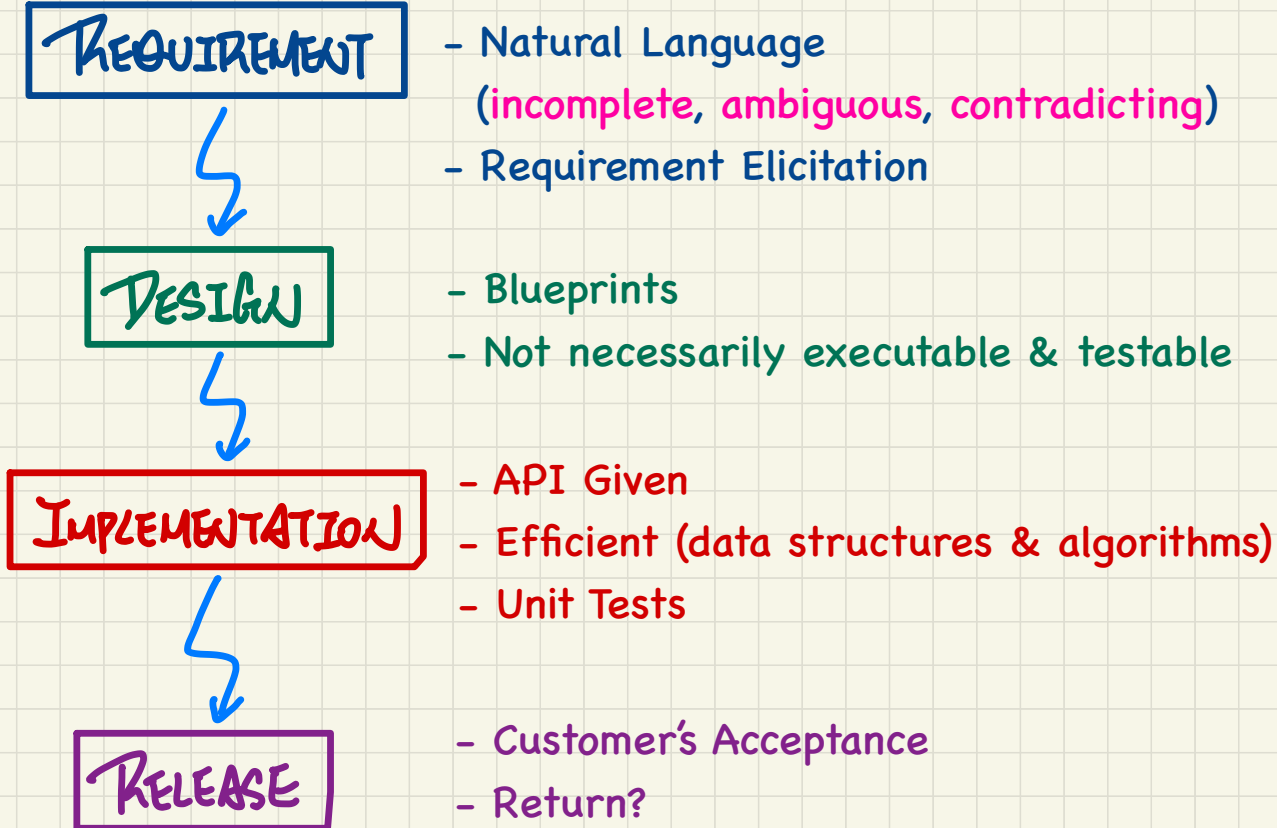


Lecture 1

Part 1

Design by Contract (DbC): Motivation & Terminology

Software Development Process



Informal Requirements

Incompleteness, Ambiguities, Contradictions

I'd like a working payment system.

Ideally, user can use it to pay their electricity bills and so on.

It should be easy to use and secure with a 4-phased authentication (face, touch, verification code, password).

