Announcements

- Program 4 has been assigned
  - Due Friday 11/10/19 , 11:59PM
  - Extra Credit due: 11/3/19, 11:59 PM
- Midterm 2
  - Wednesday 11/6 (next week)
  - Still working on Practice Problems
- Program 3 Comments

Program 4

Motivation: Learning more about the underlying “magic” that a Compiler performs

What is program 4?

- You will create a **mini compiler** that will understand a mini math programming language

Key Concepts:

- Polymorphism
- Inheritance
- Generics
- Interfaces / Abstract Classes

Program 4 (cont.)

Mini Math Programming Language

```
5 6 +      -------->          5 + 6

30 4 %   -------->          30 % 4

50 10 /   -------->           50 / 10
```

etc...

Program 4 (cont.)

Suggestions and Hints

- Remember to close your files
- You might find the String methods: Trim and isEmpty useful
- You might find the Scanner methods: hasNextLine and nextLine useful
- You might find Integers method: parseInt useful (notice that it throws and exception)
- If you are feeling unsure of how to use an ArrayList or some of the File IO objects, be sure to reread zybooks: ch 7.12 and ch 10.5.

    - You can also search on the internet for java's documentation on these objects (remember that java is owned by Oracle, so the documentation website url should be something along the lines of: docs.oracle.com/…etc)
- You may find the String class's "split" method useful
- I suggest for your delimiter you use the regex: "\s+" I leave it to you to figure out what this does
- Remember you can test your code by creating your own file, let's say for example: Application.java, creating a class called: Application, and creating a "main" method. Then you can create your own tokenizer object and test some of the methods. (Just remember not to turn in this extra file)
- Do not turn in Token.java or Tokenize.java with a "main" method in either of these files.

    - So don't have this: public static void main(String[] args) … etc

Program 4 (cont.)

Questions?

Midterm 2

Types of questions to expect:

- Multiple choice
  - Majority of the questions
  - Approximately 60% of your test grade
- Free response: read a block of code and figure out the output
  - Will have a few of these
  - Approximately 30% of your test grade
- Free response: write code to solve some problem
  - Will be syntactically forgiving but not too forgiving…
  - Only one question
  - Approximately 10% of your test grade

Midterm 2 (cont.)

Midterm Material:

- Chapter 8
  - Recommended Zybooks Sections: 8.3, 8.5, 8.7
  - Polymorphism
    - Know the difference between runtime and compile time polymorphism
    - Know the classic compile time polymorphism example (method overloading)
    - Understand how to use inheritance to create an example of runtime polymorphism
  - Understand the difference between the Is-a versus a Has-a relationship

Midterm 2 (cont.)

- Chapter 9
  - Recommended Zybooks Sections: 9.1 - 9.4
  - What is an exception?
  - What is error-checking code?
  - Try / Catch blocks
    - What are they?
    - How to use them
  - Throwing an exception
    - Programmatically how to throw an exception
    - Are there any extra lines needed in the class or method signatures?
  - Checked vs Unchecked exceptions
  - Exception Inheritance
    - Throwable, Exception, RuntimeException
    - IOException, IndexOutOfBoundsException, NullPointerException

Midterm 2 (cont.)

- Chapter 10
  - Recommended Zybooks Sections: 10.1 - 10.5
  - OutputStream
    - PrintStream
    - FileOutputStream
      - Relationship between FileOutputStream and PrintStream
  - InputStream
    - Scanner
    - FileInputStream
      - Relationship between FileInputStream and Scanner
  - Closing file streams
  - Flush

Midterm 2 (cont.)

- Chapter 11
  - Recommended Zybooks Sections: 11.1, 11.2, 11.4, 11.5
  - Abstract Class
    - What it is
    - How to create one and how to use one
    - Abstract method
  - Interface
    - What it is
    - How to create one and how to use one
  - The differences between an Abstract class and an Interface

Midterm 2 (cont.)

- Chapter 12
    - Generics
        - Generic Type
            - Naming convention
        - Generic Method
        - Generic Class
        - What it is and how to use it
    - Why we use generics
    - Comparable Interface
        - How it relates to generics
        - Why it is useful

Midterm 2 (cont.)

Questions?

