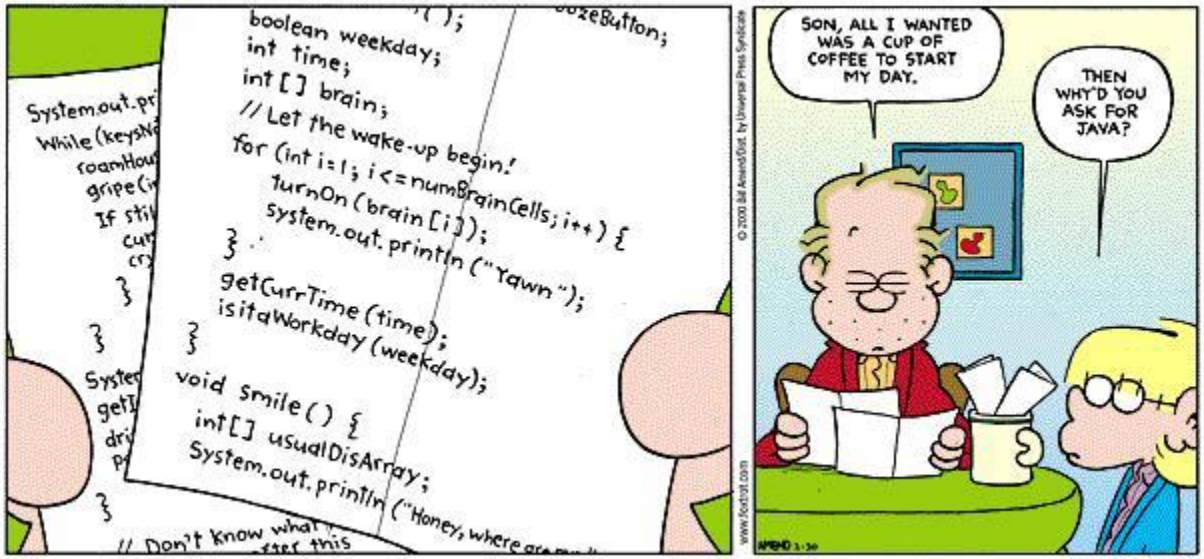
CS 142, Spring 2024

Lecture 1: introductions, review



Thank you to Marty Stepp and Stuart Reges for parts of these slides

Welcome to CS 142!

I'm Allison Obourn

Email: allison.obourn@edmonds.edu

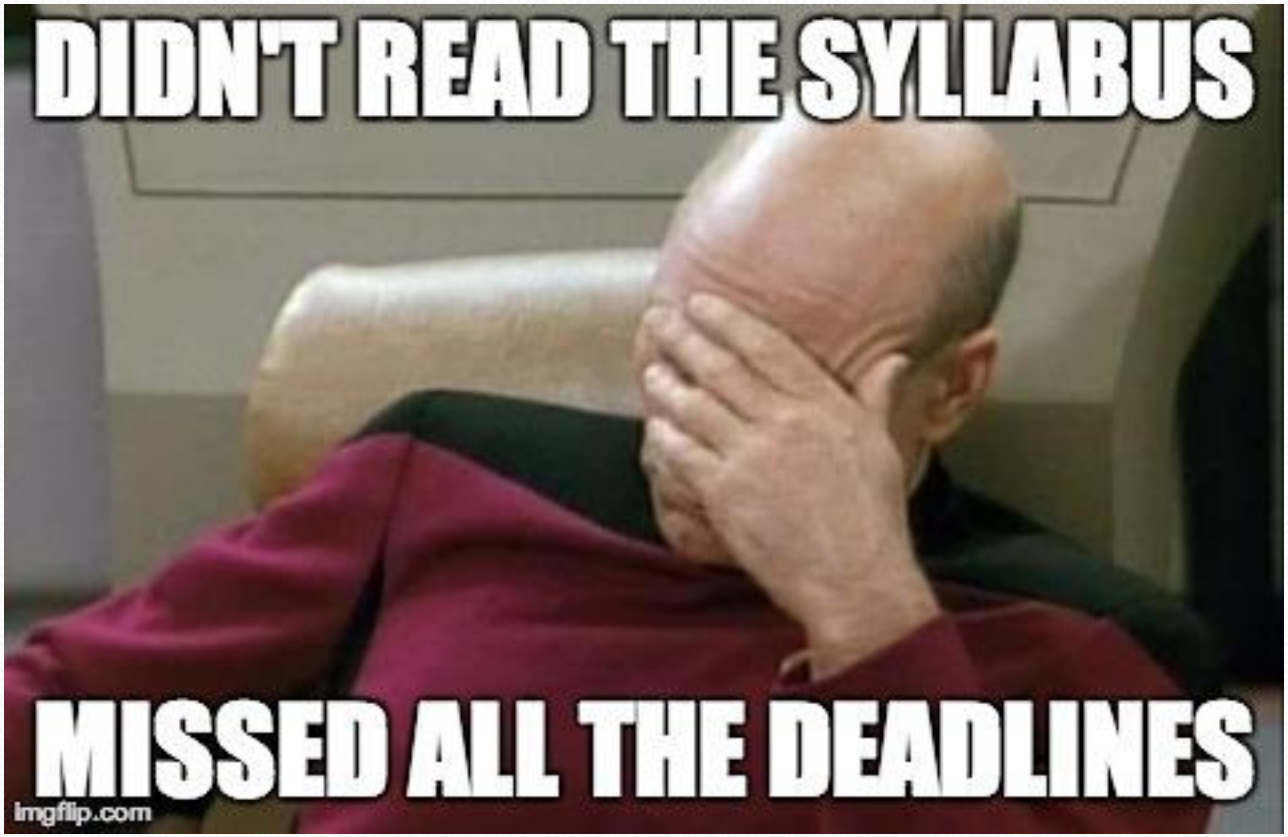
Course website: https://allisonobourn.com/edmonds/142