Roadmap of this Course

Design

Abstract Data types (ADTs)

Cohesion Principle
Single Choice Principle
Open-Closed Principle
Design Document
Justified Design Decisions

Architecture: Client-Supplier Relation

Architecture: Inheritance Relation

Program to Interface,
Not to Implementation

Modularity: Classes

Design Patterns

(Iterator, Singleton, State, Template,
Composite, Visitor, Strategy,
Observer, Event-Driven Design)
Anti-Patterns

Code Reuse via Inheritance

Substitutibility

Polymorphism (esp. *Polymorphic Collections*)

Type Casting

Static Typing, Dynamic Binding

Unit Testing

OOP

Design by Contract (DbC):

Class Invariant, Pre-/Post-condition

Information Hiding Principle

Eiffel Testing Framework (ETF)

Abstraction (via Mathematical Models)

Regression Testing

Acceptance Testing

Void Safety

Generics

Multiple Inheritance

Sub-Contracting

Architectural Design Diagrams

Eiffel

Syntax: Implementation vs. Specification
agent expression, **across** constructs
expanded types, **export** status
Runtime Contract Checking
Debugger

Specification: *Predicates*

Contracts of Loops: Invariant & Variant

Program Correctness

Weakest Precondition (**WP**)

Hoare Triples

Specification: Higher-Order Functions

Axioms, Lemmas, Theorems

Equational Proofs

Proof by Contradiction (*witness*)

Logic

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);  
    }  
}
```

Lecture 1

Part 2

***Supporting DbC in Java:
Preconditions,
Class Invariant,
Postconditions***

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 }
```

Client

Supplier

```
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 }
```

Client

Supplier

```
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 }
```

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
```

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 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    {
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    }
18 }
```

Client

Supplier

```
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        }
11    }
12 }
```

```
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 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 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        }
20    }
```

Supplier

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.
```

Compared
with
Version 1

Bank Accounts in Java: Version 2 Critique (3)

```
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        }
20    }
```

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        }
```

Requirement

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
```


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    }
```

Bank Accounts in Java: Version 3 Critique (1)

Compared
with
Version 2

```
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

Supplier

```
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"; }
```

Client

Supplier

```
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
```

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"; }
```

Bank Accounts in Java: Version 5 Critique

Compared
with
Version 4

```
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"; }
```

Client

Supplier

```
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
```

Design by Contract in Java

```
public class AccountV5 {  
    public void withdraw(int amount) throws  
        WithdrawAmountNegativeException, WithdrawAmountTooLargeException  
    {  
        int oldBalance = this.balance;  
        if (amount < 0) { /* negated precondition */  
            throw new WithdrawAmountNegativeException(); }  
        else if (balance < amount) { /* negated precondition */  
            throw new WithdrawAmountTooLargeException(); }  
        else { this.balance = this.balance - amount; }  
        assert this.getBalance() > 0 : "Invariant: positive balance";  
        assert this.getBalance() == oldBalance - amount :  
            "Postcondition: balance deducted"; }  
}
```

Supplier

Client

```
public static void main(String[] args) {  
    System.out.println("Create an account for Jim with balance 100:");  
    try {  
        AccountV2 jim = new AccountV2("Jim", 100);  
        System.out.println(jim);  
        System.out.println("Withdraw 100 from Jim's account:");  
        jim.withdraw(100);  
        System.out.println(jim);  
    }  
    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.");  
    }  
}
```


Design by Contract in Eiffel

Contract View

```
class ACCOUNT
create
    make

feature -- Attributes
    owner : STRING
    balance : INTEGER

feature -- Constructors
    make(nn: STRING; nb: INTEGER)
        require -- precondition
            positive_balance: nb > 0
        end

feature -- Commands
    withdraw(amount: INTEGER)
        require -- precondition
            non_negative_amount: amount >= 0
            affordable_amount: amount <= balance -- problematic, why?
        ensure -- postcondition
            balance_deducted: balance = old balance - amount
        end

invariant -- class invariant
    positive_balance: balance > 0
end
```

```
class ACCOUNT
create
    make

feature -- Attributes
    owner : STRING
    balance : INTEGER

feature -- Constructors
    make(nn: STRING; nb: INTEGER)
        require -- precondition
            positive_balance: nb > 0
        do
            owner := nn
            balance := nb
        end

feature -- Commands
    withdraw(amount: INTEGER)
        require -- precondition
            non_negative_amount: amount > 0
            affordable_amount: amount <= balance -- problematic
        do
            balance := balance - amount
        ensure -- postcondition
            balance_deducted: balance = old balance - amount
        end

invariant -- class invariant
    positive_balance: balance > 0
end
```

