Strings and Arrays

COMP 401, Fall 2018

Lecture 4

String, our first object

- In Java, a string is an immutable sequence of characters.
  - Strings are objects.
- Objects have a type.
  - The name of the class that defines them.
  - Example: String
- Objects have methods
  - Dereferenced using the "." operator
  - Example:

```
String s = "This is a string";
int length = s.length();
```

- Objects have fields
  - Properties that can be directly accessed as values.
  - Accessed via the "." operator like methods

`reference.field`

Creating Strings

- As a literal.
  - Enclosed in double quotes.
  - Escape sequences for common non-printable or untypeable characters.
    - \", \\, \t, \n, \u####
- Using the "new" operator.
  - Generally almost never need to do this.
- As the result of a string concatenation operator
- Lecture 4, Example 1

Useful String methods

- length()
- charAt()
- equals()
- substring()
- trim()
- indexOf()
- Lecture 4, Example 2

Strings are immutable

- Once created, can't change.
  - Some other languages treat strings as an array of characters. Not Java.
- Any operation that manipulates a string is creating a new string.
- Why immutability?
  - If the same string occurs twice, can simply reuse the same object.
    - This is an optimization that Java performs automatically if it can.
    - It may *appear* that == can be used to test character-by-character equality, but you should never do that.
      - Always use .equals() method of one string, passing the other as the parameter.
    - Lecture 4, Example 3

Arrays

- Arrays hold an indexed sequence of values
  - Indices start at 0
- Another object type … with a twist
  - A little different because it is a type that combines with another type.
    - The array structure itself is of type Array, but the type of the individual elements must also be specified.
      - Can't have an array of different types mixed together.
  - Also different from other objects in its creation syntax.
- Arrays are fixed length.
  - Must be specified when created.
  - Once created, can not be resized.

Creating / Initializing Arrays

- Type indicator for an array is the type name of the individual elements followed by []
- Using the new operator

  type`[] vname = new` *type*`[`*length*`];`

  – Array will be created, and initialized with default values.
    - For numeric types and char: 0
    - For boolean: false
    - For reference types: null
- Example:

```
String[] names = new String[3];
names[0] = “Alice”;
names[1] = “Bob”;
names[2] = “Carol”;
```

Literal Arrays

- When you know the elements in advance.
  - Comma-separated, in curly braces
- Syntax if combined with variable declaration

```
int[] iarray = {1, 2, 3};
String[] names = {“Abhinandan”,
                  “Bhagavateeprasaad”,
                  “Chaanakya”};
```

- Syntax if used to set an existing variable.

```
iarray = new int[] {4, 5, 6};
```

Indexing Arrays

- 0-based indexing
- Length is provided by *length* field
  - Note, for String objects, length() was a method
  - Here, length is a field
- Size of array can not change once created, but individual elements may change.
- Lecture 4, Example 4

null

- Special value that is always valid for any reference type.
  - Indicates “no value”
  - Any reference type variable can be set to null.
  - Default value for reference type arrays.

Arrays as Reference Types

- Same reference, same array
  - Implication for arrays passed to methods
    - When an array is passed to a method, any changes that the method makes to its elements are permanent.
- Array cloning
  - Easy way to create a “shallow” copy of an array
  - Just call *clone()* method
    - Result will be a new array of same size with same values or references
- Lecture 4, Example 5

Multidimensional Arrays

- Multidimensional array is simply an array of arrays
  - Fill out dimensions left to right.

```
int[][] marray = new int[5][];
for(int i=0; i<5; i++) {
    marray[i] = new int[10];
}
```

- Each subarray can have an independent size.
  - Sometimes known as as a “ragged” or “uneven” array

```
int[][] marray = new int[5][];
for (int i=0; i<5; i++) {
    marray[i] = new int[i+1];
}
```

- If each sub-dimension is same size, we can create it with a single *new* statement

```
int[][] marray = new int[5][10];
```

Arrays utility class

- *Arrays* is a library of useful functions for manipulating arrays
  - Note "s" in *Arrays*
  - Like Math class, all methods are static
- binarySearch
- sort
- filling and copying subranges
- http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

Lecture 4, Example 6

- Uses scanner to read input.
- Expects input to be a number indicating a size and then one of the following words:
  - integer, real, string
- Creates an array of that size of the corresponding type (i.e., int, double, or String)
- Uses a loop to read in that many of the appropriate type into the array.
- Prints the array.
- Does it all over again indefinitely.