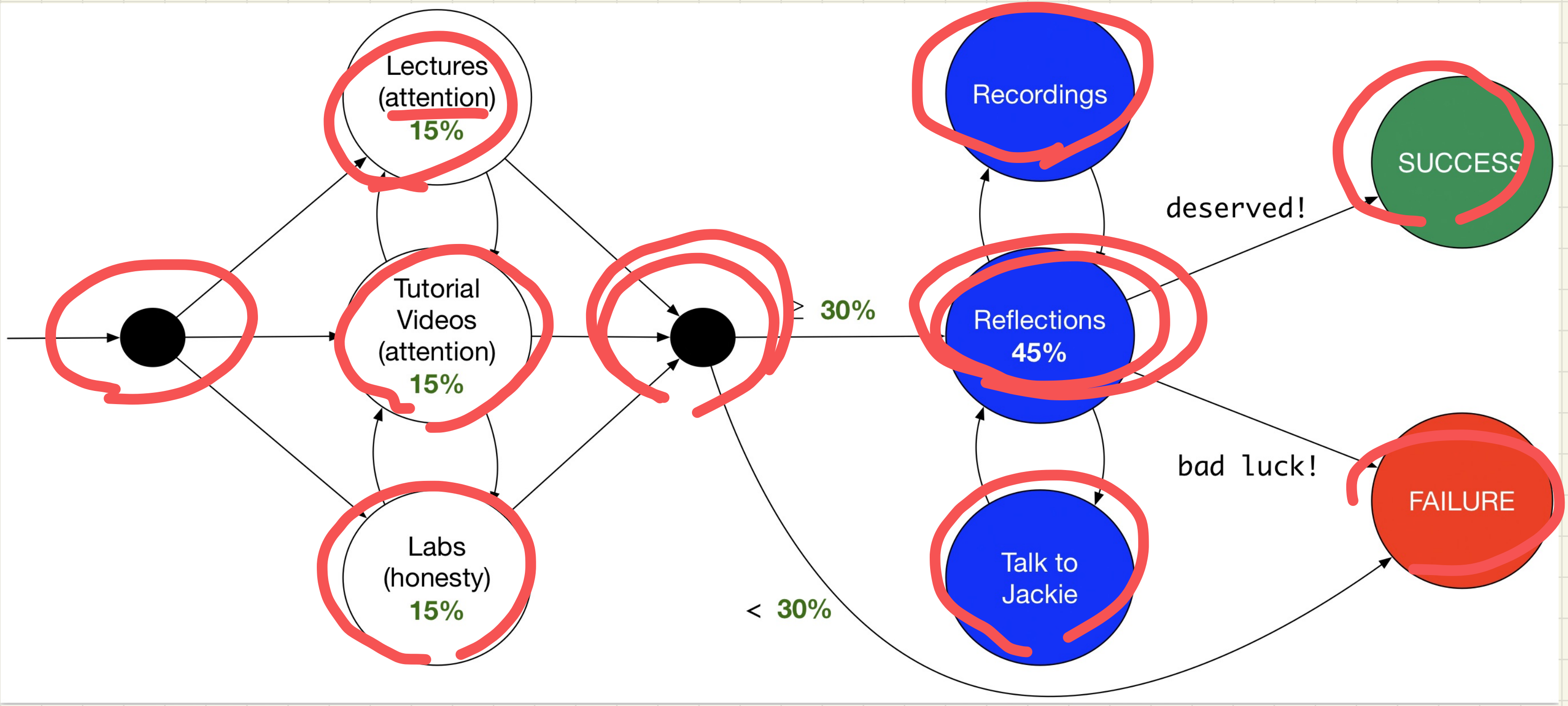
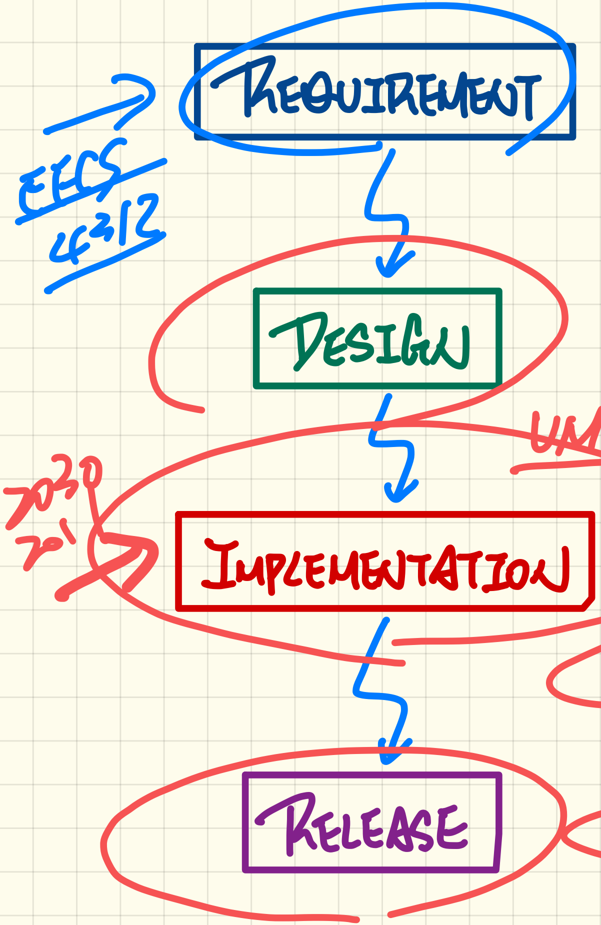
LECTURE 01

