JMLUnitNG, Representation Hiding

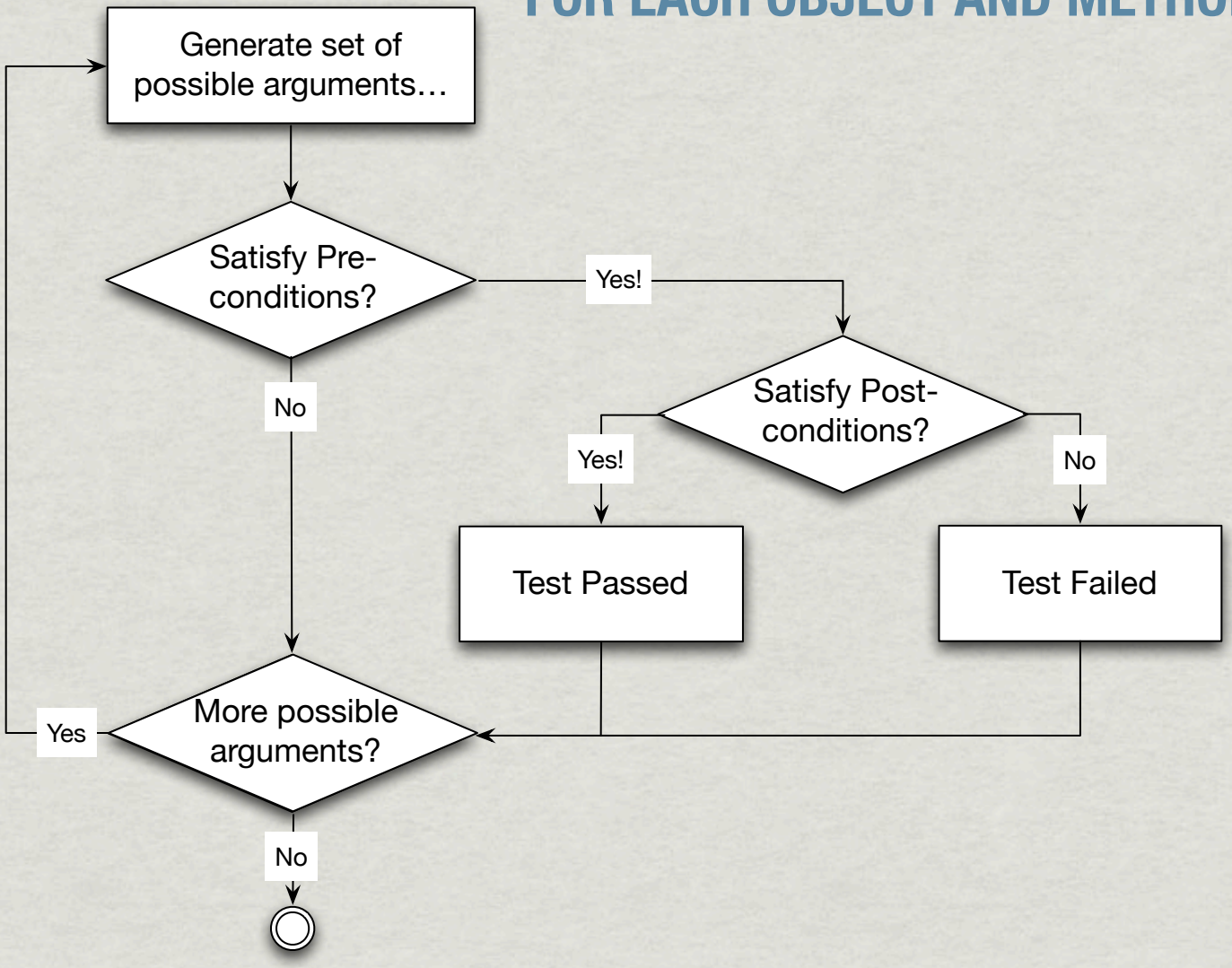
Curt Clifton
Rose-Hulman Institute of Technology

Goals for Today

- See how JMLUnitNG can automate unit testing
- See how JML can be used to specify interfaces while maintaining encapsulation

JMLUnitNG uses JML specifications as "test oracles" for automated unit testing.

FOR EACH OBJECT AND METHOD TO BE TESTED…



To use JMLUnitNG you need to install TestNG in Eclipse.

- Select Help / Install new software / Add...
- Enter:
  - Name: **TestNG**
  - Location: **http://beust.com/eclipse**
- Back in the Install dialog, check TestNG
- Follow the dialogs
- Restart Eclipse when prompted

Check out the **JMLUnitTestExample** project in Eclipse from your individual SVN repo.

JMLUnitNG lets us specify test data if we want more control.

- Uses the Strategy design pattern
- Different Strategy objects to specify custom data for different tests

http://formalmethods.insttech.washington.edu/software/jmlunitng/usage.html

To specify a class, specify its data, its invariants, and its behaviors.

- 1. Specify the data…
  - Using **spec_public** fields for now
  - Later we'll see how to be more abstract
- 2. Specify **invariant**s
- 3. Specify each public method/constructor using:
  - **requires**, **assignable**, and **ensures**

Writing specs in terms of private fields violates information hiding.

