Primitive and Object Variables

15-110 Summer 2010

Margaret Reid-Miller

Primitive type **variables** **hold** *values*

(e.g., int, double, char)

Primitive Types

- Variables of primitive types name a storage location in memory in which we can store a value.

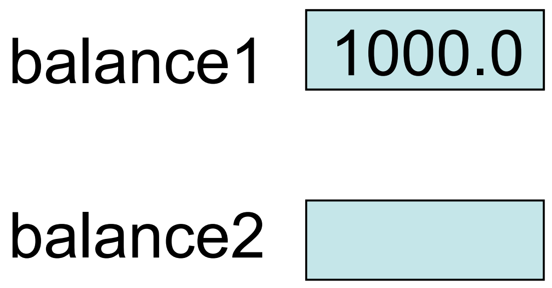
```
double balance1 = 1000.0;
```

balance1 $\boxed{1000.0}$

Primitive Types

- Simply declaring a local variable does not provide a value for the storage location. You cannot use the variable until it is assigned a value.

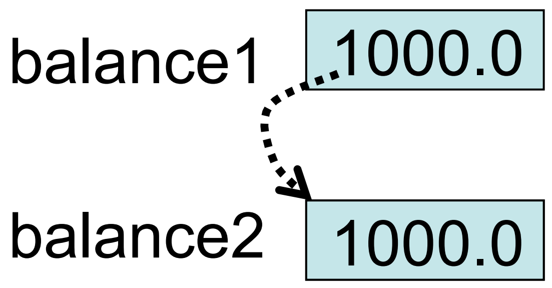
```
double balance1 = 1000.0;
double balance2;
```



Primitive Types

- Assigning the value of the one variable to another copies the value:

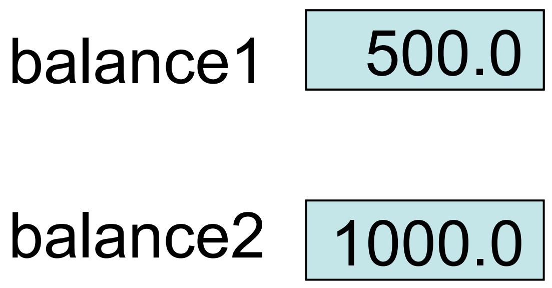
```
double balance1 = 1000.0;
double balance2;
balance2 = balance1;
```



Primitive Types

- You can assign a new value to a variable. The previous value is lost.

```
double balance1 = 1000.0;
double balance2;
balance2 = balance1;
balance1 = 500;
```

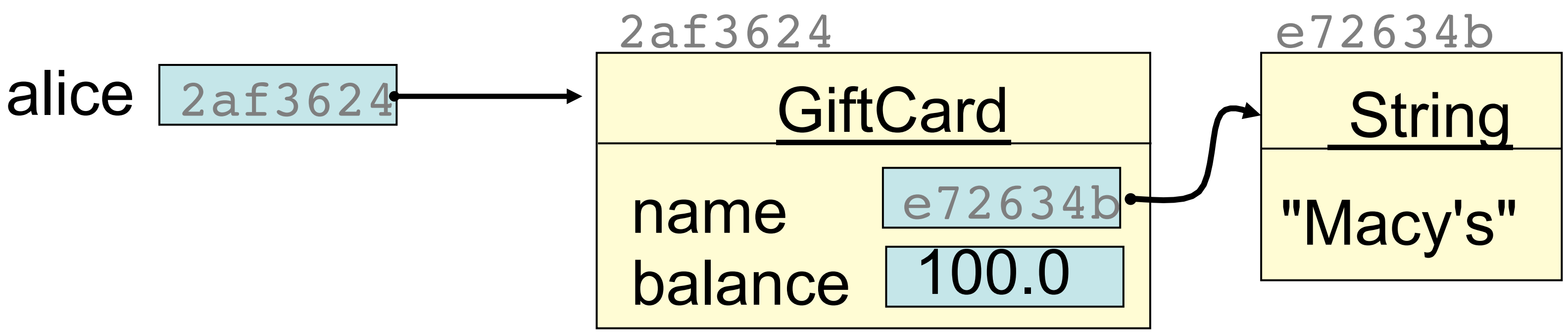


Object type variables hold *references* to objects.