MONDAY JANUARY 06

Surviving through this Course



Software Development Process



- Natural Language
  (incomplete, ambiguous, contradicting)
- Requirement Elicitation

working payment system

easy to use
and
4-phase authentication
(face, toch, code, pass)

web interface
and
deployable on mobile devices

- Blueprints
- Not necessarily executable & testable

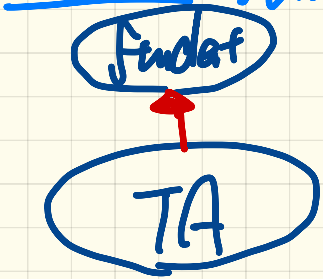
- API Given
- Efficient (data structures & algorithms)
- Unit Tests

- Customer's Acceptance
- Return?

Relationships between modules
classes

1. Inheritance

```
class Student {
} - -
class TA extends Student{
} . -
```



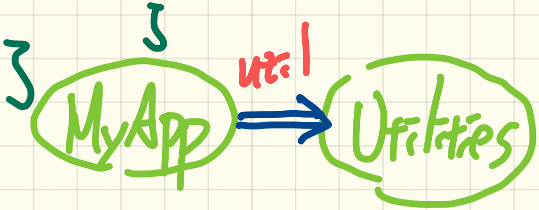
2. Client-Supplier relation.

client.

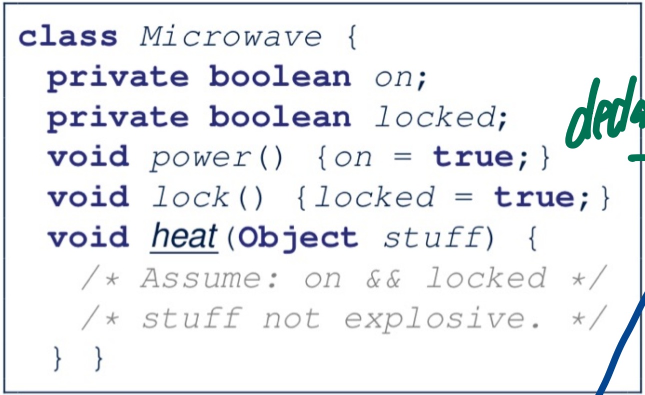
```
class MyApp {
  Utilities util = new - -
  util.sort(...);
}
```

supplier

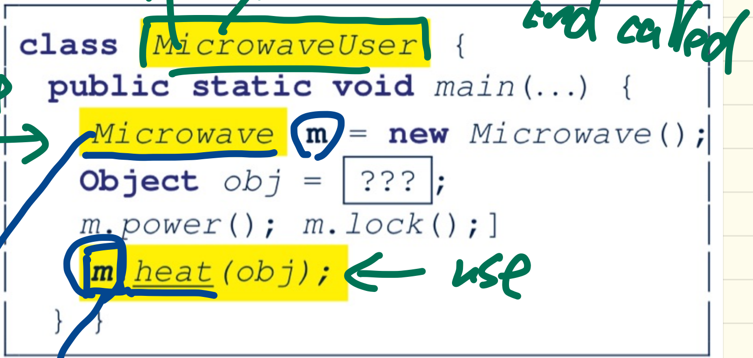
```
class Utilities {
  - - sort(int[] - -) {
  }
}
```