- Solution:
  - **model** fields
  - *data groups*
  - **represents** clauses

Model fields give an abstract representation of an object's state.

- Choose a model representation that is logically clear, even if it is inefficient

Warning!

- I’m using model fields for the example.
- For HW10 and P3 you want to use direct specifications like in the Eclipse projects.

Interface Model

INDICATES THAT THIS IS A "SPECIFICATION ONLY" FIELD

FIELDS IN JAVA INTERFACES ARE STATIC BY DEFAULT

```
public interface IBoundedStack {
    //@ public model instance int size;
    //@ public model instance int capacity;
    /*@ public model instance
      @                       JMLObjectSequence contents;
      @*/
```

JML USES A REGULAR JAVA LIBRARY TO REPRESENT MATHEMATICAL IDEAS

Interface Invariant


```
//@ public invariant 0 <= size;
//@ public invariant size <= capacity;
//@ public invariant size == contents.length();
```

We need to say how the concrete fields match the model.

- Use data groups and represents clauses

Data groups say that whenever a model field changes, this concrete field can too.

- Denoted by **in** and **maps**

Represents clauses let the tools check that private fields match public specifications.

- **represents modelField <- *<expr>***
- **represents modelField \such_that *<pred>***

Data Groups and Represents Clauses—I

DATA GROUP

```
public class BoundedStack {
    private int _size; //@ in size;
        //@ represents size = _size;
```

Data Groups and Represents Clauses—2


```
private Object[] _elems; //@ in capacity, contents
    //@ represents capacity = _elems.length;
    /*@ represents contents \such_that
      @     (\forall int k; 0 <= k && k < _size;
      @           _elems[k] == contents.itemAt(k);
      @*/
```

INITIALLY CLAUSES CONSTRAIN ALL CONSTRUCTORS

```
//@ public initially size == 0;
```

SPECIAL DATA GROUP CONTAINING ALL FIELDS

```
/*@ requires n > 0;
  @ assignable objectState;
  @ ensures capacity == n;
  @*/
public BoundedStack(int n) { … }
```

Original Rep. Exposure!

```
/*@ requires n > 0;
  @ assignable _elems, _size;
  @ ensures _elems.length == n && _size == 0;
  @*/
```

Does Implementation Meet Specification?

CLASS' CODE `this._elems = new Object[n];`

BY SEMANTICS OF ASSIGNMENT

CONCRETE POSTCONDITION `_elems.length == n;`

BY REPRESENTS CLAUSE

MODEL'S POSTCONDITION `capacity == n;`

Method Specifications Using the Model


```
/*@ requires size < capacity;
  @ assignable size, contents;
  @ ensures size == \old(size) + 1;
  @ ensures contents.itemAt(size - 1) == x;
  @*/
```

COMPARE TO ORIGINAL SPEC USING CONCRETE REPRESENTATION

```
/*@ requires _size < _elems.length;
  @ assignable _elems[_size], _size;
  @ ensures _size == \old(_size) + 1;
  @ ensures _elems[_size-1] == x;
  @*/
```

WITH MODEL-BASED SPECIFICATION
NO ADDITIONAL SPEC IS NEEDED FOR CONCRETE IMPLEMENTATION

```
public interface IBoundedStack {
        //@ public model instance int size;
        //@ public model instance int capacity;
        //@ public model instance JMLObjectSequence contents;

        //@ public invariant 0 <= size;
        //@ public invariant size <= capacity;
        //@ public invariant size == contents.length();

        //@ public initially size == 0;

        /*@ requires size < capacity;
          @ assignable size, contents;
          @ ensures size == \old(size) + 1;
          @ ensures contents.itemAt(size - 1) == x;
          @*/
        public void push(Object x);

        /*@ requires size > 0;
          @ assignable size, contents;
          @ ensures size == \old(size - 1);
          @*/
        public void pop();

        /*@ requires size > 0;
          @ assignable \nothing;
          @ ensures \result == contents.itemAt(size - 1);
          @*/
        public Object top();
}
```

```
public class BoundedStack {
        private int _size; //@ in size;
                //@ represents size = _size;
        private Object[] _elems; //@ in capacity, contents
                //@ represents capacity = _elems.length;
                /*@ represents contents \such_that
                  @     (\forall int k; 0 <= k && k < _size;
                  @          _elems[k] == contents.itemAt(k);
                  @*/

        //@ private invariant \typeof(_elems) == \type(Object[]);
        /*@ private invariant (\forall int i;
          @                              size <= i && i < capacity;
          @                              _elems[i] == null);
          @*/

        /*@ requires n > 0;
          @ assignable objectState;
          @ ensures capacity == n;
          @*/
        public BoundedStack(int n) {
                this._elems = new Object[n];
                this._size = 0;
        }

        public void push(Object x) {
                this._elems[this._size] = x;
                this._size++;
        }

        public void pop() {
                this._size--;
                this._elems[this._size] = null;
        }

        public Object top() {
                return this._elems[this._size - 1];
        }
}
```

Warning!

- I used model fields for the example.
- For HW10 and P3 you want to use direct specifications like in the Eclipse projects.