Lecture 31
• Converting a "deck of card" to String: toString()
  
  We will make a `toString()` method that return a `String` of the cards stored inside the `array deckOfCards[]`.

  We will return **13 cards on 1 line**

  (We use the `newline character \n` to `separate` the `lines`)

  
  • The `toString()` method:

    ```java
    public String toString()
    {
        String s = "";
        int k;

        k = 0;
        for ( int i = 0; i < 4; i++ )
        {
            for ( int j = 1; j <= 13; j++ )
                s += ( deckOfCards[k++] + " " );

            s += "\n"; // Add NEWLINE after 13 cards
        }
        return ( s );
    }
    ```
• Operations on a deck of cards
  
  What can you do with a deck of playing cards:

<table>
<thead>
<tr>
<th>Shuffle up and...</th>
<th>Deal !!!</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Shuffle card image]</td>
<td>![Deal card image]</td>
</tr>
</tbody>
</table>
- **Deal a card:**

  - First, we **return** the Card object stored at the `currentCard` position.
    
    (In this example, the Card object that will be returned is **Ac (Ace of Club)**)

  - Then, we **advance** the `currentCard` position to the **next** Card object:

    ```
    currentCard++;
    ```

  Result:

  ```
  Deck of Cards:

  Array of Card Objects

  deckOfCard[0]  Ac
  deckOfCard[1]  Ks
  deckOfCard[2]  9h
  deckOfCard[3]  6d
  ```

  "deal this card out of the deck"
public static final int NCARDS = 52;

private Card[] deckOfCards; // Contains all 52 cards
private int currentCard; // deal THIS card in deck

public DeckOfCards( )
{
    deckOfCards = new Card[ NCARDS ];

    int i = 0;

    for ( int suit = Card.SPADE; suit <= Card.DIAMOND; suit++ )
        for ( int rank = 1; rank <= 13; rank++ )
            deckOfCards[i++] = new Card(suit, rank);

    currentCard = 0; // Fresh deck of card...
}

/* --------------------------------------------------------------
 * deal();  deal the next card in the deck
 * i.e.  deal deckOfCards[currentCard] out
 * -------------------------------------------------------------- */

public Card deal()
{
    if ( currentCard < NCARDS )
    {
        return ( deckOfCards[ currentCard++ ] );
    }
    else
    {
        System.out.println("Out of cards error");
        return ( null ); // Error;
    }
}
Shuffling a deck of cards

- The **classic solution** (a well-known trick in Computer Science) used to **shuffle objects stored an array** is the following **algorithm**:

```plaintext
repeat for many times
{
    select 2 random object in the array
    exchange the selected objects
}
```

**Example:** (shuffling an array of integers)

- **Initial** array of 10 integers:
  
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
</table>

- Pick 2 elements randomly and exchange them:
  
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
</table>
  **Result:**
  
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>9</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>3</th>
<th>10</th>
</tr>
</thead>
</table>

- Pick 2 elements randomly and exchange them:
  
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>9</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>3</th>
<th>10</th>
</tr>
</thead>
</table>
  **Result:**
  
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>9</th>
<th>4</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>8</th>
<th>3</th>
<th>10</th>
</tr>
</thead>
</table>

- And so on.... (the elements will become more and more random)
Pseudo code:

```
// Input: n = # exchange operations performed
// Returns nothing....
public void shuffle(int n)
{
    for (k = 1, 2, 3, ..., n)
    {
        i = a random integer number between [0 .. 51];
        j = another random integer number between [0 .. 51];
        exchange: deckOfCard[i] and deckOfCard[j];
    }
}
```

- The `shuffle()` method:

```
/* -----------------------------------------------
shuffle(n): shuffle the deck using n exchanges
----------------------------------------------- */
public void shuffle(int n)
{
    int i, j, k;
    for (k = 0; k < n; k++)
    {
        i = (int) (NCARDS * Math.random());  // Pick 2 random cards
        j = (int) (NCARDS * Math.random());  // in the deck

        /* -----------------------------------------------
Swap these randomly picked cards
----------------------------------------------- */
        Card tmp = deckOfCards[i];
        deckOfCards[i] = deckOfCards[j];
        deckOfCards[j] = tmp;
    }

    currentCard = 0;  // Reset current card to deal from top of deck
```
Using the DeckOfCards class

