CS 2100: Data Structures & Algorithms 1

Static; Writing Functions/Methods; Using Arrays

Dr. Nada Basit // basit@virginia.edu

Spring 2022

Friendly Reminders

- Masks are **required** at all times during class (University Policy)
- If you forget your mask (or mask is lost/broken), I have a few available
  - Just come up to me at the start of class and ask!
- No eating or drinking in the classroom, please
- Our lectures will be **recorded** (see Collab) – please allow 24-48 hrs to post
- If you feel **unwell**, or think you are, please stay home
  - *We will work with you!*
  - At home: eye mask instead! Get some rest ☺

Static

(no not that kind...)

- What does static mean?
  - Anything ***static*** is accessible without an object of the class
  - (Accessed / "called" directly)

- The main method is needed to run things in your program, and it is a *static* method!

```
public static void main (String[] args) {
        System.out.println("In body of main method.");
}
```

Java API ~ Application Programming Interface

- The Java API can tell you all of the objects Java has built in (very useful!)
  - https://docs.oracle.com/javase/10/docs/api/overview-summary.html#JavaSE

Objects

Declaring Variables

- Declaring variables is an **assignment** statement
  - Copy the right side to the left side
  - `int x = 4;`
    - Create space for an integer, name it ‘x’, and put the number 4 in that space

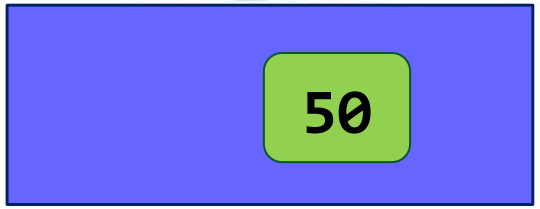
- **Primitive** variables create space on the stack (at *compile* time)

```
float radius = 1.246;
```

- **Reference** variables use space on the heap (at *run* time)

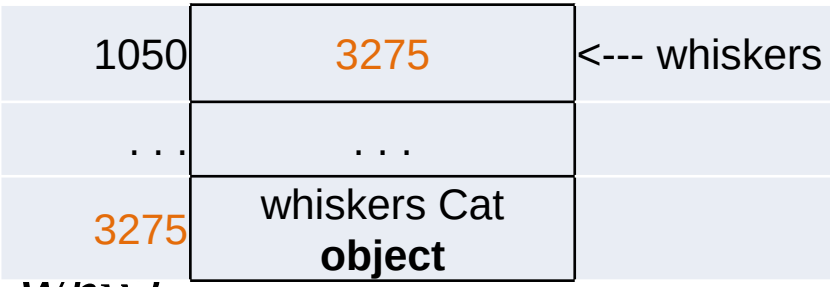
```
Cat whiskers = new Cat(“Mr. Whiskers”);
```

Brief: Primitive vs. Reference Data Types



- **Primitive data types** (*built into* Java)
  - "Literal" values, refers to literal value on **stack**
  - A "box" or chunk of memory holding the *value* itself
  - May be compared with double equal sign, `a == b`

- **Reference types** (defined from *classes*)
  - The "object" refers to the chunk of memory that holds the data
  - The variable "points to" the object in memory
  - Create new chunks (on **heap**) with **`new`** keyword
    - Calls a **constructor** for that class
  - <u>Must</u> be compared with special `.equals()` method. *Why?*



Objects: "Random" example

Java provides a random number generator

In this file we will import **`java.util.Random`**

All other variables in Java are objects!

```
/* Strings in Java are objects */
String s1 = "Hi There";

/* Many other objects exist, here's one example */
/* Note new variables use new keyword to create */

// *** Random Example - example of reference type
Random randomGenerator = new Random(); // using key-word new
int randomInt = randomGenerator.nextInt(25); // Random number 0-24 (calling method nextInt())
// .nextInt(n) means generating a random number between 0 and n (exclusive).

System.out.println(randomInt);
System.out.println("---------");
```

nextInt() method exists in the Random class: `java.util.Random`

Output: (one run)

```
14
---------
```

Objects: "Scanner" example

Java provides a set of object types for reading input from the user

```
import java.util.Scanner;
```

```
public class InputExample {
    public static void main(String args[]) {
        System.out.println("Enter two numbers: ");
        Scanner in = new Scanner(System.in);

        int x1 = in.nextInt(); // reading in 1st number entered by user
        int x2 = in.nextInt(); // reading in 2nd number entered by user
        if(x1 > x2) System.out.println("First one is bigger!");
        else if(x2 > x1) System.out.println("Second is bigger!");
        else System.out.println("They are the same!");
        /* Scanner also contains nextFloat(), nextLine(), etc. for other types */
    }
}
```

