Data Structure

Recitation II
• Primitive data types VS Reference type
• Two dimensional array
• Lab1 questions
Primitive data types

- `int num = 1;`

- A primitive type is predefined by the language and is named by a reserved keyword.

- *Store in memory stack*

- In addition to `int`, the Java programming language supports seven other *primitive data types*. 
It's not always necessary to assign a value when a field is declared. Fields that are declared but not initialized will be set to a reasonable default by the compiler. Generally speaking, this default will be zero or null, depending on the data type. Relying on such default values, however, is generally considered bad programming style.
In addition to its eight primitive types, Java defines two additional categories of data types: classes and arrays.

Java programs consist of class definitions; each class defines a new data type that can be manipulated by Java programs.

For example, a program might define a class named Point and use it to store and manipulate X,Y points in a Cartesian coordinate system.

This makes Point a new data type in that program.
Reference type - Class

- Class types, provided by:

- Java standard libraries: String, Integer, Date, System, ...

- Programmer: Person, Animal, Savings, HelloWorldApp
Reference type - Array

- An array type represents a list of values of some other type.

- char is a data type, and an array of char values is another data type, written char[].

- An array of Point objects is a data type, written Point[]. And an array of Point arrays is yet another type, written Point[][][].
Memory Structure

- Primitive, static, store in “Stack”
  - Size already known

- Reference, dynamic, store in “Heap”
  - Allocate during run time
When this line is executed, the compiler allocates a small amount of memory in the stack. The stack is responsible for keeping track of the running memory needed in your application.

Now the execution moves to the next step. As the name says stack, it stacks this memory allocation on top of the first memory allocation. You can think about stack as a series of compartments or boxes put on top of each other. Memory allocation and de-allocation is done using LIFO (Last In First Out) logic. In other words memory is allocated and de-allocated at only one end of the memory, i.e., top of the stack.

In line 3, we have created an object. When this line is executed it creates a pointer on the stack and the actual object is stored in a different type of memory location called ‘Heap’. ‘Heap’ does not track running memory, it’s just a pile of objects which can be reached at any moment of time. Heap is used for dynamic memory allocation.

One more important point to note here is reference pointers are allocated on stack. The statement, Class1 cls1; does not allocate memory for an instance of Class1, it only allocates a stack variable cls1 (and sets it to null). The time it hits the new keyword, it allocates on "heap".
Why Primitive data types?

- Creating an object using `new` isn't very efficient because `new` will place objects on the heap.
- This approach would be very costly for small and simple variables.
- Instead of create variables using `new`, Java can use primitive types to create `automatic` variables that are not references.
- The variables hold the value, and it's place on the stack so its much more efficient.
• `int i = 5;`
• `Integer i = new Integer(5);`
Differences

**Primitive types** are the basic types of data
- byte, short, int, long, float, double, boolean, char
- primitive variables store **primitive values**
- *Static, stored in stack*

**Reference types** are any instantiable class as well as arrays
- String, Scanner, Random, Die, int[], String[], etc.
- reference variables store **addresses**
- *Dynamic, stored in heap*
java.util.Date d = new java.util.Date();
Savings accountA = new Savings(300);
Animal[] zoo = new Animal[50];
Assignment Differences

- Copies the contents of RHS variable into LHS variable
- primitives: the primitive value is copied
- references: the address is copied
- implications: for references the object is not copied, it is shared (reference variables are aliases)
Assignment Differences

pointed-to by) accountA? accountB?

//(the object pointed-to by) accountA
has a balance of $300

accountA

balance is $300

Savings accountB = accountA;

accountB

balance is $300

accountB.deposit(150);

accountB

balance is $450
Passing Parameters

Differences

- **Terminology:**
  - formal parameter: the parameter variable that is listed (along with its type) in the method declaration
  - actual parameter: the parameter that is given when the method is called

- Public int foo(int a) - formal parameter
- Foo(5) - actual parameter
Passing Parameters Differences

- Copies the contents of actual parameter into the formal parameter
  
  primitives: the primitive value is copied (*pass-by-value*)
  references: the address is copied (*pass-by-reference*)

- **primitives**: changing the formal parameter's value doesn't affect the actual parameter's value

- **references**: changing the formal parameter's object does change the actual parameter's object since they refer to the same object
void increaseByOneFifty(Savings cash) {
    cash.deposit(150);
}

...  
//accountA has a balance of $300
increaseByOneFifty(accountA);
public class MyNumber{
    public int num;
    public MyNumber(int num){
        this.num = num;
    }
}

class Demo {
    public static void primitiveAdd(int a){
        a += 5;
    }
}

class Demo {
    public static void referenceAdd(MyNumber a){
        a.num += 5;
    }
}

class Demo {
    public static void main(String[] args){
        int primitiveNum = 1;
        MyNumber referenceNum = new MyNumber(1);
        primitiveAdd(primitiveNum);
        referenceAdd(referenceNum);
        System.out.println(primitiveNum);
        System.out.println(referenceNum.num);
    }
}
Comparison Differences

Compares the contents of the variables

- primitives: the primitive values are compared
- references: the addresses are compared
- implications: for references the contents of the objects are not compared
Comparison Differences

- For references p, q consider: p == q
- Compares address for equality. Do they refer to the same object?
Comparison Differences

- How do we test if objects are equal?
- Define a boolean method equals()
- Mydate p = new Mydate(...);
- Mydate q = new Mydate(...);
- p.equals(q) / p == q
Two-dimensional array
Two-dimensional arrays

- Two-dimensional arrays are used whenever the model data is best represented with rows and columns, or has two varying aspects (e.g., gender and age, weight and height, ...) or graphics that specifies a two-dimensional position with an x and y coordinate.

- **Terminology.** Other terms you will see for a two-dimensional array are *matrix* or *table.*
Two-dimensional arrays

- 2-dimensional arrays are usually represented with a row-column "spreadsheet" style. Assume we have an array, `a`, with two rows and four columns.

- `int[][] a = new int[2][4];`
Initiate Value for Two-dimensional arrays

- `int[][] board = new int[][] {{1,0,0},{0,1,0},{1,2,1}};`
Traverse Two-dimensional arrays

```java
int[][] a = new int[400][500]
for(int i  = 0; i < 400; i++){
    for(int j = 0; j < 500; j++){
        A[i][j] = i*j;
    }
}
```

What is the big-o value of traverse a Two-dimensional arrays? Say `int a = new int[n][n]`
Lab 1
public class Pair {
    private double x;
    private double y;
    public Pair(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public double sum() {
        return this.x + this.y;
    }
    public double product() {
        return this.x * this.y;
    }
}

demo class DemoPair {
    public static void main(String[] args) {
        Pair[] myArray = new Pair[100];
        for (int i = 0; i < myArray.length; i++) {
            myArray[i] = new Pair(Math.random(), Math.random());
        }
        for (int i = 0; i < myArray.length; i++) {
            ...
        }
    }
}
Part 4

- How to read input from user

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

String input;

Input = in.readLine();
Part 4

- N - size of array
- N items - fill in the array with these N numbers
- Values - then read value and find it in the array, print yes or no
- 0 - end