Lecture 2

Tuesday September 10

- waiting list ?
- Lab 1
- office hours : 4 ~ 6
  Mon Tue Wed
- tomorrow's lab session : syntax demo

How is DbC Useful in Guiding System Development?

Client's View:

- A console application.
- Keep entering names randomly until done.
- Keep inquiring if a name exists until quit.

Supplier's Implementation Strategy

- Store names in an array.
- Upon an inquiry: Binary Search,

Expected Run

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
a  b  c  d  e
Enter a name, or `quit` to stop inquiring: a
a exists!
Enter a name, or `quit` to stop inquiring: b
b exists!
Enter a name, or `quit` to stop inquiring: c
c exists!
Enter a name, or `quit` to stop inquiring: d
d exists!
Enter a name, or `quit` to stop inquiring: e
e exists!
Enter a name, or `quit` to stop inquiring: f
f does not exist!
Enter a name, or `quit` to stop inquiring: g
g does not exist!
Enter a name, or `quit` to stop inquiring: quit
```

Version 1: Wrong Implementation, No Contracts

```
class interface
  DATABASE_V1

create
    make

feature -- Constructor

  add_name (n: STRING_8)
        -- Add `n` to database.

  data_exists (n: STRING_8): BOOLEAN
        -- Does `n` exist in the database?

  make
        -- Create an empty database.

end -- class DATABASE_V1
```

C. S.

```
class interface
  UTILITIES_V1

create
    default_create

feature -- Binary Search

    search (a: ARRAY [STRING_8]; a_name: STRING_8): BOOLEAN

end -- class UTILITIES_V1
```

Bin Search Correct Imp.

- Data array in DATABASE is not kept sorted.
- Binary search in UTILITIES does not require a sorted input array.
- When user enters names in an unsorted order, output is wrong.
- But no contract violation!
- A bad design is when something goes wrong, there is no party to blame.

Version 1: User Interaction Session

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
e  c  d  a  b
Enter a name, or `quit` to stop inquiring: a
a does not exist!
Enter a name, or `quit` to stop inquiring: b
b does not exist!
Enter a name, or `quit` to stop inquiring: c
c does not exist!
Enter a name, or `quit` to stop inquiring: d
d exists!
Enter a name, or `quit` to stop inquiring: e
e does not exist!
Enter a name, or `quit` to stop inquiring: f
f does not exist!
Enter a name, or `quit` to stop inquiring: g
g does not exist!
Enter a name, or `quit` to stop inquiring: quit
```

Version 2: Wrong Implementation, Proper Precondition

```
class interface
    DATABASE_V2

create
    make

feature -- Constructor

    add_name (n: STRING_8)
            -- Add `n` to database.

    data_exists (n: STRING_8): BOOLEAN
            -- Does `n` exist in the database?

    make
            -- Create an empty database.

end -- class DATABASE_V2
```

```
class interface
    UTILITIES_V2

create
    default_create

feature -- Binary Search

    search (a: ARRAY [STRING_8]; a_name: STRING_8): BOOLEAN
        require
            array_sorted: across
                a.lower |..| (a.upper - 1) is
            all
                a [i] < a [i + 1]
            end

end -- class UTILITIES_V2
```

- Data array in DATABASE is not kept sorted.
- Binary search in UTILITIES now requires a sorted input array.
- When an unsorted array is passed for search, a contract violation occurs!
- A good design is when something goes wrong, there is one party to blame.

