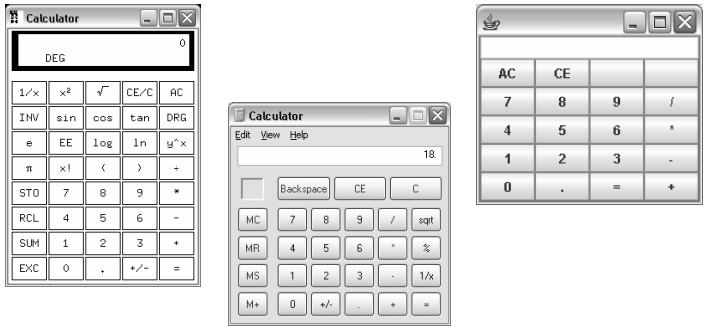
CS133: Developing Programming Principles

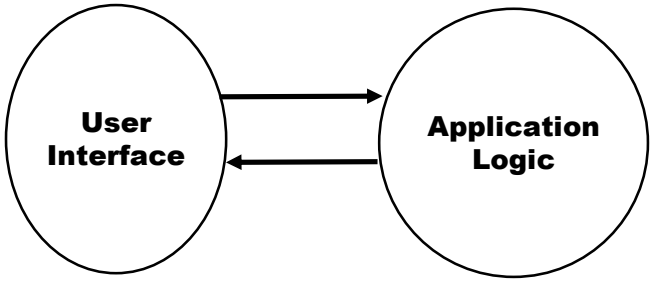
Lecture 22

A Desk Calculator

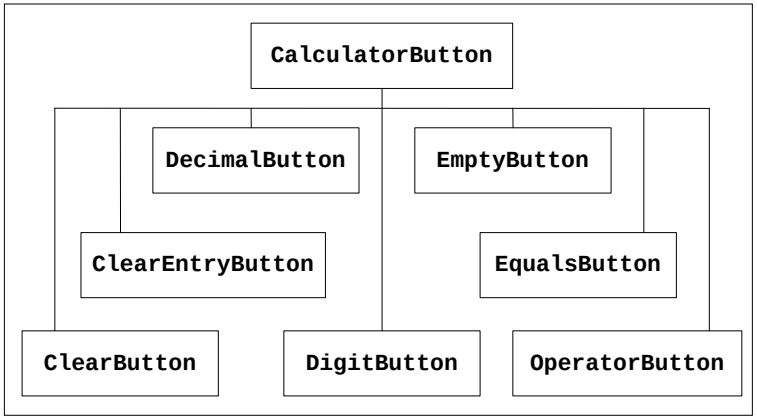
Desk Calculators



Implementation



User Interface



CalculatorButton

```
public abstract class CalculatorButton
    extends JButton
    implements ActionListener
{

  private Calculator calculator;
  private JTextField display;

  // constructor and methods (next two slides)

}
```


Constructor

```
public CalculatorButton (
    String label, Calculator calculator,
    JTextField display)
{
    super(label);
    this.calculator = calculator;
    this.display = display;
    addActionListener(this);
}
```

Methods

```
public Calculator getCalculator() {
  return calculator;
}

public void actionPerformed(ActionEvent e)
{
  pressed();
  display.setText(calculator.toString());
}

public abstract void pressed();
```


Abstract Classes

`CalculatorButton` is an example of an ***abstract class***, which is used only as a base class to derive other classes. You can't create an object of an abstract class.

The **`pressed`** method is an example of an ***abstract method***, which must be overridden in the derived class.

DigitButton

```
public class DigitButton extends CalculatorButton {
  private int digit;

  public DigitButton(
    int digit,Calculator calculator,JTextField display
  ) {
    super(Integer.toString(digit), calculator, display);
    this.digit = digit;
  }

  public void pressed() {
    getCalculator().digit(digit);
  }
}
```


EqualsButton

```
public class EqualsButton extends CalculatorButton {

  public EqualsButton(
    Calculator calculator, JTextField display
  ) {
    super("=", calculator, display);
  }

  public void pressed() {
    getCalculator().equals();
  }
}
```

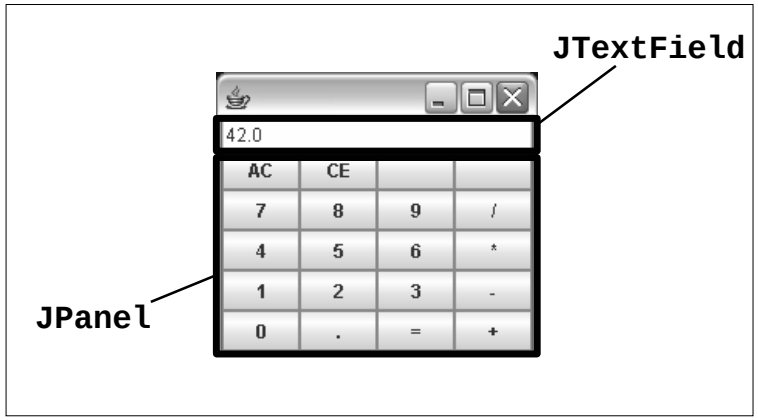
ClearButton

```
public class ClearButton extends CalculatorButton {

  public ClearButton(
    Calculator calculator, JTextField display
  ) {
    super("AC", calculator, display);
  }

  public void pressed() {
    getCalculator().clear();
  }
}
```


