Lecture 14: Modular Programming

Object Oriented Programming

Objected oriented programming (OOP).
- Programming paradigm based on data types.
- An object stores a data type value; variable name refers to object.
- "Everything" in Java is an object.

OOP enables:
- Data abstraction.
- Modular programming.
- Encapsulation.
- Inheritance.

Religious wars ongoing.

Modular Programming

Modular programming.
- Break a large program into smaller independent modules.
  - Ex: Card, Deck, Player, Blackjack, Casino.
  - Ex: Switch, Gate, Adder, ALU, FlipFlop, Decoder, Memory, TOY.

Advantages.
- Debug pieces independently.
- Divide work for multiple programmers.
- Reuse code.

Modular programming in Java.
- Define new classes in terms of old ones.
- Keep classes small.

Review

Data type: set of values and operations on those values.

A Java class allows us to define a data type by:
- Specifying a set of variables.
- Defining operations on those values.

Break up a program into smaller pieces.
- Class = program that defines a data type.
- Client = program that uses a data type.
Cards Data Type

Set of values.
- 2♠, 3♠, 4♠, ..., A♠.
- Two images, for displaying front and back.

Operations.
- Initialize.
- Draw the front or back using our graphics library.
- Convert to string representation.

<table>
<thead>
<tr>
<th>Clubs</th>
<th>Diamonds</th>
<th>Hearts</th>
<th>Spades</th>
</tr>
</thead>
<tbody>
<tr>
<td>Card</td>
<td>Card #</td>
<td>Card</td>
<td>Card #</td>
</tr>
<tr>
<td>2♠</td>
<td>0</td>
<td>2♥</td>
<td>2♣</td>
</tr>
<tr>
<td>3♠</td>
<td>1</td>
<td>3♥</td>
<td>3♣</td>
</tr>
<tr>
<td>4♠</td>
<td>2</td>
<td>4♥</td>
<td>4♣</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>K♠</td>
<td>11</td>
<td>K♥</td>
<td>K♣</td>
</tr>
<tr>
<td>A♠</td>
<td>12</td>
<td>A♥</td>
<td>A♣</td>
</tr>
<tr>
<td>card  = 37</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
suit = 37 / 13 = 2 |
rank = 37 % 13 = 11 |
public class Game {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Player N = new Player("North", 300, 375);
        Player E = new Player("East", 550, 225);
        Player S = new Player("South", 300, 75);
        Player W = new Player("West", 50, 225);
        deck.shuffle();
        while(!deck.isEmpty()) {
            N.draw();
            E.draw();
            S.draw();
            W.draw();
            StdDraw.show();
        }
    }
}

public class Deck {
    private Card[] cards;
    private int N;
    public Deck() {
        N = 52;
        cards = new Card[N];
        for (int i = 0; i < N; i++)
            cards[i] = new Card(i, i + ".gif", "back.gif");
    }
    public Card dealFrom() { return cards[--N]; }
    public boolean isEmpty() { return (N == 0); }
    public void shuffle() {
        for (int i = 0; i < N; i++) {
            int r = (int) (Math.random() * (i+1));
            Card swap = cards[i];
            cards[i] = cards[r];
            cards[r] = swap;
        }
    }
}

Set of values.
- Sequence of remaining cards.

Operations.
- Initialize a new deck of 52 cards.
- Shuffle it.
- Deal a card (and remove it from deck).
- Create a string representation of the deck.

public class Player {
    private String name;
    private int score;
    private String location;
    public Player(String name, int score, String location) {
        this.name = name;
        this.score = score;
        this.location = location;
    }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public int getScore() { return score; }
    public void setScore(int score) { this.score = score; }
    public String getLocation() { return location; }
    public void setLocation(String location) { this.location = location; }
    public String toString() { return name + " (Score: " + score + ", Location: " + location + ");"; }
}

Player Data Type

Set of values.
- Pile of cards.
- Name.
- Location for drawing.

Operations.
- Deal a card to the player.
- Display the pile using our standard graphics library.
- Create a string representation of the player.
Player Data Type: Java Implementation

```java
public class Player {
    private Card[] cards;
    private int N = 0;
    private int x, y;
    private String name;

    public Player(String name, int x, int y) {
        this.name = name;
        this.x = x;
        this.y = y;
        this.cards = new Card[52];
    }

    public void dealTo(Card c) { cards[N++] = c; }

    public void draw() {
        StdDraw.go(x, y);
        for (int i = 0; i < N; i++) {
            cards[i].drawFront();
            StdDraw.goForward(17);
        }
    }
}
```

Layers of Abstraction

Relationships among data types.