Implementation View

Lecture 1

Part 3

DbC in Eiffel: Runtime Contract Checking

Design by Contract in Eiffel

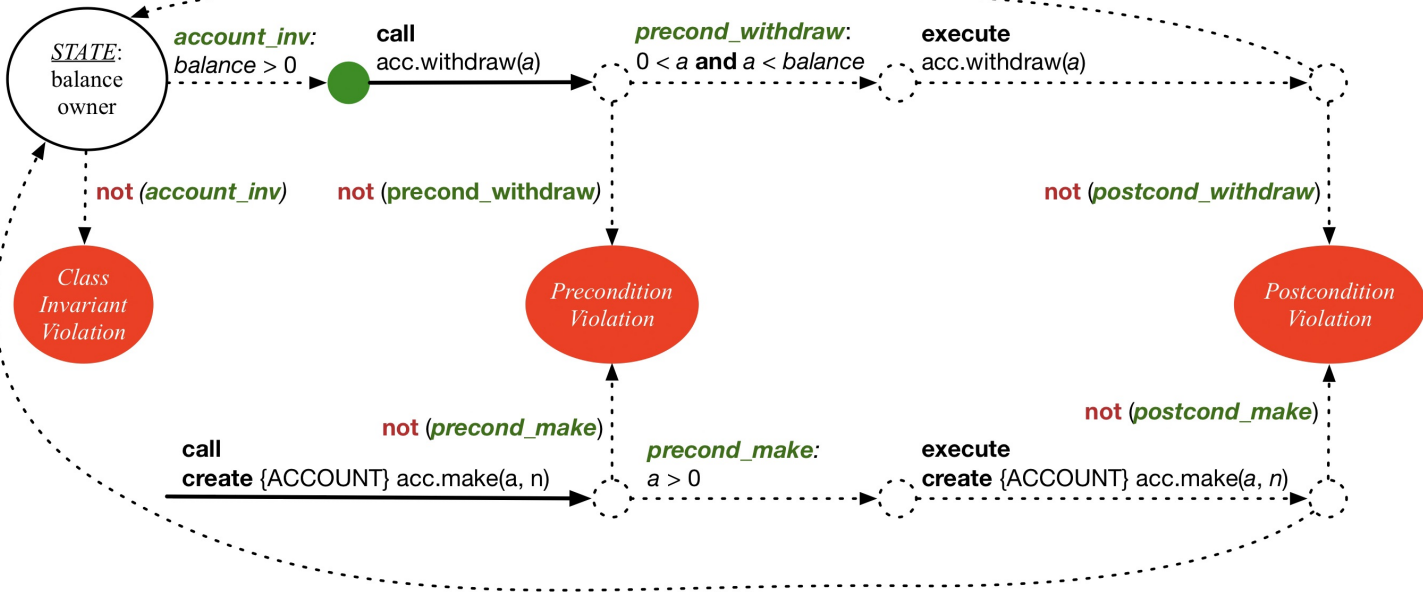
Implementation View

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(nn: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    do
      owner := nn
      balance := nb
    end
  ensure
    init: balance = nb and owner = nn
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount > 0
      affordable_amount: amount <= balance -- problematic
    do
      balance := balance - amount
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```

Runtime Monitoring of Contracts

```
acc: ACCOUNT  
create acc.make(a, n)  
acc.withdraw(a)
```

postcond_withdraw:
acc.balance = old acc.balance - a and acc.owner ~ old acc.owner



Precondition Violation: `positive_balance`

Feature view of feature 'make' of class ACCOUNT

```
make (nn: STRING_8; nb: INTEGER_32)
  require
    positive_balance: nb >= 0
  do
    owner := nn
    balance := nb
  end
```

Call Stack

Status = Implicit exception pending

positive_balance: PRECONDITION_VIOLATION raised

In Feature	In Class	From Class	@
make	ACCOUNT	ACCOUNT	1
make	APPLICATION	APPLICATION	1

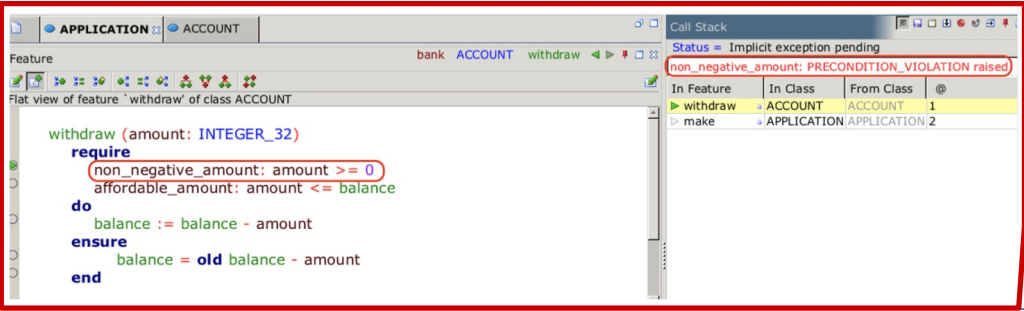
Supplier

Client

```
class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
    -- Run application.
  local
    alan: ACCOUNT
  do
    -- A precondition violation with tag
    create {ACCOUNT} alan.make ("Alan", -10)
  end
end
```

