

# 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;`



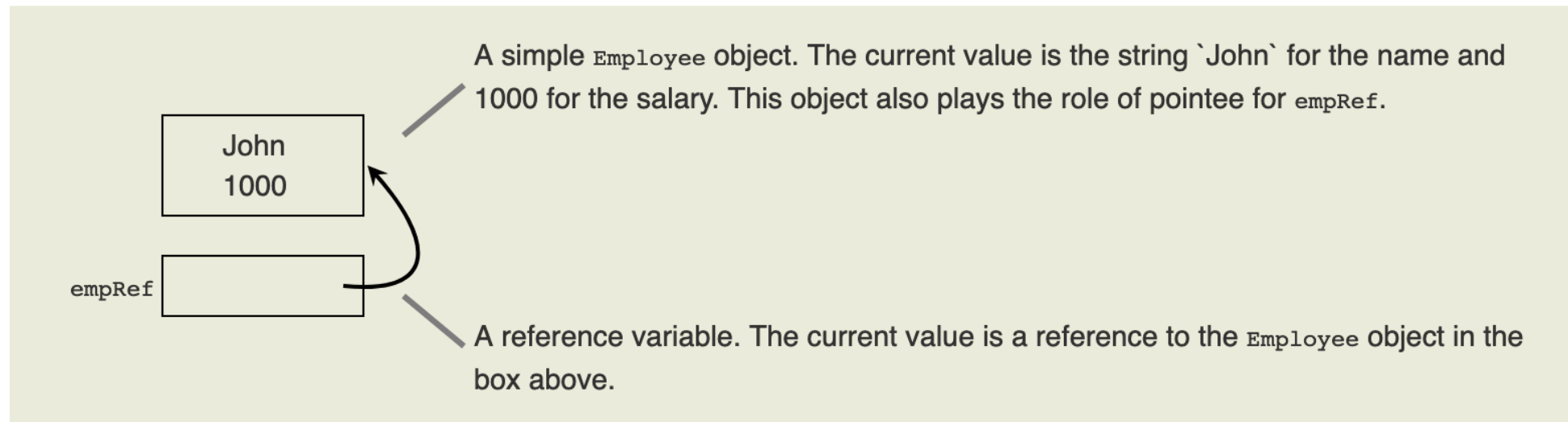
# Data Types in Java

- 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**
- Example:

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

```
public class Employee {  
    private String name;  
    private int salary;  
    public Employee(String name, int salary) {  
        this.name = name;  
        this.salary = salary;  
    }  
    ...  
}
```

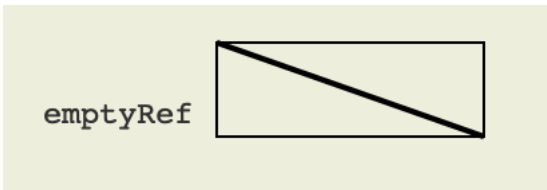


# Dereferencing

- 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
  - Examples:
    - 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

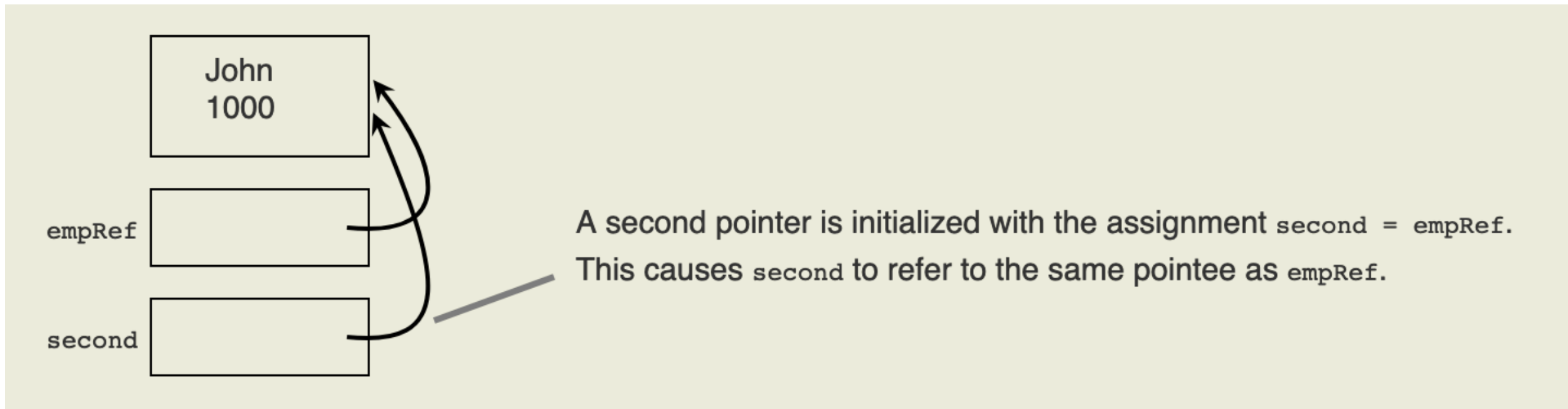
- **A reference must be assigned a pointee** before dereference operations will work.
- Not assigning a pointee to a reference will cause a **NullPointerException**
- **null**: special reference value that encodes the idea of "*points to nothing*".
  - Initial value of references



