Lecture 2

Wednesday January 8

- Lab0

- Textbook: OOSC2 on course wiki

- Slides for self-study:
    * Eiffel: Overvies of Syntax
    * Eiffel: Common Error
    * BON Design Diagrams

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

```
1  public class AccountV1 {
2        private String owner;
3        private int balance;
4        public String getOwner() { return owner; }
5        public int getBalance() { return balance; }
6        public AccountV1(String owner, int balance) {
7              this.owner = owner; this.balance = balance;
8        }
9        public void withdraw(int amount) {
10             this.balance = this.balance - amount;
11       }
12       public String toString() {
13             return owner + "'s current balance is: " + balance;
14       }
15 }
```

Bank Accounts in Java: Version 1 Critique (1)

```
1  public class AccountV1 {
2        private String owner;
3        private int balance;
4        public String getOwner() { return owner; }
5        public int getBalance() { return balance; }
6        public AccountV1(String owner, int balance) {
7              this.owner = owner; this.balance = balance;
8        }
9        public void withdraw(int amount) {
10             this.balance = this.balance - amount;
11       }
12       public String toString() {
13             return owner + "'s current balance is: " + balance;
14       }
15 }
```

Supplier

Client

```
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);
```

Console Output:

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

Bank Accounts in Java: Version 1 Critique (2)

```
1  public class AccountV1 {
2        private String owner;
3        private int balance;
4        public String getOwner() { return owner; }
5        public int getBalance() { return balance; }
6        public AccountV1(String owner, int balance) {
7              this.owner = owner; this.balance = balance;
8        }
9        public void withdraw(int amount) {
10             this.balance = this.balance - amount;
11       }
12       public String toString() {
13             return owner + "'s current balance is: " + balance;
14       }
15 }
```

-1M

Supplier

Client

```
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
```

Bank Accounts in Java: Version 1 Critique (3)

```
1  public class AccountV1 {
2        private String owner;
3        private int balance;
4        public String getOwner() { return owner; }
5        public int getBalance() { return balance; }
6        public AccountV1(String owner, int balance) {
7              this.owner = owner; this.balance = balance;
8        }
9        public void withdraw(int amount) {
10             this.balance = this.balance - amount;
11       }
12       public String toString() {
13             return owner + "'s current balance is: " + balance;
14       }
15 }
```

150 100 150

Supplier

Client

```
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
```

X

Precondition

↳ service

divide(10, 0)

Exception

↳ error

double divide(x, y)

require ← precond

y != 0

B

0

F

precondition violation

↳ client

caller

double divide(x, y)

