References in Java
Variables, Before

A variable is like a "box" inside of which a piece of data is placed.
Variables, Now

A variable is a **named portion of memory** that contains data of a particular type **and has a fixed size**.

- For *primitive types*, data can be stored directly inside of a variable.
- For *object* or *reference types*, data is stored in a separate portion of the computer's memory. Instead of storing the data directly, variables of these types tell us how to find the data elsewhere!

Let's drill down.
Data Types in Java

Primitive Data Types

- byte, short, int, long, float, double, boolean, char
- Primitive types variable works like a box that can store a single value
- Example: `int num = 42;`
Data Types in Java

References

- Reference variables do not store simple values directly!
- Reference variables store a reference to some object
  - Literally: an address that describes where the object is stored in the computer's memory.
- The object that the reference refers to is known as its pointee
Why Make the Distinction?

All variables have the same fixed size.*

- Primitive types have a fixed "size". For example, all \( \text{int} \) values can be expressed using \( \leq 32 \) bits (ones and zeroes)
- Objects & Records can be huge, consisting of multiple primitive & object values! They couldn't possibly be squeezed into any fixed size container.

*This is only a little bit of a lie 😊
An "Employee" Class

Two instance variables, name and salary, along with a simple constructor. (Assume setters & getters written beneath).

```java
public class Employee {
    private String name;
    private int salary;
    public Employee(String name, int salary) {
        this.name = name;
        this.salary = salary;
    }
    ... // the rest omitted for space
}
```
A Picture of the Reference

If we create a new `Employee` object `empRef = new Employee("john", 1000)`; we can represent the memory abstractly with the following diagram:

- `Employee` object: `John 1000`
- `empRef`: Reference variable. The current value is a reference to the `Employee` object in the box above.

A simple `Employee` object. The current value is the string `"John"` for the name and `1000` for the salary. This object also plays the role of pointee for `empRef`. 
**Rules for Drawing References**

1. Variables are always drawn as boxes with names outside of the boxes.
   
i. Variables that store primitive types are drawn as boxes that contain the values directly.

   ii. Variables that store reference (object) types are drawn as boxes that contain arrows representing references.

2. References are drawn as arrows **from variables to object data**.

3. Object data is drawn as a box containing the fields (instance variables) for the object.
   
i. Instance variables that store reference types will feature arrows to other variables.

4. A **null** value is always represented as a box with a line crossed through it.
Dereferencing

• Accessing the value of the pointee for some reference variable
  ○ Variables storing primitive types cannot be dereferenced—why?

• Is done with the dot operator (\texttt{.}) to access a field or method of an object

• Symbolically, this is like following the arrow stored in a reference variable to the object data it points to.
Dereferencing

String myName = empRef.getName() dereferences empRef to call the getName method for that object.

A simple Employee object. The current value is the string 'John' for the name and 1000 for the salary. This object also plays the role of pointee for empRef.

A reference variable. The current value is a reference to the Employee object in the box above.
Referencing & \texttt{null}

A reference must be assigned a pointee before dereference operations will work.

- Not assigning a pointee to a reference will cause a \texttt{NullPointerException}
- \texttt{null}: special reference value that encodes the idea of "pointing to nothing", which is also the initial value of references
**Reference Assignments**

An assignment of one reference variable to another makes them the same pointee.

```java
Employee empRef = new Employee("john", 1000);
Employee second = empRef;
second == empRef; // true!
```

A second pointer is initialized with the assignment `second = empRef`. This causes `second` to refer to the same pointee as `empRef`. 
Sharing

Two references which both refer to a single pointee are said to be **sharing**, and each is an **alias** for the other.

A second pointer is initialized with the assignment `second = empRef`. This causes `second` to refer to the same pointee as `empRef`. 
**Shallow and Deep Copying**

Shallow copy (of a reference) is achieved through sharing

Deep copy creates a new copy of the pointee
Shallow and Deep Copying Example

Shallow copy: copies only the reference

```
Employee second = empRef;
```

Deep copy: constructs a new object and copies the values of the other object's instance variables.

```
Employee deepCopy = new Employee(empRef.getName(), empRef.getSalary());
```
Shallow and Deep Comparing

Double equals (==) checks if two reference variables are referencing the same object
- Returns true for shallow copies
- Returns false for deep copies

The equals method checks if the values (data fields) of the two objects are the same
- Returns true for shallow copies
- Returns true for deep copies
public class Employee {
    private String name;
    private int salary;
    public Employee(String name, int salary) {
        this.name = name;
        this.salary = salary;
    }

    public boolean equals(Employee emp) {
        if (this == emp) {
            return true;
        } else {
            return this.name == emp.name &&
                this.salary == emp.salary;
        }
    }
}