Object Types

- Alice gets a $100 gift card from Macy's.

```
GiftCard alice = new GiftCard("Macy's", 100.0);
```

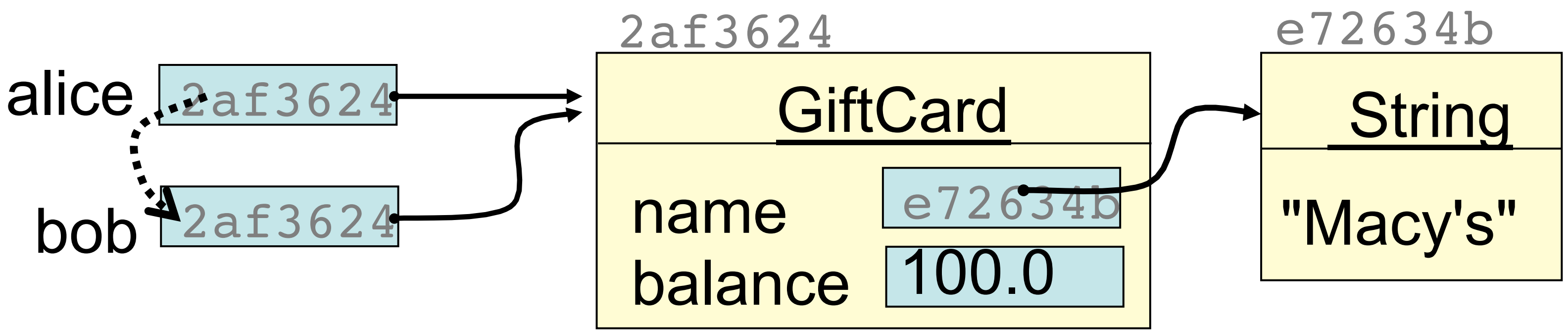


- Object type variables also name a memory location. But the memory is too small to hold an object. It can only hold a reference (pointer) to the object.

Object References

- Bob takes Alice's gift card.

```
GiftCard alice = new GiftCard("Macy's", 100.0);
GiftCard bob = alice;
```

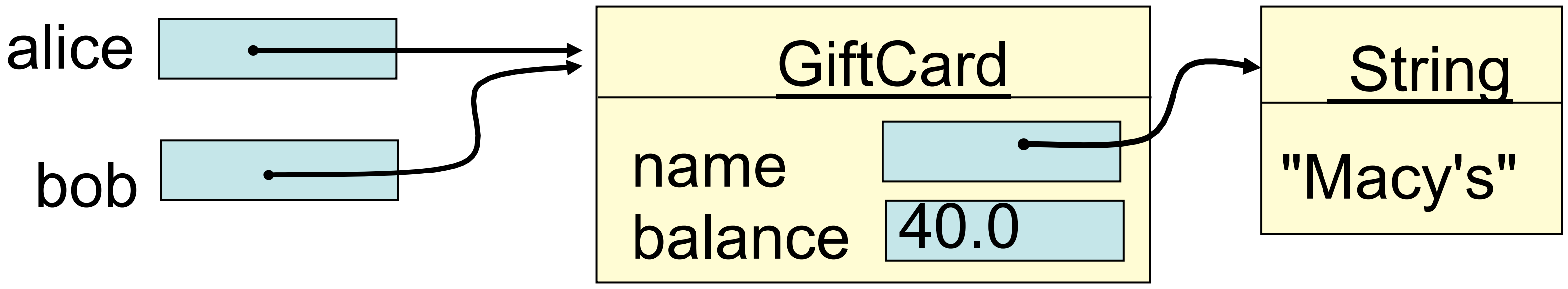


Assigning alice to bob copies the **reference** from alice to bob. We say bob is an ***alias*** for alice.

Object References

- Bob spends $60. Alice can see that her card now has only $40.

