

# CSE 143

## Computer Programming II

## Welcome to CSE 143!



### Outline

- 1 Administrivia
- 2 Code Reviews
- 3 Back to CSE 142
- 4 ArrayLists

### Course Goals

1

#### CSE 142 vs. CSE 143: The Big Picture

In **CSE 142**, you learned how to use logic, control flow, and decomposition to write programs.

In **CSE 143**, you will learn to solve **more complex** and **larger** tasks **efficiently**.

#### Big Learning Goals

- Abstraction (implementation vs. client)
- Data Structures (organizing complex data)
- Algorithms (standard ways of completing common tasks)

We're going to build some **really cool** programs. And have a lot of fun!

### Boring Administrivia

2

#### Course Website

<http://cs.uw.edu/143>

#### Section

We have **two** sections a week.

Each section has a **warm-up**; these are **completely optional**.

#### Grading

- 50% programming projects, 20% midterm, 30% final
- Weekly programming projects assigned **Fridays**, due on **Thursdays**
- 5 "free late days"; 10% off for subsequent days late; up to **2** days late on each HW

### Support and Asking for Help

3

#### Resources

- **TWO** sections a week
- Tons of TAs!
- The IPL (and my office hours!)
- Practice-It

**Asking for help is not a sign of weakness; it's a sign of strength.**

What does it mean for a program to be “correct”?

What does this code do?

```

- ( __ , __ , __ ) { __ / __ <= 1 ? ( __ , __ + 1 , __ ) : ! ( __ % __ ) ? ( __ , __ + 1 , 0 ) : __
% __ = __ / __ && ! __ ? ( printf ( "%d\t" , __ / __ ) , ( __ , __ + 1 , 0 ) ) : __ %
> 1 && __ < __ / __ ? ( __ , 1 + __ , __ + ! ( __ / __ % ( __ % __ ) ) ) : __ < * __
? ( __ , __ + 1 , __ ) : 0 ; } main ( ) { _ ( 100 , 0 , 0 ) ; }
    
```

Programs must be written for people to read, and only incidentally for machines to execute. (Abelson & Sussman)

What “Program Correctness”?

- Your code does the right thing on **all** inputs
- The code is easy to read
- The code is well documented
- The code is well formatted
- The code is efficient
- ...

How Can We Determine If A Program is Correct?

- Write tests
- Think about “edge cases”
- Ask someone experienced to review your code!

This last one is actually really important!

Industry calls it a “code review”.

Google has a “legalese” document describing “Google Style”:

<https://google.github.io/styleguide/javaguide.html>

If a Google programmer submits code that misses even **one** of these guidelines, it is

**REJECTED!**

Each programming language (C++, Java, Python, etc.) has different guidelines.

The actual guidelines themselves aren’t important; Facebook, for example, has different ones.

The relevant skill here is **being able to follow style guidelines**.

We will grade your programming assignments by code review.

Graders will check that your code follows “CSE 143 Style”:

<http://courses.cs.washington.edu/courses/cse143/16au/style>

Beware! The style guide may not include **everything!**

Google’s doesn’t either.

Words Exercise

Write code to read a file and display its words in reverse order.

(Bad) Solution with Arrays

```

1 String[] words = new String[1000];
2 int i = 0;
3
4 Scanner inp = new Scanner(new File("words.txt"));
5 while (inp.hasNext()) {
6     String word = inp.next();
7     words[i] = word;
8     i++;
9 }
10 for (int j = i - 1; j >= 0; j--) {
11     System.out.println(words[j]);
12 }
    
```

Arrays are one way to store **many** values of the same type (int, String, DrawingPanel, etc.).

```
int[] arr = new int[8];
```

arr[0]	12	arr[1]	49	arr[2]	-2	arr[3]	26	arr[4]	2	arr[5]	6	arr[6]	26	arr[7]	11
--------	----	--------	----	--------	----	--------	----	--------	---	--------	---	--------	----	--------	----

“Element #3 is 26”

“arr has size 8”

Limitations of Arrays

- Fixed, upfront size (once you create the array, it will remain that size)
- Adding and removing can get complicated
- No methods (and weird “.length” syntax)  
Functionality for arrays is in the Arrays class:
  - Arrays.copyOf
  - Arrays.equals
  - Arrays.sort
  - Arrays.toString

## Collections

Collections store **many** pieces of data of **the same type**.

