

Welcome to CS2030 Lab 2!

4 February 2022 [10J]



Agenda

1. Login to PE node, showcase ProxyJump.
2. Speed Run Lab 1
3. Lesson proper
4. Try Lab 2; Answer questions

Lab 1 Recap - Imports

```
import java.lang.*;
```

Note that these libraries are automatically imported!

As a result, you do not need to import them in your java file.

Lab 1 Recap - Style - Use of Spaces

Use spaces after operators and punctuation marks

Good

```
int x = 1 + 2;  
String.format("%s", "hi");
```

Bad

```
int x=1+2;  
String.format("%s","hi");
```

Lab 1 Recap - Style - Variable Naming

Use more descriptive variable names

```
public class Circle {  
    private final Point p; // Not advisable  
    private final Point midpoint; // Better variable name  
}
```

Spell out variables in full (e.g. maxDiscCoverage instead of mdc)

The convention in Java is to use camelCase (helloWorld instead of hello_world)

Lab 1 Recap - Style - Variable Naming

```
public class Circle {  
    private final double radius;  
    public Circle (double r) {  
        radius = r;  
    }  
    // OR  
    public Circle (double radius) {  
        this.radius = radius; // same name require `this` keyword.  
        // [Enrichment: Pointers]  
    }  
}
```

Lab 1 Recap - Style - if Statements

Use braces after single line `if` statements

```
// works but not advisable; could result in bugs if not careful  
if (condition)  
    // some code
```

```
if (condition) {  
    // code here;  
}
```

Lab 1 Recap - Style - Boolean Expressions

```
// Redundant if-else statement
if (booleanMethod(parameter) == true) { // if true == true???
    return true; // return true if true == true
} else {
    return false; // return false if false ≠ true
}
```

```
// Better
return booleanMethod(parameter); // Just return the result
```


Lab 1 Recap - Style - Line Wrapping

- Wrap lines instead of letting them get too long!
- CS2030 sets **100? characters** as the line length limit.
- It is usually appropriate to wrap lines **after operators** or at appropriate junctures for Strings (e.g. after a full-stop)

Lab 1 Recap - Style - String.format()

- `String.format()` can be used to format entire strings instead of being called multiple times.
- The following lines return the same `String` (assume that `x` and `y` are `int` variables)

```
return "Coordinates: " + String.format("(%d, %d)", x, y);
```

```
return String.format("Coordinates: (%d, %d)", x, y);
```

- Use `%s` as the placeholder for a `String`

Recall Lab 1 used `%f` for `double/float`!

Lab 1 Recap - Style - Variable Initialization

Not recommended

```
int numberOfPoints;  
// some other code  
numberOfPoints = sc.nextInt();
```

Better — declare and initialize variables within the scope that they are needed

```
// Other code  
int numberOfPoints = sc.nextInt(); // Declare and initialize here
```

Lab 1 Recap - Style - Variable Initialization

Not Recommended

```
int i, j, k; // i, j, k all outside the scope of the for loops
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        // Other code
    }
}
```

Better — declare and initialize variables within the scope that they are needed

```
for (int i = 0; i < n; i++) { // i in the scope of this for loop
    for (int j = 0; j < n; j++) { // j in the scope of this for loop
        // Other code
    }
}
```

Lab 1 Recap - Style - Arranging Methods

👎 **Bad**

```
if (!isUnusualCase) {  
    if (!isErrorCase) {  
        start();  
        process();  
        cleanup();  
        exit();  
    } else {  
        handleError();  
    }  
} else {  
    handleUnusualCase();  
}
```

👍 **Good**

```
if (isUnusualCase) {  
    handleUnusualCase();  
    return;  
}  
  
if (isErrorCase) {  
    handleError();  
    return;  
}  
  
start();  
process();  
cleanup();  
exit();
```

Lab 1 Recap - Style - CS2030 Style Guide

1. [Original style guide](#)
2. [Github Wiki](#)

Lab 1 Recap - Style - Modularisation

- Break up length methods into shorter ones that ultimately have the same functionality.
- If need be, it is also recommended to abstract out lines of code into helper functions even if it's only being used once.
- This makes your code easier to debug and is an important part of designing code.

Lab 1 Recap - Style - Modularisation

Example of modularisation:

```
boolean containsPoint(Point q) {  
    return center.distTo(q) ≤ radius;  
}
```

You can make use of the above method as necessary instead of repeatedly calling `centre.distTo(q) ≤ radius`.