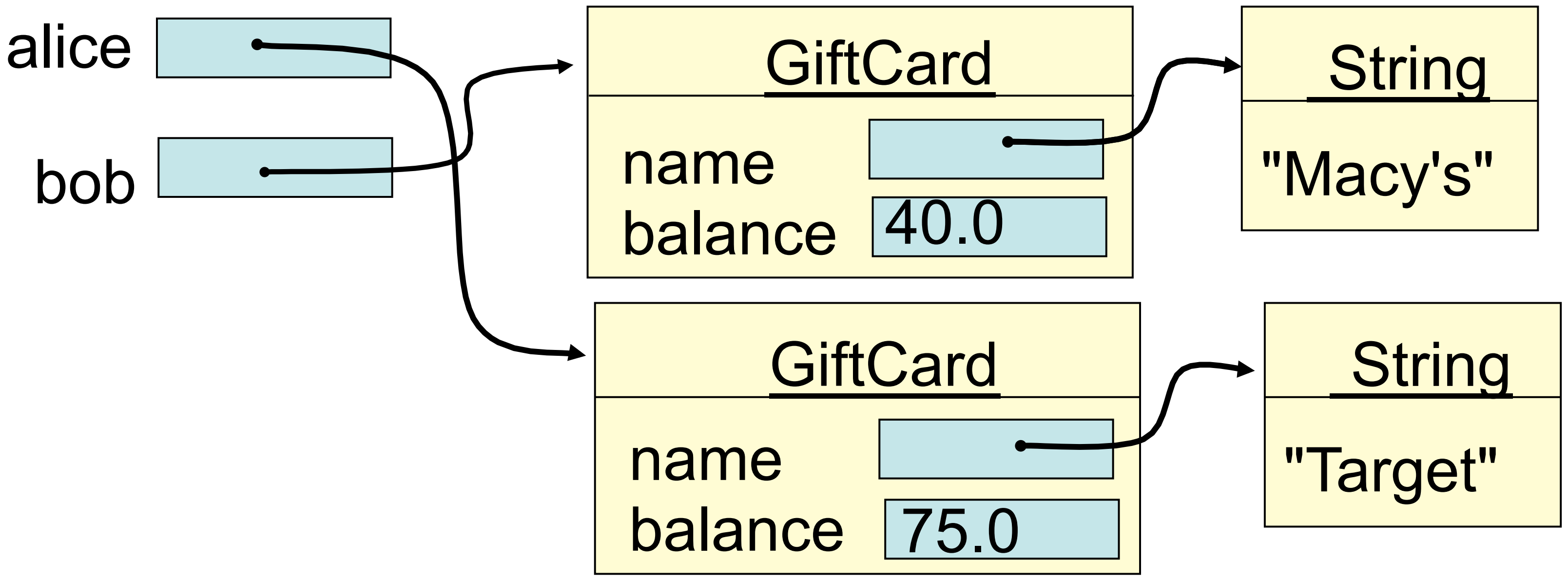
```
GiftCard alice = new GiftCard("Macy's", 100.0);
GiftCard bob = alice;
bob.buyGoods(60.0);
```



Object References

- Alice buys a $75 gift card from Target.

