

## 3.2: Modular Programming

---

### Review

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

A Java `class` allows us to define data types by:

- Specifying a set of variables.
- Defining a set of methods.

↙ Java terminology for a function in a data type

Break up a program into smaller pieces.

- Class = program that defines a data type.
- Client = program that uses a data type.

### Object Oriented Programming

Object oriented programming (OOP).

- Programming paradigm based on data types.
- An object is a data type value.
- "Everything" in Java is an object.

OOP enables:

- Data abstraction.
- ➔ • Modular programming.
- Encapsulation. *next week*
- Inheritance. *playing with fire to delve too deeply into advanced concepts*

Religious wars ongoing.

### Data Abstraction

René Magritte.

- "This is not a pipe."



Java.

- This is not a deck of cards.

```
Deck d = new Deck ();  
d.shuffle ();  
System.out.println (d);
```

OOP natural vehicle for studying abstract models of the real world.

## 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.

← needed in any programming paradigm

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

### Modular programming in Java.

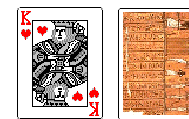
- Define new classes in terms of old ones.
- Keep classes small.

5

## 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 in Turtle graphics.
- 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

6

## 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() { Turtle.spot(front); }
    public void drawBack() { Turtle.spot(back); }

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

↑  
i<sup>th</sup> character in string

7

## Sample Client

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

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

    Card c2 = new Card(51, "51.gif", "back.gif");
    Turtle.fly(100, 60);
    c2.drawFront();
    System.out.println(c1);

    Turtle.destroy();
}
```

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

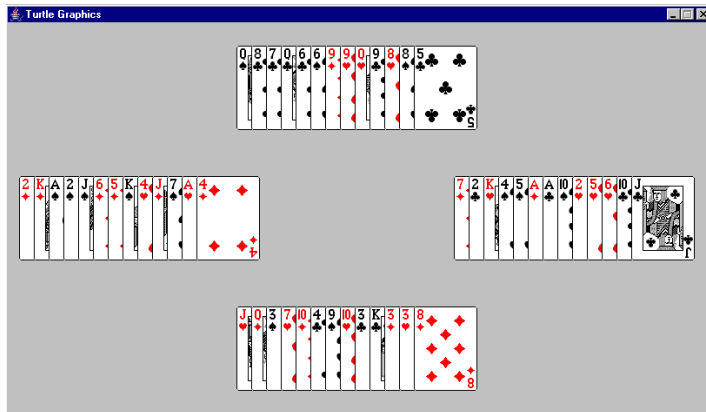


8

## A Card Game

### Bridge game (partial).

- Create a deck of cards and 4 players.
- Shuffle and deal cards.
- Display with Turtle graphics.



9

## 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());
        }

        Turtle.create(810, 450); // draw 4 hands
        N.draw();
        E.draw();
        S.draw();
        W.draw();
        Turtle.destroy();
    }
}
```

10

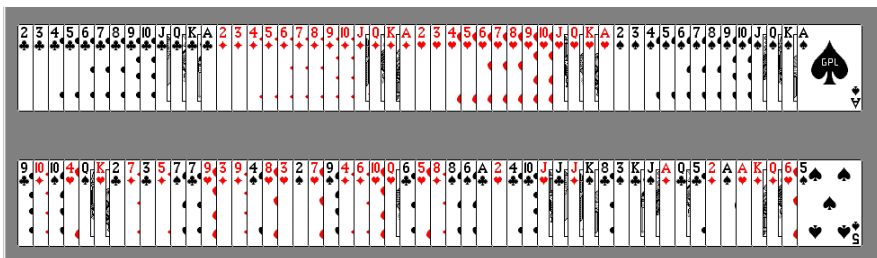
## Deck Data Type

### 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.



11

## Deck Data Type: Java Implementation

```
public class Deck {
    private Card[] cards;
    private int N;

    public Deck() {
        N = 52;
        cards = new Card[N]; // cards.jar contains 0.gif, ..., 51.gif
        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);
            Card swap = cards[i];
            cards[i] = cards[r];
            cards[r] = swap;
        }
    }
}
```

12

## Player Data Type

### Set of values.

- Pile of cards.
- Name.
- Location for drawing Turtle graphics.

### Operations.

- Deal a card to the player.
- Display the pile using Turtle graphics.
- Create a string representation of the player.

13

## 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];
    }

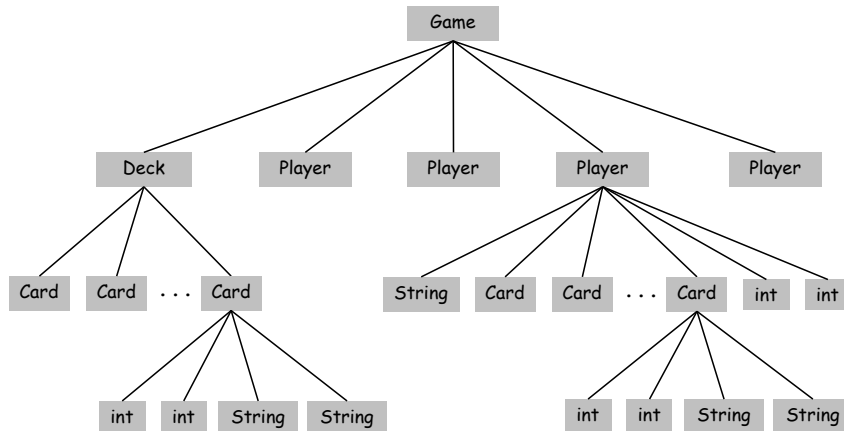
    public void dealTo(Card c) { cards[N++] = c; }

