Functions with Variants

```
; An animal is either
;  - snake
;  - dillo
;  - ant

; animal-is-lighter? : animal num -> bool
(define (animal-is-lighter? a n)
  (cond
    [(snake? a) (snake-is-lighter? s n)]
    [(dillo? a) (dillo-is-ligheter? s n)]
    [(ant? a) (ant-is-lighter? s n)]))

; snake-is-lighter? : snake num -> bool
(define (snake-is-lighter? s n) ...)

; dillo-is-lighter? : dillo num -> bool
(define (dillo-is-lighter? d n) ...)

; ant-is-lighter? : ant num -> bool
(define (ant-is-lighter? a n) ...)
```

Methods with Variants

```
abstract class Animal {
  abstract boolean isLighter(double n);
}

class Snake extends Animal {
  ...
  boolean isLighter(double n) { ... }
}

class Dillo extends Animal {
  ...
  boolean isLighter(double n) { ... }
}

class Ant extends Animal {
  ...
  boolean isLighter(double n) { ... }
}
```

Translating Functions to Methods

```
; An animal is either
;  - snake
;  - dillo
;  - ant

; animal-is-lighter? : animal num -> bool
(define (animal-is-lighter? a n)
  (cond
    [(snake? a) (snake-is-lighter? s n)]
    [(dillo? a) (dillo-is-ligheter? s n)]
    [(ant? a) (ant-is-lighter? s n)]))

; snake-is-lighter? : snake num -> bool
(define (snake-is-lighter? s n) ...)

; dillo-is-lighter? : dillo num -> bool
(define (dillo-is-lighter? d n) ...)

; ant-is-lighter? : ant num -> bool
(define (ant-is-lighter? a n) ...)
```

```
abstract class Animal {
  abstract boolean isLighter(double n);
}

class Snake extends Animal {
  ...
  boolean isLighter(double n) { ... }
}

class Dillo extends Animal {
  ...
  boolean isLighter(double n) { ... }
}

class Ant extends Animal {
  ...
  boolean isLighter(double n) { ... }
}
```

Translating Functions to Methods

```
; An animal is either
;  - snake
;  - dillo
;  - ant

; animal-is-lighter? : animal num -> bool
(define (animal-is-lighter? a n)
  (cond
    [(snake? a) (snake-is-lighter? s n)]
    [(dillo? a) (dillo-is-ligheter? s n)]
    [(ant? a) (ant-is-lighter? s n)]))

; snake-is-lighter? : snake num -> bool
(define (snake-is-lighter? s n) ...)

; dillo-is-lighter? : dillo num -> bool
(define (dillo-is-lighter? d n) ...)

; ant-is-lighter? : ant num -> bool
(define (ant-is-lighter? a n) ...)
```

Data definition turns into class declarations

```
abstract class Animal {
  abstract boolean isLighter(double n);
}

class Snake extends Animal {
  ...
  boolean isLighter(double n) { ... }
}

class Dillo extends Animal {
  ...
  boolean isLighter(double n) { ... }
}

class Ant extends Animal {
  ...
  boolean isLighter(double n) { ... }
}
```

Translating Functions to Methods

```
; An animal is either
;  - snake
;  - dillo
;  - ant

; animal-is-lighter? : animal num -> bool
(define (animal-is-lighter? a n)
  (cond
    [(snake? a) (snake-is-lighter? s n)]
    [(dillo? a) (dillo-is-ligheter? s n)]
    [(ant? a) (ant-is-lighter? s n)]))

; snake-is-lighter? : snake num -> bool
(define (snake-is-lighter? s n) ...)

; dillo-is-lighter? : dillo num -> bool
(define (dillo-is-lighter? d n) ...)

; ant-is-lighter? : ant num -> bool
(define (ant-is-lighter? a n) ...)
```

Variant functions turn into variant methods — all with the same contract after the implicit argument

```
abstract class Animal {
  abstract boolean isLighter(double n);
}

class Snake extends Animal {
  ...
  boolean isLighter(double n) { ... }
}

class Dillo extends Animal {
  ...
  boolean isLighter(double n) { ... }
}

class Ant extends Animal {
  ...
  boolean isLighter(double n) { ... }
}
```