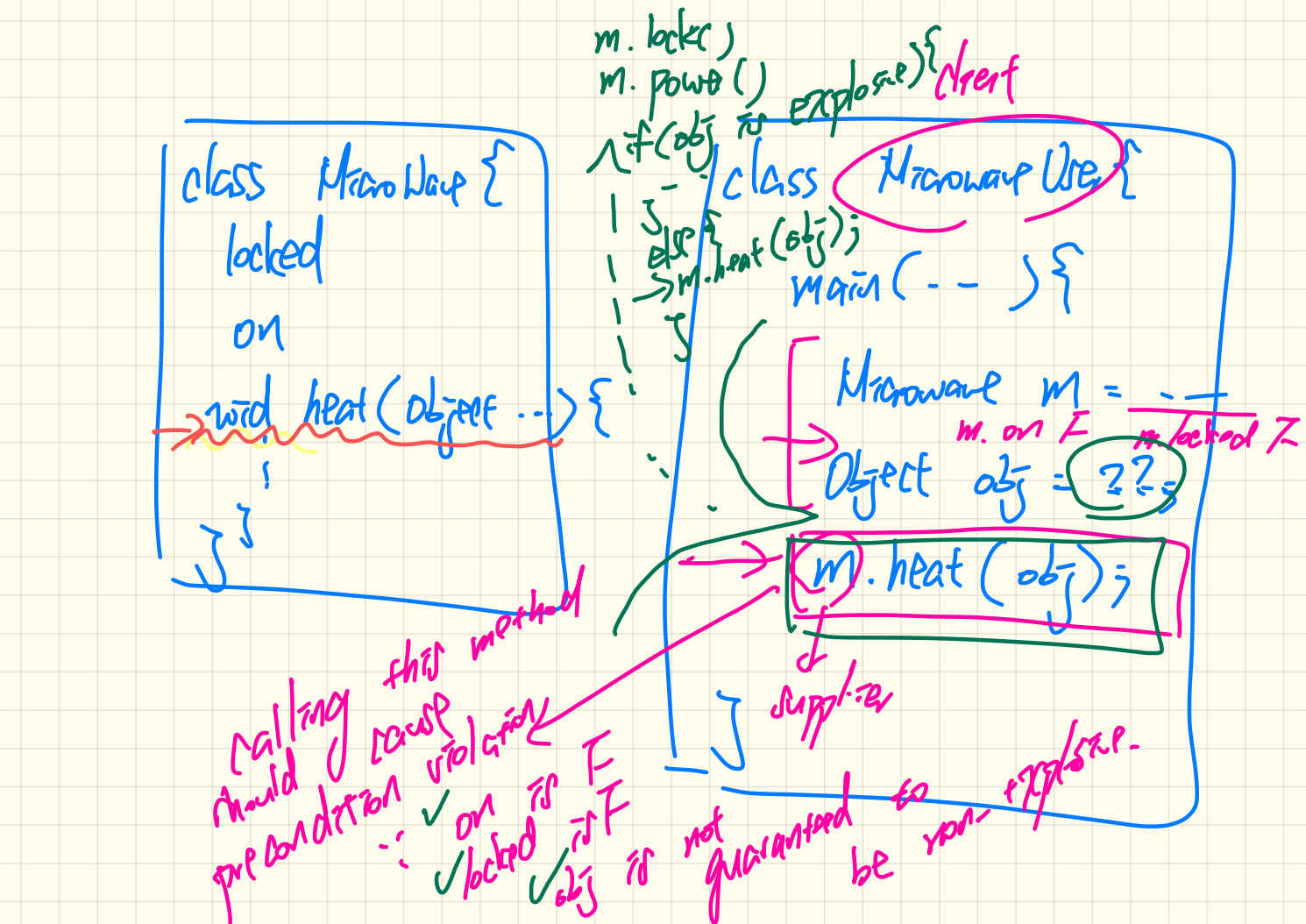
if(y == 0) {

!B

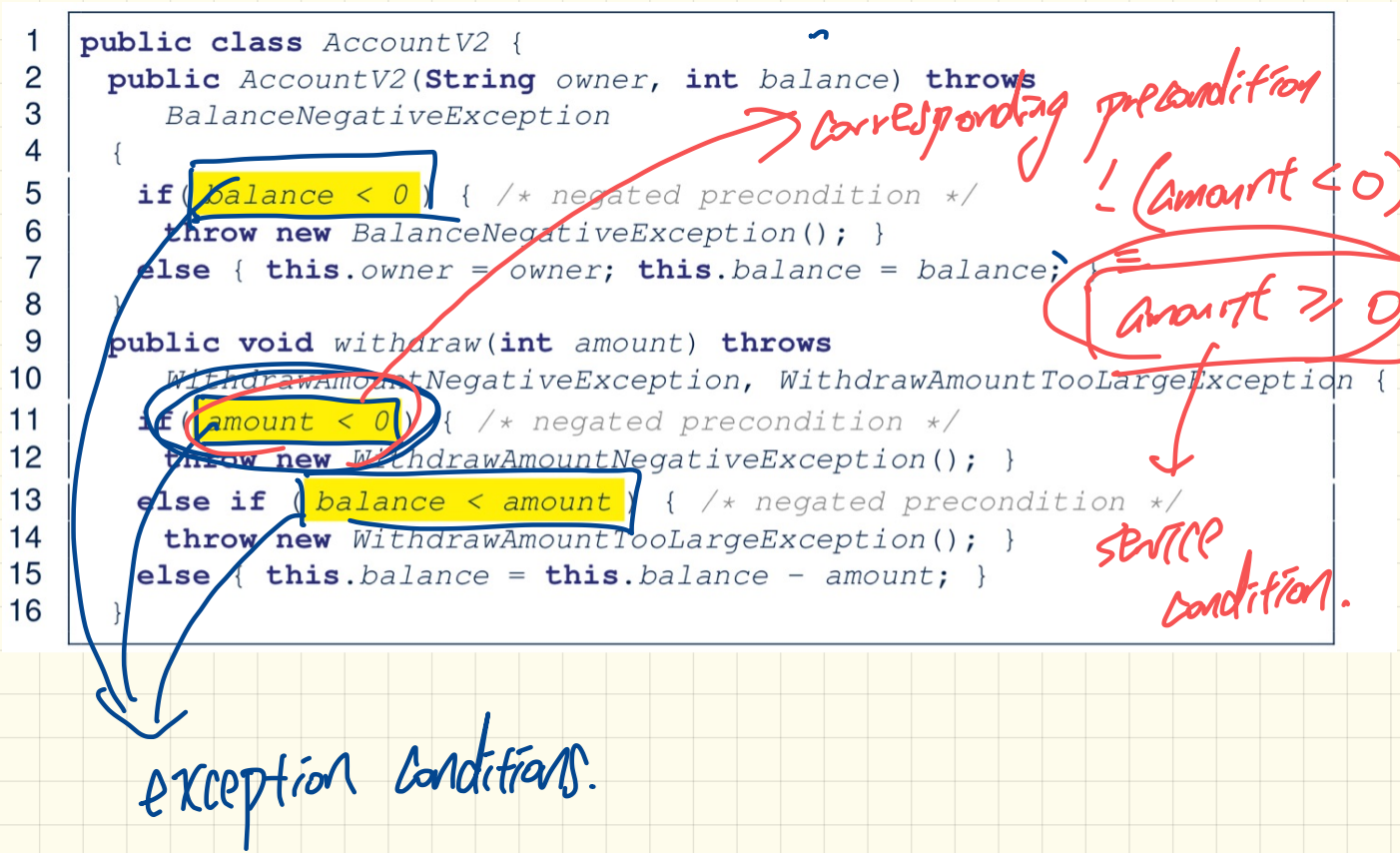
throw new IAE.