Algorithms

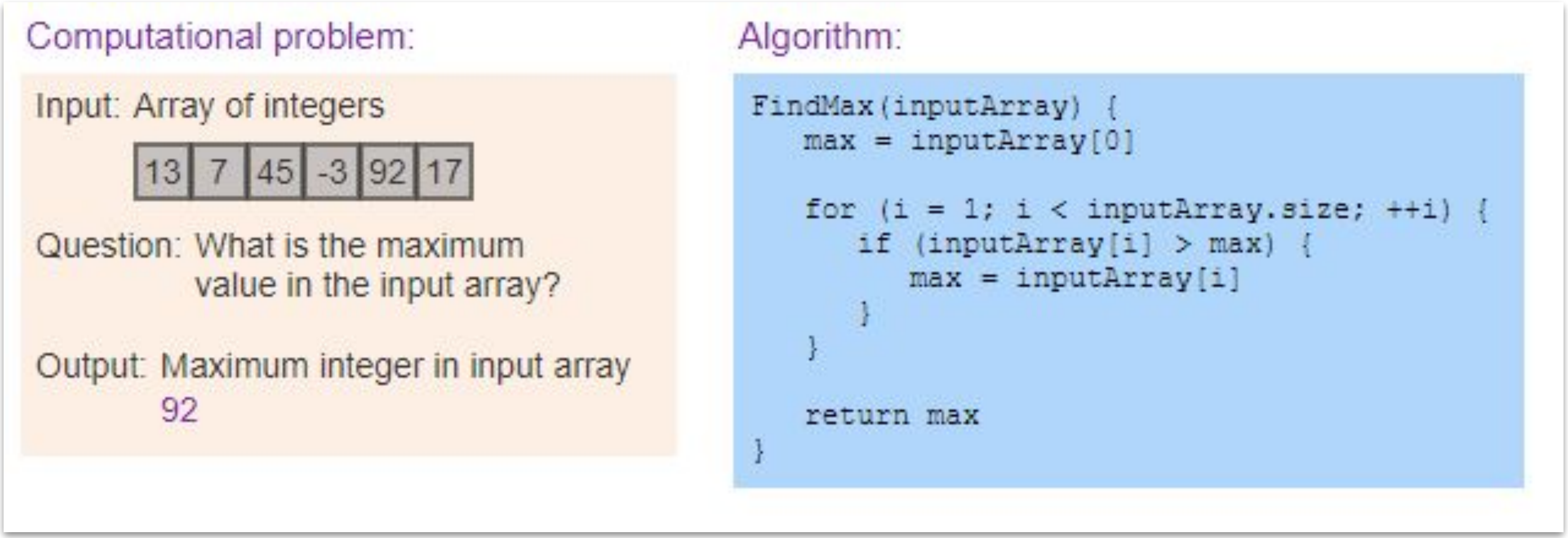
An algorithm describes a sequence of steps to solve a computational problem or perform a calculation.

A computational problem specifies:

- An input
- A question about the input
- A desired output

Algorithms (cont.)

Example:



Computational problem:

Input: Array of integers

13	7	45	-3	92	17

Question: What is the maximum value in the input array?

Output: Maximum integer in input array
92

Algorithm:

```
FindMax(inputArray) {
   max = inputArray[0]

   for (i = 1; i < inputArray.size; ++i) {
      if (inputArray[i] > max) {
         max = inputArray[i]
      }
   }

   return max
}
```

Linked Lists (cont.)

Linked List traversal algorithm: visits all nodes in the list once and performs an operation on each node.

- A common traversal operation prints all list nodes.
- The computational problem:
    - Input: Linked List
    - Question: Can all of the nodes in a linked list be visited?
    - Desired Output: Print all of the nodes in the linked list.

Linked Lists (cont.)

- The algorithm starts by assigning a node reference called: **curNode**, to the list’s head node.
- Then loop through all of the elements in the linked list by setting the curNode reference to the curNode’s “next” pointer, printing the elements as the linked list is traversed.
- The stopping condition for this algorithm is when curNode eventually becomes null, in which case the end of the list has been reached.

Linked Lists (cont.)

Example (pseudo code):

```
curNode = Head;

while ( curNode != null ) {
    print ( curNode.data );
    curNode = curNode.next;
}
```

Let’s look at an example on the board...

Linked Lists (cont.)

Now that we have a basic understanding of a linked list let us try to make one on the board together...

Linked Lists (cont.)

A **doubly-linked list** is similar to singly linked list except now each node has a pointer to the previous node as well as the next node.

The **doubly-linked** in this case refers to being to having two pointers or “links” to other the previous and next nodes. This allows a doubly-linked list to be **bi-directional** (i.e. you can traverse the list in either direction).

We will go over four common doubly-linked list operations: Append, Prepend, Insert, and Remove.

Linked Lists (cont.)

- The **append** operation for a doubly-linked list, inserts a new node after the list's tail node (java tends to call this operation: **addLast**)
- This is the same as for a singly-linked list
- The implementation is nearly identical except now you must also worry about the "previous" pointer that a node contains

Linked Lists (cont.)

- Append to an empty list: If the list’s head pointer is null (i.e. it is empty), the list’s head and tail now point to (i.e. reference) the new node.
- Append to a non-empty list: If the list’s head pointer is not null (i.e. there is at least one element in the list), the tail node’s “next” pointer must reference the new node, then point the new nodes “previous” pointer to reference the tail node, then the reference to the tail node must change to be the newNode

Let’s look at an example...