Output: (entering #s in)

```
Enter two numbers:
24 66
Second is bigger!
```

Casting

Reminder... Java is Strongly Typed

- Variables in **Python** are NOT strongly typed. I can reassign a double to a variable that was a String:

```
x = "hello there"
x = 5.2
```

- In **Java**, most of these are *invalid*:

```
int x = 5.23;     //Error: cannot convert from double to int
String s1 = 9;    //Error: cannot convert from int to String
double d1 = 5.2 //This one looks fine
d1 = 'e'          //Error: cannot convert from char to double
```

Primitive (numeric) Data Type ranking (High→Low)

- double, float, long, int, short, byte

HIGH LOW

---------- need to cast ---------→

←---------- no casting ----------

Reminder of ranking [high to low]:
double, float, long, int, short, byte

Casting (conversion between types)

```
// When converting High --> Low, need to cast the type

double d = 208.4;   // double is a higher ranking than int
int i = 2;
```

low high

```
//int res = d / i; // Error - Java won't automatically cast (High -> Low)
double res = d / i; // now this is OK
System.out.println(res); // OUTPUT: 104.2

int res2 = (int) (d / i); // Need to cast due to information loss High --> Low
System.out.println(res2); // OUTPUT: 104

// String s1 = (String)9; // Error cannot cast from int into to String
// Sometimes Java does not know how to force the conversion
```

Methods

(Called “Functions” in Python)

Methods

- **nextInt**() (see code snippet below) is called a **method**
  - Like a function, but operates on a specific instance of that type
  - In this case, get the nextInt() specifically from the object ‘in’
  - How do you know what methods are available?
    - See the **Java API** !!
- More methods coming soon!

```
int x1 = in.nextInt(); // From the previous slide
```

Basic Methods

- Methods use the ***static*** syntax

```
public class BasicMethods {

   public static void main(String[] args) {
      // declare and initialize two int variables:
      int a = 5;
      int b = 7;
      System.out.println("The sum is: " + add(a,b) ); // call add() method
   }

   public static int add(int x, int y) {
      return x + y; // add the two numbers and return the result
   }
}
```

Layout of the Class "BasicFunctions"

- There are no nested methods in a Java program. Each method sits inside the class. Order doesn't matter!

```
public class BasicMethods {

  public static void main(String[] args) {
    // declare and initialize two int variables:
    int a = 5;
    int b = 7;
    System.out.println("The sum is: " + add(a,b) ); // call add() method
  }

  public static int add(int x, int y) {
    return x + y; // add the two numbers and return the result
  }
}
```

Basic Functions

- Functions use the ***static*** syntax

```
public class BasicFunctions {

   public static void main(String[] args) {
      // declare and initialize two int variables:
      int a = 5;
      int b = 7;
      System.out.println("The sum is: " + add(a,b) ); // call or "invoke" the add() method
   }

   public static int add(int x, int y) {
      return x + y; // add the two numbers and return the result
   }
}
```

This is a <u>method</u> called "**add**".