CS 142

- **141:** can automate basic tasks using a programming language (logic, control flow, decomposition)
- **142:** can work with and create your own object classes and create graphical user interfaces
- **143:** learn tools for automating complex tasks efficiently
  - Reinforce abstraction (client vs. implementation)
  - Data structures
  - Algorithms

Being Successful

- Determination, hard work, focus
- Investing time (~15 hours a week)
  - Starting early
  - Developing problem-solving strategies
- Knowing when to ask for help
  - Talk to me after class, during office hours
  - Go to the STEM study room or MESA study center
- Studying together
  - Projects 1 - 5 are individual but studying in groups pays off



Every quarter a few students complete everything EXCEPT the projects and are surprised they fail the class.

YOU CAN'T PASS UNLESS YOU DO THE PROJECTS

Logistics

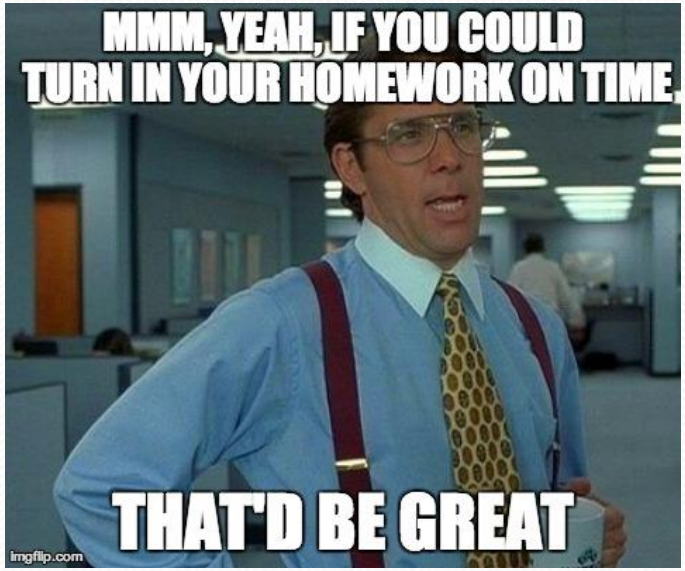
Get to know https://allisonobourn.com/edmonds/142

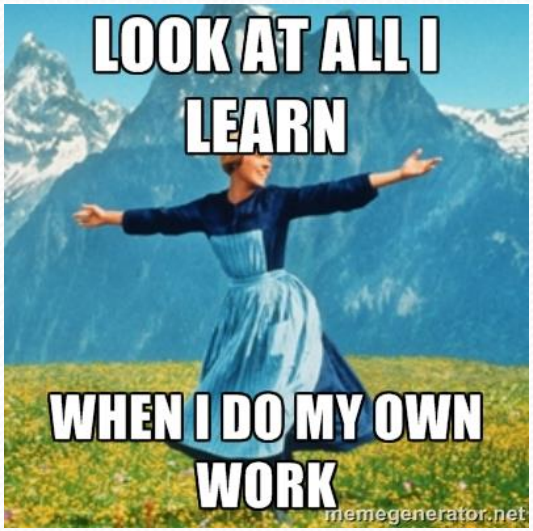
- Lab activities
  - We will do lab activities every class. If you attend and participate you will get credit regardless of how many problems you complete.
- Exercises
  - Due Fridays and Mondays
  - Meant as a practice aid
  - Half credit if late
- Quizzes

Programming projects 1 - 5

- Must be complete **individually**

- 4 "free late days“; you can use a max of 3 on one assignment; -1 for subsequent days late





Group final project

- You will work in groups of 2 - 3
- Your group will create an animated simulation of your choosing.
- To improve your group work experience, there will be several intermediate deadlines to help you stay on track
- Your experience and final product will be much better if you write code together (viewing the same screen at the same time)

What are you still confused about?