Client vs. Supplier in OOP



```
class Microwave {
  private boolean on;
  private boolean locked;
  void power() {on = true;}
  void lock() {locked = true;}
  void heat(Object stuff) {
    /* Assume: on && locked */
    /* stuff not explosive. */
  } }
```



```
class MicrowaveUser {
  public static void main(...) {
    Microwave m = new Microwave();
    Object obj = ???;
    m.power(); m.lock();]
    m.heat(obj);
  } }
```

client the class where the supplier obj is declared and called

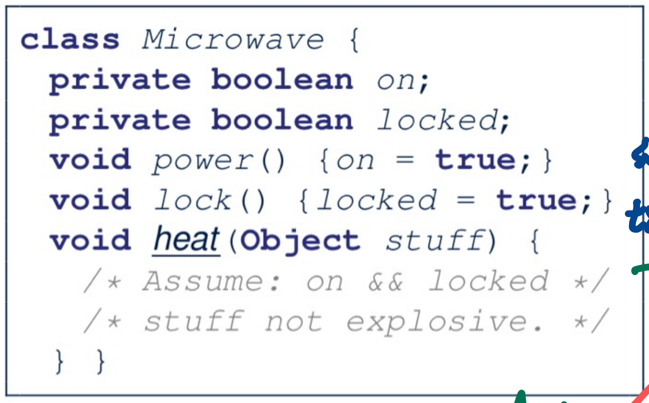
declare

use

type of C.O. ⇓ type of supplier

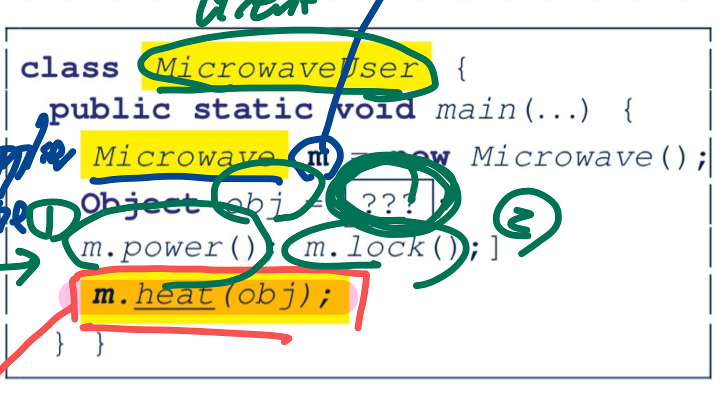
Context object

supplier.



```
class Microwave {
  private boolean on;
  private boolean locked;
  void power() {on = true;}
  void lock() {locked = true;}
  void heat(Object stuff) {
    /* Assume: on && locked */
    /* stuff not explosive. */
  } }
```

client



```
class MicrowaveUser {
  public static void main(...) {
    Microwave m = new Microwave();
    Object obj = ???;
    m.power(); m.lock(); ]
    m.heat(obj);
  } }
```

before-state
pre-state
→ satisfied

if any ob. of client not met's do not exec. method.

obligation of client
benefits of supplier.

on? ①
locked? ②
non-explosive? ✗

m. heat(obj)

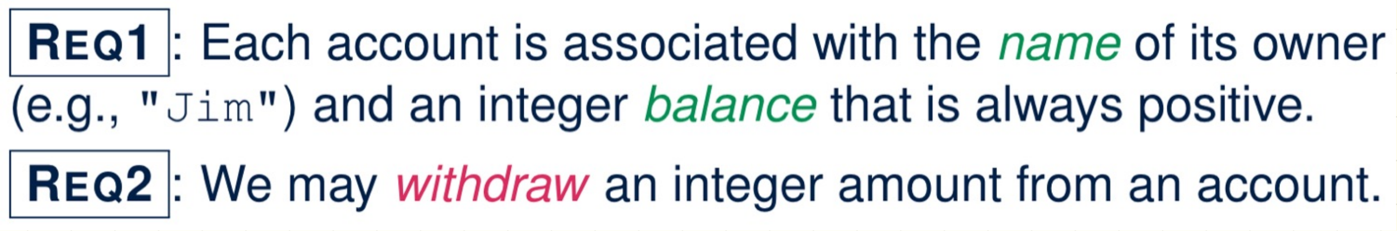
post-state.
after-state

→ obligat. of supplier.
bef. of client.