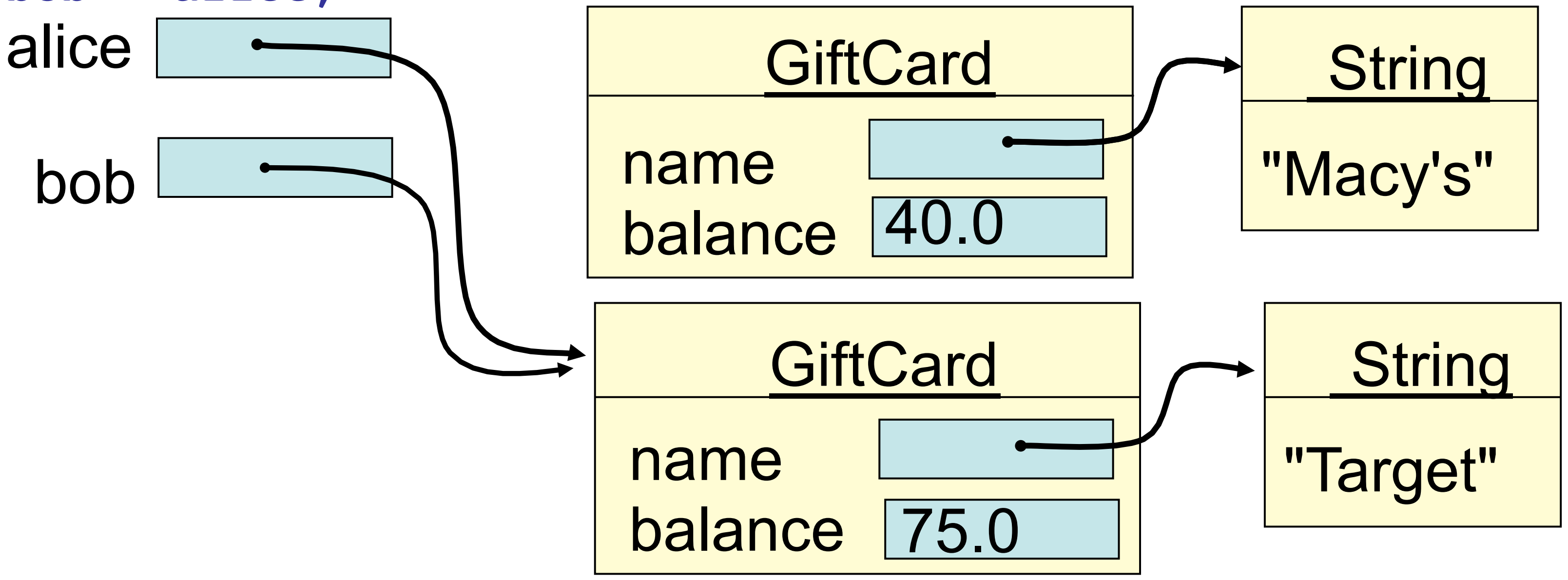
```
GiftCard alice = new GiftCard("Macy's", 100.0);
GiftCard bob = alice;
bob.buyGoods(60.0);
alice = new GiftCard("Target", 75.0);
```



Object References

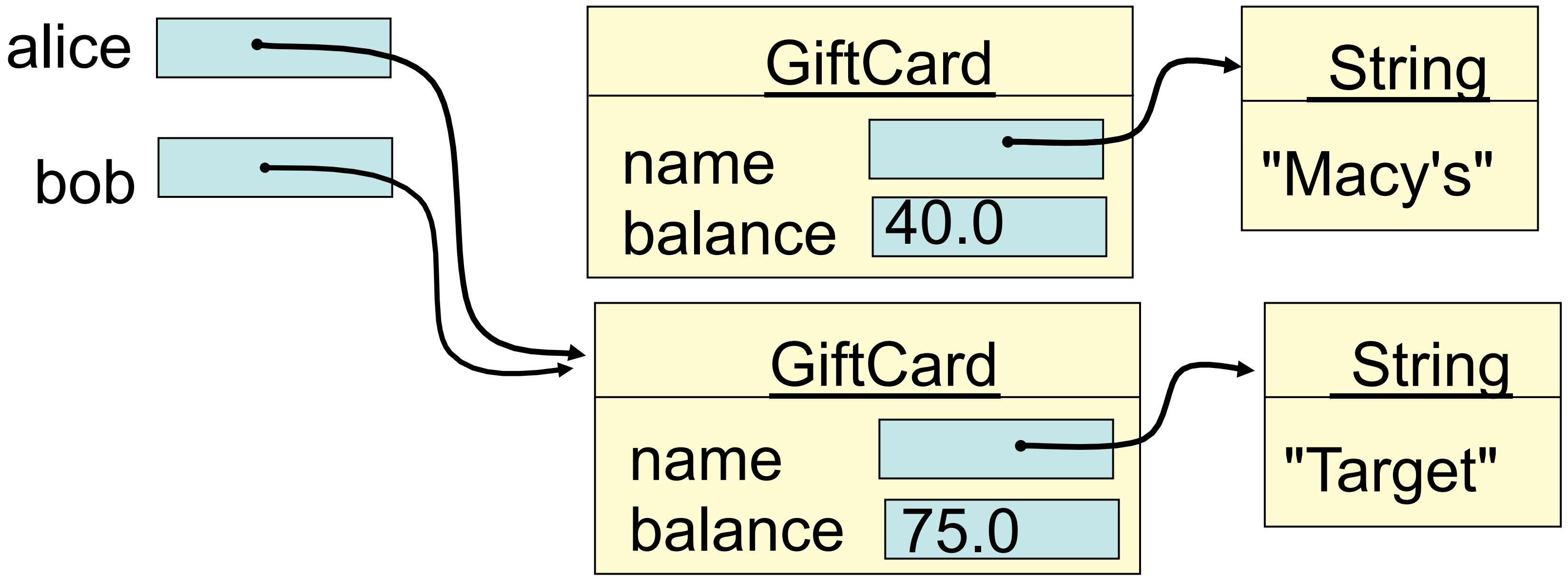
- Bob takes Alice’s Target card and loses Macy’s card.

```
GiftCard alice = new GiftCard("Macy's", 100.0);
GiftCard bob = alice;
bob.buyGoods(60.0);
alice = new GiftCard("Target", 75.0);
bob = alice;
```



