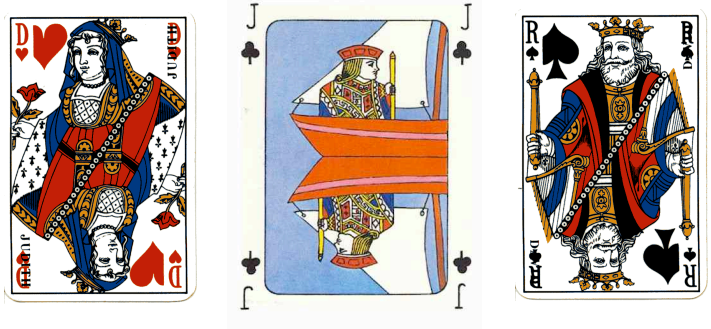
Lecture 14: Modular Programming



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.

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.

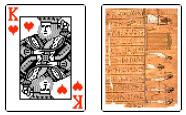
Modular programming in Java.

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

Cards Data Type

Set of values.

- 2 ♣, 3 ♣, 4 ♣, . . . , A ♠.
- Two images, for displaying front and back.



```
card = 37
suit = 37 / 13 = 2
rank = 37 % 13 = 11
```

Operations.

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

Clubs		Diamonds		Hearts		Spades	
Card	#	Card	#	Card	#	Card	#
2 ♣	0	2 ♦	13	2 ♥	26	2 ♠	39
3 ♣	1	3 ♦	14	3 ♥	27	3 ♠	40
4 ♣	2	4 ♦	15	4 ♥	28	4 ♠	41
. . .	. .	. . .	. .	. . .	. .	. . .	. .
K ♣	11	K ♦	24	K ♥	37	K ♠	50
A ♣	12	A ♦	25	A ♥	38	A ♠	51

Card Data Type: Java Implementation

```
public class Card {
   private int suit, rank;
   private String front, back;

   public Card(int card, String front, String back) {
      this.rank  = card % 13;
      this.suit  = card / 13;
      this.front = front;
      this.back  = back;
   }

   public void drawFront() { StdDraw.spot(front); }
   public void drawBack()  { StdDraw.spot(back);  }

   public String toString() {
      String ranks = "23456789TJQKA";
      String suits = "CDHS";
      return ranks.charAt(rank) + "" + suits.charAt(suit);
   }                          ⬆
}                    ith character in string
```


Sample Client

```
public static void main(String[] args) {
   StdDraw.create(200, 180);
   StdDraw.clear(Color.gray);

   Card c1 = new Card(27, "27.gif", "back.gif");
   StdDraw.go(100, 120);
   c1.drawBack();
   System.out.println(c1);

   Card c2 = new Card(51, "51.gif", "back.gif");
   StdDraw.go(100, 60);
   c2.drawFront();
   System.out.println(c2);

   StdDraw.show();
}
```

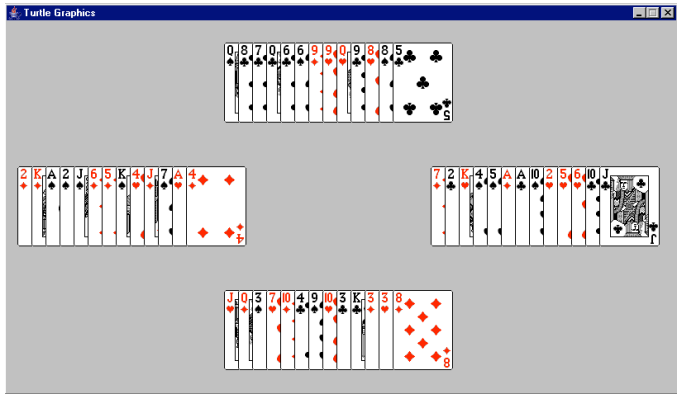



```
% java -classpath .:cards.jar Card
3H                  ⬆                ⬆
AS          tell Java where    Java archive contains
            to find files      53 images
```

A Card Game

Bridge game (partial).

- Create a deck of cards and 4 players.
- "Shuffle up and deal."
- Display with standard graphics library.



A Card Game Client

```
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();                                  shuffle and deal
      while(!deck.isEmpty()) {
         N.dealTo(deck.dealFrom());
         E.dealTo(deck.dealFrom());
         S.dealTo(deck.dealFrom());
         W.dealTo(deck.dealFrom());
      }

      StdDraw.create(810, 450);                        draw 4 hands
      N.draw();
      E.draw();
      S.draw();
      W.draw();
      StdDraw.show();
   }
}
```

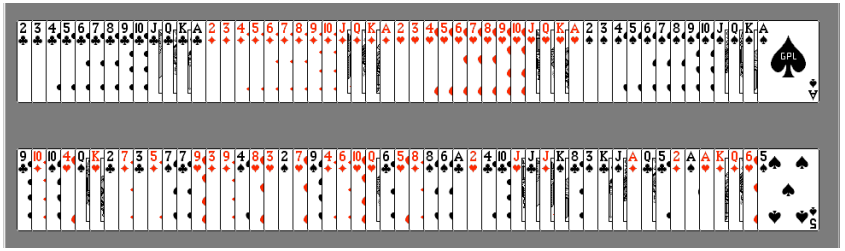

Deck Data Type

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.



Deck Data Type: Java Implementation

```
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;
      }
   }
}
```


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.

