Test-Driven Development (TDD)

EECS3311 A: Software Design
Fall 2018

CHEN-WEI WANG

DbC: Supplier

DbC is supported natively in Eiffel for **supplier**:

```
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, why?
              do
                      balance := balance - amount
              ensure -- postcondition
                      balance_deducted: balance = old balance - amount
              end
invariant -- class invariant
      positive_balance: balance > 0
end
```

DbC: Contract View of Supplier

Any potential **client** who is interested in learning about the kind of services provided by a **supplier** can look through the *contract view* (without showing any implementation details):

```
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
```

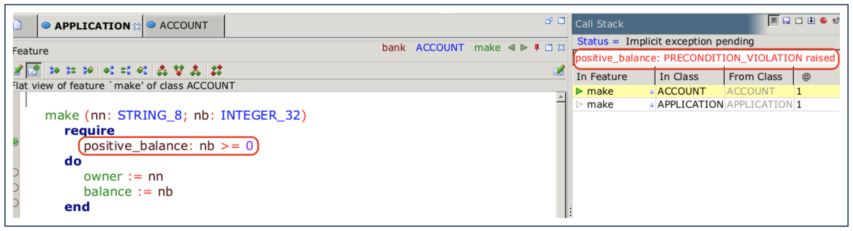
DbC: Testing Precondition Violation (1.1)

The **client** need not handle all possible contract violations:

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

By executing the above code, the runtime monitor of Eiffel Studio will report a *contract violation* (precondition violation with tag "positive_balance").

DbC: Testing for Precondition Violation (1.2)



APPLICATION | ACCOUNT

Feature — bank ACCOUNT make

Flat 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

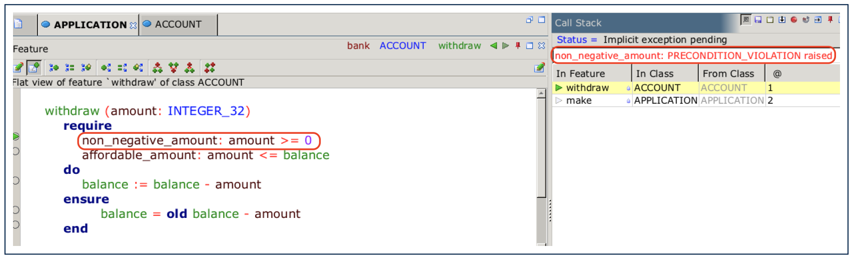
In Feature	In Class	From Class	@
make	ACCOUNT	ACCOUNT	1
make	APPLICATION	APPLICATION	1

DbC: Testing for Precondition Violation (2.1)

```
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
```

By executing the above code, the runtime monitor of Eiffel Studio will report a *contract violation* (precondition violation with tag "non_negative_amount").

DbC: Testing for Precondition Violation (2.2)



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

non_negative_amount: PRECONDITION_VIOLATION raised

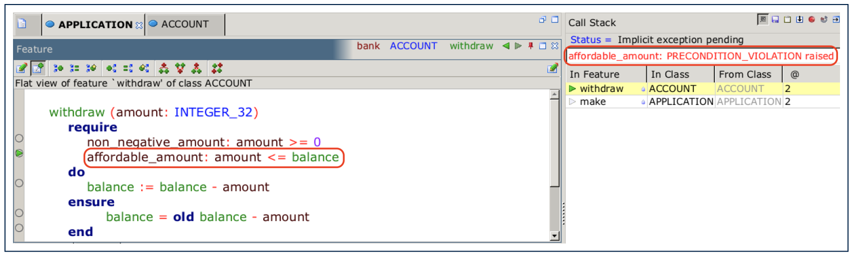
In Feature	In Class	From Class	@
withdraw	ACCOUNT	ACCOUNT	1
make	APPLICATION	APPLICATION	2

DbC: Testing for Precondition Violation (3.1)

```
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 "affordable_amount"
    tom.withdraw(150)
  end
end
```

By executing the above code, the runtime monitor of Eiffel Studio will report a *contract violation* (precondition violation with tag "affordable_amount").