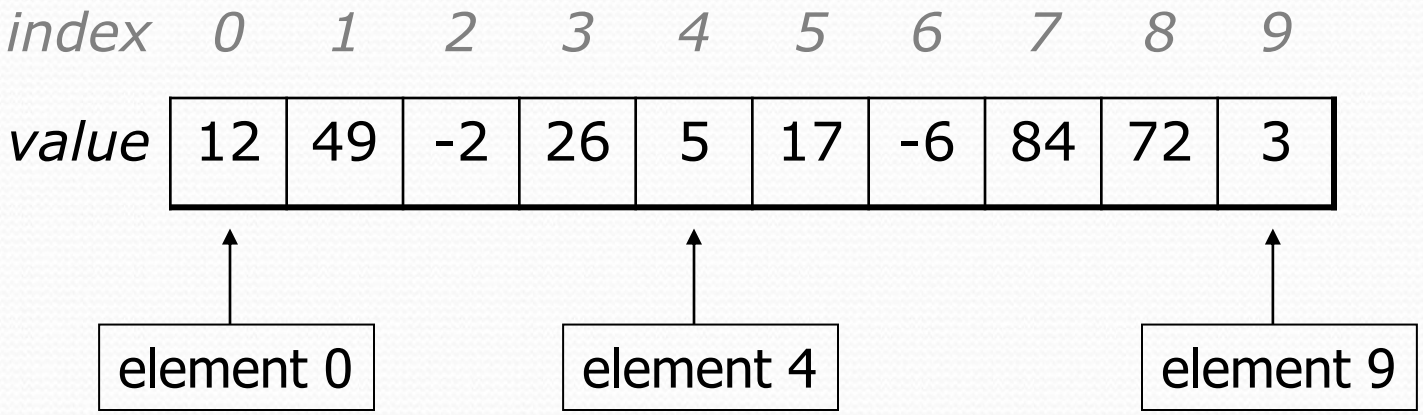
- Although we expect you to be familiar with the material, we know everyone has areas where they struggle.

141 Topics

- variables and expressions
- string manipulation
- `boolean` expressions
- `if` statements
- `while` loops
- `for` loops
- file I/O and reading from the console with `Scanner`
- arrays
- using objects like `Scanner` and `Random`
- creating your own objects

Array Review

- **array**: object that stores many values of the same type.
  - **element**: One value in an array.
  - **index**: A 0-based integer to access an element from an array.



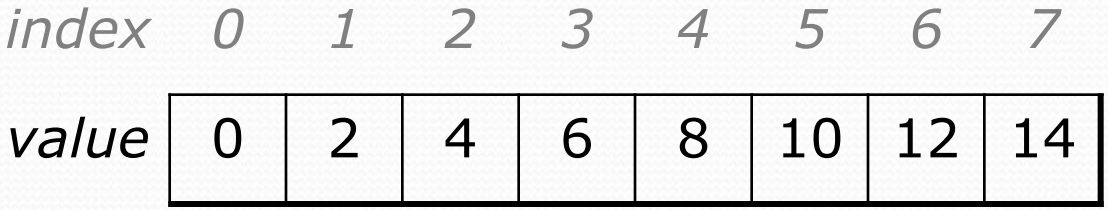
Arrays and `for` loops

- It is common to use `for` loops to access array elements.

```
for (int i = 0; i < 8; i++) {
    System.out.print(numbers[i] + " ");
}
System.out.println();  // output: 0 4 11 0 44 0 0 2
```

- Sometimes we assign each element a value in a loop.

```
for (int i = 0; i < 8; i++) {
    numbers[i] = 2 * i;
}
```



Limitations of arrays

- You cannot resize an existing array:

```
int[] a = new int[4];
a.length = 10;              // error
```

- You cannot compare arrays with == or `equals`:

```
int[] a1 = {42, -7, 1, 15};
int[] a2 = {42, -7, 1, 15};
if (a1 == a2) {  ... }              // false!
if (a1.equals(a2)) {  ... }         // false!
```

- An array does not know how to print itself:

```
int[] a1 = {42, -7, 1, 15};
System.out.println(a1);             // [I@98f8c4]
```

