

# Same Bucket

Methods: passing and returning references of an object



# Two sticky notes, one house

PAUSE AND TRY

**You copy your address onto a sticky note for a friend. What happens in each case?**

**SITUATION 1**

Your friend goes to the address on the note and paints the house blue. Is your house blue?

**SITUATION 2**

Your friend crosses out the address and writes a different one. Does your note change?

## By the end of class you can...

- 1 Explain what a parameter holds when the argument is an object.
- 2 Predict when a method changes the caller's object, and when it can't.
- 3 Leave an object passed to you unchanged unless the specification says otherwise.
- 4 Trace a method that returns an object reference, not a copy.
- 5 Decide whether a method may use the private data of a parameter.

A white rounded square with a dark teal number '1' inside, tilted slightly to the right.

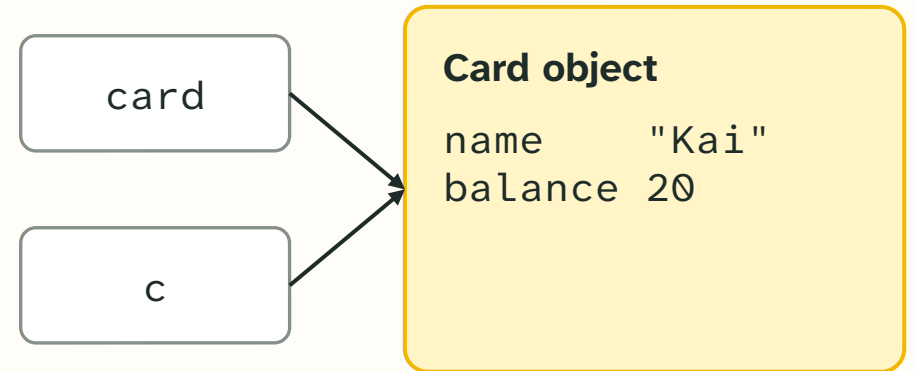
# 1

PART 1

# Passing an object reference

# The parameter gets a copy of the reference

```
class Card {  
    private String name;  
    private int balance;  
    public Card(String owner, int amount) {  
        name = owner;  
        balance = amount;  
    }  
    public String getName() {  
        return name;  
    }  
    public int getBalance() {  
        return balance;  
    }  
    public void load(int amount) {  
        balance = balance + amount;  
    }  
}
```



**card** is main's variable. **c** is the parameter. Both hold the same address.

Passing an object copies its **reference**. It never builds a new object. Only **new** does that.

# A mutator call through the parameter

## CASHIER.JAVA

```
class Cashier {  
    public void addBonus(Card c) {  
        c.load(5);  
    }  
}
```

## MAIN

```
Card card = new Card("Kai", 20);  
Cashier shop = new Cashier();  
shop.addBonus(card);  
System.out.println(card.getBalance());
```

## OUTPUT

25

- 1 card refers to a Card with balance 20.
- 2 addBonus(card) copies the reference into c.
- 3 c.load(5) changes that same Card to 25.
- 4 main prints card's balance: 25.

# Pointing the parameter elsewhere changes nothing



Calling a mutator through the parameter changes the caller's object. **Assigning** the parameter a new object does not.

## CASHIER.JAVA

```
class Cashier {  
    public void replace(Card c) {  
        c = new Card("New", 100);  
        System.out.println(c.getBalance());  
    }  
}
```

```
Card card = new Card("Kai", 20);  
Cashier shop = new Cashier();  
shop.replace(card);  
System.out.println(card.getBalance());
```

## OUTPUT

```
100  
20
```

card

**Kai's card**  
balance 20

# Load, then let go

PAUSE AND TRY

**What does main print?**

```
class Cashier {  
    public void process(Card c) {  
        c.load(10);  
        c = new Card("Temp", 0);  
        c.load(50);  
    }  
}
```

**MAIN**

```
Card card = new Card("Ana", 15);  
Cashier shop = new Cashier();  
shop.process(card);  
System.out.println(card.getBalance());
```

**YOUR OUTPUT, AND WHY**

A white rounded square with a thin black border, containing the number '2' in a bold, dark teal font.

# 2

PART 2

# Change it only when asked

## KEY IDEA

# Follow the specification

**Specification:** return the balance left after buying something that costs cost. Do not change the card.

## MEETS THE SPEC

```
public int left(Card c, int cost) {  
    return c.getBalance() - cost;  
}
```

```
Card card = new Card("Kai", 20);  
Planner p = new Planner();  
System.out.println(p.left(card, 8));  
System.out.println(card.getBalance());  
System.out.println(p.leftBad(card, 8));  
System.out.println(card.getBalance());
```

## SIDE EFFECT

```
public int leftBad(Card c, int cost) {  
    c.load(-cost);  
    return c.getBalance();  
}
```

## OUTPUT

```
12  
20  
12  
12
```



# A String can't be changed at all

## GREETER.JAVA

```
class Greeter {  
    public void shout(String s) {  
        s = s + "!";  
        System.out.println("inside: " + s);  
    }  
}
```

## MAIN

```
String word = "hi";  
Greeter g = new Greeter();  
g.shout(word);  
System.out.println("main: " + word);
```

## OUTPUT

```
inside: hi!  
main: hi
```



A **String** is **immutable**: once built, it never changes. A **Card** is **mutable**: load changes it. Only mutable objects can be changed through a parameter.



# 3

## PART 3

# Returning a reference

# Return hands back the same object

## CASHIER.JAVA

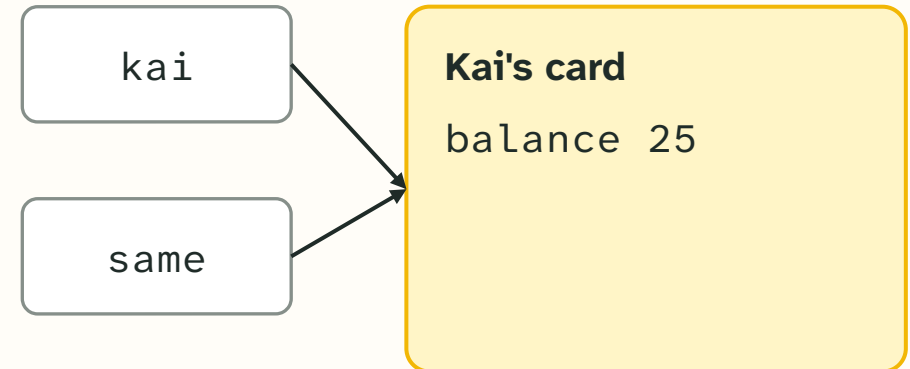
```
class Cashier {  
    public Card topUp(Card c, int amount) {  
        c.load(amount);  
        return c;  
    }  
}
```

## MAIN

```
Card kai = new Card("Kai", 20);  
Cashier shop = new Cashier();  
Card same = shop.topUp(kai, 5);  
System.out.println(same == kai);  
System.out.println(kai.getBalance());
```

## OUTPUT

```
true  
25
```



**Only new makes a copy.**

# Returning a private object lets others change it

## WALLET.JAVA

```
class Wallet {  
    private Card card;  
    public Wallet() {  
        card = new Card("Kai", 20);  
    }  
    public Card getCard() {  
        return card;  
    }  
}
```

```
Wallet mine = new Wallet();  
Card stolen = mine.getCard();  
stolen.load(-20);  
Card again = mine.getCard();  
System.out.println(again.getBalance());
```

## OUTPUT WITH THIS GETCARD

0

## FIX: RETURN A COPY INSTEAD

```
public Card getCard() {  
    return new Card(card.getName(),  
                    card.getBalance());  
}
```

## OUTPUT WITH THE FIXED GETCARD

20

# Follow the returns

PAUSE AND TRY

**What does this print? How many Card objects are there?**

MAIN

```
Card a = new Card("Ana", 10);
Cashier shop = new Cashier();
Card b = shop.topUp(a, 5);
Card c = shop.topUp(b, 5);
System.out.println(a.getBalance());
System.out.println(a == c);
```

```
public Card topUp(Card c, int amount) {
    c.load(amount);
    return c;
}
```

YOUR ANSWER



# 4

## PART 4

# Private data of a parameter

## KEY IDEA

# Same class? You may look inside

### INSIDE CARD: ALLOWED

```
public boolean beats(Card other) {  
    return balance > other.balance;  
}
```

KAI.BEATS(MO) PRINTS  
false

### INSIDE CASHIER: NOT ALLOWED

```
public int total(Card a, Card b) {  
    return a.balance + b.balance;  
}
```

COMPILER SAYS  
error: balance has private access in Card



Code may use the private variables and methods of a parameter **only when the parameter's type is the class the code is in**. Everywhere else, use the accessors.

## Which line compiles in Cashier?

PAUSE AND TRY

**A Cashier method has a parameter Card a. Which line compiles?**

**A** `return a.balance;`

**B** `return a.getBalance();`

**C** `a.balance = 0;`

**D** `return a.name.length();`

# Words to know

## Reference

The address of an object; what a variable of a class type holds.

## Alias

One of two or more variables that refer to the same object.

## Mutable

An object whose state can change after it is built.

## Immutable

An object that can never change, like a String.

## Specification

What a method is supposed to do; callers rely on it.

## Side effect

A change to a passed object that the specification did not ask for.

## Same object, or a new one?



**Tell your partner one way a method can change your object, and one way it can't.**

### SENTENCE STARTER

The parameter gets a copy of the \_\_\_\_\_, so calling a \_\_\_\_\_ through it changes my object, but assigning it a \_\_\_\_\_ object doesn't.

Next: the Same Bucket puzzle worksheet.