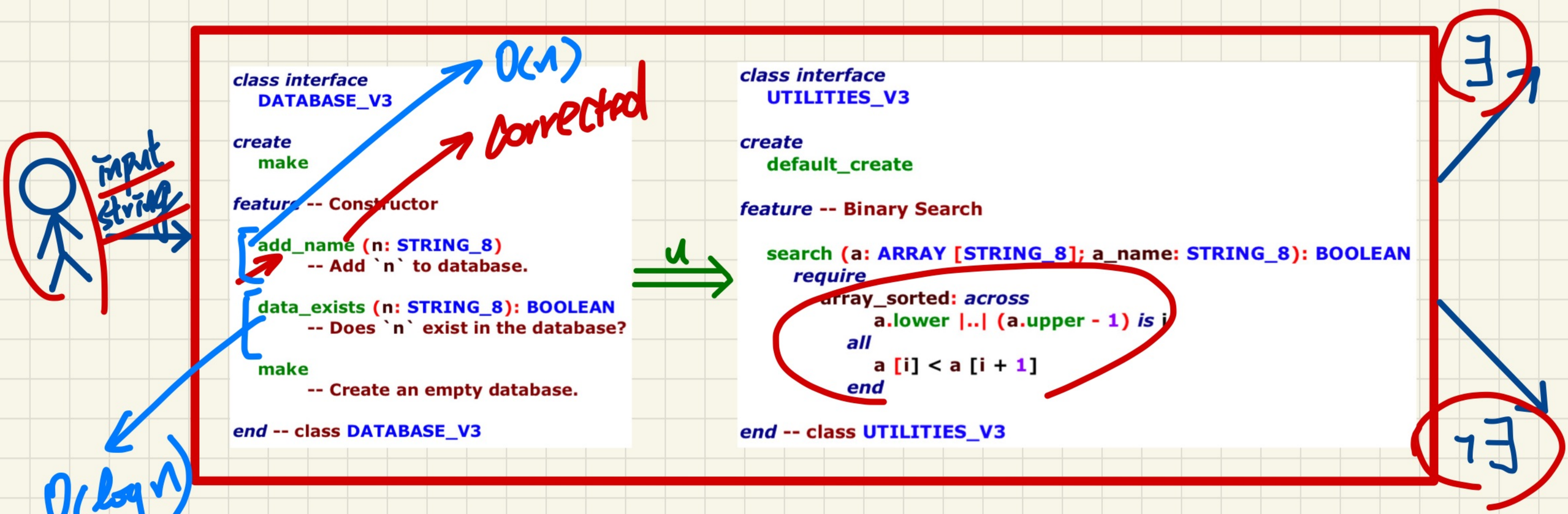
Version 2: User Interaction Session

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
e  c  d  a  b
Enter a name, or `quit` to stop inquiring: a

why-dbc-useful: system execution failed.
Following is the set of recorded exceptions:

******************************** Thread exception ********************************
In thread            Root thread            0x0 (thread id)
**********************************************************************************
--------------------------------------------------------------------------------
Class / Object       Routine                Nature of exception          Effect
--------------------------------------------------------------------------------
UTILITIES_V2         search @1              array_sorted:
<000000010EFF0FB8>                          Precondition violated.       Fail
--------------------------------------------------------------------------------
DATABASE_V2          data_exists @2
<000000010EFEFDF8>                          Routine failure.             Fail
--------------------------------------------------------------------------------
ROOT                 make @30
<000000010EFEF548>                          Routine failure.             Fail
--------------------------------------------------------------------------------
ROOT                 root's creation
<000000010EFEF548>                          Routine failure.             Exit
--------------------------------------------------------------------------------
```

Version 3: Fixed Implementation, Proper Precondition



- Data array in DATABASE is now kept sorted (so as to avoid contract violation).
- Binary search in UTILITIES still requires a sorted input array.
- A sorted array is always passed for search, a contract violation never occurs!
- Now finalize/deliver the working system with contracts checking turned off.

Version 3: User Interaction Session

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
a  b  c  d  e
Enter a name, or `quit` to stop inquiring: a
a exists!
Enter a name, or `quit` to stop inquiring: b
b exists!
Enter a name, or `quit` to stop inquiring: c
c exists!
Enter a name, or `quit` to stop inquiring: d
d exists!
Enter a name, or `quit` to stop inquiring: e
e exists!
Enter a name, or `quit` to stop inquiring: f
f does not exist!
Enter a name, or `quit` to stop inquiring: g
g does not exist!
Enter a name, or `quit` to stop inquiring: quit
```