Translating Functions to Methods

```
; An animal is either
;  - snake
;  - dillo
;  - ant

; animal-is-lighter? : animal num -> bool
(define (animal-is-lighter? a n)
  (cond
    [(snake? a) (snake-is-lighter? s n)]
    [(dillo? a) (dillo-is-ligheter? s n)]
    [(ant? a) (ant-is-lighter? s n)]))

; snake-is-lighter? : snake num -> bool
(define (snake-is-lighter? s n) ...)

; dillo-is-lighter? : dillo num -> bool
(define (dillo-is-lighter? d n) ...)

; ant-is-lighter? : ant num -> bool
(define (ant-is-lighter? a n) ...)
```

Function with variant-based **cond** turns into just an **abstract** method declaration

```
abstract class Animal {
  abstract boolean isLighter(double n);
}

class Snake extends Animal {
  ...
  boolean isLighter(double n) { ... }
}

class Dillo extends Animal {
  ...
  boolean isLighter(double n) { ... }
}

class Ant extends Animal {
  ...
  boolean isLighter(double n) { ... }
}
```

Lists of Things

```
abstract class ListOfThing {
  abstract int length();
}

class EmptyListOfThing extends ListOfThing {
   EmptyListOfThing() { }
   int length() { return 0; }
}

class ConsListOfThing extends ListOfThing {
   Thing first;
   ListOfThing rest;
   ConsListOfThing(Thing first, ListOfThing rest) {
     this.first = first;
     this.rest = rest;
   }
   int length() { return 1 + this.rest.length(); }
}
```

Copy

Trees of Things

```
abstract class TreeOfThing {
  abstract int count();
}

class EmptyTreeOfThing extends TreeOfThing {
   EmptyTreeOfThing() { }
   int count() { return 0; }
}

class ConsTreeOfThing extends TreeOfThing {
   Thing v;
   TreeOfThing left;
   TreeOfThing right;
   ConsTreeOfThing(Thing v, TreeOfThing left, TreeOfThing right) {
     this.v = v;
     this.left = left;
     this.right = right;
   }
   int count() { return 1 + this.left.count()
                          + this.right.count(); }
}
```

Copy

Implementing Methods Directly

Some Scheme methods on **`animal`** can be implemented with other **`animal`** functions:

```
; animal-light? : animal -> bool
; Determines whether a is less than 10 lbs
(define (animal-light? a)
  (animal-lighter? a 10))
```

Implementing Methods Directly

Some Scheme methods on `animal` can be implemented with other `animal` functions:

```
; animal-light? : animal -> bool
; Determines whether a is less than 10 lbs
(define (animal-light? a)
  (animal-lighter? a 10))
```

In Java, this corresponds to a non-**`abstract`** method in an **`abstract`** class:

```
abstract class Animal {
   ...
   boolean isLight() {
     return this.isLighter(10);
   }
}
```

Conditionals

Some uses of **cond** where not based on the data definition:

```
(define (posn-big-part p)
  (cond
    [(> (posn-x p) (posn-y p)) (posn-x p)]
    [else (posn-y p)]))
```

For these, we use **if** in Java:

```
class Posn { ...
  double bigPart() {
    if (this.x > this.y)
      return this.x;
    else
      return this.y;
  }
}
```

Conditionals

In general:

```
(cond                          ⇒    if question1
  [question1 answer1]                 return answer1;
  [question2 answer2]               else if question2
  ...                                 return answer2;
  [else answerN])                   ...
                                    else
                                      return answerN;
```

Primitive Operations on Numbers

```
1 + 2  "should be" 3
1 - 2  "should be" -1
1 * 2  "should be" 2
1 / 2  "should be" 0
1.0 / 2.0  "should be" 0.5

1 < 2  "should be" true
1 > 2  "should be" false
1 <= 2  "should be" true
1 >= 2  "should be" false
1 == 2  "should be" false
1 == 1  "should be" true
```

Primitive Operations on Booleans

```
!true "should be" false
!false "should be" true

true && true "should be" true
true && false "should be" false
true || false "should be" true
false || false "should be" false
```