} else {

}


m.lock()
m.powe()
if(obj is explosive){ client
class MicroWave {
locked
on
void heat(Object ..) {
}
}
else{ m.heat(obj);
}
class MicrowaveUse {
main(--){
Microwave m = --
m.on F
m.locked T
Object obj = ??;
m.heat(obj);
supplier
calling this method should cause precondition violation
on is F
locked is F
obj is not guaranteed to be non-explosive.

Bank Accounts in Java: Version 2



Bank Accounts in Java: Version 2 Critique (1)

Compared with Version 1

```
1  public class AccountV2 {
2    public AccountV2(String owner, int balance) throws
3      BalanceNegativeException
4    {
5      if( balance < 0 ) { /* negated precondition */
6        throw new BalanceNegativeException(); }
7      else { this.owner = owner; this.balance = balance; }
8    }
9    public void withdraw(int amount) throws
10     WithdrawAmountNegativeException, WithdrawAmountTooLargeException
11     if( amount < 0 ) { /* negated precondition */
12       throw new WithdrawAmountNegativeException(); }
13     else if ( balance < amount ) { /* negated precondition */
14       throw new WithdrawAmountTooLargeException(); }
15     else { this.balance = this.balance - amount; }
16   }
```

Supplier

Client

```
1  public class BankAppV2 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Alan with balance -10:");
4      try {
5        AccountV2 alan = new AccountV2("Alan", -10);
6        System.out.println(alan);
7      }
8      catch ( BalanceNegativeException bne) {
9        System.out.println("Illegal negative account balance.");
10     }
```

```
Create an account for Alan with balance -10:
Illegal negative account balance.
```

Bank Accounts in Java: Version 2 Critique (2)

Compared with Version 1

```java
public class AccountV2 {
  public AccountV2(String owner, int balance) throws
     BalanceNegativeException
  {
    if( balance < 0 ) { /* negated precondition */
      throw new BalanceNegativeException(); }
    else { this.owner = owner; this.balance = balance; }
  }
  public void withdraw(int amount) throws
     WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
    if( amount < 0 ) { /* negated precondition */
      throw new WithdrawAmountNegativeException(); }
    else if ( balance < amount ) { /* negated precondition */
      throw new WithdrawAmountTooLargeException(); }
    else { this.balance = this.balance - amount; }
  }
```