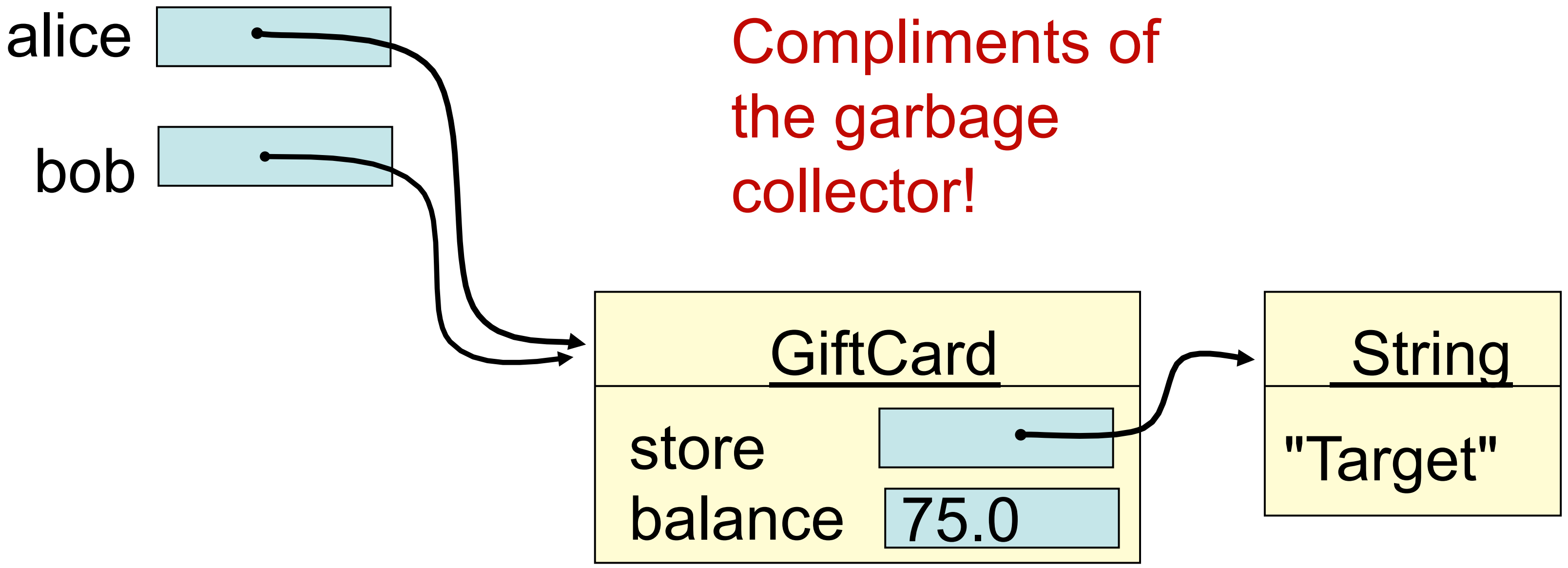
Garbage

- But now the program cannot access the Macy’s gift card any more.
- Such objects are considered “garbage” because they still take up memory space.