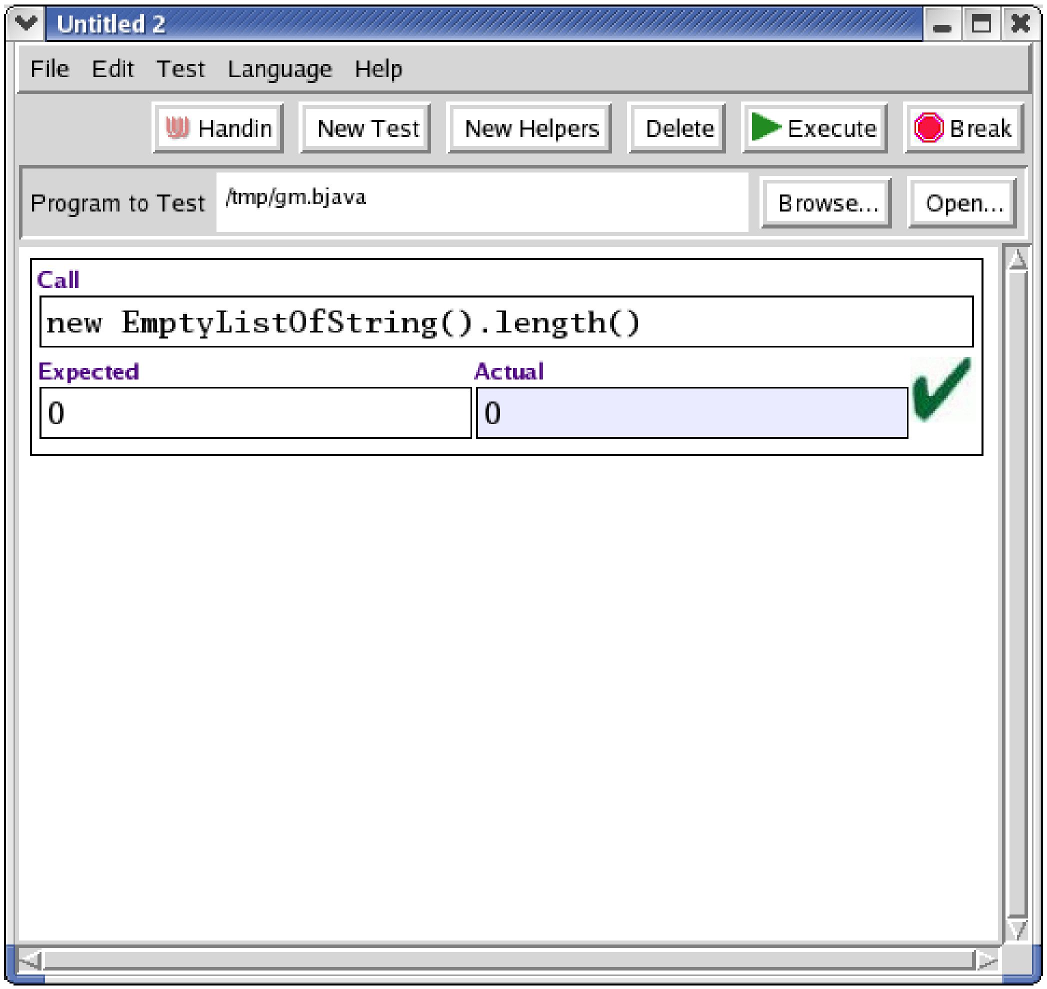
Primitive Operations on Strings

```
"hello".equals("bye") "should be" false
"hello".equals("hello") "should be" true

"good".concat(" bye") "should be" "good bye"

"good bye".startsWith("good") "should be" true
"good bye".startsWith("bye") "should be" false

"good bye".endsWith("good") "should be" false
"good bye".endsWith("bye") "should be" true
```

These operations are really method calls, and that's why **`String`** is capitalized

Testing Java Code

Select **New Test Suite** from the **File** menu



Testing Tool Warnings

- The test tool is tied to the program via the file name
  - If you rename your file, you must fix the test window
  - You must save changes to your program before testing
- To open a saved test suite, use **Open Test Suite...**
- The test tool incorrectly reports failures when the result is not a number or boolean

For HW 12, you must handin your test suite as well as your implementation