COMP 210: Data Structures

Review of References

In Java, a **reference** is a variable that stores the memory address of an object, not the object itself. Think of it as a pointer or a handle to an object, similar to how a street address pornts to a specific house. You use this reference to access the object's methods and fields. Actual objects are created with the `new` keyword, and can then be assigned to a reference variable. If a reference variable has not yet been assigned an object, it is a `null` reference.

```
MyObject obj;           // Creates a null reference, no object
obj = new MyObject();  // Creates an object; obj now refers to it
obj.myMethod();
obj.myField = 10;
```

Primitive Types vs. References

It's crucial to understand the difference between how Java handles **primitive types** (like `int`, `double`, `boolean`) and **reference types** (all classes, including `String`). When a variable of a primitive type is defined, space in memory is set aside for a value of that type. On the other hand, when a variable of a class type is defined, that merely creates space in memory for a *reference* to an object of that type, but not for the object itself. No space is created for the actual object until you use the `new` keyword, invoking the constructor and initializing the object.

- **Primitive Types:** A variable directly stores the value.

  ```
  int i;     // Space for an int is set aside, but it has no initial value
  int a = 5; // Space for an int is set aside, and it is initialized to 5
  ```

- **Reference Types:** A variable stores a reference (think of a memory address) or `null`. The expression `new MyObject()` creates the actual object in memory (specifically, on the **heap**).

  ```
  MyObject obj1;                    // Creates a null reference, no object
  MyObject obj2 = new MyObject();  // Creates an object and a reference to it
  ```

Copied Values vs. Aliases

- **Primitive Types:** When you assign one primitive to another, the value is copied.

  ```
  int a = 5; // Space for an int is set aside, and it is initialized to 5
  ```

```
int b;      // Separate space is created for b
b = a;      // The value 5 is copied from 'a' to 'b'.
a = 10;     // 'b' remains 5.
```

- **Reference Types:** When you assign one reference variable to another, **the reference itself is copied, not the object**. This means both variables point to the *exact same object* in memory. Two (or more) references to the same object are known as *aliases* of each other. In the example below, if you modify the object using variable `obj2`, the change will be visible through `obj1` because they are both looking at the same thing.

```
MyObject obj1 = new MyObject();
MyObject obj2 = obj1; // The reference (memory address) is copied.
                      // Both obj1 and obj2 now point to the same object.
```

Null References

A reference variable holds the special value `null` when it doesn't point to any object at all.

```
MyObject nullObj1;
MyObject nullObj2 = null;
```

If you try to call a method or access a field on a `null` reference, you'll get a `NullPointerException` at runtime. This is a very common error for students to encounter. It's a key part of memory management in Java and is a frequent topic in debugging.

Alyce Brady, based on an early draft by Gemini, 26 September, 2025.