- Example Java program showing how to use the DeckOfCards class:

```java
public class TestDeck
{
    public static void main(String[] args)
    {
        DeckOfCards a;

        a = new DeckOfCards();
        System.out.println(a); // What a new deck look like
        System.out.println("Shuffle cards....");
        a.shuffle(1000); // Shuffle deck of card "a"
        System.out.println(a); // Deck after shuffling

        Card b;

        b = a.deal();
        System.out.println("Deal a card: "+b);
        b = a.deal();
        System.out.println("Deal a card: "+b);
        b = a.deal();
        System.out.println("Deal a card: "+b);
        b = a.deal();
        System.out.println("Deal a card: "+b);
        b = a.deal();
        System.out.println("Deal a card: "+b);
    }
}
```
Introduction to the game of *Poker*

- [http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/10/poker.html](http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/10/poker.html)
Desiging a Poker Playing library: Overview

- **Functionality needed to play Poker**
  - What do you need to **play Poker**?

  - **Information** on the cards held by a player
    - We can store this information using an array of Card object:
      ```java
      Card[] player = new Card[5];
      ```

  - **Operations** to check the hand of a player
    - Operations:
      - `isFlush(playerHand)`: returns `true` if `playerHand` has a **Flush**, and returns `false` otherwise.
      - `isStraight(playerHand)`: returns `true` if `playerHand` has a **Straight**, and returns `false` otherwise.
      - `is4s(playerHand)`: returns `true` if `playerHand` has a **Four Of a Kind**, and returns `false` otherwise.
      - `isFullHouse(playerHand)`: returns `true` if `playerHand` has a **Full House**, and returns `false` otherwise.
      - And so on....

  - **Operation** to assign a value to the hand of a player
    - `valueHand(playerHand) = a number` that represents the **strength** of the `playerHand`
    - The **assigned value** must be such that a **better Poker hand** is assigned a **higher value** !!!
      (So that we can simply compare the values (integer numbers) to find out which is the **better Poker hand**)
• Representing a Player's (Poker) hand
  
  - As stated above, we will use an **array of 5 card objects**
  
  - **Example:**

```java
public class TestPoker1 {
    public static void main(String[] args) {
        DeckOfCards a;
        Card[] player1 = new Card[5];
        Card[] player2 = new Card[5];
        int i;

        a = new DeckOfCards();
        System.out.println(a);

        System.out.println("Shuffle cards....");
        a.shuffle(1000);
        System.out.println(a);

        /* -------------------------------
           Deal cards to 2 players
           ------------------------------- */
        for (i = 0; i < 5; i++) {
            player1[i] = a.deal();
            player2[i] = a.deal();
        }

        System.out.print("player1's hand: ");
        for (i = 0; i < 5; i++)
            System.out.print( player1[i] + " ");
        System.out.println();

        System.out.print("player2's hand: ");
        for (i = 0; i < 5; i++)
            System.out.print( player2[i] + " ");
        System.out.println();
    }
}
```
Designing the Poker library: Checking for Poker hands

- Methods used to check for Poker hands
  - The *different types* of *Poker hands* that need to be checked:
    - Flush
    - Straight (*Straight flush* is a *Straight* that is *also* a *Flush*)
    - Four of a Kind
    - Full House
    - Set (Three of a Kind)
    - Two pairs
    - One pair

- Checking for each *type of Poker hand* will be performed by *one method*
What kind of method should we use: class or instance method?

- If the information is stored inside an object, the operation is implemented as an instance method.
- If the information is NOT stored inside an object, the operation is implemented as a class method.
We should use *class* method

**Diagram:**
- Input of the method
- `isFlush()` function
- Outputs: `true`, `false`
The **Poker class** will then look something like this:

```java
public class Poker {
    ... define class variables (if necessary) ....

    // Use class methods to check on
    // the input poker hand h

    public static boolean isFlush( Card[] h )
    {
        ....
    }

    public static boolean isStraight( Card[] h )
    {
        ....
    }

    ...
}
```