Supplier

Client

```java
public class BankAppV2 {
  public static void main(String[] args) {
    System.out.println("Create an account for Mark with balance 100:");
    try {
      AccountV2 mark = new AccountV2("Mark", 100);
      System.out.println(mark);
      System.out.println("Withdraw -1000000 from Mark's account:");
      mark.withdraw(-1000000);
      System.out.println(mark);
    }
    catch (BalanceNegativeException bne) {
      System.out.println("Illegal negative account balance.");
    }
    catch (WithdrawAmountNegativeException wane) {
      System.out.println("Illegal negative withdraw amount.");
    }
    catch (WithdrawAmountTooLargeException wane) {
      System.out.println("Illegal too large withdraw amount.");
    }
```

Console Output:

```
Create an account for Mark with balance 100:
Mark's current balance is: 100
Withdraw -1000000 from Mark's account:
Illegal negative withdraw amount.
```

Bank Accounts in Java: Version 2 Critique (3)

Compared with Version 1

```
1  public class AccountV2 {
2    public AccountV2(String owner, int balance) throws
3       BalanceNegativeException
4    {
5      if( balance < 0 ) { /* negated precondition */
6        throw new BalanceNegativeException(); }
7      else { this.owner = owner; this.balance = balance; }
8    }
9    public void withdraw(int amount) throws
10     WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
11   if( amount < 0 ) { /* negated precondition */
12     throw new WithdrawAmountNegativeException(); }
13   else if ( balance < amount ) { /* negated precondition */
14     throw new WithdrawAmountTooLargeException(); }
15     else { this.balance = this.balance - amount; }
16   }
```

Supplier

Client

```
1  public class BankAppV2 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Tom with balance 100:");
4      try {
5        AccountV2 tom = new AccountV2("Tom", 100);
6        System.out.println(tom);
7        System.out.println("Withdraw 150 from Tom's account:");
8        tom.withdraw(150);
9        System.out.println(tom);
10     }
11     catch (BalanceNegativeException bne) {
12       System.out.println("Illegal negative account balance.");
13     }
14     catch (WithdrawAmountNegativeException wane) {
15       System.out.println("Illegal negative withdraw amount.");
16     }
17     catch (WithdrawAmountTooLargeException wane) {
18       System.out.println("Illegal too large withdraw amount.");
19     }
```

Console Output:

```
Create an account for Tom with balance 100:
Tom's current balance is: 100
Withdraw 150 from Tom's account:
Illegal too large withdraw amount.
```

Bank Accounts in Java: Version 2 Critique (4)

Supplier

```
1  public class AccountV2 {
2    public AccountV2(String owner, int balance) throws
3       BalanceNegativeException
4    {
5      if( balance < 0 ) { /* negated precondition */
6        throw new BalanceNegativeException(); }
7      else { this.owner = owner; this.balance = balance; }
8    }
9    public void withdraw(int amount) throws
10      WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
11   if( amount < 0 ) { /* negated precondition */
12     throw new WithdrawAmountNegativeException(); }
13   else if ( balance < amount ) { /* negated precondition */
14     throw new WithdrawAmountTooLargeException(); }
15   else { this.balance = this.balance - amount; }
16   }
```

Client

```
1  public class BankAppV2 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Jim with balance 100:");
4      try {
5        AccountV2 jim = new AccountV2("Jim", 100);
6        System.out.println(jim);
7        System.out.println("Withdraw 100 from Jim's account:");
8        jim.withdraw(100);
9        System.out.println(jim);
10     }
11     catch (BalanceNegativeException bne) {
12       System.out.println("Illegal negative account balance.");
13     }
14     catch (WithdrawAmountNegativeException wane) {
15       System.out.println("Illegal negative withdraw amount.");
16     }
17     catch (WithdrawAmountTooLargeException wane) {
18       System.out.println("Illegal too large withdraw amount.");
19     }
```

Req:

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

Console Output:

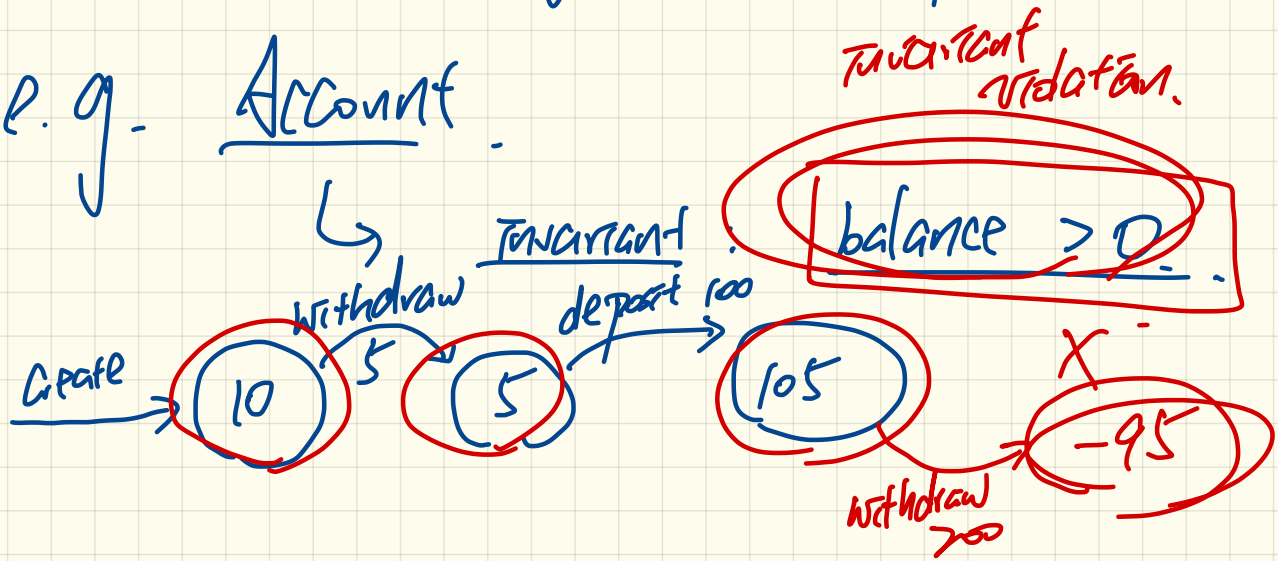
```
Create an account for Jim with balance 100:
Jim's current balance is: 100
Withdraw 100 from Jim's account:
Jim's current balance is: 0
```

class invariant → not change

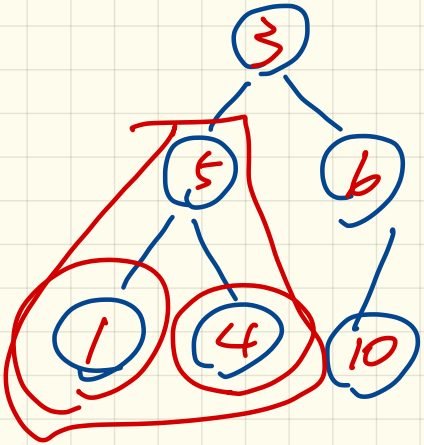
↳ property that holds true for all objects of a particular class

e.g. Account.

↳ invariant.



