

Data Modeling with JML

Curt Clifton

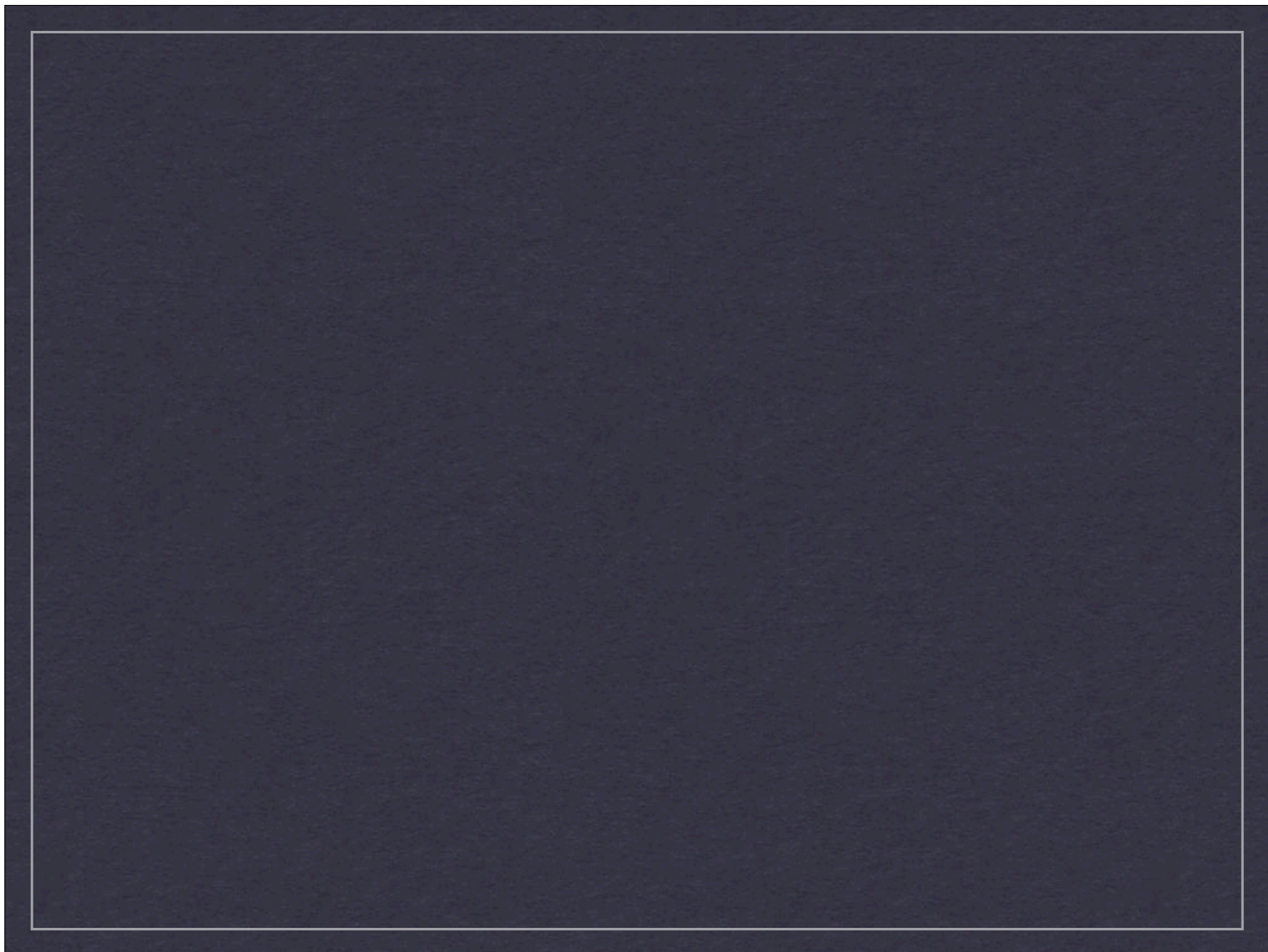
Rose-Hulman Institute of Technology

DBC for Methods

- * Pre-conditions – **requires**
- * Post-conditions – **ensures**
- * Frame axioms – **assignable**

DBC for Objects

- * Besides specifying what methods do, we also have to constrain what the objects can do
- * We use *class invariants* for this
- * Example: `//@ invariant size <= capacity;`



An Example

An Example, but first...

Example

- * BoundedStack
 - * A stack with a fixed maximum size
- * Live-coding in Mobius PVE...

Starting Projects in Mobius PVE

- * Create a new workspace
- * Create a new Java project
- * Set the project's "nature" to ESC/Java
 - * Select project
 - * Choose *JML* → *Setup* → *Enable ESC/Java nature*

The Outline

The Outline

```
* public class BoundedStack {  
    public BoundedStack(int maxSize) {...}  
    public void push(Object x) {...}  
    public void pop() {...}  
    public Object top() {...}  
}
```

1. Specify the Data

```
public class BoundedStack {  
    private /*@ spec_public @*/ int size;  
    private /*@ spec_public @*/  
        Object[] elems;  
    ...  
}
```

1. Specify the Data

LETS US TALK ABOUT size IN OUR SPECIFICATION

```
public class BoundedStack {  
    private /*@ spec_public @*/ int size;  
    private /*@ spec_public @*/  
        Object[] elems;  
    ...  
}
```

2. Specify the Invariant

```
public class BoundedStack {  
    private /*@ spec_public @*/ int size;  
    private /*@ spec_public @*/ Object[] elems;
```

2. Specify the Invariant

```
public class BoundedStack {
    private /*@ spec_public @*/ int size;
    private /*@ spec_public @*/ Object[] elems;
    //@ public invariant 0 <= size
    //@ public invariant size <= elems.length;
    //@ public invariant elems != null;
    //@ public invariant \typeof(elems) == \type(Object[]);
    /*@ public invariant (\forall int i;
        @           size <= i && i < elems.length;
        @           elems[i] == null);
    @*/
    //@ public invariant this.elems.owner == this;
```

2. Specify the Invariant

```
public class BoundedStack {
    private /*@ spec_public @*/ int size;
    private /*@ spec_public @*/ Object[] elems;
    //@ public invariant 0 <= size
    //@ public invariant size <= elems.length;
    //@ public invariant elems != null;
    //@ public invariant \typeof(elems) == \type(Object[]);
    /*@ public invariant (\forall int i;
        @           size <= i && i < elems.length;
        @           elems[i] == null);
    @*/
    //@ public invariant this.elems.owner == this;
```

DISALLOWS JAVA WEIRDNESS
LIKE `elems = new String[]`

2. Specify the Invariant

```
public class BoundedStack {  
    private /*@ spec_public @*/ int size;  
    private /*@ spec_public @*/ Object[] elems;  
    /*@ public invariant 0 <= size  
    //@ public invariant size <= elems.length;  
    //@ public invariant elems != null;  
    //@ public invariant \typeof(elems) == \type(Object[]);  
    /*@ public invariant (\forallall int i;  
        @           size <= i && i < elems.length;  
        @           elems[i] == null);  
    @*/  
    //@ public invariant this.elems.owner == this;
```

TELLS ESC/JAVA2 THAT
ELEMS SHOULDN'T BE
ALIASED

DISALLOWS JAVA WEIRDNESS
LIKE elems = new String[]

3. Specify Public Constructors/Methods

3. Specify Public Constructors/Methods

```
/*@  
  @  
  @  
  @*/  
public BoundedStack(int n) {  
  elems = new Object[n];  
  size = 0;  
  //@ set this.elems.owner = this;  
}
```

3. Specify Public Constructors/Methods

```
/*@ requires n > 0;  
@  
@  
@*/  
public BoundedStack(int n) {  
    elems = new Object[n];  
    size = 0;  
    //@ set this.elems.owner = this;  
}
```

3. Specify Public Constructors/Methods

```
/*@ requires n > 0;  
@ assignable elems, size;  
@  
@*/  
public BoundedStack(int n) {  
    elems = new Object[n];  
    size = 0;  
    //@ set this.elems.owner = this;  
}
```