DbC: Testing for Precondition Violation (3.2)



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

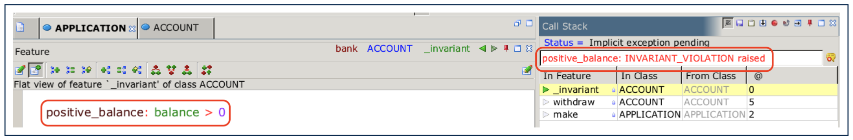
In Feature	In Class	From Class	@
withdraw	ACCOUNT	ACCOUNT	2
make	APPLICATION	APPLICATION	2

DbC: Testing for Class Invariant Violation (4.1)

```
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
```

By executing the above code, the runtime monitor of Eiffel Studio will report a *contract violation* (class invariant violation with tag "positive_balance").

DbC: Testing for Class Invariant Violation (4.2)



APPLICATION | ACCOUNT

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

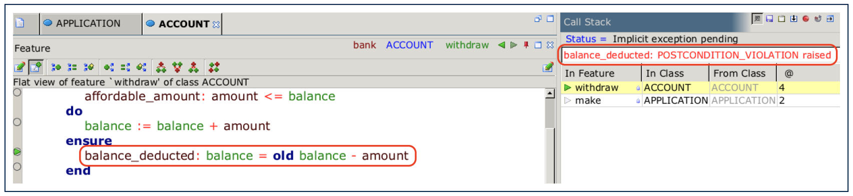
In Feature	In Class	From Class	@
_invariant	ACCOUNT	ACCOUNT	0
withdraw	ACCOUNT	ACCOUNT	5
make	APPLICATION	APPLICATION	2

DbC: Testing for Class Invariant Violation (5.1)

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

By executing the above code, the runtime monitor of Eiffel Studio will report a *contract violation* (postcondition violation with tag "balance_deducted").

DbC: Testing for Class Invariant Violation (5.2)



Flat view of feature `withdraw' of class ACCOUNT

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

Status = Implicit exception pending

balance_deducted: POSTCONDITION_VIOLATION raised

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

TDD: Test-Driven Development (1)

- How we have tested the software so far:
  - Executed each test case **manually** (by clicking `Run` in EStudio).
  - Compared **with our eyes** if ***actual results*** (produced by program) match ***expected results*** (according to requirements).
- Software is subject to numerous revisions before delivery.
  - ⇒ Testing manually, repetitively, is tedious and error-prone.
  - ⇒ We need *automation* in order to be cost-effective.
- *Test-Driven Development*
  - Test Case:
    - ***normal*** scenario (**expected** outcome)
    - ***abnormal*** scenario (**expected** contract violation).
  - Test Suite: Collection of test cases.
    - ⇒ A test suite is supposed to measure "correctness" of software.
    - ⇒ The larger the suite, the more confident you are.

TDD: Test-Driven Development (2)

- Start writing tests as soon as your code becomes *executable* :
  - with ***a unit of functionality*** completed
  - or even with ***headers*** of your features completed

```
class STACK[G]
create make
-- No implementation
feature -- Queries
  top: G do end
feature -- Commands
  make do end
  push (v: G) do end
  pop do end
end
```

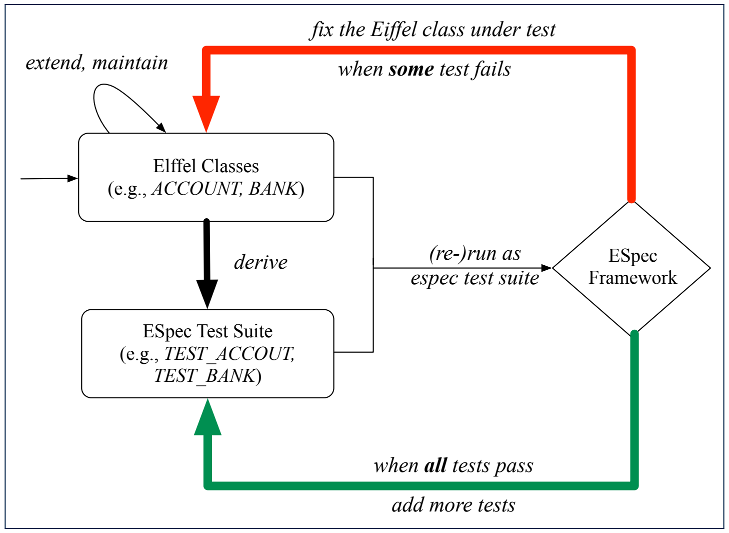
```
class TEST_STACK
...
  test_lifo: BOOLEAN
    local s: STACK[STRING]
    do create s.make
        s.push ("Alan") ; s.push ("Mark")
        Result := s.top ~ "Mark"
        check Result end
        s.pop
        Result := s.top ~ "Alan"
    end
end
```

- Writing tests should ***not*** be an isolated, last-staged activity.
- Tests are a precise, executable form of *documentation* that can guide your design.

