File Reading, Command Line Arguments, & Arrays

Reading with `In.java`

Purpose

In is another library (like **PennDraw**) that provides a series of functions you can use to read information from the user or from a file!

- Instead of putting all of our data directly in a program, we can direct the program to read data from elsewhere on our computer.

We'll use this throughout the course, starting in HW02 (Personality Quiz)

Getting Started with In: User Input

```
In userInput = new In();
// ^declaration ^initialization
```

Declare a variable called `userInput` with type `In` and initialize it so that it's set to read data typed in at the terminal.

- `userInput` is just a variable, so technically you can call it anything!
- `new In()` is a specific instruction that sets up reading from the terminal. This has to be written exactly like this.

Set it and forget it! Usually, you'll just do this once at the top of your program.

The `In` Toolkit for Reading User Input

Once you have initialized an **In** object and stored it in a variable, you can use any of these functions to read data from the command line!

```
userInput.readInt(); // reads in an int from the terminal
userInput.readDouble(); // reads in a double from the terminal
userInput.readBoolean(); // reads in a boolean from the terminal
userInput.readString(); // reads in a string from the terminal
```

Reading Behavior

When reading user input from the terminal, a call to `readXYZ` will *pause* your program and wait for the user to type in a value in the terminal and then press enter/return.

This allows your program to interact with a user during its execution!

Example Program: `BraggingFriend.java`

```
public class BraggingFriend {
    public static void main(String[] args) {
        In userInput = new In();
        System.out.println("How many classes are you taking this semester?");

        int response = userInput.readInt();
        System.out.println("Oh, " + response + " classes?");
        System.out.println("That's cool. I'm taking " + (response + 1));
    }
}
```

Types & Reading

What happens if the program asks for an `int` but I provide a double? Remember that we use `readInt()` in the previous example to ask for an `int` from the user.

```
$ java BraggingFriend
How many classes are you taking this semester?
four
Exception in thread "main" java.util.InputMismatchException:
attempts to read an 'int' value from the input stream,
but the next token is "four"
        at In.readInt(In.java:350)
        at BraggingFriend.main(BraggingFriend.java:6)
```

Types & Reading

`InputMismatchException` is raised and your program crashes if the type provided is not the type expected.

- There's not that much you can do about it if the user is just giving you unexpected garbage, but be aware!
- Make sure to read the error carefully! Look out for `java.util.InputMismatchException` to see when this happens.

`SayHello.java`

Prompt the user for a name to say hello to and a volume at which to say it.

```
public class SayHello {
    public static void main(String[] args) {
      In userInput = new In();
      System.out.println("Who am I saying hello to?");
      String name = userInput.readString();
      System.out.println("How loud should I be?");
      int volume  = userInput.readInt();

      if (volume <= 1) {
         System.out.println("hello " + name);
      } else if (volume == 2) {
         System.out.println("Hello, " + name);
      } else if (volume == 3) {
         System.out.println("HELLO, " + name);
      } else {
         System.out.print("HELLO, " + name.toUpperCase());
         for (int i = 0; i < volume; i++) {
               System.out.print("!");
         }
         System.out.println();
      }
   }
}
```

Reading from a File

`In` can be used to read text from a file instead of the terminal!

- Not interactive, but allows you to change how your program behaves based on a bunch of data stored in one place
- We use `In` in a similar way, but we have to specify the source of our information: the name of the file to read from.

Getting Started with In: User Input

```
String filename = "some_file.txt";
In inStream = new In(filename);
// ^declaration ^initialization
```

Declare a variable called `inStream` with type `In` and initialize it so that it's set to read data from the file called `filename`.

- `inStream` is just a variable, so technically you can call it anything!
- `new In(filename)` is a specific instruction that sets up reading from the file with name `filename`. This line changes based on the name of the file you want to read from!

The `In` Toolkit for Reading From a File

```
inStream.isEmpty(); // boolean value that is true if there are no more values, false otherwise
inStream.readInt(); // reads in an int from inStream
inStream.readDouble(); // reads in a double from inStream
inStream.readBoolean(); // reads in a boolean from inStream
inStream.readString(); // reads in a string from inStream
inStream.readLine(); // reads in an entire line from inStream (as a String)
inStream.readAll(); // reads in the entire file from inStream (as a String)
```

Once you have initialized an **In** object and stored it in a variable, you can use any of these functions to read data from the file!

