References (pointers) in Java
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 variable **does not store a simple value directly**
  - Reference variable **stores a reference to some object**
  - The object that the reference refers to is known as its **pointee**
Data Types in Java

- References
  - Example:

  ```java
  Employee empRef = new Employee("john", 1000);
  ```

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.
Dereferencing

- Dereferencing:
  - Accessing the value of the *pointee* for some *reference* variable
  - Is done with the "*dot" (.)" operator to access a field or method of an object
Dereferencing

- Dereferencing:
  - dereferencing `empRef` in the previous slide gives back its pointee, the Employee object
  - `String myName = empRef.getName();`

  dereferences `empRef` to call the `getName` method for that object
Referencing

- **A reference must be assigned a pointee** before dereference operations will work.
- Not assigning a pointee to a reference will cause a `NullPointerException`.
- **null**: special reference value that encodes the idea of "points to nothing".
  - Initial value of references
Reference Assignments

- An assignment (using equals) of one reference to another makes them point to the same pointee

  - Example:

  Employee empRef = new Employee("john", 1000);
  Employee second = empRef;

  After the assignment, testing for second == empRef would return true
Sharing

- Two references which both refer to a single pointee are said to be sharing.
- Each (shared reference) 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

Employee firstEmployee = new Employee("Sam");
Employee shallowCopyEmployee = firstEmployee;
shallowCopyEmployee.setName("John");
System.out.println(firstEmployee.getName());
Employee secondEmployee = new Employee("Patrice");
Employee deepCopyEmployee = new Employee();
deepCopyEmployee.setName(secondEmployee.getName());
deepCopyEmployee.setName("John");
System.out.println(secondEmployee.getName());
Shallow and Deep Comparing

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

- **The equals method checks** if the values (data fields) of the two objects are the same
  - Returns true for shallow copies
    - firstEmployee.equals(shallowCopyEmployee)
  - Returns true for deep copies
    - secondEmployee.equals(deepCopyEmployee)