Sorting the Hands

Goal: display each hand in "sorted" order.

```
0  Turtle Graphics
```

```
public boolean less(Card c) {
    if (suit < c.suit) return true;
    else if (suit > c.suit) return false;
    else if (rank < c.rank) return true;
    else return false;
}
```

```
public void sort() {
    for (int i = 0; i < N; i++) {
        for (int j = i; j > 0; j--) {
            if (cards[j-1].less(cards[j])) {
                Card swap = cards[j];
                cards[j] = cards[j-1];
                cards[j-1] = swap;
            }
        }
    }
}
```

Sorting the Hands

Goal: display each hand in "sorted" order.

- Need method `less` in `Card` to compare cards.
- Need method `sort` in `Player` to sort hand.
**I Doubt It**

```java
public boolean less(Card c) {
    return (rank < c.rank);
}
```

**Card.java**

**A Bridge Experiment**

Determine strength of bridge hand.
- Face cards and aces.
- Uneven suit distribution.

<table>
<thead>
<tr>
<th>Each Occurrence</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ace</td>
<td>4</td>
</tr>
<tr>
<td>King</td>
<td>3</td>
</tr>
<tr>
<td>Queen</td>
<td>2</td>
</tr>
<tr>
<td>Jack</td>
<td>1</td>
</tr>
<tr>
<td>Void Suit</td>
<td>2</td>
</tr>
<tr>
<td>Singleton Suit</td>
<td>1</td>
</tr>
<tr>
<td>Singleton Suit</td>
<td>2</td>
</tr>
</tbody>
</table>

- Need method `rank` in `Card` to check for aces.
- Need method `suit` in `Card` to count cards in each suit.

```java
public int rank() { return rank; }
public int suit() { return suit; }
```

**Card.java**

**Experiment**

```java
public class BridgeExperiment {
    public static void main(String[] args) {
        Histogram hist1 = new Histogram(ACE);
        Histogram hist2 = new Histogram(ACE);
        while (true) {
            Deck deck = new Deck();
            deck.shuffle();
            Player N = new Player("North", 300, 375);
            Player E = new Player("East", 550, 225);
            Player S = new Player("South", 300, 75);
            Player W = new Player("West", 50, 225);
            while (!deck.isEmpty()) {
                N.dealTo(deck.dealFrom());
                E.dealTo(deck.dealFrom());
                S.dealTo(deck.dealFrom());
                W.dealTo(deck.dealFrom());
                hist1.add(N.points());
                hist2.add(N.points() + S.points());
            }
        }
    }
}
```

**Player.java**

```java
public int points() {
    int sum = 0;
    for (int i = 0; i < N; i++) {  
        int rank = cards[i].rank();
        if (rank == 12) sum = sum + 4;
        else if (rank == 11) sum = sum + 3;
        else if (rank == 10) sum = sum + 2;
        else if (rank == 9) sum = sum + 1;
    }
    int[] suits = new int[4];
    for (int i = 0; i < N; i++)
        suits[cards[i].suit()]++;
    for (int j = 0; j < 4; j++) {
        if (suits[j] == 0) sum = sum + 2;
        else if (suits[j] == 1) sum = sum + 1;
    }
    return sum;
}
```

**Counting Points**
### Summary

**Modular programming.**
- Break a large program into smaller independent modules.  
  *Ex: Card, Deck, Player, Game, Casino.....*
  
**Debug and test each piece independently (unit testing).**
- Each class can have its own `main`.  
- Spend less overall time debugging.

**Divide work for multiple programmers.**
- Software architect specifies data types.  
- Each programmer writes, debugs, and tests one.

**Reuse code.**
- *Ex: reuse Histogram with gambler’s ruin.*  
- *Ex: reuse Card, Deck to make blackjack or poker game.*

---

### Announcements

**Thinking about majoring in Computer Science?**
**Or doing the Certificate in Applications of Computing?**

Then: visit the all-new "Life in the Computer Science Department: A Guide for the Humble Undergraduate":
- a handy FAQ that answers many many questions

And/Or: Come talk to me.

**AND CERTAINLY attend at least one of:**
- C.S. open house for BSE freshmen Tuesday March 29, Friend Convocation Room, 5:45 (PM!): tours, demos, pizza (AB’s welcome)
- C.S. open house for AB sophomores Tuesday April 5, C.S. Tea Room, 4 PM (but no pizza, and maybe fewer demos) (BSE’s welcome)

---

### Histogram Data Type

```java
public class Histogram {
    private int[] freq;  // freq[i] = # occurrences of i
    private Draw draw;   // separate turtle for each histogram

    public Histogram(int N) {
        freq = new int[N + 1];  // data between 0 and N
        draw = new Draw(512, 512);
    }

    public void add(int i) {
        freq[i]++;
        draw();
    }

    public void draw() {
        draw.go(x, y);
        // Draw object
    }
}
```

---

### Histograms of Points in a Bridge Hand

![Histograms of Points in a Bridge Hand](imageurl)