The Reading Model

`inStream` will start reading from the beginning of the file (top left)

Each time a function (like `readDouble()`) is called, `inStream` attempts to read the next (unread) data from the file and interpret it as the type that you asked for

- an error will occur if the data cannot be parsed to the requested type.

The next time a read function is called, `inStream` moves to the next item in the file.

The Reading Model

`sample.txt`:

```
|4 5
Hello true 3.0
32
342
193 true
23901 391 30 3
Yes green orange
Think
```

```
In inStream = new In("sample.txt");
```

After creating the inStream, the file is set to start reading from the top left. (The cursor is marked as `|`).

The Reading Model

```
inStream.readInt(); \\ evaluates to 4
```

```
4 |5
Hello true 3.0
32
342
193 true
23901 391 30 3
Yes green orange
Think
```

Reading an int causes the **4** to be read and moves the reading position forward.

The Reading Model

```
inStream.readInt(); \\ evaluates to 5
```

```
4 5
|Hello true 3.0
32
342
193 true
23901 391 30 3
Yes green orange
Think
```

Reading an *another* int causes the **5** to be read and moves the reading position forward.

The Reading Model

`inStream.readInt(); \\ InputMismatchException!!`

```
4 5
|Hello true 3.0
32
342
193 true
23901 391 30 3
Yes green orange
Think
```

Reading an *yet another* int causes the program to crash! The next data in the file is the character `'H'`, which can't be the start of an int.

The Reading Model

`inStream.readString(); \\ evaluates to "Hello"`

```
4 5
Hello |true 3.0
32
342
193 true
23901 391 30 3
Yes green orange
Think
```

Instead, we could have tried to read it as a `String`, which reads one word at a time and moves the reading position again.

Examples:

`recitation_roster.txt`:

```
5
Harry 201
Adi 203
Becca 204
Sukya 201
Bhrajit 202
```

Our goal is to iterate through the file `recitation_roster.txt` and print out the names of students in recitation 201.

Examples:

`recitation_roster.txt`:

```
5
Harry 201
Adi 203
Becca 204
Sukya 201
Bhrajit 202
```

```
In inStream = new In("recitation_roster.txt");
```

Examples:

`recitation_roster.txt`:

```
5
Harry 201
Adi 203
Becca 204
Sukya 201
Bhrajit 202
```

```
In inStream = new In("recitation_roster.txt");
int numStudents = inStream.readInt();
```

Examples:

`recitation_roster.txt`:

```
5
Harry 201
Adi 203
Becca 204
Sukya 201
Bhrajit 202
```

```
In inStream = new In("recitation_roster.txt");
int numStudents = inStream.readInt();
for (int i = 0; i < numStudents; i++) {
   // this loop runs once per line!
}
```

Examples:

`recitation_roster.txt`:

```
5
Harry 201
Adi 203
Becca 204
Sukya 201
Bhrajit 202
```

```java
In inStream = new In("recitation_roster.txt");
int numStudents = inStream.readInt();
for (int i = 0; i < numStudents; i++) {
   // this loop runs once per line!
   String name = inStream.readString();
   int recitation = inStream.readInt();
}
```

Examples:

`recitation_roster.txt`:

```
5
Harry 201
Adi 203
Becca 204
Sukya 201
Bhrajit 202
```

```
In inStream = new In("recitation_roster.txt");
int numStudents = inStream.readInt();
for (int i = 0; i < numStudents; i++) {
   // this loop runs once per line!
   String name = inStream.readString();
   int recitation = inStream.readInt();
   if (recitation == 201) {
      System.out.println(name);
   }
}
```

Command Line Arguments

Specifying Command Line Arguments

It's possible to put information after `java MyProgramName` when running the program!

```
$ java Greet howdy partner
6

$ java Greet hello sailor
1
```

Accessing Command Line Arguments

Finally, an explanation for `String[] args`.

```
public class Greet {
   public static void main(String[] args) { // args is a "String array"
      String greeting = args[0];
      String name = args[1];
      System.out.println(greeting + ", " + name + "!");
   }
}
```

`args` is a variable that stores all of the command line arguments as `String` values.

Accessing Command Line Arguments

`args` is an array—we'll talk about what this means **very** soon—that stores all of the command line arguments in one place.

- To access the first thing written after the program's name, use `args[0]`
- To access the second thing written after the program's name, use `args[1]`
- To access the `i`th thing written after the program's name, use `args[i]`

Poll Time

If I run a program with `java PollProgram 3 Harry 5.0`, what is the value of `args[1]`?

Poll Time

If I run a program with `java PollProgram 3 Harry 5.0`, what is the value of `args[1]`?