TDD: Test-Driven Development (3)

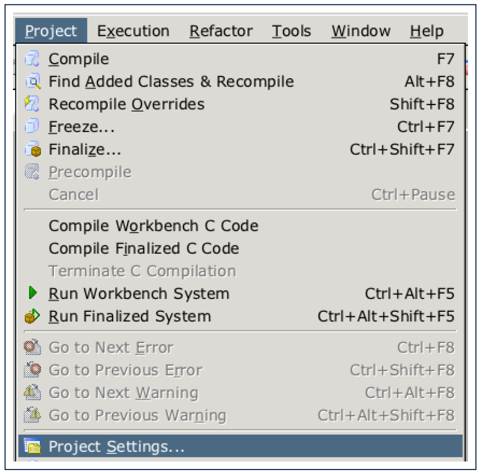
- The *ESpec* (Eiffel Specification) library is a framework for:
  - Writing and accumulating ***test cases***
    - Each list of *relevant test cases* is grouped into an `ES_TEST` class, which is just an Eiffel class that you can execute upon.
  - Executing the ***test suite*** whenever software undergoes a change
    - e.g., a bug fix
    - e.g., extension of a new functionality
- ESpec tests are *helpful client* of your classes, which may:
  - Either attempt to use a feature in a ***legal*** way (i.e., ***satisfying*** its precondition), and report:
    - ***Success*** if the result is as expected
    - ***Failure*** if the result is ***not*** as expected:
      - e.g., state of object has not been updated properly
      - e.g., a ***postcondition violation*** or ***class invariant violation*** occurs
  - Or attempt to use a feature in an ***illegal*** way (e.g., ***not satisfying*** its precondition), and report:
    - ***Success*** if precondition violation occurs.
    - ***Failure*** if precondition violation does ***not*** occur.

TDD: Test-Driven Development (4)



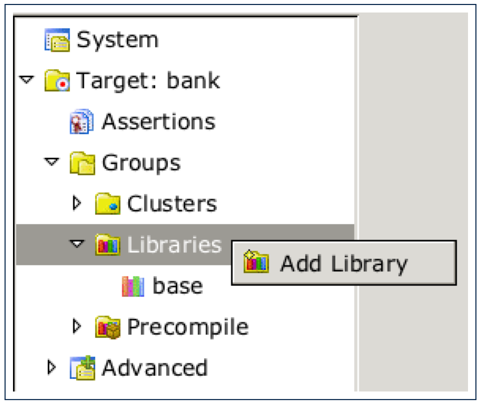
Adding the ESpec Library (1)

Step 1: Go to Project Settings.


Project Execution Refactor Tools Window Help
Compile F7
Find Added Classes & Recompile Alt+F8
Recompile Overrides Shift+F8
Freeze... Ctrl+F7
Finalize... Ctrl+Shift+F7
Precompile
Cancel Ctrl+Pause
Compile Workbench C Code
Compile Finalized C Code
Terminate C Compilation
Run Workbench System Ctrl+Alt+F5
Run Finalized System Ctrl+Alt+Shift+F5
Go to Next Error Ctrl+F8
Go to Previous Error Ctrl+Shift+F8
Go to Next Warning Ctrl+Alt+F8
Go to Previous Warning Ctrl+Alt+Shift+F8
Project Settings...

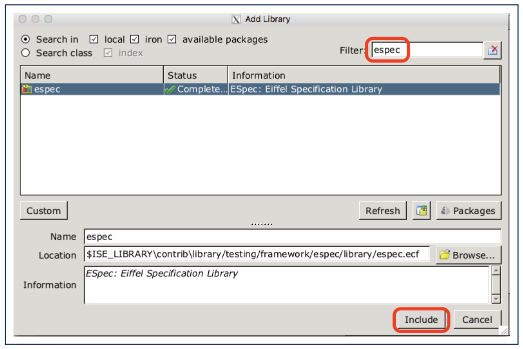
Adding the ESpec Library (2)

Step 2: Right click on `Libraries` to add a library.



Adding the ESpec Library (3)

Step 3: Search for `espec` and then include it.