when it happens to be exp: it's the client's fault ⇐ *

↳ ??? is never checked before the call

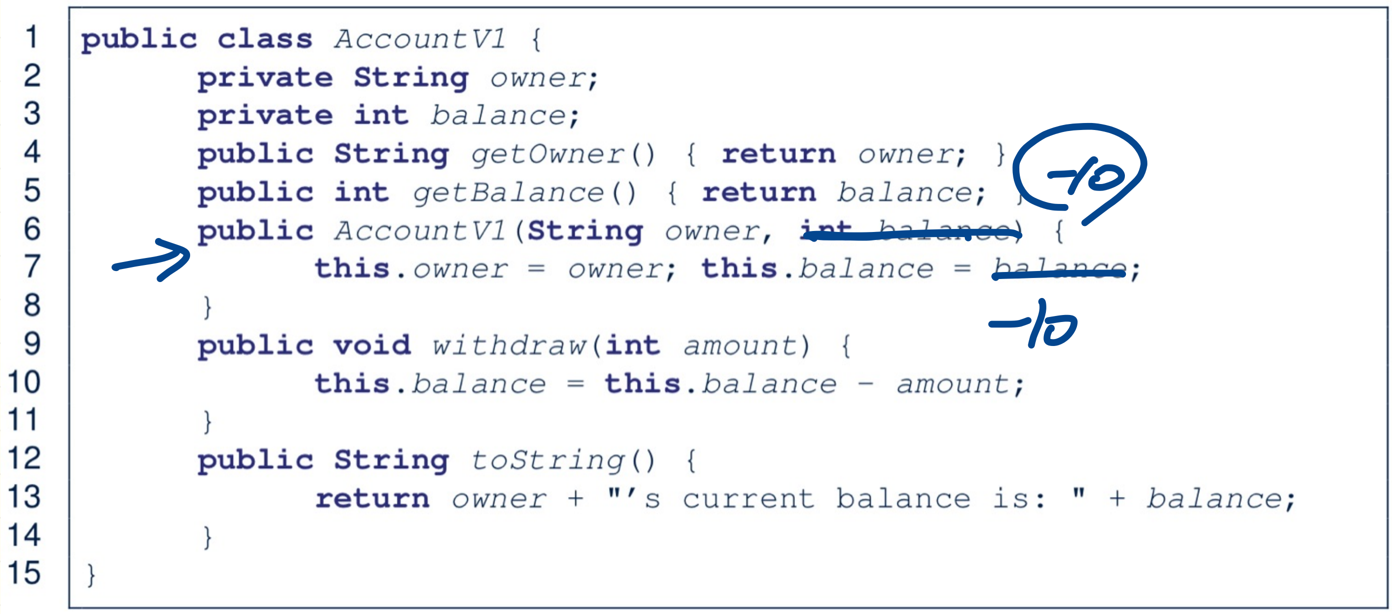
A Simple Design Problem: Bank Accounts



REQ1: Each account is associated with the *name* of its owner (e.g., `"Jim"`) and an integer *balance* that is always positive.

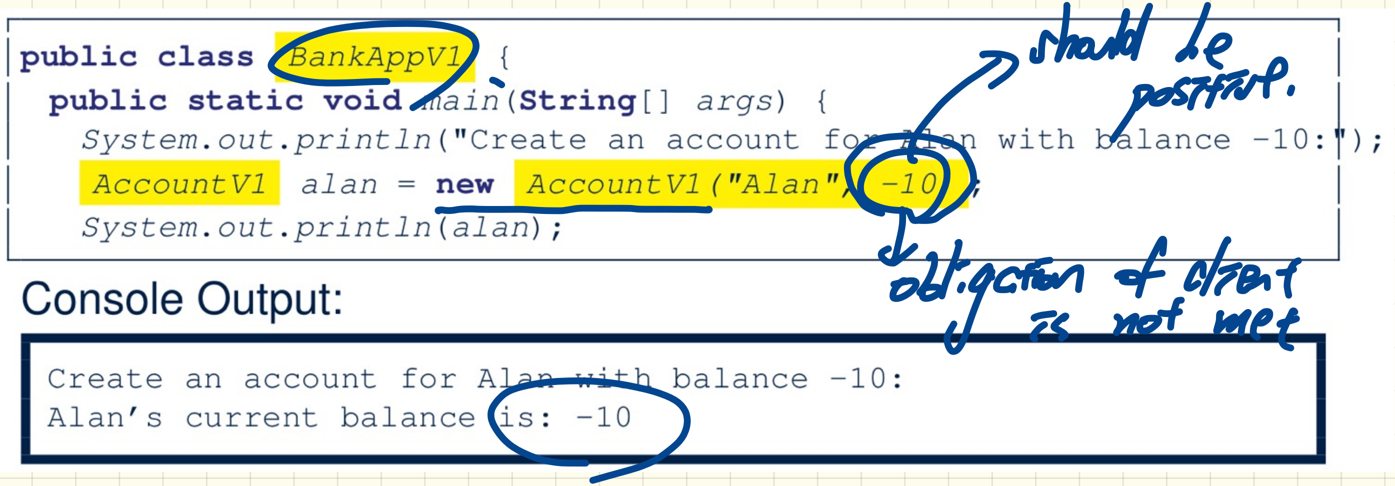
REQ2: We may *withdraw* an integer amount from an account.

