An abstract data type (ADT)

- is a type for encapsulating related data
- is abstract in the sense that it hides distracting implementation details

Associated Operations

Data

Two Parts of Every ADT

The Object of Data Abstraction and Structure, David D. Riley
© Addison Wesley pub.
From the user’s perspective...

1) What are the **operations** for this Calculator ADT?
2) What makes up the **state** of this ADT?

**Class Diagram**

(UML Notation)

<table>
<thead>
<tr>
<th>Class</th>
<th>State (Instance variables)</th>
<th>Operations (key methods)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Calculator</strong></td>
<td>- int displayedOperand</td>
<td>+ Calculator()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ void pressC()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ void pressPercent()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ void pressDivide()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ void pressMultiply()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ void pressSubtract()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ void pressAdd()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ void pressEquals()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ void pressPlusMinus()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ void pressDigit(int)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ void pressDecimalPoint()</td>
</tr>
</tbody>
</table>
Class diagrams include key members, but not usually *all* members.

### Class Specification (partial)

**Class invariant**

For each *this* object of type `Calculator`:
- `displayedOperand` is visible in the window of *this*
  - `0` if `(lastOp == '%' or lastOp == '/' or lastOp == '*' or lastOp == '-' or lastOp == '+')`

**Constructor method**

Public method `Calculator()`
- Post: A new Calculator object is instantiated
  - `displayedOperand == 0`
  - `lastOp == '='`
  - `withinOperand`

**Update methods**

Public method `pressDigit(int d)`
- Pre: `0 <= d <= 9`
- Modifies: `displayedOperand`
- Post: `(withinOperand and displayedOperand@pre + d) = displayedOperand@pre`
  - `displayedOperand@pre` is the displayed operand before calling `pressDigit`
  - `d` is the digit being pressed
  - `displayedOperand@pre` must be a string representation of a valid number or a calculation result
  - The method updates `displayedOperand` by adding the digit `d` to the current string representation
  - The method ensures that the updated string representation remains a valid number or calculation result

---

The Object of Data Abstraction and Structure, David D. Riley © Addison Wesley pub.
/** class invariant (for each this object of type Calculator) displayedOperand is visible in the window of this
 and (lastOp == '%' or lastOp == '/' or lastOp == '*' or lastOp == '=' or lastOp == '+') */

public class Calculator {
  private int displayedOperand, previousOperand;
  private boolean withinOperand;
  private char lastOp;

  /* post: A new Calculator object is instantiated and displayedOperand == 0 and lastOp == '='
   and withinOperand */
  public Calculator() {
    displayedOperand = 0;
    lastOp = '=';
    withinOperand = true;
    updateDisplay();
  }

  /* post: the value of displayedOperand has been displayed on the calculator screen*/
  private void updateDisplay() {
    ...
  }

  // Additional methods omitted.
}

/* pre: 0 <= d and d <= 9
 modifies: displayedOperand, withinOperand
 post: ((withinOperand && pre and displayedOperand && pre >= 0) implies displayedOperand == displayedOperand && pre * 10 + d)
 and ((withinOperand && pre and displayedOperand && pre <= 0) implies displayedOperand == displayedOperand && pre * 10 - d)
 and (!withinOperand && pre implies displayedOperand == d)
 and withinOperand*/
  public void pressDigit(int d) {
    if (withinOperand) {
      if (displayedOperand >= 0) {
        displayedOperand = displayedOperand * 10 + d;
      } else {
        displayedOperand = displayedOperand * 10 - d;
      }
    } else {
      displayedOperand = d;
    }
    withinOperand = true;
    updateDisplay();
  }

  // Additional methods omitted.
Design determines the software’s architecture. (What task does the software perform?)

Implementation determines the software’s code. (How does the software perform its task?)

Label each decision below as a design or implementation.

1) Whether or not to include a +/- key on the calculator.
2) Whether to use an if or switch instruction within the pressDigit method.
3) Whether to use a single pressDigit method and parameter or separate methods for each digit.
4) Whether the “C” button clears both the display and previousOperand or clears just the display. (Some calculators require two consecutive clears to reset everything.)
5) Whether to include a separate private method, like updateDisplay, or to include these instructions within the necessary methods.
6) Whether to use the given Calculator ADT or a different ADT based on GUI objects.
7) Whether to write pressEquals as a void method or a double method that returns the result of the calculation.