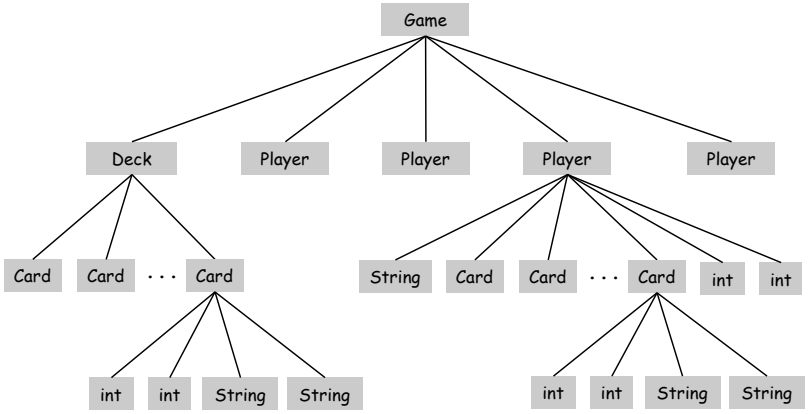
Player Data Type: Java Implementation

```
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);
      }
   }
}
```

inter-card spacing

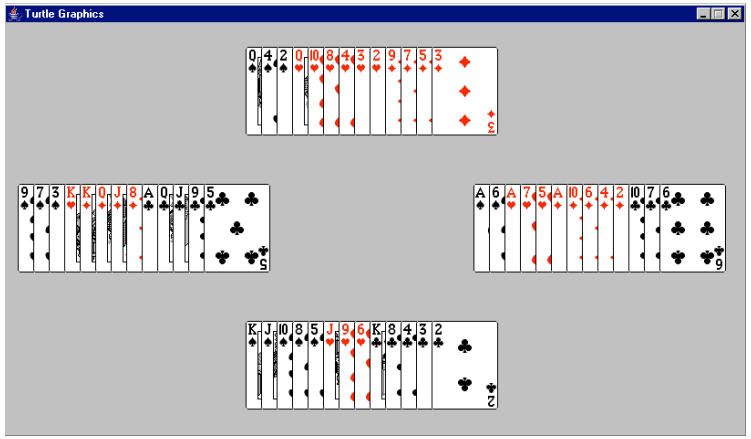
Layers of Abstraction

Relationships among data types.



Sorting the Hands

Goal: display each hand in "sorted" order.



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.

```
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;
}
```

Card.java

```
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;
         }
      }
   }
}
```

insertion sort

descending order

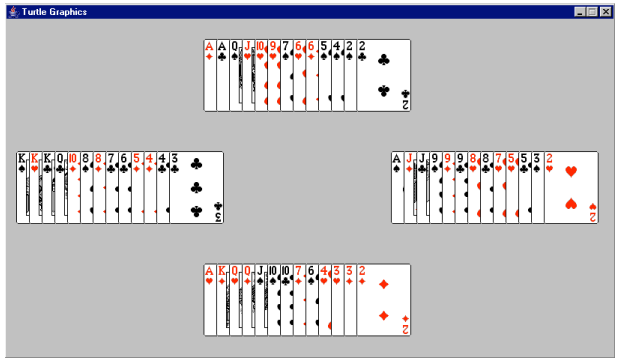
Player.java

"I Doubt It"

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

"I Doubt It" order

Card.java



A Bridge Experiment

Determine strength of bridge hand.

- Face cards and aces.
- Uneven suit distribution.
- Ex: 3♣ 7♣ 8♣ 9♣ 2♦ 9♦ Q♦ A♦ 3♠ 6♠ 9♠ K♠ A♠

  15 points: (0) ♣ + (2 + 4) ♦ + (2) ♥ + (3 + 4) ♠

Each Occurrence	Points
Ace	4
King	3
Queen	2
Jack	1
Void Suit	2
Singleton Suit	1
Singleton Suit	2

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

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

Card.java

Counting Points

```
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;
}
```

high card points

voids and singletons

Player.java

Experiment

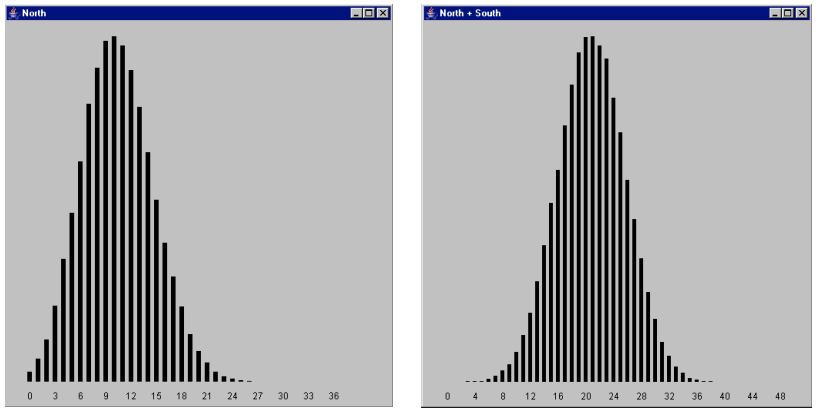
```
public class BridgeExperiment {
   public static void main(String[] args) {
      Histogram hist1 = new Histogram(38);
      Histogram hist2 = new Histogram(48);

      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());
      }
   }
}
```

max points

Histogram Data Type

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

   public Histogram(int N) {    <- data between 0 and N
      freq = new int[N + 1];
      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



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":

- http://www.cs.princeton.edu/ugradpgm/life.html
- 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)