**Summary of the methods**

- We will present the following **Poker hand checking methods** (in the order given):
  - `isFlush(Card[] h)`: detects whether the Poker hand h contains a **flush**
  - `isStraight(Card[] h)`: detects whether the Poker hand h contains a **straight**
  - `is4s(Card[] h)`: detects whether the Poker hand h contains a **Four of a Kind**
  - `isFullHouse(Card[] h)`: detects whether the Poker hand h contains a **Full House**
  - `is3s(Card[] h)`: detects whether the Poker hand h contains a **Three of a Kind**
  - `is22s(Card[] h)`: detects whether the Poker hand h contains **2 pairs**
Checking for a flush

How do we detect a flush:

Flush:
- There is one suit of cards in the Poker hand

Hint:
- Try sorting the cards first....
Algorithm in Pseudo Code:

Sort the cards in the Poker hand by the suit;

if ( lowest suit == highest suit )
    Hand contain a flush (only 1 suit of cards in the hand!);
else
    Hand does not contain a flush;

The `isFlush()` method in Java:

```java
/* ---------------------------------------------------------------
 isFlush(): true if h has a flush
             false otherwise
 --------------------------------------------------------------- */

public static boolean isFlush( Card[] h )
{
    if ( h.length != 5 )
        return(false);  // Make sure we have 5 cards....

    sortBySuit(h);    // Sort the cards by the suit values

    return( h[0].suit() == h[4].suit() );  // All cards has same suit
}
```
Helper method: sortBySuit()

- The sortBySuit() helper method:

```java
/* Sort hand by suit:
   smallest suit card first ....
   (Finding a flush is easier that way)
   ------------------------------------------------- */
public static void sortBySuit(Card[] h )
{
    int i, j, min_j;
    /* ------------------------------------------------------
    The selection sort algorithm
    ------------------------------------------------------ */
    for ( i = 0 ; i < h.length ; i ++ )
    {
        /* -----------------------------------------------
        Find array element with min. value among
        h[i], h[i+1], ..., h[n-1]
        -------------------------------------------------- */
        min_j = i;  // Assume elem i (h[i]) is the minimum
        for ( j = i+1 ; j < h.length ; j++ )
        {
            if ( h[j].suit() < h[min_j].suit() )
            {
                min_j = j;  // We found a smaller suit value, update min_j
            }
        }
        /* -----------------------------------------------
        Swap a[i] and a[min_j]
        -------------------------------------------------- */
        Card help = h[i];
        h[i] = h[min_j];
        h[min_j] = help;
    }
}
```

Which sorting algorithm is used?
Prepare to play Poker --- Assigning values to Poker hands

- Computing the **value** of a Poker hand
  - The **value** of a Poker hand depends on the **type** of Poker hand

- We can give the following **general method** that determines the **value** of a Poker hand

```java
public static int valueHand( Card[] h )
{
    if ( isFlush(h) && isStraight(h) )
        return valueStraightFlush(h);
    else if ( is4s(h) )
        return valueFourOfAKind(h);
    else if ( isFullHouse(h) )
        return valueFullHouse(h);
    else if ( isFlush(h) )
        return valueFlush(h);
    else if ( isStraight(h) )
        return valueStraight(h);
    else if ( is3s(h) )
        return valueSet(h);
    else if ( is2s(h) )
        return valueTwoPairs(h);
    else if ( is2(h) )
        return valueOnePair(h);
    else
        return valueHighCard(h) // Lowest Poker hand;
}
```

- Assigning values to Poker hands: another **encoding scheme**
  - In order to **compare** Poker hands, we want to **assign values** to Poker hands such that:

```plaintext
Value( a better Poker hand ) > Value( a worse Poker hand )
```
• *Inter* and *intra* Poker hand comparisons

  ◦ 2 kinds of comparisons:

    - *Inter Poker hands*: comparison between *different kinds* of Poker hands
    - *Intra Poker hands*: comparison between *same kind* of Poker hands

  ◦ *Inter* Poker hand comparisons:

    Royal Flush
    > Straight Flush
    > 4 of a Kind
    > Full House
    > Flush
    > Straight
    > 3 of a Kind
    > 2 pairs
    > 1 pair
    > High Card

  ◦ *Intra* Poker hand comparisons:

    Royal Flush: all equal value
    Straight Flush: highest rank card wins
    4 of a Kind: higest rank quads wins
    Full House: highest rank (3) cards wins
    Flush: highest rank card wins
    Straight: highest rank card wins
    3 of a Kind: highest rank (3) cards wins
    2 pairs: highest pair wins
    if tie, lowest pair wins
    if also tie, highest unmatched card wins
    1 pair: highest pair wins
    if tie, highest unmatched card wins
    if also tie, second highest unmatched card wins
    if also tie, lowest unmatched card wins
    High Card: highest rank card wins
    if tie, second highest rank card wins
    and so on...
- Value encoding method that take cares of both *inter* and *intra* Poker hand comparision

  - **Encoding method** for the **value** of a **Poker hand**:

    ![Diagram showing ranges of Poker hands](image)

    - **High Card range**: 1000000
    - **One Pair range**: 2000000
    - **Two Pairs range**: 3000000
    - **Set range**: 4000000
    - **Straight range**: 5000000
    - **Flush range**: 6000000
    - **Full House range**: 7000000
    - **4 of a Kind range**: 8000000

- **Within** the **same type (kind)** of **Poker hand**:

  - A **higher ranked** hand is assigned a **higher value**
  - We make sure the **assigned value** does **not exceed** the **prescribe range**
Example:

![Diagram](image)

Explanation:

- Suppose you have 3 flush hands:
  - Spade Flush with cards 2, 3, 4, 5, 7
  - Heart Flush with cards 2, 3, 4, 6, 7
  - Club Flush with cards 2, 3, 5, 6, 7

- The different flush hand must receive different values in order to tell them apart !!!
- *Inter* Poker hand encoding

  - The *inter* Poker hand encoding is used to separate the *different kinds* of Poker hand

  - We can use the following *function* to do this job:

    ```
    InterValue( Straight Flush ) = 8000000
    InterValue( Four of a Kind ) = 7000000
    InterValue( Full House ) = 6000000
    InterValue( Flush ) = 5000000
    InterValue( Straight ) = 4000000
    InterValue( 3 of a Kind ) = 3000000
    InterValue( Two pairs ) = 2000000
    InterValue( One pair ) = 1000000
    InterValue( High Card ) = 0
    ```

- Note:

  - We *must* make sure that:

    ```
    IntraValue( Poker Hand ) < 1000000
    ```

  - Otherwise, we *cannot satisfy* the value encoding method given in:
Computer playing Poker

- Playing the Poker game between 2 players
  - Writing Poker game is now very easy using the methods contained in the Poker class
  - Here is a program that simulate a 2 player Poker game:

```java
Shuffle the deck of cards;
Deal 5 cards to player1
Deal 5 cards to player2

if ( Poker.valueHand(player1) > Poker.valueHand(player2) )
    System.out.println("Player 1 wins");
else if ( Poker.valueHand(player1) < Poker.valueHand(player2) )
    System.out.println("Player 2 wins");
else
    System.out.println("Player hands are a tie");
```
public class PlayPoker
{
    public static void main(String[] args)
    {
        DeckOfCards a;

        Card[] player1 = new Card[5];
        Card[] player2 = new Card[5];

        int i;

        a = new DeckOfCards();
        System.out.println(a);

        System.out.println("Shuffle cards....");
        a.shuffle(1000);        // Shuffle up...
        System.out.println(a);

        // Deal !
        for ( int i = 0; i < 5; i++ )
        {
            player1[i] = a.deal();
            player2[i] = a.deal();
        }

        System.out.println("Player 1's hand: ");
        for ( int i = 0; i < 5; i++ )
        {
            System.out.print( player1[i] + " ");
            System.out.println(" - value = " + Poker.valueHand(player1) );
        }

        System.out.println("Player 2's hand: ");
        for ( int i = 0; i < 5; i++ )
        {
            System.out.print( player2[i] + " ");
            System.out.println(" - value = " + Poker.valueHand(player2) );
        }

        System.out.println();
        System.out.println();
        System.out.println();

        if ( Poker.valueHand(player1) > Poker.valueHand(player2) )
        {
            System.out.println("Player 1 wins");
        }
        else if ( Poker.valueHand(player1) < Poker.valueHand(player2) )
        {
            System.out.println("Player 2 wins");
        }
        else
        {
            System.out.println("Player hands are a tie");
        }
        System.out.println();
    }
}