It is situated within the "**BasicFunctions**" class but OUTSIDE of the "main" method

Basic Functions

- Functions use the ***static*** syntax

```
public class BasicFunctions {

   public static void main(String[] args) {
      // declare and initialize two int variables
      int a = 5;
      int b = 7;
      System.out.println("The sum is: " + add(a,b) ); // call or "invoke" the add() method
   }

   public static int add(int x, int y) {
      return x + y; // add the two numbers and return the result
   }
}
```

Formal parameters (e.g. x and y) must have a declared type (e.g. **int**)

The **add()** method takes as input two integer variables, invoked using 'a' and 'b', called **actual parameters**

Basic Functions

- Functions use the ***static*** syntax

```
public class BasicFunctions {

   public static void main(String[] args) {
      // declare and initialize two int variables:
      int a = 5;
      int b = 7;
      System.out.println("The sum is: " + add(a,b) ); // call or "invoke" the add() method
   }

   public static int add(int x, int y) {
      return x + y; // add the two numbers and return the result
   }
}
```

Calling the **add()** method from inside the "main" method.

Simply use the method name ("**add**") and pass in relevant parameters ('a' and 'b').

The actual parameters ('a' and 'b') do NOT have to match the **add()** method's formal parameters ('x' and 'y')

Basic Functions

- Functions use the ***static*** syntax

```
public class BasicFunctions {

   public static void main(String[] args) {
      // declare and initialize two int variables:
      int a = 5;
      int b = 7;
      System.out.println("The sum is: " + add(a,b) ); // call add() method
   }

   public static int add(int x, int y) {
      return x + y; // add the two numbers and return the result
   }
}
```

Don't worry too much about 'public' right now.
The item after 'static' is the method's **return type.**
In Java you must declare what data type a method will return. The return type can be **void** if nothing is returned by the method.

Here, the **return type** is int.
The **add()** method returns a value of type integer.
(The sum of two integers is also an integer)

Another Example Using Methods

```java
public class Example {
    public static int multiply(int a, int b) {
        return a * b;
    }

    public static String concat(String c) {
        return c + " World";
    }

    public static void main(String[] args) {
        int apple = 10;
        int cookie = 5;
        String text = "Hello";

        int crumble = multiply(apple, cookie);

        System.out.println(crumble);
        System.out.println(concat(text));
    }
}
```

What do you think the output is?

Arrays

1-dimensional Arrays

Use of the [ ] syntax in Java

A collection of variables of the same data type

- An array in java is an object
- An array variable references a group of data
- The size of an array is fixed

numbers → | 5 | 8 | 2 | 0 | 1 | 6 |

```
int[] numbers = new int[5]; // {0, 0, 0, 0, 0}

numbers[0] = 5; // {5, 0, 0, 0, 0}

numbers[2] = 8; // {5, 0, 8, 0, 0}

numbers[4] = 10; // {5, 0, 8, 0, 10}
```