"Harry"

Poll Time

If I run a program with `java PollProgram 3 Harry 5.0`, what is the value of `args[2]`?

Poll Time

If I run a program with `java PollProgram 3 Harry 5.0`, what is the value of `args[2]`?

```
5.0
```

Poll Time

If I run a program with `java PollProgram 3 Harry 5.0`, what is the value of `args[3]`?

Poll Time

If I run a program with `java PollProgram 3 Harry 5.0`, what is the value of `args[3]`?

> Triggers an `ArrayIndexOutOfBoundsException`, the first of many we'll see.

Type of Command Line Arguments

`args` is a `String[]`—a "String array". This means that all of the command line arguments are stored as `String` values.

- If you want to treat them as anything else, you have to *parse* them!
  - e.g. `Integer.parseInt(), Double.parseDouble())`

Example of Parsing Command Line Arguments

```
public class AddTwoNumbers {
    public static void main(String[] args) {
        int firstNumber = Integer.parseInt(args[0]);
        int secondNumber = Integer.parseInt(args[1]);

        System.out.println(firstNumber + secondNumber);
    }
}
```

Rules for Command Line Arguments

- Any values written after the program name when running the program are stored as `Strings` in the array variable `args`
- `args` is always in scope in `main`—you don't have to declare it yourself.
- To access the first command line argument, use `args[0]`. To get the second, use `args[1]`. And so on.
  - To figure out how many command line arguments were passed in, you can ask for `args.length`, which is an `int`
- To treat CLAs as anything other than `Strings`, you have to *parse* them!
  - e.g. `Integer.parseInt(), Double.parseDouble())`

RECAP

- `In.java` can be used two ways: read input from the user while the program is running, or read input from a file.
  - To read input from a file, declare and initialize a new `In` and pass the name of the file as a string: `In inStream = new In("filename.txt");`
  - To read user input, declare and initialize a new `In` without any arguments: `In userInput = new In();`
- `args` is an array of `String` values that are passed in before the program starts running.

All three of these techniques can be employed in the same program to do different things!

Arrays

Overview

Often, we need to store and manipulate several variables; a program will use an array to store a collection of variables of the same type

In this module we will learn how to store several variables inside an array

You can think of an array as a collection of variables in which each variable occupies a specific position

Example:

- Store your friends' phone numbers inside an array named *besties*

Learning Objectives

To be able to declare an array

To be able to initialize an array with the new keyword & with an initializer list.

To be able to access array values

To be able to modify array values

To be able to traverse an array using the for-loop

To be able to solve problems using arrays

What Does an Array Do?

Up until now: variables store a single data item (e.g. `59` or `"Hello"`)

Now: *arrays* are special variables that store a list of data items under a single name.

- Each of the data items in the array is called an *element*

```
|-------array---------|
[4, 5, 9, 10, -20, 32]
    ⬆️ one element
```

Arrays Have Indices

Since arrays are collections of data, we need some way of accessing the individual pieces of data inside the array.

Each element's location number is called the **index**.

If an array has `n` elements, then the indices of the array range from `0` to `n-1`

```
ARRAY   :   [39.02, 902.094, 94.2, -29.3]
INDICES:    0       1         2     3
```

Array Elements Can Be Accessed & Set

```
grades <-- [94, 89, 77, 100, 88]
```

- Accessing & printing the third element of the array: `System.out.println(grades[2])`
- Assigning a new value to the first element: `grades[0] = 98`

Practice with Indices

I have an array with 10 elements called `studentNames`...

1. How can I assign the first element in the list to be `"Han"`?
2. What are the valid indices for elements in `studentNames`?
3. How can I set the last element to be the value of the second element?

Practice with Indices

Answers

1. `studentNames[0] = "Han";`
2. `0` through `9`
3. `studentNames[9] = studentNames[1];`

Array Syntax in Java

Declaring & Allocating a New Array

```
dataType[] arrayName = new dataType[numElements];
```

This tells Java to declare a new variable called `arrayName`. This variable has type `dataType[]`, which reads aloud like "data type array".

The initial value of `arrayName` is set to be a new array of type `dataType[]` with a length of `numElements`.

e.g. `int[] officeHourAttendance = new int[5];`

More Rules about Indices

- Indexing into an array is an expression
  - `arr[1]` works
  - `arr[i]` works
  - `arr[i * 3 - j + 2]` works
- The type of the index must be `int`
  - `arr["Harry"]` does not work!