In Java, collections are in the `util` package:

```
import java.util.*;
```

Different collections have different properties:

- "Data ordered by indices"
- "Sorted data"
- "Data without duplicates"
- etc.

## Lists

A **list** is a collection of elements ordered by a 0-based index.

- It supports add/remove from anywhere!
- The size isn't fixed!
- There are multiple implementations; first, `ArrayList`

- Suppose we have an `ArrayList` with values: [1, 2, -6]:

Step 0: 

1	2	-6	...
0	1	2	

- Insert 5 at index 2:

Step 1: 

1	2	5	-6	...
0	1	2	3	

- Add 0 at the beginning:

Step 2: 

0	1	2	5	-6	...
0	1	2	3	4	

- Get index 3:

```
arrayList.get(3) → 5
```

<code>add(val)</code>	Appends <code>val</code> to the end of the list
<code>add(idx, val)</code>	Puts <code>val</code> at index <code>idx</code> ; all elements at indices <code>idx</code> and larger get shifted forward
<code>get(idx)</code>	Returns the value at index <code>idx</code>
<code>set(idx, val)</code>	Replaces the value at index <code>idx</code> with <code>val</code>
<code>remove(idx)</code>	Removes <b>and</b> returns the value at index <code>idx</code> ; all elements at higher indices get shifted backward
<code>clear()</code>	Removes all elements from the list
<code>size()</code>	Returns the number of elements in the list
<code>indexOf(val)</code>	Returns the smallest index such that <code>get(idx).equals(val)</code> , or -1 if there is no such index
<code>toString()</code>	Returns a string representation of the list such as [3, 42, -7, 15]



Recall that we can create arrays of different types:

```
{1, 2, 5, 2}           {"hi", "banana"}
(new int[4])          (new String[2])
```

Since the array initializations specify the **type** of the elements, the declaration for `ArrayList`'s should too:

```
[1, 2, 5, 2]           ["hi", "banana"]
(new ArrayList<Integer>) (new ArrayList<String>)
```

`ArrayList` is a **generic** class which means that it can handle any type you want! Java knows the type by what you put in `<>`:

```
ArrayList<String> arrayList = new ArrayList<String>();
```

```
String[] arr = new String[5];           → ArrayList<String> list = new ArrayList<String>();
arr[0] = "hi";                          → list.add("hi");
arr[1] = "bye";                          → list.add("bye");
String s = arr[0];                       → String s = list.get(0);
for (int i=0; i < arr.length; i++) {     for (int i = 0; i < list.size(); i++) {
    if (names[i].contains("b")) {...}    if (list.get(i).contains("b")) {...}
}
```

Note that these two pieces of code have **different** loop bounds:

```
arr.length == 5
```

```
list.size() == 2
```

`ArrayList` is just another type (like `DrawingPanel` or `String`)!

```
1 public void methodName(..., ArrayList<Type> name, ...) { ... }
2 public ArrayList<Type> methodName(...) { ... }
```

The following takes in an `ArrayList` and returns a new list containing only the words that start with `x`:

```
1 public ArrayList<String> startingWithX(ArrayList<String> list) {
2     ArrayList<String> newList = new ArrayList<String>();
3     for (int i=0; i < list.length; i++) {
4         if (list.get(i).startsWith("x")) {
5             newList.add(list.get(i));
6         }
7     }
8     return newList;
9 }
```

**Words Exercise... Now with more ArrayList!**

Write code to read a file and display its words...

- 1 in reverse order (but using an ArrayList)
- 2 with all words ending in "s" capitalized
- 3 with all words ending in "s" removed

```

1  /* Read in the words */
2  ArrayList<String> allWords = new ArrayList<String>();
3  Scanner input = new Scanner(new File("words.txt"));
4  while (input.hasNext()) {
5      String word = input.next();
6      allWords.add(word);
7  }
8
9  /* Display in Reverse Order */
10 for (int i = allWords.size() - 1; i >= 0; i--) {
11     System.out.println(allWords.get(i));
12 }

```

```

1  /* Remove All Words Ending in 's' */
2  for (int i = 0; i < allWords.size(); i++) {
3      String word = allWords.get(i);
4      if (word.endsWith("s")) {
5          allWords.remove(i);
6
7          /* This is the tricky part; since we removed a word,
8           * we're actually at the SAME index again! */
9          i--;
10     }
11 }

```

- Understand the course policies
- Learn why code reviews are important (Are you convinced?)
- Recall arrays and how they work from CSE 142
- Begin being a **client** of the ArrayList class