Introduction to Object Oriented Programming 2E Timothy A. Budd

Chapter 3

Classes and Methods

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Same Ideas, Different Terms

All OOP languages have the following concepts:

- **classes**, object type, factory object
- **instances**, objects
- **message passing**, method lookup, member function invocation, method binding
- **methods**, member function, method function
- **inheritance**, subclassing

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Encapsulation and Instantiation

Classes provide two very important capabilities:

Encapsulation: purposeful hiding of information, thereby reducing the details that must be remembered/communicated

Instantiation: create multiple instances of an abstraction

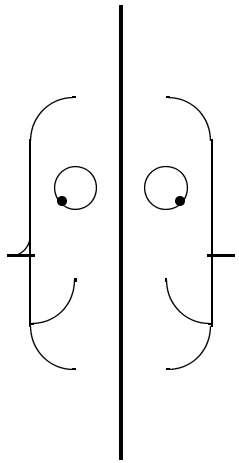
© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Internal and External Views

Push

Pop

Top



```
const limit = 300;

var currentTop : 0 .. limit;

values : array [ 1 .. limit] of integer;
```

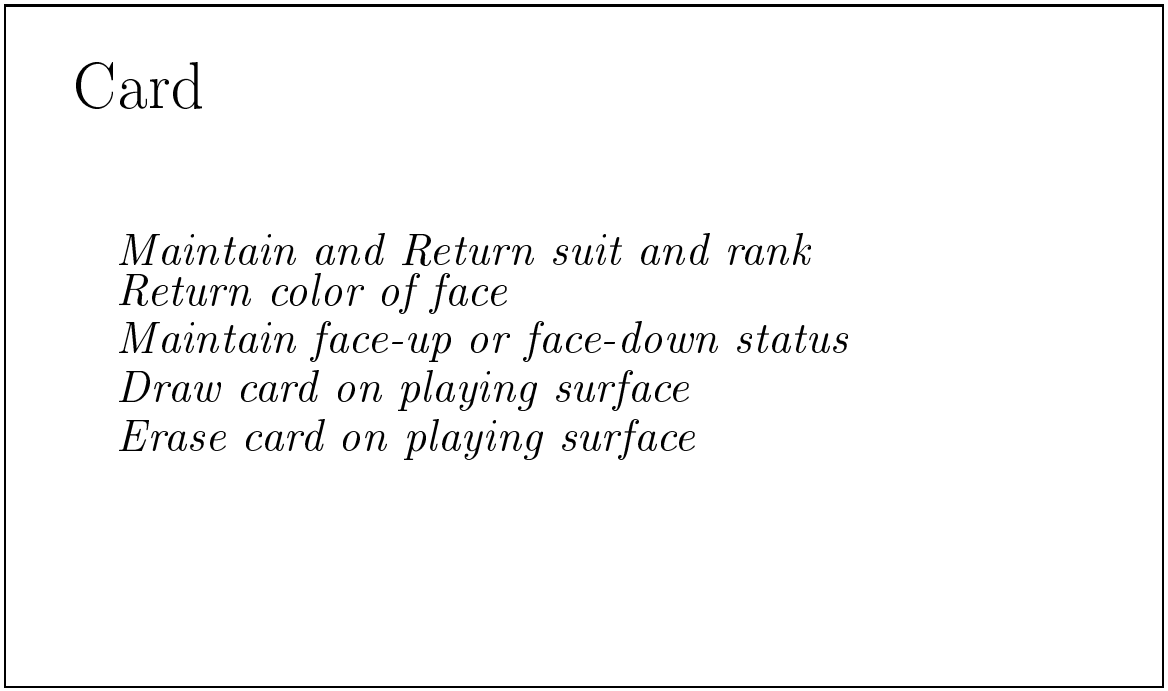
© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

State – Instance Variables

- *Instance* – a representative or example of a class
- *Instance variable* – the state, or data values, maintained internal to a class
- Record information about instance variables on back of CRC card (internal to class)
  - clients can see front of CRC card
  - implementors can see back side of CRC card

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

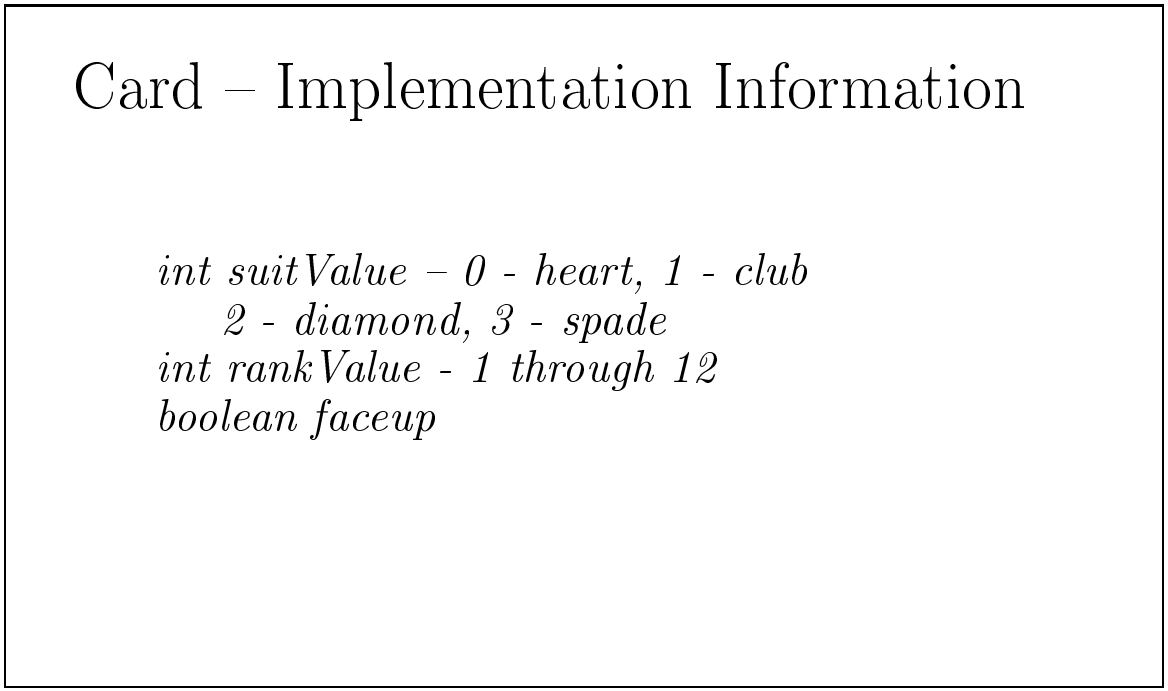
Example – Playing Card



This is the front (public face) of a typical CRC card.

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

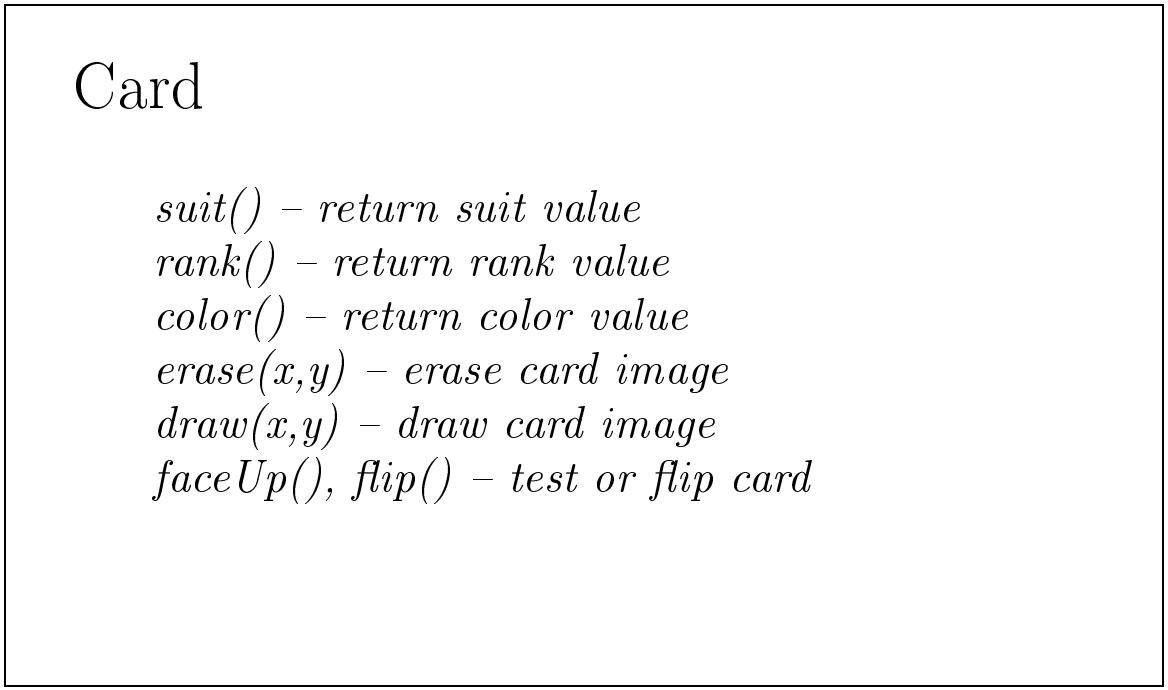
Playing Card Backside – Data Areas



The back can record information about the internal state of an instance.

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Revision of Card



We refine the behavior by providing explicit function names and argument types.

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Interface and Implementation

Distinction between interface and implementation is manifest in different ways:

- Within class, separation between interface and implementation

  Example: public and private fields in C++ and Java

- Interface descriptions and implementation bodies appearing in separate places

  Example: .h files and *.cpp files in C++

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Varieties of Classes

Data manager: principal responsibility is to maintain data values

Example: Playing card – maintains suit and value information

Data sink or source: interface to data generator or consumer, no data itself

Example: Input/Output stream objects - files, I/O devices

View or observer class: display objects graphically, not the object itself

- Viewing tasks are different from data manipulation tasks
- An object might have many different views

Facilitator or helper class: maintain little/no state, but assist with complex tasks

Example: painting assistant (Pen, Canvas, Edit box) for graphics output device

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Classes and Methods in Java

Java is similar to C++, but

- No preprocessor, global variables, enumerated data types
- Methods always defined directly in the class definitions
- Keywords public and private applied individually to features
- Boolean datatype is boolean, not bool
- Object references instead of pointers
- All methods must specify a return type

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Playing Card in Java

```
class Card {
      // static values for colors and suits
   final public int red = 0;
   final public int black = 1;
   final public int spade = 0;
   final public int heart = 1;
   final public int diamond = 2;
   final public int club = 3;

      // constructor
   Card (int sv, int rv) { s = sv; r = rv; faceup = false; }

      // mutators
   public void    flip() { faceup  =  ! faceup; }

      // accessors
   public int    rank() { return r; }
```

```
    public int    suit() { return s; }

     public int    color () {
        if (suit() == heart || suit() == diamond)
            return red;
        return black ; }

    public boolean    faceUp() { return faceup; }

        // perform other actions
    public void    draw (Graphics g, int x, int y) {
        ... /* omitted */ ...
        }

        // data fields
    private boolean faceup;
    private int  r;
    private int s;
};
```