Exercise: Our First Array

Write a program that declares an array that will store doubles named `homeworkScores`. The array should have a length of `4`.

The array stores the average homework scores on the first `4` homeworks for a class. The average for HW1 was `89.3`, for HW2 was `99.0`, for HW3 was `77.8`, and for HW4 was `82.84`. Store these values in the correct order in the array `homeworkScores`.

Solution: Our First Array

```
public class FirstArray {
    public static void main(String[] args) {
        double[] homeworkScores = new double[4];

        homeworkScores[0] = 89.3;
        homeworkScores[1] = 99.0;
        homeworkScores[2] = 77.8;
        homeworkScores[3] = 82.84;
    }
}
```

Printing Arrays

More complicated than we might hope...

```
public class FirstArray {
    public static void main(String[] args) {
        double[] homeworkScores = new double[4];
        homeworkScores[0] = 89.3;
        homeworkScores[1] = 99.0;
        homeworkScores[2] = 77.8;
        homeworkScores[3] = 82.84;
        System.out.println(homeworkScores);
    }
}
```

➡️ `[D@65e579dc` 🤔 🤔 🤔 🤔

Printing Arrays

We can still print the individual elements of the array.

```
public class FirstArray {
    public static void main(String[] args) {
        double[] homeworkScores = new double[4];
        homeworkScores[0] = 89.3;
        homeworkScores[1] = 99.0;
        homeworkScores[2] = 77.8;
        homeworkScores[3] = 82.84;
        System.out.println(homeworkScores[1]);
    }
}
```

➡ `99.0`

Loops and Arrays

We can use for loops to print out our arrays!

```
public class FirstArray {
    public static void main(String[] args) {
        double[] homeworkScores = new double[4];
        // homeworkScores elements set as before...

        for (int i = 0; i < homeworkScores.length; i++) {
            System.out.print(homeworkScores[i] + " ");
        }
    }
}
```

➡️ `89.3 99.0 77.8 82.84` ✅

Loops & Arrays

All arrays have a property `length` that stores how many elements are inside.

```
String[] names = new String[17];
System.out.println(names.length);
```

➡️ `17`

To loop over every index in an array, start from `0` and increment until the last valid index, `arr.length - 1`.

```
for (int i = 0; i < arr.length; i++) {...}
```

Check-in

```
In userInput = new In();
int numStudents = 55;
int numTAs = 5;
String[] pennkeys = new String[numStudents + numTAs];
for (int i = 0; i < pennkeys.length; i++) {
    System.out.println("Enter Student/TA PennKey:");
    pennkeys[i] = userInput.readString();
}
```

How many iterations does this for loop run for? What is the value of `pennkeys.length`?

Check-in

```
In userInput = new In();
int numStudents = 55;
int numTAs = 5;
String[] pennkeys = new String[numStudents + numTAs];
for (int i = 0; i < pennkeys.length; i++) {
    System.out.println("Enter Student/TA PennKey:");
    pennkeys[i] = userInput.readString();
}
```

60

Alternative Array Initialization

Only at the same time that you declare the array variable, you can write out the elements of the array manually:

```
int[] newArray = {5, 7, 11};
```

- The elements are automatically set
  - `newArray[0] = 5, newArray[1] = 7, newArray[2] = 11`
- The length of the array is automatically interpreted
  - Don't need to provide that the length is `3`
  - `newArray.length == 3` automatically

Exercise: Printing in Reverse

Can you write a loop that prints out the elements of an array in reverse? Think carefully about loop variable! Where should it start? How should it be updated? When should it stop?

```
int[] toPrint = {1, 2, 3, 4, 5};

// your code here
// Should print out 5 4 3 2 1
```

Solution 1: Printing in Reverse

```
int[] toPrint = {1, 2, 3, 4, 5};

for (int i = toPrint.length - 1; i >= 0; i--) {
    System.out.println(toPrint[i]);
}
```

Iterate backwards from `toPrint.length - 1` down to (and including) `0`, decrementing `i` by `1` every iteration.

Solution 2: Printing in Reverse

```
int[] toPrint = {1, 2, 3, 4, 5};

for (int i = 0; i < toPrint.length; i++) {
    System.out.println(toPrint[toPrint.length - (i + 1)]);
}
```

Iterate forwards the normal way, but access the indices using the formula `toPrint.length - (i + 1)`. *Why does this work?*

Bad Solution 1: Printing in Reverse

```
int[] toPrint = {1, 2, 3, 4, 5};

for (int i = toPrint.length - 1; i > 0; i--) {
    System.out.println(toPrint[i]);
}
```

What's wrong?