Lab 1 Recap - Style - Data Hiding

- Declare instance and class variable as private (except for special cases like constants).
- This is part of *encapsulation* and hides data that only the class or instance needs to know about.

Lab 1 Recap - Style - Immutability

- Make objects immutable so that they cannot be tampered with; especially important for reference types
- Use the `final` keyword for instance variables and use constructors to get new object instances.
- No setters (e.g. a `void setX(double x)` method)! We do not change the state of immutable objects once they are created.

static keyword

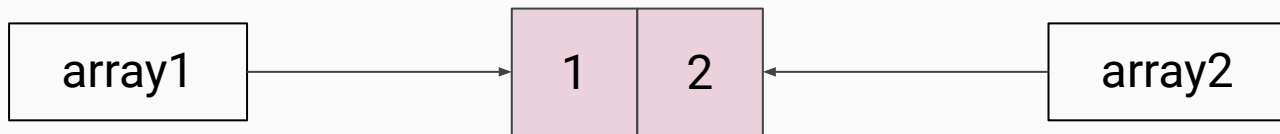
- The `static` keyword is used to declare class level attributes.
- One copy of each static variable (attribute) is stored across all instances of a class (instead of one copy per instance).

```
class Dog {  
    private static String sound = "woof";  
}
```

- For example, in the above class, the "woof" sound is shared across any instance of a Dog.

Lab 2: Abstraction and Encapsulation; Immutable List

Shallow Copying



```
int[] array1 = {1, 2};
```

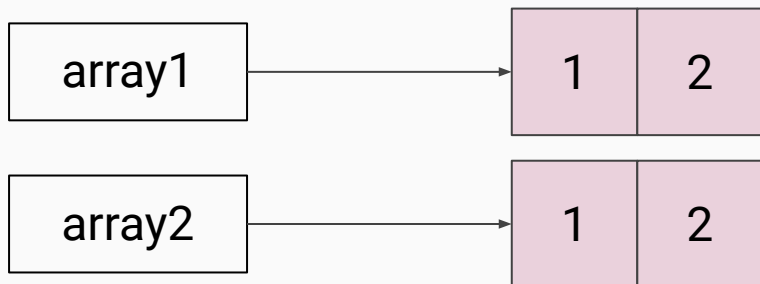
```
int[] array2 = array1;
```

Note that a shallow copy is created! (**array1** and **array2** point to the same array)

Modifying the contents from either variable will also cause the other one to be affected

This applies to any reference type, which includes classes

Deep Copying



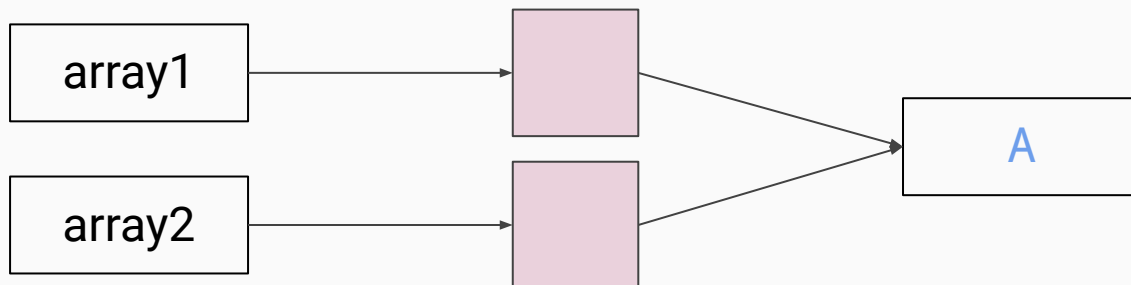
Need to copy over the individual contents from the array to make a deep copy (i.e. exact copy of the array, without pointing to the same memory address)

```
int[] array1 = {1, 2};
```

```
int[] array2 = new int[2];
```

```
for (int i = 0; i < array1.length; i++) {  
    array2[i] = array1[i];  
}
```

Deep Copying



Since **A** is a reference type, the code snippet in the previous slide will NOT create a deep copy of the array (it creates a shallow copy instead)

You should iterate over **array1** and create copies of the **A** instances

This applies to any reference type, which includes N-dimensional arrays