# Object References

CPSC 1181 – O.O.

Jeremy Hilliker

Summer 2017

Langara.
THE COLLEGE OF HIGHER LEARNING.

# Objectives

- Learn how *objects* are stored
  - How it differs from *primitive* types

- Garbage Objects
- Garbage Collection

- Memory Leaks

# Storing Variables

- A variable can be visualized as a box.
- Storing a *primitive* (non-object) variable:

```
int i;
```


- Assignment:

```
i++;
```

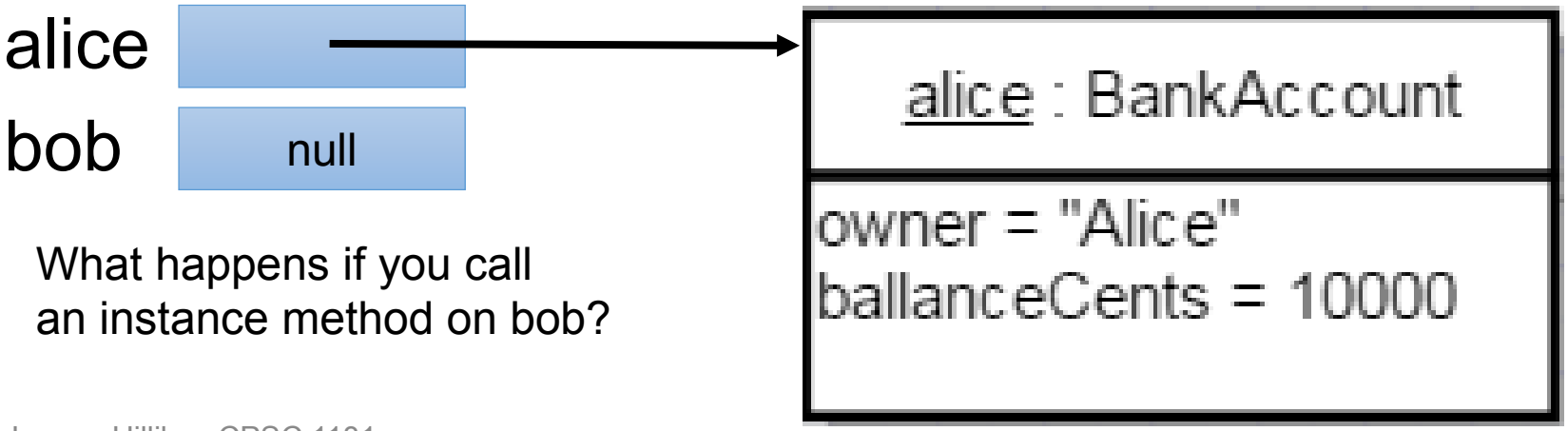

```
i = 15;
```


```
i += 10;
```

# Storing Objects

- In java, objects are not stored “in” a variable.
- The variable holds a ***reference*** (a/k/a pointer) to an object (returned by the *new* operator)

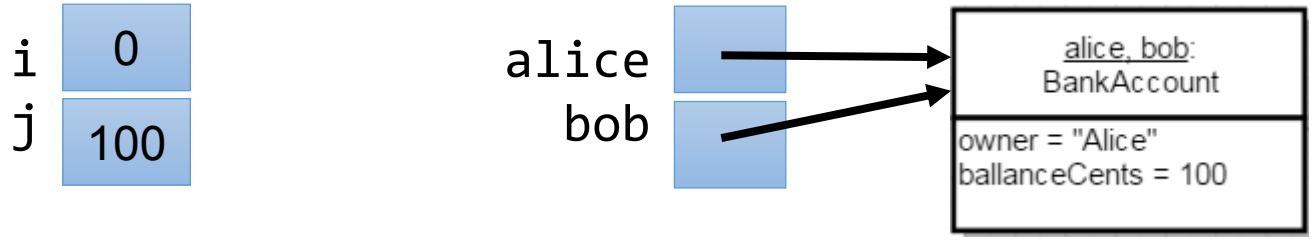
```
1   BankAccount alice = new BankAccount("Alice");
2   alice.deposit(10000);
3   BankAccount bob = null;
4   bob.deposit(100);    // does this make sense?
5
```




What happens if you call an instance method on bob?

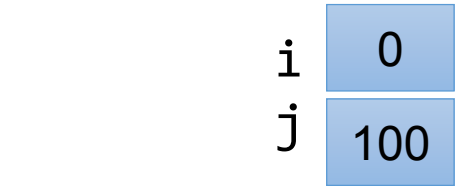
# Aliases

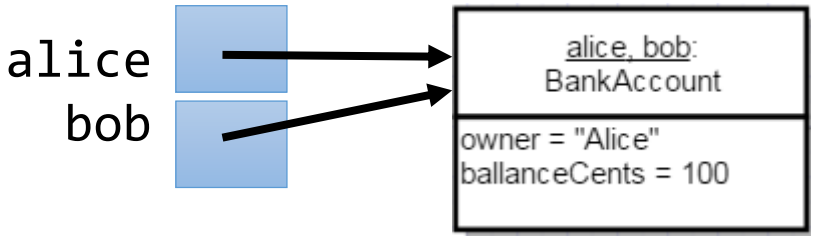
- Different variables can refer to the same object!