(●) Search in ☑ local ☑ iron ☑ available packages
( ) Search class ☑ index
Filter: espec

Name	Status	Information
espec	✓ Complete...	ESpec: Eiffel Specification Library

Name: espec
Location: $ISE_LIBRARY\contrib\library/testing/framework/espec/library/espec.ecf
Information: *ESpec: Eiffel Specification Library*

This will make two classes available to you:

- `ES_TEST` for adding test cases
- `ES_SUITE` for adding instances of `ES_TEST`.
    - To run, an instance of this class must be set as the `root`.

ES_TEST: Expecting to Succeed (1)

```
1  class TEST_ACCOUNT
2  inherit ES_TEST
3  create make
4  feature -- Add tests in constructor
5   make
6     do
7      add_boolean_case (agent test_valid_withdraw)
8     end
9  feature -- Tests
10  test_valid_withdraw: BOOLEAN
11    local
12     acc: ACCOUNT
13    do
14     comment("test: normal execution of withdraw feature")
15     create {ACCOUNT} acc.make ("Alan", 100)
16     Result := acc.balance = 100
17     check Result end
18     acc.withdraw (20)
19     Result := acc.balance = 80
20    end
21 end
```

ES_TEST: Expecting to Succeed (2)

- **L2**: A test class is a subclass of `ES_TEST`.
- **L10 – 20** define a `BOOLEAN` test *query*. At runtime:
  - ***Success***: Return value of `test_valid_withdraw` (final value of variable **`Result`**) evaluates to *true* upon its termination.
  - ***Failure***:
    - The return value evaluates to *false* upon termination; or
    - Some contract violation (which is *unexpected*) occurs.
- **L7** calls feature `add_boolean_case` from `ES_TEST`, which expects to take as input a *query* that returns a Boolean value.
  - We pass *query* `test_valid_withdraw` as an input.
  - Think of the keyword `agent` acts like a function pointer.
    - `test_invalid_withdraw` alone denotes its return value
    - **`agent`** `test_invalid_withdraw` denotes address of *query*
- **L14**: Each test feature ***must*** call **`comment(...)`** (inherited from `ES_TEST`) to include the description in test report.
- **L17**: Check that *each* intermediate value of `Result` is ***true***.

ES_TEST: Expecting to Succeed (3)

- Why is the `check Result end` statement at **L7** necessary?
  - When there are two or more *assertions* to make, some of which (except the last one) may ***temporarily falsify*** return value **Result**.
  - As long as the last *assertion* assigns ***true*** to **Result**, then the entire *test query* is considered as a ***success***.

    ⇒ A ***false positive*** is possible!
- For the sake of demonstrating a false positive, imagine:
  - Constructor `make` ***mistakenly*** deduces 20 from input amount.
  - Command `withdraw` ***mistakenly*** deducts nothing.

```
1  test_query_giving_false_positive: BOOLEAN
2   local acc: ACCOUNT
3   do comment("Result temporarily false, but finally true.")
4      create {ACCOUNT} acc.make ("Jim", 100) -- balance set as 80
5      Result := acc.balance = 100 -- Result assigned to false
6      acc.withdraw (20) -- balance not deducted
7      Result := acc.balance = 80 -- Result re-assigned to true
8      -- Upon termination, Result being true makes the test query
9      -- considered as a success ==> false positive!
10  end
```

Fix? [insert ***check Result end***] between **L6** and **L7**.

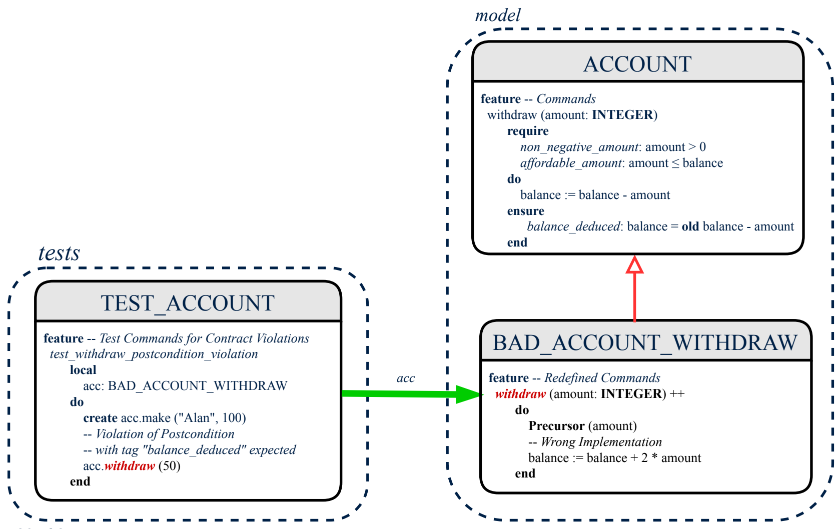
ES_TEST: Expecting to Fail Precondition (1)

```
class TEST_ACCOUNT
inherit ES_TEST
create make
feature -- Add tests in constructor
 make
  do
   add_violation_case_with_tag ("non_negative_amount",
    agent test_withdraw_precondition_violation)
  end
feature -- Tests
 test_withdraw_precondition_violation
  local
   acc: ACCOUNT
  do
   comment("test: expected precondition violation of withdraw")
   create {ACCOUNT} acc.make ("Mark", 100)
   -- Precondition Violation
   -- with tag "non_negative_amount" is expected.
   acc.withdraw (-1000000)
  end
end
```

