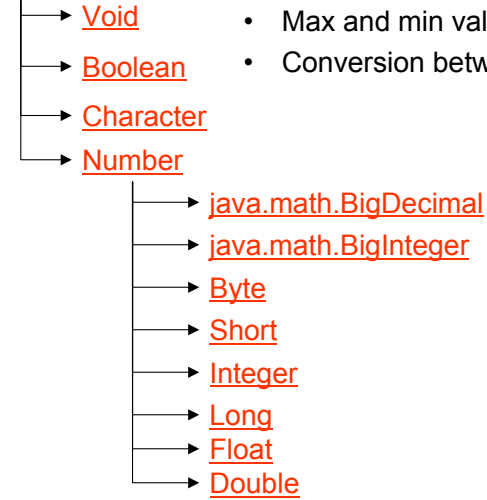


## Object-Based Programming

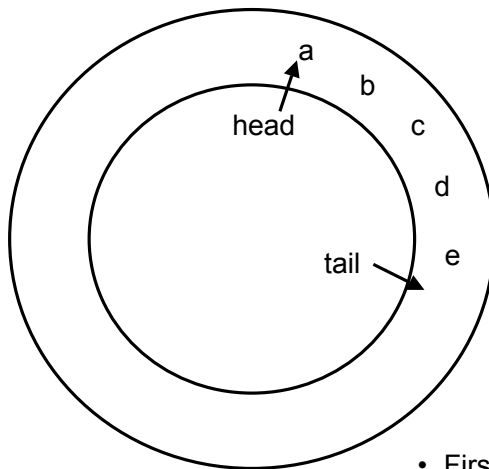
## Wrapper Classes

Object



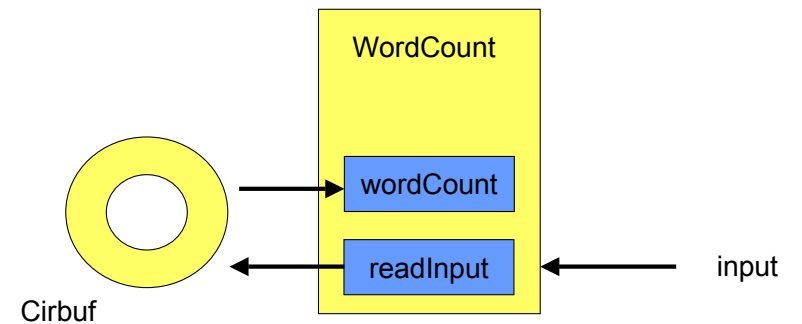
- Converting primitive types into ordinary classes
- Supplies methods related to strings
- Max and min value constants
- Conversion between numeric and string values

## Circular Queue



- First-In-First-Out
- head: the first available item
- tail: the first available slot