Bank Accounts in Java: Version 1



```
public class AccountV1 {
        private String owner;
        private int balance;
        public String getOwner() { return owner; }
        public int getBalance() { return balance; }        -10
        public AccountV1(String owner, ~~int balance~~) {
                this.owner = owner; this.balance = ~~balance~~;
        }                                                  -10
        public void withdraw(int amount) {
                this.balance = this.balance - amount;
        }
        public String toString() {
                return owner + "'s current balance is: " + balance;
        }
}
```

Bank Accounts in Java: Version 1 Critique (1)



```
public class BankAppV1 {
  public static void main(String[] args) {
    System.out.println("Create an account for Alan with balance -10:");
    AccountV1 alan = new AccountV1("Alan", -10);
    System.out.println(alan);
```

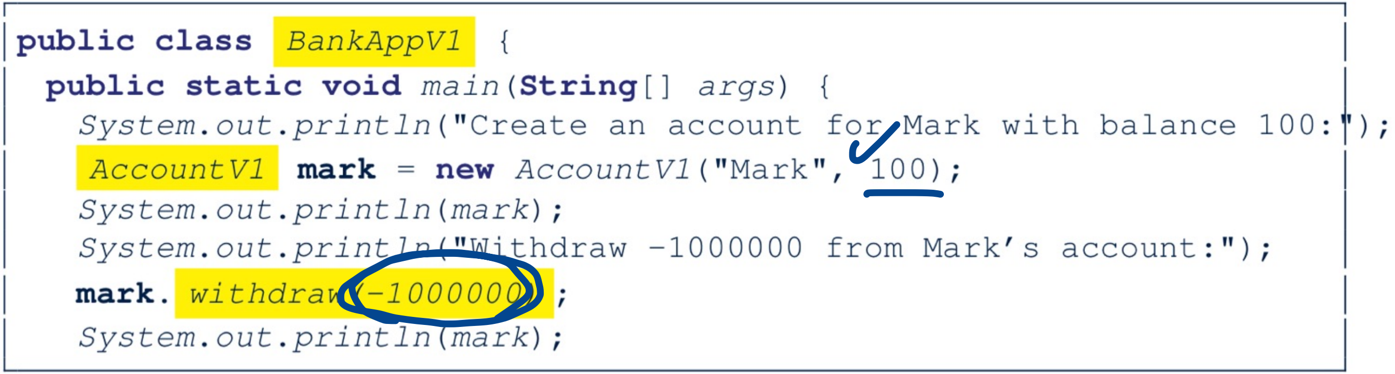
should be postfixed.

obligation of client is not met

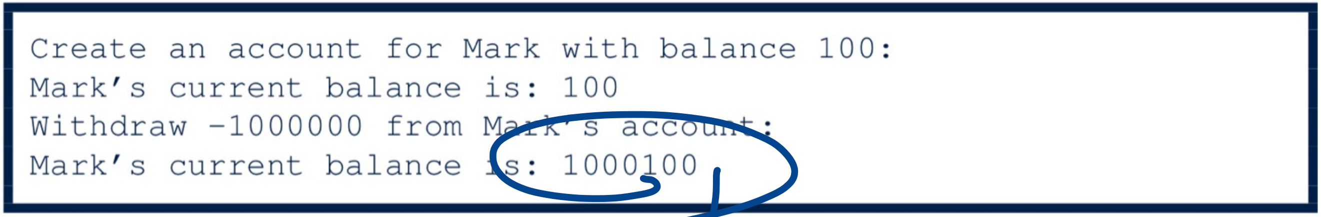
Console Output:

```
Create an account for Alan with balance -10:
Alan's current balance is: -10
```