    public void draw() {
        Turtle.fly(x, y);
        for (int i = 0; i < N; i++) {
            cards[i].drawFront();
            Turtle.flyForward(17);
        }
        ↑
        inter-card spacing
    }
}
```

14

## Layers of Abstraction

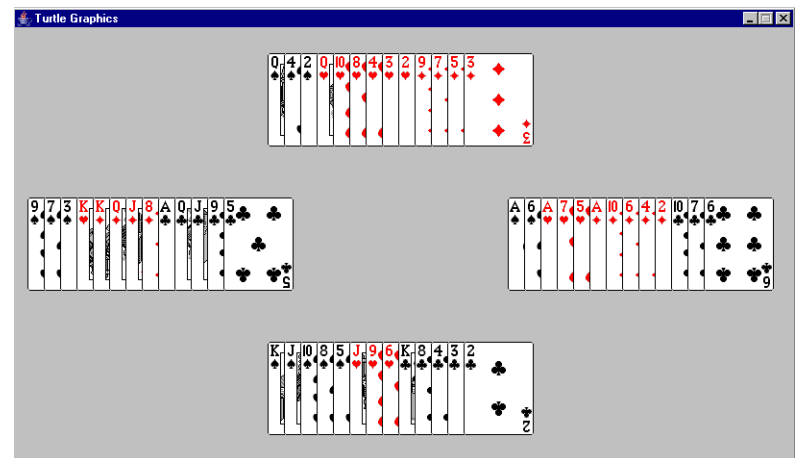
### Relationships among data types.



15

## Sorting the Hands

### Goal: display each hand in "sorted" order.



16

## 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() {
    // insertion 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;
            }
        }
    }
}
Player.java
```

descending order

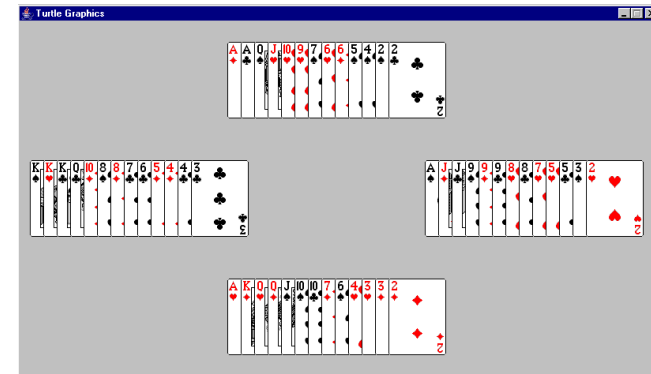
17

## "I Doubt It"

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

"I Doubt It" order

Card.java



18

## A Bridge Experiment

Determine strength of bridge hand.

- Face cards and aces.
- Uneven suit distribution.
- Ex:  $3\spadesuit 7\heartsuit 8\clubsuit 9\spadesuit 2\diamond 9\heartsuit Q\diamond A\diamond 3\spadesuit 6\spadesuit 9\spadesuit K\spadesuit A\spadesuit$   
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

- Need method `getRank` or `isAce` in `Card` to check for aces.
- Need method `getSuit` in `Card` to count cards in each suit.

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

Card.java

19

## Counting Points

```
public int points() {
    int sum = 0;

    for (int i = 0; i < N; i++) {
        // high card points
        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;
    }

    // voids and singletons
    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;
}
Player.java
```

20

## 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());
        }
    }
}
```

21

## Histogram Data Type

```
public class Histogram {
    private int freq[];
    private TurtleFrame turtle;

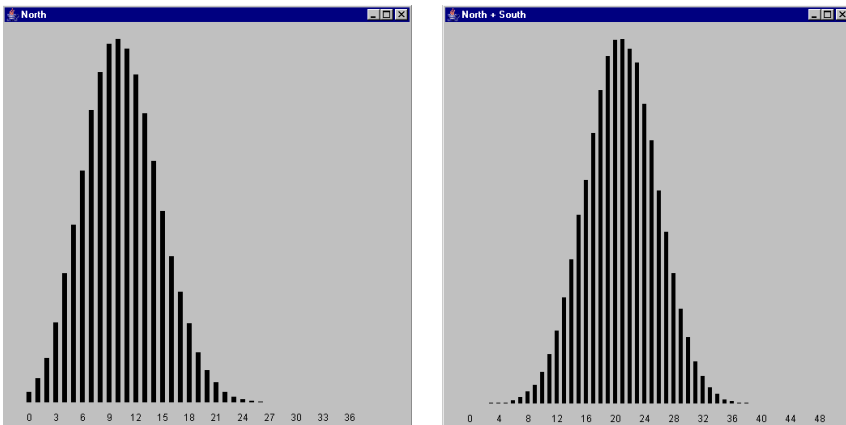
    public Histogram(int N) {
        freq = new int[N + 1];
        turtle = new TurtleFrame(512, 512);
    }

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

    public void draw() {
        . . .
        turtle.fly(x, y);
        . . .
    }
}
```

22

## Histograms of Points in a Bridge Hand



23

## Summary

### Modular programming.

- Break a large program into smaller independent modules.
- Ex: Card, Deck, Player, Game, Casino, . . . .

### Advantages.

- Debug pieces independently.
  - each class can have its own main
- Divide work for multiple programmers.
  - software architect specifies data types
  - each programmer writes and debugs one
- Reuse code.
  - Ex: reuse Histogram with gambler's ruin
  - Ex: reuse Card, Deck to make Blackjack game

24