 7, 9 }
for (int i = 0; i < arr.length; i++) {
    sum += arr[i];
}
```

```
System.out.println(sum / arr.length);
```

1.1.3 References

Computer Science: An Interdisciplinary Approach, Robert Sedgewick and Kevin Wayne.