Mover.java | Alias.java

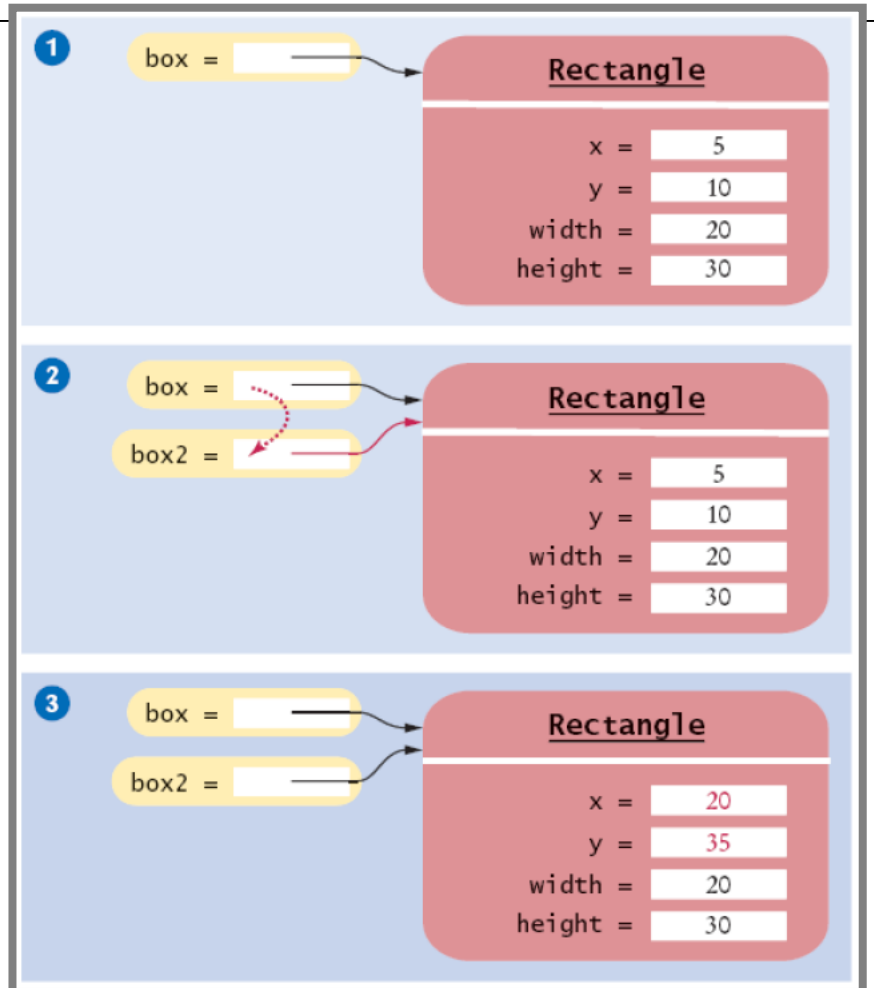
```java
public class Alias {

  public static void main(String[] args) {
    int i = 10;
    int j = i;
    System.out.println("i: " + i);    // i: 10
    System.out.println("j: " + j);    // j: 10
    j = 100;
    System.out.println("i: " + i);    // i: 10
    System.out.println("j: " + j);    // j: 100

    BankAccount alice = new BankAccount("Alice");
    BankAccount bob = alice;
    System.out.println("alice: " + alice.getBallanceInCents());
    System.out.println("  bob: " + bob.getBallanceInCents());
    // alice: 0 \n bob: 0
    bob.deposit(100);
    System.out.println("alice: " + alice.getBallanceInCents());
    System.out.println("  bob: " + bob.getBallanceInCents());
    // alice: 100 \n bob: 100

    bob = new BankAccount("Bob");
    bob.deposit(500);
    System.out.println("alice: " + alice.getBallanceInCents());
    System.out.println("  bob: " + bob.getBallanceInCents());
    // alice: 100 \n bob: 500
  }
}
```



# Copying Object References

```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
box2.translate(15, 25);
```



# Keywords: final

- must be set

```java
public class Final_Unset {
    private final int unset;
    public Final_Unset() {
    }
}
```

```
$ javac Final_Unset.java
Final_Unset.java:4: error: variable unset might not have been initialized
    }
    ^
1 error
```