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)

```
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)

```
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)

```
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
```

divide(x, y)

Precondition: Service

y != 0

Exceptions: Error

y == 0

Bank Accounts in Java: Version 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   }
```

Bank Accounts in Java: Version 2 Critique (1) (Compared with Version 1)

```
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)

```
1  public class BankAppV2 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Mark with balance 100:");
4      try {
5        AccountV2 mark = new AccountV2("Mark", 100);
6        System.out.println(mark);
7        System.out.println("Withdraw -1000000 from Mark's account:");
8        mark.withdraw(-1000000);
9        System.out.println(mark);
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 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 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:

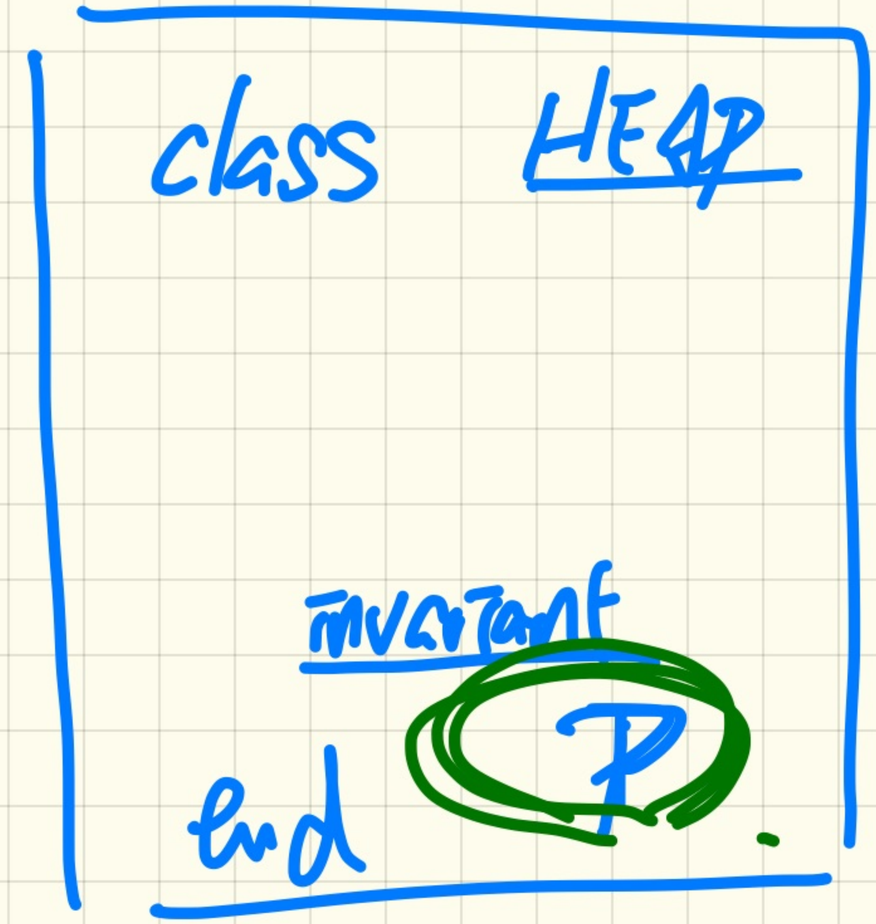
REQ1: Each account is associated with the *name* of its owner (e.g., "Jim") and an integer *balance* that is always positive.

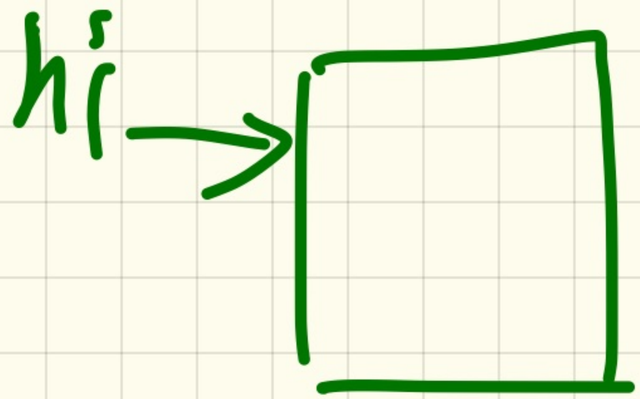
Console Output:

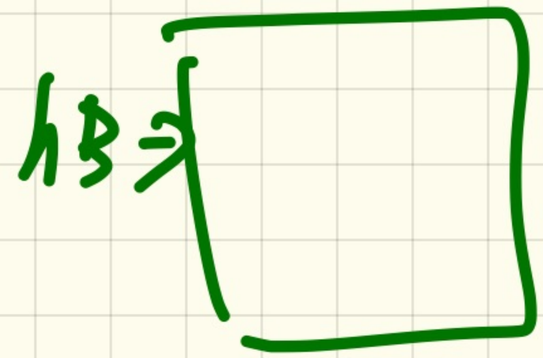
```
Create an account for Jim with balance 100:
Jim's current balance is: 100
Withdraw 100 from Jim's account:
Jim's current balance is: 0
```

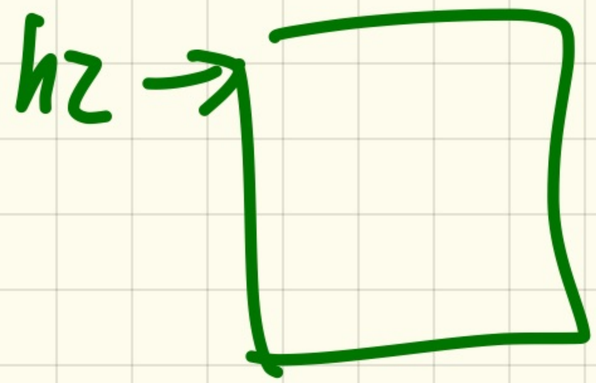
class invariant

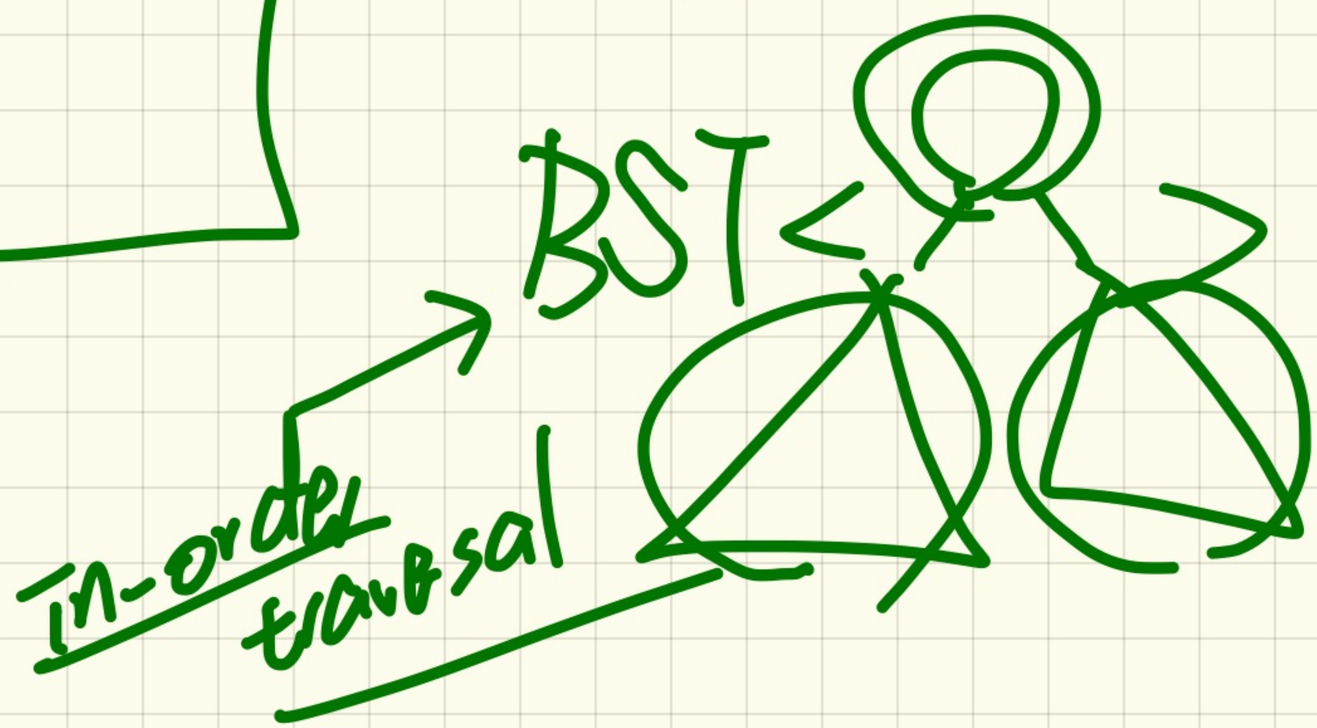
Compile time











API

withdraw(---)

if(---){

throw

}

assert(---)

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   }
```