# Reference Assignments

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

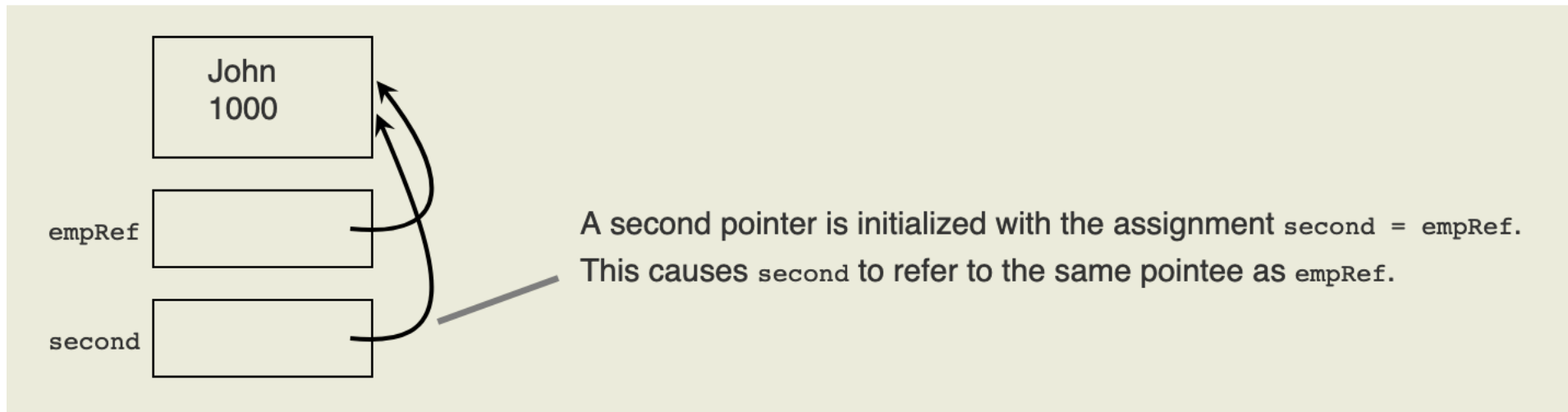
```
Employee empRef = new Employee("john", 1000);  
Employee second = empRef;
```



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

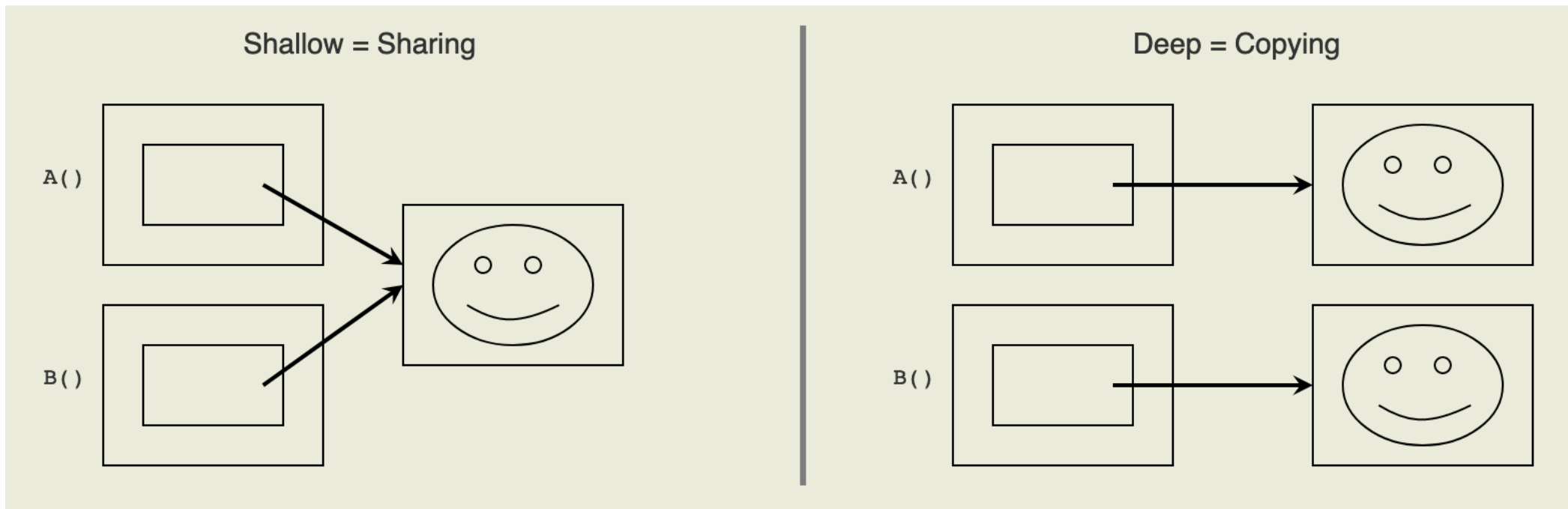
# Sharing

- Two references which both refer to a single pointee are said to be **sharing**
- Each (shared reference) is an **alias** for the other



# 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; ← alias  
shallowCopyEmployee.setName("John"); ← update alias  
System.out.println(firstEmployee.getName()); ← // John  
Employee secondEmployee = new Employee("Patrice");  
Employee deepCopyEmployee = new Employee(); ← deep copy  
deepCopyEmployee.setName(secondEmployee.getName()); ← deep copy  
deepCopyEmployee.setName("John"); ← update deep copy  
System.out.println(secondEmployee.getName()); ← // Patrice
```

# 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)`

```
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;
    }
    ...
}
```