How Can We Improve the State of Experimental Evaluation in Computer Science

Peter Sweeney
IBM Research, TJ Watson
Friday, April 12, 12:00
Kendade 307

Stack

- Data structure that only allows access to the last item added to it.

class Stack<E> {
  // Add item to end
  public E push(E item) {...}

  // Get last item added
  public E peek() {...}

  // Remove last item added
  public E pop() {...}

  // Is it empty?
  public boolean empty() {...}

  // How many items are in it?
  public int size() {...}
}

Stack Traces

Exception in thread "AWT-EventQueue-0" java.lang.NullPointerException
at model.Profile.addFriend(Profile.java:85)
at gui.FacePamphlet.addFriend(FacePamphlet.java:219)
at gui.FacePamphlet.actionPerformed(FacePamphlet.java:184)
at javax.swing.JComboBox.fireActionEvent(JComboBox.java:1197)
at javax.swing.JComboBox.setSelectedItem(JComboBox.java:561)
at javax.swing.JComboBox.setSelectedIndex(JComboBox.java:597)
... (more omitted)

- Top of the stack identifies the method where the exception occurred
- 2nd item on the stack identifies where the top method was called
- And so on...
Magic!!!
**Factorial**

- Function commonly used in calculating probabilities.
- \( \text{factorial}(n) = n \times (n-1) \times (n-2) \times \ldots \times 1 \)
- What is \( \text{factorial}(n-1) \)?
- \( \text{factorial}(n-1) = (n-1) \times (n-2) \times \ldots \times 1 \)
- Therefore...
  \[ \text{factorial}(n) = n \times \text{factorial}(n-1) \]

```java
private int factorial (int n) {
    assert n >= 0;
    if (n == 0) {
        return 1;
    }
    return n * factorial(n-1);
}
```

Let's write a factorial method that way!

```java
private int factorial (int n) {
    assert n >= 0;
    if (n == 0) {
        return 1;
    }
    return n * factorial(n-1);
}
```

But does it work???

- \( \text{factorial}(3) => 3 \times \text{factorial}(2) \)
  \( \text{factorial}(2) => 2 \times \text{factorial}(1) \)
  \( \text{factorial}(1) => 1 \times \text{factorial}(0) \)
  \( \text{factorial}(0) => 1 \)

\( \text{factorial}(3) \Rightarrow 3 \times \text{factorial}(2) = 2 \times 1 = 6 \)
\( \text{factorial}(2) \Rightarrow 2 \times \text{factorial}(1) = 1 \times 1 = 2 \)
\( \text{factorial}(1) \Rightarrow 1 \times \text{factorial}(0) = 1 \times 1 = 1 \)
\( \text{factorial}(0) \Rightarrow 1 \)
Recursive Thinking

- Define the solution to a problem in terms of a solution to one or more "smaller" subproblems
- Define a base case, a subproblem that can be solved directly.

Factorial

```java
private int factorial (int n) {
  assert n >= 0;
  if (n == 0) {
    return 1;
  }
  return n * factorial(n-1);
}
```

General Recursive Algorithm

```
E func (...) {
  if (problem is simple) {
    solve it;
    return;
  }
  solve 1 or more subproblems;
  combine subproblem solutions;
  Solution to recursive case
}
```
Recursive Drawing

// Base case
if (radius is small) {
    draw center;
    return;
}

// Recursive case
draw ring;
draw smaller bulls eye

Recursive Drawing

Recursive Stairs

Subproblems

Base case

Wednesday, April 10, 13
Recursive Stairs

// Base case
if (small stairs) {
    draw rectangle;
    return;
}

// Recursive case
    draw rectangle;
    draw stairs above rectangle;
    draw stairs to right of rectangle;

MAGIC!!