Bad Solution 1: Printing in Reverse

```
int[] toPrint = {1, 2, 3, 4, 5};

for (int i = toPrint.length - 1; i > 0; i--) {
    System.out.println(toPrint[i]);
}
```

What's wrong? *We never print the first element of the array since we stop before* `i = 0`

Bad Solution 2: Printing in Reverse

```
int[] toPrint = {1, 2, 3, 4, 5};

for (int i = 0; i < toPrint.length; i++) {
    System.out.println(toPrint[toPrint.length - i]);
}
```

What's wrong?

Bad Solution 2: Printing in Reverse

```
int[] toPrint = {1, 2, 3, 4, 5};

for (int i = 0; i < toPrint.length; i++) {
    System.out.println(toPrint[toPrint.length - i]);
}
```

What's wrong? *We try to access* `toPrint[toPrint.length - 0]` *in the first iteration, which is not a valid index. Only* `0` *through* `toPrint.length - 1` *are valid.*

Finding the Biggest Element in an Array

Biggest Element Solution

```
public static void main(String[] args) {
    In userInput = new In();
    int NUM_INPUTS = 5;
    double[] inputs = new double[NUM_INPUTS];

    for (int i = 0; i < inputs.length; i++) {
        inputs[i] = userInput.readDouble();
    }

    double biggestSoFar = inputs[0];
    for (int i = 0; i < inputs.length; i++) {
        if (inputs[i] > biggestSoFar) {
            biggestSoFar = inputs[i];
        }
    }
    System.out.println("Biggest element: " + biggestSoFar);
}
```

Exercise: `GradeCalculator.java`

Prompt the user for for their homework score, their average exam score, their final project score, and their attendance score. Use these values to calculate their final grade in the class.

Component	Weight
HW	60%
Exam	20%
Final Project	10%
Attendance	10%

Swapping

Swapping array elements is a little more complicated than it might seem.

What happens here?

```
String[] names = {"Harry", "Adrian", "Vivian"};
names[0] = names[2];
names[2] = names[0];
```

Swapping

Swapping array elements is a little more complicated than it might seem.

What happens here?

```
String[] names = {"Harry", "Adrian", "Vivian"};
names[0] = names[2];
names[2] = names[0];
```

`names` becomes `{"Vivian", "Adrian", "Vivian"}` since `"Harry"` gets overwritten by `"Vivian"`.

Swapping

Swapping correctly:

```
String[] names = {"Harry", "Adrian", "Vivian"};
String temp = names[0]; // temp is "Harry", this doesn't change with names

names[0] = names[2]; // We get {"Vivian", "Adrian", "Vivian"}
names[2] = temp; // We get {"Vivian", "Adrian", "Harry"}
```

Fundamental Rules of Arrays

- Once created, an array's length cannot change
- Copying an array into another variable does not create a new array

```
int[] x = {3, 4, 5};
int[] y = x;
x[1] = 17;
System.out.println(y[1]); // --> Prints 17!!!
```

- In order to duplicate an array, you need to initialize an entirely new array of the same size.

Copying an Array

```
int[] x = {3, 4, 5};
int[] y = new int[x.length]; // init new array of the same size
for (int i = 0; i < x.length; i++) {
    y[i] = x[i]; // copy each element individually
}
x[1] = 10;
y[1] = -10;


System.out.println(x[1]); // --> prints 10;
System.out.println(y[1]); // --> prints -10
```

Exercise: Reversed Copy

Write a program that copies the values of an array into another new array, but in reverse!

e.g. `{4, 3, 6} -> {6, 3, 4}`

Solution: Reversed Copy

Write a program that copies the values of an array into another new array, but in reverse!

```
int[] x = {3, 4, 5};
int[] reversed = new int[x.length]; // init new array of the same size
for (int i = 0; i < x.length; i++) {
    reversed[reversed.length - i - 1] = x[i]; // copy each element individually
}
```

Put `x[0]` into `reversed[2]`, `x[1]` into `reversed[1]`, and `x[2]` into `reversed[0]`

Default Values

When we initialize an array with **new**, Java will make each element some “default” value depending on the type

Examples

`int[] myArray = new int[6]`

- integers in the array will start off as `0`

`String[] studentsArray = new String[10]`

- Strings (or any other object type) in the array will start as `null`
  - `null` leads to crashing programs, so be careful!

`double` starts as `0.0`, `boolean` starts as false.