class invariant for BST

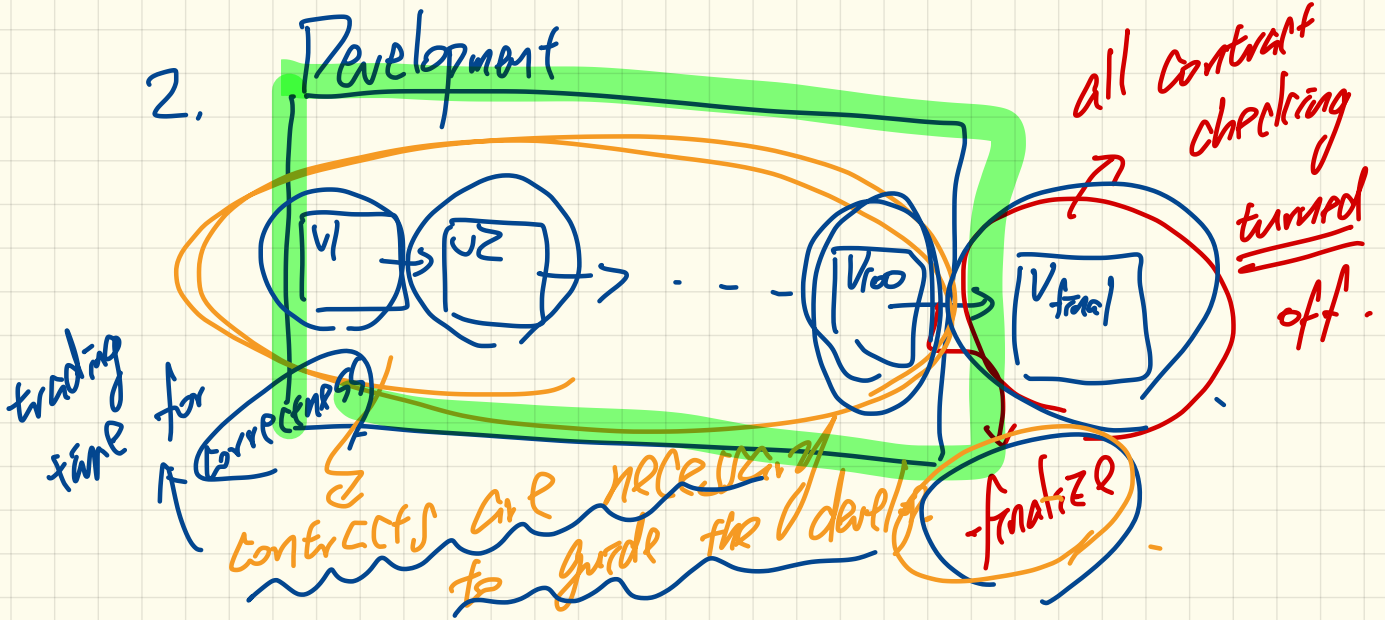


search

```
1. bool isBST(node){
    if(l){
    l<node ; isBST(l)
    }
    if(r){
    r> node
    }
}
```

1. When the data structure is large, checking contract (e.g. class inv) is inefficient.

2.



```
1  public class AccountV2 {
2    public AccountV2(String owner, int balance) throws
3       BalanceNegativeException
4    {
5      if( balance < 0 ) { /* negated precondition */
6        throw new BalanceNegativeException(); }
7      else { this.owner = owner; this.balance = balance; }
8    }
9    public void withdraw(int amount) throws
10      WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
11     if( amount < 0 ) { /* negated precondition */
12       throw new WithdrawAmountNegativeException(); }
13     else if ( balance < amount ) { /* negated precondition */
14       throw new WithdrawAmountTooLargeException(); }
15     else { this.balance = this.balance - amount; }
16   }
```

Single Choice Principle

assert bala. ≥ 0

assert balance ≥ 0.

class invariant.

assert bala. ≥ 0

Bank Accounts in Java: Version 3

```
1  public class AccountV3 {
2    public AccountV3(String owner, int balance) throws
3       BalanceNegativeException
4    {
5      if(balance < 0) { /* negated precondition */
6        throw new BalanceNegativeException(); }
7      else { this.owner = owner; this.balance = balance; }
8      assert this.getBalance() > 0 : "Invariant:  positive balance";
9    }
10   public void withdraw(int amount) throws
11      WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
12     if(amount < 0) { /* negated precondition */
13       throw new WithdrawAmountNegativeException(); }
14     else if (balance < amount) { /* negated precondition */
15       throw new WithdrawAmountTooLargeException(); }
16     else { this.balance = this.balance - amount; }
17     assert this.getBalance() > 0 : "Invariant:  positive balance";
18   }
```

C.I.

Bank Accounts in Java: Version 3 Critique (1)

Compared with Version 2

Supplier

```
1  public class AccountV3 {
2    public AccountV3(String owner, int balance) throws
3       BalanceNegativeException
4    {
5     if(balance < 0) { /* negated precondition */
6       throw new BalanceNegativeException(); }
7     else { this.owner = owner; this.balance = balance; }
8      assert this.getBalance() > 0 : "Invariant:  positive balance";
9    }
10   public void withdraw(int amount) throws
11      WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
12    if(amount < 0) { /* negated precondition */
13      throw new WithdrawAmountNegativeException(); }
14    else if (balance < amount) { /* negated precondition */
15      throw new WithdrawAmountTooLargeException(); }
16    else { this.balance = this.balance - amount; }
17     assert this.getBalance() > 0 : "Invariant:  positive balance";
18   }
```