ES_TEST: Expecting to Fail Precondition (2)

- **L2**: A test class is a subclass of `ES_TEST`.
- **L11 – 20** define a test *command*. At runtime:
  - ***Success***: A precondition violation (with tag `"non_negative_amount"`) occurs at **L19** before its termination.
  - ***Failure***:
    - No contract violation with the expected tag occurs before its termination; or
    - Some other contract violation (with a different tag) occurs.
- **L7** calls feature `add_violation_case_with_tag` from `ES_TEST`, which expects to take as input a *command*.
  - We pass *command* `test_invalid_withdraw` as an input.
  - Think of the keyword `agent` acts like a function pointer.
    - `test_invalid_withdraw` alone denotes a call to it
    - `agent test_invalid_withdraw` denotes address of *command*
- **L15**: Each test feature ***must*** call `comment(...)` (inherited from `ES_TEST`) to include the description in test report.

ES_TEST: Expecting to Fail Postcondition (1)



model
ACCOUNT
feature -- Commands
withdraw (amount: INTEGER)
require
non_negative_amount: amount > 0
affordable_amount: amount ≤ balance
do
balance := balance - amount
ensure
balance_deducted: balance = old balance - amount
end
tests
TEST_ACCOUNT
feature -- Test Commands for Contract Violations
test_withdraw_postcondition_violation
local
acc: BAD_ACCOUNT_WITHDRAW
do
create acc.make ("Alan", 100)
-- Violation of Postcondition
-- with tag "balance_deduced" expected
acc.withdraw (50)
end
acc
BAD_ACCOUNT_WITHDRAW
feature -- Redefined Commands
withdraw (amount: INTEGER) ++
do
Precursor (amount)
-- Wrong Implementation
balance := balance + 2 * amount
end

ES_TEST: Expecting to Fail Postcondition (2.1)

```
1  class
2    BAD_ACCOUNT_WITHDRAW
3  inherit
4    ACCOUNT
5      redefine withdraw end
6  create
7    make
8  feature -- redefined commands
9    withdraw(amount: INTEGER)
10     do
11       Precursor(amount)
12       -- Wrong implementation
13       balance := balance + 2 * amount
14     end
15 end
```

- **L3–5**: `BAD_ACCOUNT_WITHDRAW.withdraw` inherits postcondition from `ACCOUNT.withdraw`: `balance = old balance - amount`.
- **L11** calls *correct* implementation from parent class `ACCOUNT`.
- **L13** makes overall implementation *incorrect*.

ES_TEST: Expecting to Fail Postcondition (2.2)

```
class TEST_ACCOUNT
inherit ES_TEST
create make
feature -- Constructor for adding tests
  make
    do
      add_violation_case_with_tag ("balance_deducted",
       agent test_withdraw_postcondition_violation)
    end
feature -- Test commands (test to fail)
  test_withdraw_postcondition_violation
    local
      acc: BAD_ACCOUNT_WITHDRAW
    do
      comment ("test: expected postcondition violation of withdraw")
      create acc.make ("Alan", 100)
      -- Postcondition Violation with tag "balance_deduced" to occur.
      acc.withdraw (50)
    end
end
```

Exercise

Recall from the “Writing Complete Postconditions” lecture:

```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    do ... -- Put Correct Implementation Here.
    ensure
      ...
      others_unchanged :
        across old accounts.deep_twin as cursor
        all cursor.item.owner /~ n implies
              cursor.item ~ account_of (cursor.item.owner)
        end
    end
end
```