3. Specify Public Constructors/Methods

```
/*@ requires n > 0;  
  @ assignable elems, size;  
  @ ensures elems.length == n && size == 0;  
  @*/  
public BoundedStack(int n) {  
    elems = new Object[n];  
    size = 0;  
    //@ set this.elems.owner = this;  
}
```

3. Specify Public Constructors/Methods

```
/*@ requires n > 0;  
  @ assignable elems, size;  
  @ ensures elems.length == n && size == 0;  
  @*/  
public BoundedStack(int n) {  
    elems = new Object[n];  
    size = 0;  
    //@ set this.elems.owner = this;  
}
```

TELLS ESC/JAVA2 THAT
ELEMS ISN'T ALIASED

Don't Get Pushy

Don't Get Pushy

```
/*@  
  @  
  @  
  @  
  @*/  
public void push(Object x) {  
    elems[size] = x;  
    size = size + 1;  
}
```

Don't Get Pushy

```
/*@ requires size < elems.length;  
@  
@  
@  
@*/  
public void push(Object x) {  
    elems[size] = x;  
    size = size + 1;  
}
```

Don't Get Pushy

```
/*@ requires size < elems.length;  
  @ assignable elems[size], size;  
  @  
  @  
  @*/  
public void push(Object x) {  
    elems[size] = x;  
    size = size + 1;  
}
```

Don't Get Pushy

```
/*@ requires size < elems.length;  
  @ assignable elems[size], size;  
  @ ensures size == \old(size) + 1;  
  @  
  @*/  
public void push(Object x) {  
    elems[size] = x;  
    size = size + 1;  
}
```

Don't Get Pushy

```
/*@ requires size < elems.length;  
  @ assignable elems[size], size;  
  @ ensures size == \old(size) + 1;  
  @ ensures elems[size-1] == x;  
  @*/  
public void push(Object x) {  
    elems[size] = x;  
    size = size + 1;  
}
```

All Around the Shoemaker's Bench...

All Around the Shoemaker's Bench...

```
/*@  
  @  
  @  
  @*/  
public void pop() {  
    size = size - 1;  
    elems[size] = null;  
}
```

All Around the Shoemaker's Bench...

```
/*@ requires size > 0;  
  @  
  @  
  @*/  
public void pop() {  
    size = size - 1;  
    elems[size] = null;  
}
```

All Around the Shoemaker's Bench...

```
/*@ requires size > 0;  
  @ assignable size, elems[size-1];  
  @  
  @*/  
public void pop() {  
    size = size - 1;  
    elems[size] = null;  
}
```

All Around the Shoemaker's Bench...

```
/*@ requires size > 0;  
  @ assignable size, elems[size-1];  
  @ ensures size == \old(size-1);  
  @*/  
public void pop() {  
    size = size - 1;  
    elems[size] = null;  
}
```

All Around the Shoemaker's Bench...

```
/*@ requires size > 0;  
  @ assignable size, elems[size-1];  
  @ ensures size == \old(size-1);  
  @*/  
public void pop() {  
    size = size - 1;  
    elems[size] = null;  
}
```

REMEMBER THE INVARIANT INCLUDED:

$(\forall \text{forall int } i; \text{size} \leq i \ \&\& \ i < \text{elems.length}; \text{elems}[i] == \text{null})$

I'm on the Top of the
World...

I'm on the Top of the World...

```
/*@  
  @  
  @  
  @*/  
public Object top() {  
    return elems[size-1];  
}
```

I'm on the Top of the World...

```
/*@ requires size > 0;  
@  
@  
@*/  
public Object top() {  
    return elems[size-1];  
}
```

I'm on the Top of the World...

```
/*@ requires size > 0;  
  @ assignable \nothing;  
  @  
  @*/  
public Object top() {  
    return elems[size-1];  
}
```

I'm on the Top of the World...

```
/*@ requires size > 0;  
  @ assignable \nothing;  
  @ ensures \result == elems[size-1];  
  @*/  
public Object top() {  
    return elems[size-1];  
}
```

Steps for Specifying a Class

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