The `Arrays` class

- Class `Arrays` in package `java.util` has useful static methods for manipulating arrays:

Method name	Description
`binarySearch(`**array**`, `**value**`)`	returns the index of the given value in a *sorted* array (or < 0 if not found)
`copyOf(`**array**`, `**length**`)`	returns a new copy of an array
`equals(`**array1**`, `**array2**`)`	returns `true` if the two arrays contain same elements in the same order
`fill(`**array**`, `**value**`)`	sets every element to the given value
`sort(`**array**`)`	arranges the elements into sorted order
`toString(`**array**`)`	returns a string representing the array, such as `"[10, 30, -25, 17]"`

- Syntax: `Arrays.`**methodName**`(`**parameters**`)`

"Array mystery" problem

- **traversal**: An examination of each element of an array.
- What element values are stored in the following array?

```
int[] a = {1, 7, 5, 6, 4, 14, 11};
for (int i = 0; i < a.length - 1; i++) {
    if (a[i] > a[i + 1]) {
        a[i + 1] = a[i + 1] * 2;
    }
}
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	1	7	10	12	8	14	22

Array reversal question

- Write code that reverses the elements of an array.
  - For example, if the array initially stores:

    ```
    [11, 42, -5, 27, 0, 89]
    ```

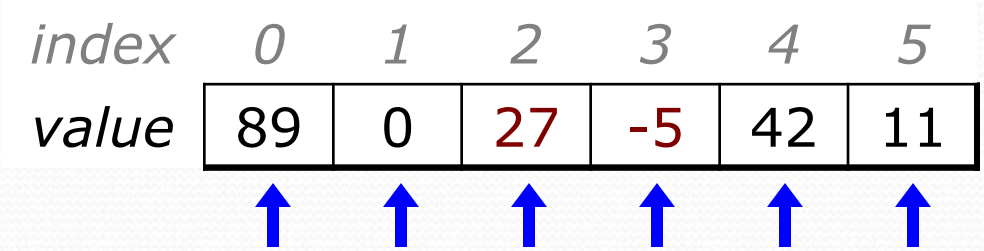
  - Then after your reversal code, it should store:

    ```
    [89, 0, 27, -5, 42, 11]
    ```

    - The code should work for an array of any size.
    - Hint: think about swapping various elements...

Algorithm idea

- Swap pairs of elements from the edges; work inwards:



Swapping values

```
public static void main(String[] args) {
    int a = 7;
    int b = 35;

    // swap a with b?
    a = b;
    b = a;

    System.out.println(a + " " + b);
}
```

- What is wrong with this code? What is its output?

- The red code should be replaced with:

```
int temp = a;
a = b;
b = temp;
```

Array reverse question 2

- Turn your array reversal code into a `reverse` method.
  - Accept the array of integers to reverse as a parameter.

```
int[] numbers = {11, 42, -5, 27, 0, 89};
reverse(numbers);
```

  - How do we write methods that accept arrays as parameters?
  - Will we need to return the new array contents after reversal?

    ...

A `swap` method?

- Does the following `swap` method work? Why or why not?

```
public static void main(String[] args) {
    int a = 7;
    int b = 35;

    // swap a with b?
    swap(a, b);

    System.out.println(a + " " + b);
}

public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Value semantics

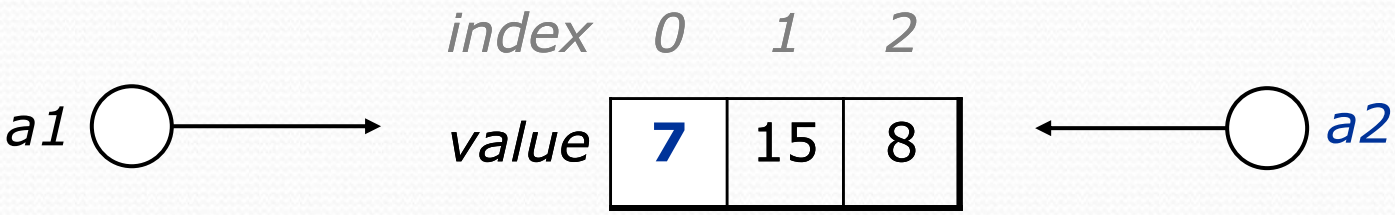
- **value semantics**: Behavior where values are copied when assigned, passed as parameters, or returned.
  - All primitive types in Java use value semantics.
  - When one variable is assigned to another, its value is copied.
  - Modifying the value of one variable does not affect others.

```
int x = 5;
int y = x;        // x = 5, y = 5
y = 17;           // x = 5, y = 17
x = 8;            // x = 8, y = 17
```

Reference semantics (objects)

- **reference semantics**: Behavior where variables actually store the address of an object in memory.
  - When one variable is assigned to another, the object is *not* copied; both variables refer to the *same object*.
  - Modifying the value of one variable *will* affect others.