How do you create a “bad” descendant of BANK that violates this postcondition?

```
class BAD_BANK_DEPOSIT
inherit BANK redefine deposit end
feature -- redefined feature
  deposit_on_v5 (n: STRING; a: INTEGER)
    do Precursor (n, a)
      accounts[accounts.lower].deposit(a)
    end
end
```

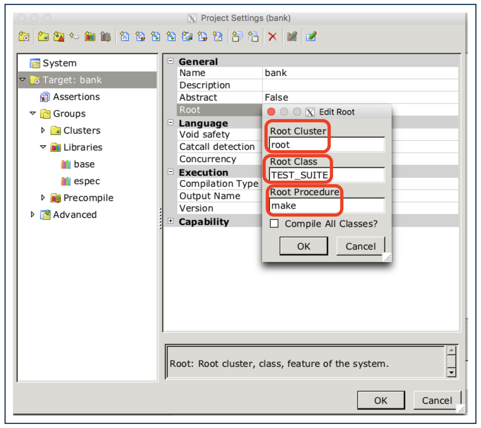
ES_SUITE: Collecting Test Classes

```
1   class TEST_SUITE
2   inherit ES_SUITE
3   create make
4   feature -- Constructor for adding test classes
5     make
6       do
7         add_test (create {TEST_ACCOUNT}.make)
8         show_browser
9         run_espec
10      end
11  end
```

- **L2**: A test suite is a subclass of ES_SUITE.
- **L7** passes an **anonymous** object of type TEST_ACCOUNT to add_test inherited from ES_SUITE).
- **L8 & L9** have to be entered in this order!

Running ES_SUITE (1)

Step 1: Change the *root class* (i.e., entry point of execution) to be TEST_SUITE.



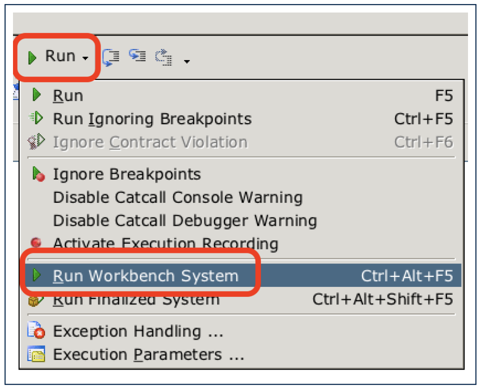
Name: bank
Description:
Abstract: False
Root

Root Cluster: root
Root Class: TEST_SUITE
Root Procedure: make

Root: Root cluster, class, feature of the system.

Running ES_SUITE (2)

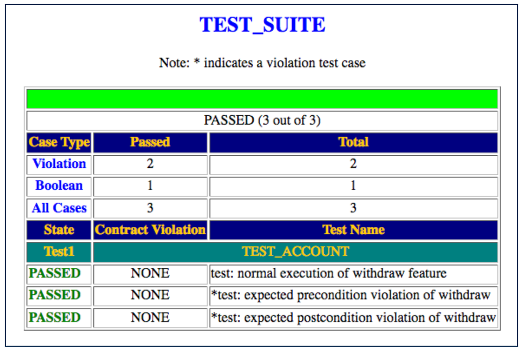
Step 2: Run the **Workbench System**.



```
Run                               F5
Run Ignoring Breakpoints          Ctrl+F5
Ignore Contract Violation         Ctrl+F6
Ignore Breakpoints
Disable Catcall Console Warning
Disable Catcall Debugger Warning
Activate Execution Recording
Run Workbench System              Ctrl+Alt+F5
Run Finalized System              Ctrl+Alt+Shift+F5
Exception Handling ...
Execution Parameters ...
```

Running `ES_SUITE` (3)

Step 3: See the generated test report.



TEST_SUITE

Note: * indicates a violation test case

🟩		
PASSED (3 out of 3)		
Case Type	**Passed**	**Total**
Violation	2	2
Boolean	1	1
All Cases	3	3
State	**Contract Violation**	**Test Name**
Test1	**TEST_ACCOUNT**	
PASSED	NONE	test: normal execution of withdraw feature
PASSED	NONE	*test: expected precondition violation of withdraw
PASSED	NONE	*test: expected postcondition violation of withdraw

Beyond this lecture...

- Study this tutorial series on DbC and TDD:
  `https://www.youtube.com/playlist?list=PL5dxAmCmjv_6r5VfzCQ5bTznoDDgh__KS`

Index (1)

Index (2)

Index (3)