100

violates Single choice principle.

Bank Accounts in Java: Version 3 Critique (1) (Compared with Version 2)

```
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";
```

10^6

wrong sign?

→ nothing will signal an error.

When 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 { /* WRONT 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 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) */
```

```
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
```

Precondition & Postcondition Exercise

∀ ∃

change_at (a: **ARRAY**[**STRING**]; i: **INTEGER**; ns; **STRING**)

- - Change index `i` in array `a` to string `ns`

require

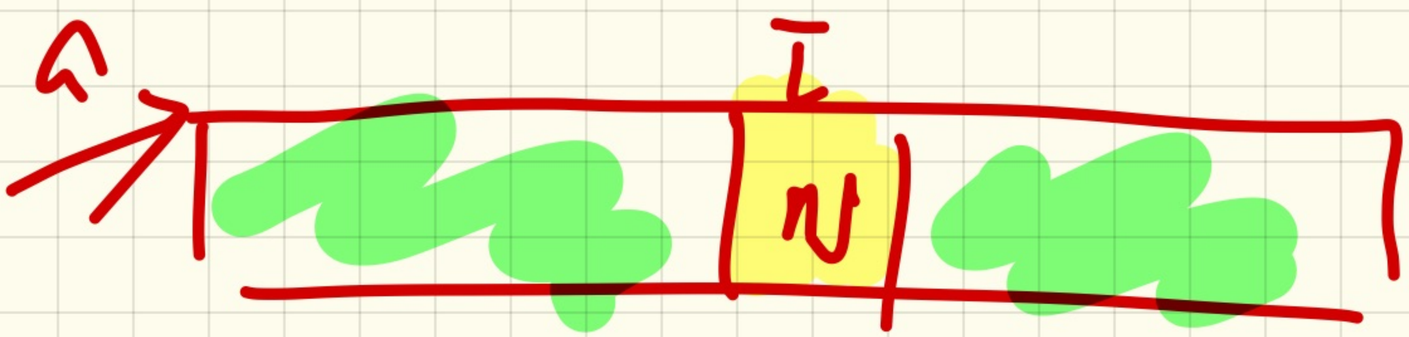
??

1 <= i and i <= a.count

ensure

??

→ ✓ a[i] ~ ns



Eiffel Supplie:

a[i] = ns;

a[i-1] = null;

Bank Accounts in Java : Version 5

current balance : 100

50

```
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"; }
```

100

150

100

50

T

Bank Accounts in Java : Version 5 Critique (compared with Version 4)

```
1  public class BankAppV5 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Jeremy with balance 100:")
4      try { AccountV5 jeremy = new AccountV5("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) */
```

```
Create an account for Jeremy with balance 100:
Jeremy's current balance is: 100
Withdraw 50 from Jeremy's account:
Exception in thread "main"
  java.lang.AssertionError: Postcondition: balance deducted
```