Garbage Collector

- To reclaim the memory space, Java has a garbage collector that periodically “cleans up” memory so that it can be reused.
  - Without it, programs can easily have a “memory leak” if not programmed with extreme care.

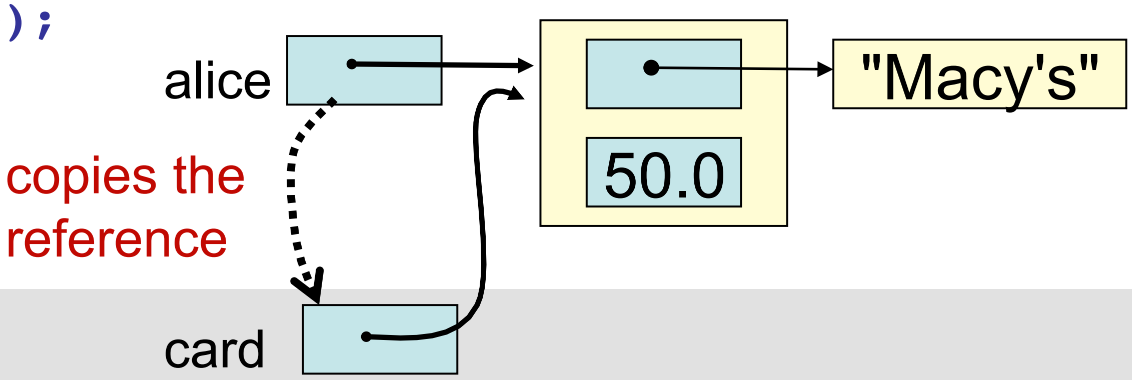


Compliments of the garbage collector!

Object Types as Parameters

- An object type parameter is an alias of the argument.

```
GiftCard alice = new GiftCard("Macy's", 50.0);
goShopping(alice);
```

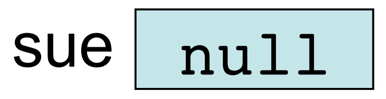


```
public static void goShopping(Giftcard card) {
   while (card.getBalance > 0) {
        card.buyGoods(10.0)
     }
}
```

The `null` Pointer

If we do not instantiate an object, the variable holds a special value `null` that represents a nonexisting object.

```
GiftCard sue;
```



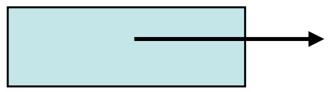
If we try to use the variable as an object, we get a `NullPointerException` at runtime.

```
sue.addMoney(30);
```

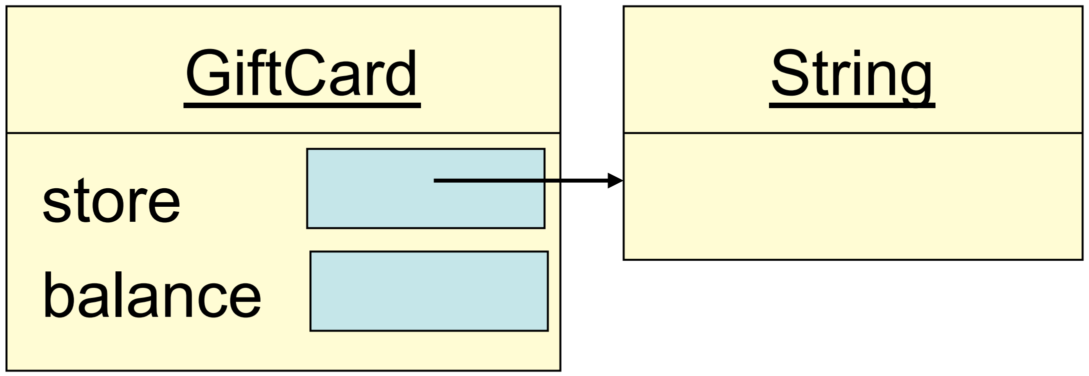
Tip: Methods that have object parameters should test whether the parameter is `null` before using it!

The `equals` Method Revisited

- The == operator tests whether two variables have the same **references** (identity);



- Whereas the `equals` method tests whether two variables refer to objects that have the same **state** (content).


GiftCard
store
balance
String