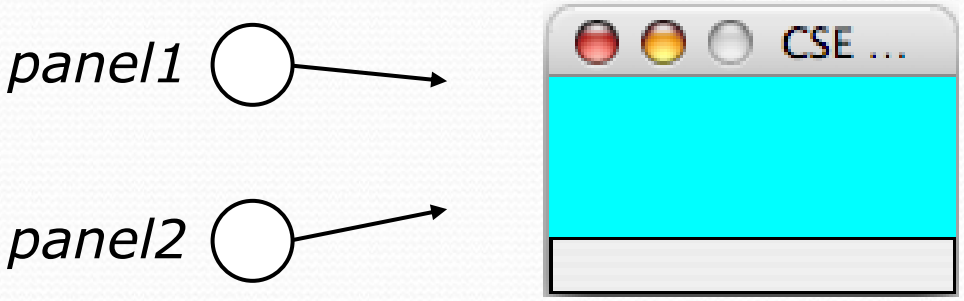
```
int[] a1 = {4, 15, 8};
int[] a2 = a1;              // refer to same array as a1
a2[0] = 7;
System.out.println(Arrays.toString(a1)); // [7, 15, 8]
```



References and objects

- Arrays and objects use reference semantics. Why?
  - *efficiency.* Copying large objects slows down a program.
  - *sharing.* It's useful to share an object's data among methods.

```
DrawingPanel panel1 = new DrawingPanel(80, 50);
DrawingPanel panel2 = panel1;   // same window
panel2.setBackground(Color.CYAN);
```

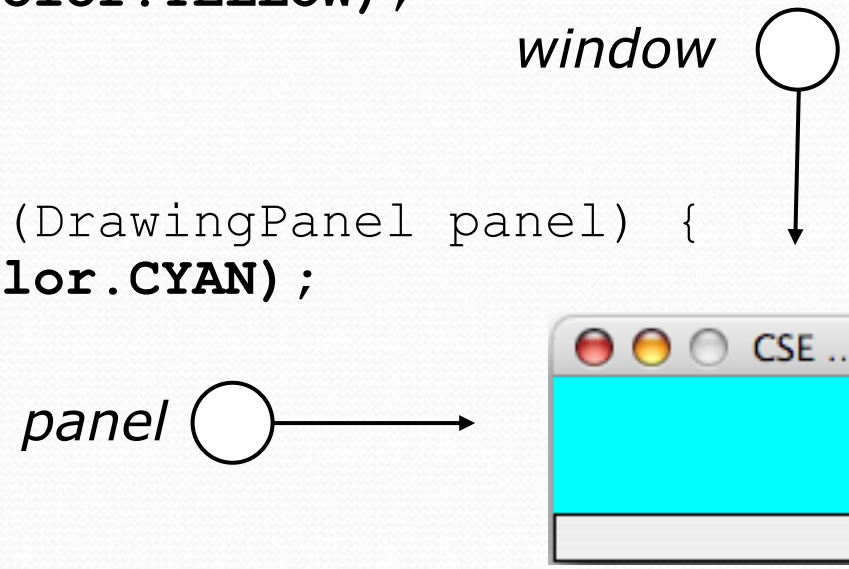


Objects as parameters

- When an object is passed as a parameter, the object is *not* copied. The parameter refers to the same object.
  - If the parameter is modified, it *will* affect the original object.

```
public static void main(String[] args) {
    DrawingPanel window = new DrawingPanel(80, 50);
    window.setBackground(Color.YELLOW);
    example(window);
}

public static void example(DrawingPanel panel) {
    panel.setBackground(Color.CYAN);
    ...
}
```

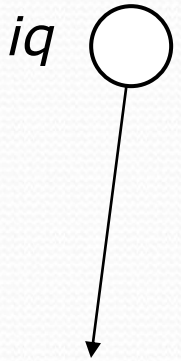


Arrays pass by reference

- Arrays are passed as parameters by *reference*.
  - Changes made in the method are also seen by the caller.

```
public static void main(String[] args) {
    int[] iq = {126, 167, 95};
    increase(iq);
    System.out.println(Arrays.toString(iq));
}

public static void increase(int[] a) {
    for (int i = 0; i < a.length; i++) {
        a[i] = a[i] * 2;
    }
}
```



  - Output:

```
[252, 334, 190]
```