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(nn: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    end
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount >= 0
      affordable_amount: amount <= balance -- problema
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```

Precondition Violation: non_negative_amount



Supplier

Client

```
class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
  -- Run application.
local
  mark: ACCOUNT
do
  create {ACCOUNT} mark.make ("Mark", 100)
  -- A precondition violation with tag "non_negative_amount"
  mark.withdraw(-1000000)
end
end
```

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(nn: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    end
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount >= 0
      affordable_amount: amount <= balance -- problema
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```

Precondition Violation:

affordable_amount

```
APPLICATION: ACCOUNT
Feature bank ACCOUNT withdraw
Flat view of feature 'withdraw' of class ACCOUNT

withdraw (amount: INTEGER_32)
  require
    non_negative_amount: amount >= 0
    affordable_amount: amount <= balance
  do
    balance := balance - amount
  ensure
    balance = old balance - amount
end
```

Call Stack
Status = Implicit exception pending
affordable_amount: PRECONDITION_VIOLATION raised

In Feature	In Class	From Class	@
withdraw	ACCOUNT	ACCOUNT	2
make	APPLICATION	APPLICATION	2

Supplier

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(nn: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    end
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount >= 0
      affordable_amount: amount <= balance -- problema
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```

Client

```
class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
    -- Run application.
local
  tom: ACCOUNT
do
  create {ACCOUNT} tom.make ("Tom", 100)
  -- A precondition violation with tag "
  tom.withdraw(150)
end
end
```

Class Invariant Violation: **positive_balance**

Feature: bank ACCOUNT _invariant

Flat view of feature '_invariant' of class ACCOUNT

```
positive_balance: balance > 0
```

Call Stack

Status = Implicit exception pending

positive_balance: INVARIANT_VIOLATION raised

In Feature	In Class	From Class	@
_invariant	ACCOUNT	ACCOUNT	0
withdraw	ACCOUNT	ACCOUNT	5
make	APPLICATION	APPLICATION	2

Supplier

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(nn: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    end
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount >= 0
      affordable_amount: amount <= balance -- problema
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```

Client

```
class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
    -- Run application.
local
  jim: ACCOUNT
do
  create {ACCOUNT} tom.make ("Jim", 100)
  jim.withdraw(100)
  -- A class invariant violation with tag "positive_balance"
end
end
```


Postcondition Violation: **balance_deducted**

Feature view of feature `withdraw` of class ACCOUNT

```
affordable_amount: amount <= balance
do
  balance := balance + amount
ensure
  balance_deducted: balance = old balance - amount
end
```

Call Stack

Status = Implicit exception pending

balance_deducted: POSTCONDITION_VIOLATION raised

In Feature	In Class	From Class	@
withdraw	ACCOUNT	ACCOUNT	4
make	APPLICATION	APPLICATION	2

Supplier

Client

```
class BANK_APP
inherit ARGUMENTS
create make
feature -- Initialization
  make
    -- Run application.
  local
    jeremy: ACCOUNT
  do
    -- Faulty implementation of withdraw in ACCO
    -- balance := balance + amount
    create {ACCOUNT} jeremy.make ("Jeremy", 100)
    jeremy.withdraw(150)
    -- A postcondition violation with tag "balance_deducted"
  end
end
```

```
class ACCOUNT
create
  make
feature -- Attributes
  owner : STRING
  balance : INTEGER
feature -- Constructors
  make(n: STRING; nb: INTEGER)
    require -- precondition
      positive_balance: nb > 0
    end
feature -- Commands
  withdraw(amount: INTEGER)
    require -- precondition
      non_negative_amount: amount ≥ 0
      affordable_amount: amount <= balance -- problema
    ensure -- postcondition
      balance_deducted: balance = old balance - amount
    end
invariant -- class invariant
  positive_balance: balance > 0
end
```

Runtime Monitoring of Contracts