Client

```
1  public class BankAppV3 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Jim with balance 100:");
4      try { AccountV3 jim = new AccountV3("Jim", 100);
5            System.out.println(jim);
6            System.out.println("Withdraw 100 from Jim's account:");
7            jim.withdraw(100);
8            System.out.println(jim); }
9            /* catch statements same as this previous slide:
10            * Version 2: Why Still Not a Good Design? (2.1) */
```

```
Create an account for Jim with balance 100:
Jim's current balance is: 100
Withdraw 100 from Jim's account:
Exception in thread "main"
    java.lang.AssertionError: Invariant: positive balance
```

Bank Accounts in Java: Version 3 Critique (2)

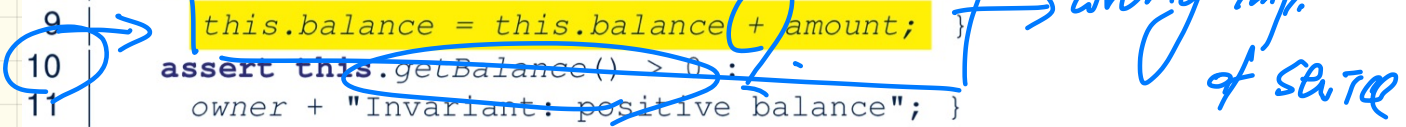
```
1  public class AccountV3 {
2    public void withdraw(int amount) throws
3        WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
4      if ( amount < 0 ) { /* negated precondition */
5        throw new WithdrawAmountNegativeException(); }
6      else if ( balance < amount ) { /* negated precondition */
7        throw new WithdrawAmountTooLargeException(); }
8      else { this.balance = this.balance - amount; }
9      assert this.getBalance() > 0 : "Invariant: positive balance"; }
```

When the amount is neither negative nor too large,
is there any obligation on the supplier of withdraw?

Bank Accounts in Java: Version 4

with an evil supplier

```
1  public class AccountV4 {
2    public void withdraw(int amount) throws
3      WithdrawAmountNegativeException, WithdrawAmountTooLargeException
4    { if(amount < 0) { /* negated precondition */
5        throw new WithdrawAmountNegativeException(); }
6      else if (balance < amount) { /* negated precondition */
7        throw new WithdrawAmountTooLargeException(); }
8      else { /* WRONG IMPLEMENTATION */
9        this.balance = this.balance + amount; }
10     assert this.getBalance() > 0 :
11       owner + "Invariant: positive balance"; }
```



Bank Accounts in Java: Version 4 Critique

```
1  public class AccountV4 {
2    public void withdraw(int amount) throws
3      WithdrawAmountNegativeException, WithdrawAmountTooLargeException
4    { if(amount < 0) { /* negated precondition */
5      throw new WithdrawAmountNegativeException(); }
6      else if (balance < amount) { /* negated precondition */
7      throw new WithdrawAmountTooLargeException(); }
8      else { /* WRONT IMPLEMENTATION */
9        this.balance = this.balance + amount; }
10     assert this.getBalance() >= 0 :
11       owner + "Invariant: positive balance"; }
```

balance = x

acc.withdraw(a)

balance = y.

y = x - a.

100 → 150

wrong - 150

Supplier

Client

```
1  public class BankAppV4 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Jeremy with balance 100:")
4      try { AccountV4 jeremy = new AccountV4("Jeremy", 100);
5            System.out.println(jeremy);
6            System.out.println("Withdraw 50 from Jeremy's account:");
7            jeremy.withdraw(50);
8            System.out.println(jeremy); }
9      /* catch statements same as this previous slide:
10      * Version 2: Why Still Not a Good Design? (2.1) */
```

missing delegation for supplier

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

Bank Accounts in Java: Version 5

```
1  public class AccountV5 {
2    public void withdraw(int amount) throws
3       WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
4     int oldBalance = this.balance;
5     if(amount < 0) { /* negated precondition */
6       throw new WithdrawAmountNegativeException(); }
7     else if (balance < amount) { /* negated precondition */
8       throw new WithdrawAmountTooLargeException(); }
9     else { this.balance = this.balance - amount; }
10    assert this.getBalance() > 0 :"Invariant: positive balance";
11    assert this.getBalance() == oldBalance - amount :
12      "Postcondition:  balance deducted"; }
```

postcondition

new value

ensure balance = old balance - a.