Image from Neso Academy

```
int[] arr = { 34, 21, 2, 66, 567 };
```

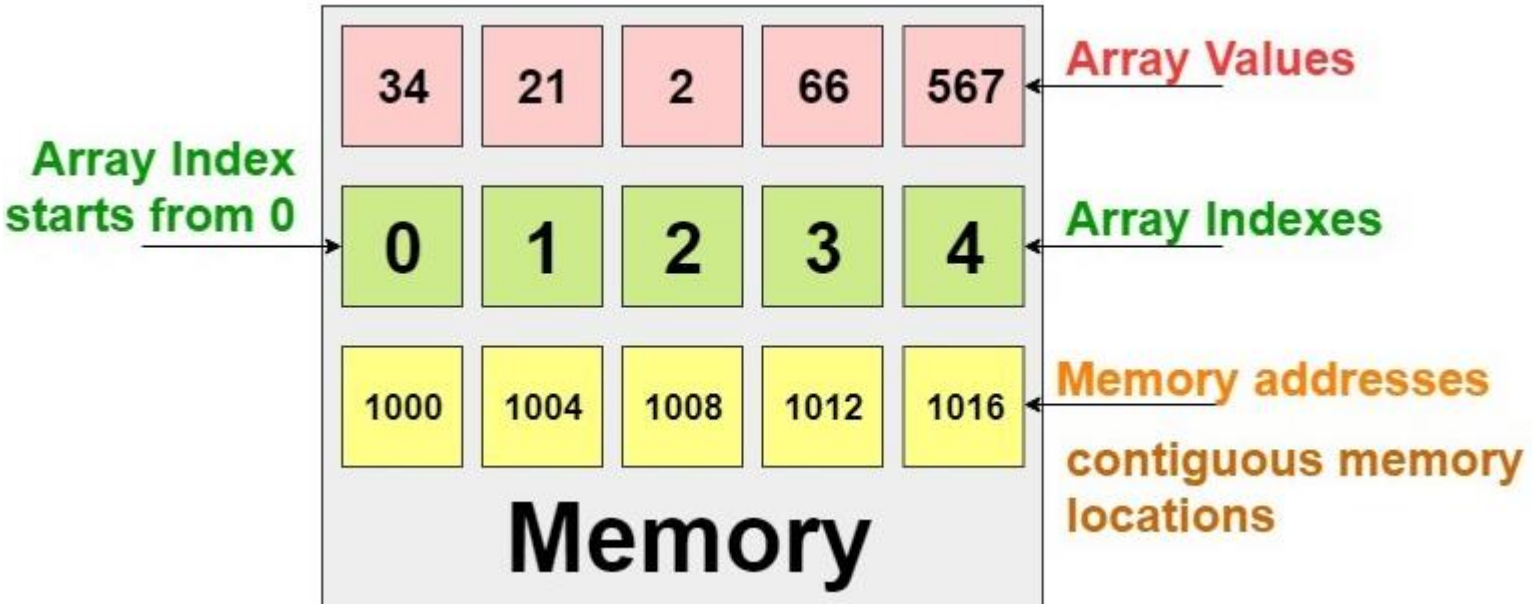


Image from SimpleSnippets.tech

Declaration and Creation

```
public class ArrayExample {
  public static void main(String[] args) {
    // [1] No import statements required to use an Array
    // [2] Creating an Array
    int size = 4; // number of elements in the array (aka size)
    double[] dblArray; // DECLARATION
    dblArray = new double[size]; // CREATION

    // DECLARATION and CREATION
    double[] myDblArray = new double[size];

    // Another example: int Array called 'arr'
    int[] arr = new int[3]; // .. Array .. // Need to specify size (here: size=3)
    . . .
```

Arrays have a fixed length. They CANNOT grow and shrink.

If you need more space, you need to allocate a new larger array and manually move items over.

Adding Elements – Arrays hold *one* kind of data

```
// Another example: int Array called 'arr'
int[] arr = new int[3]; // .. Array .. // Need to specify size (here: size=3)

// [3] Data // Arrays can only hold *one* kind of data type

// [4] Adding elements //
// .. Array ..
arr[0] = 1;
arr[1] = 2;
arr[2] = 3;
//arr[3] = 4; // CANNOT do; size cannot change (grow or shrink)
```

If not specified, contents of an Array are all set to **default values** (based on the data type of the Array) (0 for integers, "" for Strings, etc.)

If you access **OUTSIDE the range** of the array (e.g. arr[20], Java throws an **`ArrayIndexOutOfBoundsException`**

Declaration, Creation, and Initialization // Accessing

```
// DECLARATION, CREATION, and INITIALIZATION (all in one line!):
String[] cities = {"Charlottesville", "Edinburgh", "Lisbon", "Jakarta", "Dubai", "Tokyo",
                   "Melbourne", "Buenos Aires"};

// So, what is its size? As many elements as you add. In this case, 8.



// [5] Accessing specific element // use square brackets and put the index inside
// Arrays are zero-based, that is, the FIRST element is at index position zero (0)

System.out.println(cities[0]); // get the first Array element (Gets "Charlottesville")
System.out.println(cities[5]); // get the sixth Array element (Gets "Tokyo")
```

Output:

Charlottesville
Tokyo

Length of Array and Quick Printing

```
    // [6] What is the "size" of the data structure? – Use "length" attribute

    // IF you use the following you can print an Array easily: import java.util.Arrays;
    System.out.println("Array: " + Arrays.toString(cities) + " Size: " + cities.length);

    System.out.println("Size of arr is: " + arr.length);
  } // END main

} // END Class
```

Output:

```
Array: [Charlottesville, Edinburgh, Lisbon, Jakarta, Dubai, Tokyo, Melbourne, Buenos Aires]
Size: 8
Size of arr is: 3
```