## Word Counting Using Circular Buffer

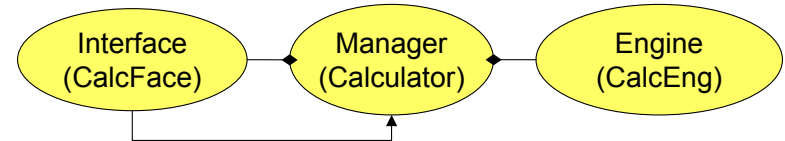


## Pocket Calculator

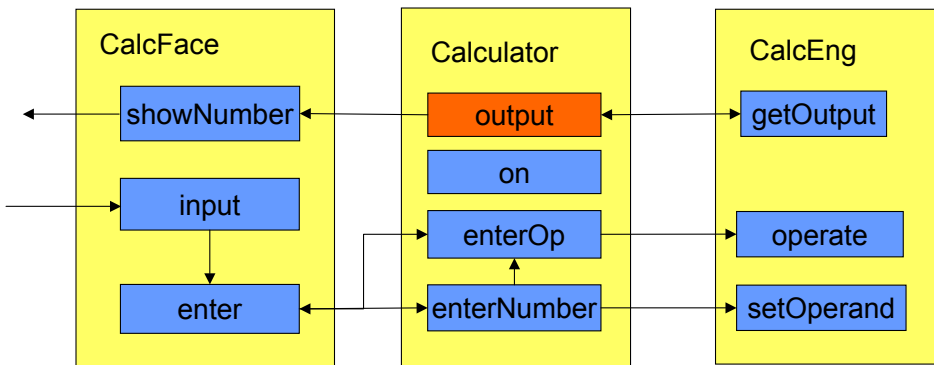
- Operation: +, -, \*, /, =, C (clear), A (all clear), N (negate)
- Example session: 32 + 66 – 86 = N EOF
  - Calc: 0
  - 32 +
  - Calc: 32
  - 66 –
  - Calc: 98
  - 86 =
  - Calc: 12
  - N
  - Calc: -12
- Try

## Pocket Calculator Program Design

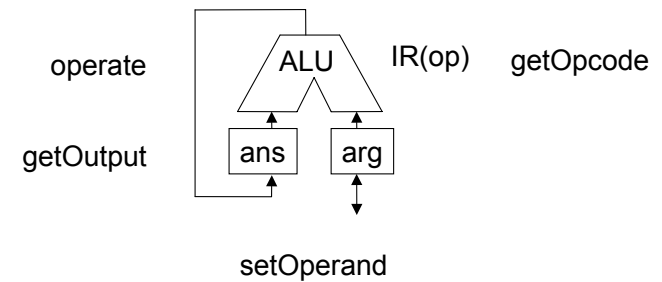
- Manager: Calculator
- User interface: CalcFace
- Computing Engine; CalcEng



## Manager: Class Calculator



## Engine: Class CalcEng



Fields: registers  
Methods: ALU and control unit

## Class CalcEng

```
class CalcEng {
    public CalcEng() { allClear(); }

    public char getOpcode()           // returns current opcode
    public double getOutput()         // returns current argument
    public void operate(char nc)      // CU, nc is next opcode
    public void setOperand(double in)

    // private members
    private void compute()           //ALU

    private final String KEYS = "+-*/=NAC";
    private final byte PREC = 8;
    private double ans, arg;
    private char op;                 // operation code
    private int argcnt;              // argument count
}
```

Engine  
(CalcEng)

## Interface

```
class CalcFace {
    public CalcFace(String k, byte pr)
    public void setCalc(Calculator ca)

    public void showNumber(String s) // System.out.println(prompt + s);
    public void input() throws IOException
    private void enter(char c)       // works with calculator manager

    private String prompt="Calc: ";
    private Calculator calc;
    private String keys;             // keys recognized (operations)
    private StringBuffer nbuf;      // buffer for input number
    private byte prec;              // max no of chars displayable
    private boolean before_point = true;
    private boolean num = false;
}
```

Interface  
(CalcFace)

## Manager: Class Calculator

```
public class Calculator {

    public Calculator(CalcEng e, CalcFace f)

    public void on() throws java.io.IOException
    public void enterNumber(String number, char op)
    public void enterOp( char op )
    private void output()

    private CalcEng eng = null;
    private CalcFace cf = null;
}
```

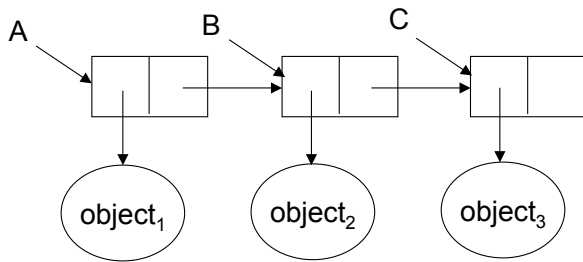
Manager  
(Calculator)

## Class RunCalc

```
public class RunCalc {
    public static void main(String[] args) throws java.io.IOException {
        CalcEng e = new CalcEng();
        CalcFace f = new CalcFace(e.keys(), e.precision());
        Calculator x = new Calculator(e, f);
        x.on();
        return;
    }
}
```

## Linked List

- Recursive data structure
- $List ::= Nil \mid Element \ List$
- Views of the linked list from A, B and C are the same



## Cell of Character Linked List

```
class ListCell {  
  
    ListCell(char c, ListCell cell) { item=c; next=cell; }  
    ListCell() { }  
    char content() { return item; }  
  
    // fields  
    char item = '\0';  
    ListCell next = null;  
}
```

## CharList

```
public class CharList {  
  
    public CharList() { } // empty list constructor  
    // single element list  
    public CharList(char c) { head=new ListCell(c,null); }  
  
    public ListCell first() { return head.item; } // first cell  
  
    public ListCell rest() { return head.next; } // list after the first cell  
  
    public void putOn(char c) { // insert in front  
        head = new ListCell(c,head);  
    }  
}
```

```
public boolean isEmpty() { return head==null; }  
  
public String toString() { return toString(head); }  
  
public String toString(ListCell p)  
{ String s = "(";  
  while ( p != null )  
  { s = s + p.item;  
    if ( (p = p.next) != null ) s = s + " ";  
  }  
  return s + " )";  
}  
private ListCell head = null; // first cell of list  
}
```

## Other Useful Methods

```
public ListCell last() { // last cell
    ListCell p = head;
    while( p != null && p.next != null ) p = p.next;
    return p;
}
public ListCell find(char c) { // first item == c
    for( ListCell p = head; p!=null ; p = p.next )
        if( p.item == c ) return p;
    return null; // c not on list
}
public boolean substitute(char r, char s) { // r for first s on list
    ListCell p = find(s);
    if( p == null ) return false; // s not on list
    p.item = r;
    return true;
}
```

```
public int remove(char c) { // c from entire list
    ListCell p = head;
    int count = 0;
    if ( p == null ) return count;
    while (p.next != null) // treat all but head cell
        if ((p.next).item == c) {
            count++;
            p.next = (p.next).next;
        } else
            p = p.next;
    if( head.item == c ) { // treat head cell
        head = head.next;
        count++;
    }
    return count; // number of items removed
}
```

```
public int shorten(int n) { // remove first n cells
    while (n-- > 0 && head != null)
        head = head.next;
    return --n;
}
public void insert(char c, ListCell e) { // insert after cell
    e.next = new ListCell(c,e.next);
}
public void append(char c) { insert(c, last()); } // insert at end
```

## Testing Linked List

```
public class TestCharList {
    public static void main(String[] args) {
        CharList a = new CharList('B');
        a.putOn('A');
        a.append('D'); a.append('E'); a.append('F');
        System.out.println("a = " + a);
        ListCell lp = a.find('B');
        System.out.println(a.toString(lp));
        a.insert('C', lp);
        System.out.println("a = " + a);
        a.remove('E');
        a.shorten(2);
        System.out.println("a = " + a);
        a.remove('C');
        System.out.println("a = " + a);
    }
}
```

exec

## Resolution of a Call to an Overloaded Method

- Accessible
- Applicable – type matching
- Method with the most specific signature is selected
- Example
  - $f(\text{long}, \text{double}) > f(\text{int}, \text{double}) > f(\text{int}, \text{float}) > f(\text{short}, \text{float})$
  - No order: given  $g(\text{long}, \text{float})$ ,  $g(\text{int}, \text{double})$ , try type matching  $g(\text{int}, \text{float})$

type + method name

## Storage Allocation and Management

- Variables: compile time
- Data objects
  - primitive types: compile time
  - reference types: by new from heap memory
- Garbage collection (no explicit deallocation)
- Memory model
  - static: code and global variables
  - stack: method invocation
  - heap: dynamic memory allocation