Bank Accounts in Java: Version 1 Critique (2)



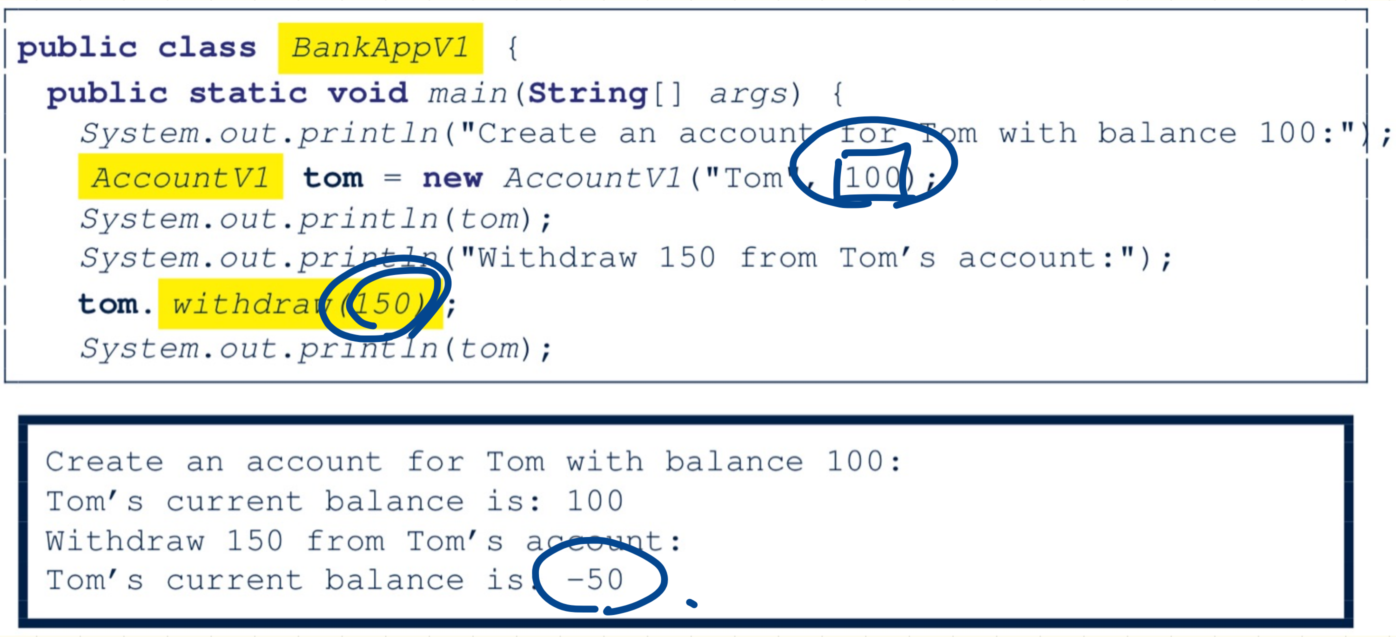
```
public class BankAppV1 {
  public static void main(String[] args) {
    System.out.println("Create an account for Mark with balance 100:");
    AccountV1 mark = new AccountV1("Mark", 100);
    System.out.println(mark);
    System.out.println("Withdraw -1000000 from Mark's account:");
    mark.withdraw(-1000000);
    System.out.println(mark);
```



```
Create an account for Mark with balance 100:
Mark's current balance is: 100
Withdraw -1000000 from Mark's account:
Mark's current balance is: 1000100
```

not good ∵ amount of withdraw is neg.

Bank Accounts in Java: Version 1 Critique (3)



```
public class BankAppV1 {
  public static void main(String[] args) {
    System.out.println("Create an account for Tom with balance 100:");
    AccountV1 tom = new AccountV1("Tom", 100);
    System.out.println(tom);
    System.out.println("Withdraw 150 from Tom's account:");
    tom.withdraw(150);
    System.out.println(tom);
```

```
Create an account for Tom with balance 100:
Tom's current balance is: 100
Withdraw 150 from Tom's account:
Tom's current balance is: -50
```

Precondition vs. Exception

↳ service cond.

↳ error cond.

```
double divide(double x, double y)
```

y == 0

!(y != 0)

① y == 0

① ≡ ¬②,

② y != 0

Precondition

```
if (y ~~!=~~ == 0) {
    throw IAE(...);
}
```