- Cannot change

```java
public class Final_Changed {
    private final int changed;
    public Final_Changed() {
        changed = 0;
    }
    public void increment() {
        changed++;
    }
}
```

```
$ javac Final_Changed.java
Final_Changed.java:7: error: cannot assign a value to final variable changed
        changed++;
        ^
1 error
```

```
$ javac Final_Changed.java
Final_Changed.java:5: error: variable changed might already have been assigned
        changed++;
        ^
1 error
```

# Keywords: static

- there is only one copy

StaticDemo.ja

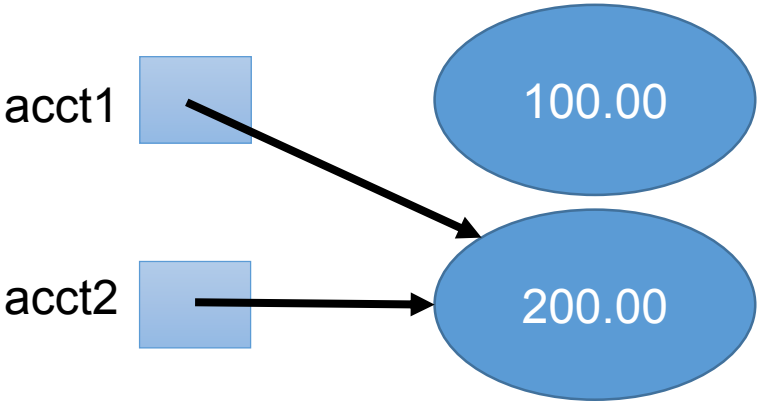
```java
public class StaticDemo {
    private static int globalCount = 0;

    public void inc() {
        globalCount++;
    }
    public int getCount() {
        return globalCount;
    }

    public static void main(String[] args) {
        StaticDemo s1 = new StaticDemo();
        StaticDemo s2 = new StaticDemo();
        System.out.printf("s1: %d, s2: %d\n",
            s1.getCount(), s2.getCount());

        s1.inc(); s1.inc(); s2.inc();
        System.out.printf("s1: %d, s2: %d\n",
            s1.getCount(), s2.getCount());
    }
}
```

```
$ javac StaticDemo.java ; java StaticDemo
s1: 0, s2: 0
s1: 3, s2: 3
```

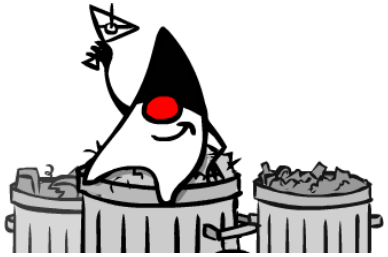
# Garbage

- When an object is no longer referenced by any variables, it is said to be ***orphaned*** and ***garbage***. [That's harsh!]

```
account acct1 = new Account(100.00);
account acct2 = new Account(200.00);
acct1 = acct2;
```

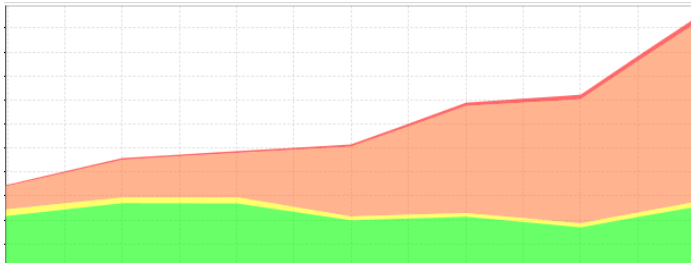


# Garbage Collection



- One of java's main features is that it automatically reclaims the memory occupied by orphaned/garbage objects.
  - This is knows as ***garbage collection (or GC)***.
  - The tool that does it (at runtime) is called the ***garbage collector (or also GC)***.
- Why does this matter?
  - YOU don't have to manage memory.
  - The JVM does it for you.
    - See: Fred Brooks Jr. (1987). *"No Silver Bullet—Essence and Accidents of Software Engineering"*. Computer. 20 (4): 10. doi:10.1109/MC.1987.1663532
  - There is a performance cost.
    - GC has to regularly evaluate reachability of objects.

# Memory Leaks



- Other languages rely on the programmer to *explicitly* release memory.
  - More efficient, but MUCH more error prone.
- Failing to reclaim spaced used by garbage wastes memory (called a ***memory leak***)
- Over time, a program with a memory leak can exhaust its available memory
- But releasing memory prematurely can cause values in use to be overwritten, leading to failure.
- Java is not immune to memory leaks.
  - Keeping references to old objects that you don't need any more.
  - Objects that you do need holding references to objects that you don't need. [Imagine an "undo" history.]

# Recap

- Objects are stored in a variable as a reference to the actual object
  - Object can be reference by many variable names
  - Changes made through one name effect the state of the object for all names
- Orphaned objects are called “garbage”
  - Their memory space is reclaimed by the “garbage collector”
- Memory can be leaked by failing to orphan no-longer needed objects