Components



CalculatorPanel

```
public class CalculatorPanel extends JPanel
{

  private static final int DISPLAY = 16;

  // see next two slides

}
```


Constructor

```
public CalculatorPanel() {
  Calculator calculator = new Calculator();
  JTextField display = new JTextField(DISPLAY);
  display.setEditable(false);
  display.setBackground(Color.WHITE);
  JPanel buttons =
    calculatorButtons(calculator, display);
  setLayout(new BorderLayout());
  add(display, BorderLayout.NORTH);
  add(buttons, BorderLayout.SOUTH);
}
```

calculatorButtons

```
private JPanel calculatorButtons(
    Calculator calculator, JTextField display
) {
  JPanel buttons = new JPanel();

  buttons.setLayout(new GridLayout(5,4));

  buttons.add(new ClearButton(calculator,display));
  buttons.add(new ClearEntryButton(calculator,display));
  buttons.add(new EmptyButton(calculator,display));
  buttons.add(new EmptyButton(calculator,display));
  buttons.add(new DigitButton(7,calculator,display));
  buttons.add(new DigitButton(8,calculator,display));
  // etc…
}
```


DeskCalculator

```
public class DeskCalculator extends JFrame {

  public DeskCalculator() {
    Container content = getContentPane();
    CalculatorPanel calculator = new CalculatorPanel();
    content.add(calculator);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    pack();
    setVisible(true);
  }
}
```

Application Logic

```
public class Calculator {

   public Calculator() {... }

   public void clear() {... }
   public void clearEntry() {... }
   public void digit(int digit) {... }
   public void decimal() {... }
   public void operator(Operator operator) {... }
   public void equals() {... }

   public String toString() {... }

  ...
}
```


Operators

```
public static final Operator ADD =
   new Operator("+");
public static final Operator SUBTRACT =
   new Operator("-");
public static final Operator MULTIPLY =
   new Operator("*");
public static final Operator DIVIDE =
   new Operator("/");
public static final Operator MOVE =
   new Operator(" ");
```

(argument to the **`operator`** method)

Testing the Class

```
public static void main(String[] args) {
    Calculator c = new Calculator();
    c.digit(1);
    System.out.println("1 = " + c);
    c.digit(0);
    System.out.println("10 = " +  c);
    c.operator(ADD);
    System.out.println("10 = " +  c);
    c.digit(2);
    System.out.println("2 = " +  c);
    c.equals();
    System.out.println("12 = " +  c);
}
```


Hidden Information

In addition to what's visible on the display, a calculator contains lots of hidden information:

1) a running total (accumulator)
2) an arithmetic operation (**+**, **-**, *****, **/**)
3) display update status

Instance Variables

```
// Values are entered into the display variable.
// Results are computed and stored in the accumulator.
private double display, accumulator;

// The operation to apply when updating the accumulator.
private Operator pendingOperation;

// Do we clear the display variable before entering digits?
private boolean clearEntryPending;

// When a digit is pressed is it before the decimal point?
private boolean beforeDecimal;

// Position for next digit after the decimal point
private double decimalPlace;
```


Constructor and Clearing

```
public Calculator()
{
  clear();
}

public void clearEntry()
{
  display = 0.0;  clearEntryPending = false;
  beforeDecimal = true;  decimalPlace = 0.1;
}
public void clear()
{
  accumulator = 0.0;  pendingOperation = MOVE;
  clearEntry();
}
```

Entering Digits

```
public void digit(int digit) {
  assert digit >= 0 && digit <= 9;
  if (clearEntryPending) clearEntry();
  if (beforeDecimal) {
    display = 10*display + digit;
  } else {
    display += decimalPlace*digit;
    decimalPlace *= 0.1;
  }
}

public void decimal() {
  if (clearEntryPending) clearEntry();
  beforeDecimal = false;
}
```


Equals

```
public void equals() {
  if (pendingOperation == ADD) {
    accumulator += display;
  } else if (pendingOperation == SUBTRACT) {
    accumulator -= display;
  } else if (pendingOperation == MULTIPLY) {
    accumulator *= display;
  } else if (pendingOperation == DIVIDE) {
    accumulator /= display;
  } else {
    accumulator = display;
  }
  display = accumulator;
  clearEntryPending = true;
  pendingOperation = MOVE;
}
```

Operators and Results

```
public void operator(Operator operator)
{
  equals();
  pendingOperation = operator;
}


public String toString()
{
  return Double.toString(display);
}
```


Improvements

- Correct problems with display.
- More functions (e.g., +/-, 1/x, sin, memory).
- Make it look prettier.

ERROR: undefined
OFFENDING COMMAND:

STACK: