Lecture 30
Using the Card class

- Using the card class
  - Look only at the public things in the Card class:

```java
public class Card {
    public static final int SPADE = 4;
    public static final int HEART = 3;
    public static final int CLUB = 2;
    public static final int DIAMOND = 1;

    public Card(int suit, int rank) ...

    public int suit() ...

    public String suitStr() ...

    public int rank() ...

    public String rankStr() ...

    public String toString() ...
}
```
The constant value SPADE:

- Because the definition contains the keyword static:

  ```java
  public static final int SPADE ....
  ```

  it is a class constant and the syntax used to refer to SPACE is:

  ```java
  Card.SPADE
  ```
The constructor `Card(int suit, int rank)`:

- We know that a **constructor** is used to **initialized** a newly created **card** object.
- From its **definition**, we know **this particular constructor** takes **2 int parameters**.
- **Example usage:**

  ```java
  Card x;
  x = new Card(Card.CLUB, 1);  // Ace of club
  ```

- **Note:**
  - In order to use a **method effectively**, we need to know the **meaning** of each **parameter**.
  - I.e., you need to **know** the **card encoding** method discussed above.
- The **method** `int suit()`

  - We know that it is an *instance method*  
    
    (Because it does *not have* the *keyword* `static`)
  
  - Therefore, the `suit()` *method* must be *invoked* using an *object variable*, like this:

    ```java
    Card x;
    x = new Card( Card.CLUB , 1 );
    x.suit()  // (suit()) has no parameters
    ```

  - Because the *method* `suit()` *returns* an *int value*, we need to stored the return value in an *int typed* variable:

    ```java
    Card x;
    int  r;
    x = new Card( Card.CLUB , 1 );
    r = x.suit()  // (suit()) has no parameters
    ```
Example Java program showing how to use the Card class:

```java
public class TestCard1 {
    public static void main(String[] args) {
        Card x;
        int r;
        String s;

        x = new Card(Card.CLUB, 1); // Create a card
        r = x.suit();               // Get the suit of a card
        System.out.println("x.suit() = " + r);

        r = x.rank();               // Get the rank of a card
        System.out.println("x.rank() = " + r);

        s = x.suitStr();            // Get the suit of a card (Str)
        System.out.println("x.suitStr() = " + s);

        s = x.rankStr();            // Get the rank of a card (Str)
        System.out.println("x.rankStr() = " + s);

        System.out.println(x.toString()); // Convert a card into a String
        System.out.println(x);       // toString() invoked automatically
                                      // by Java to convert object to String!!!
    }
}
```

Output:

```
x.suit() = 2 (2 = club)
x.rank() = 14 (rank of Ace)
x.suitStr() = c (club)
x.rankStr() = A (Ace)
A
A
```
The variable definitions in `main()` first create the local variables:

```java
public static void main(String[] args)
{
    Card x;
    int r;
    x = new Card(Card.CLUB, 1);
    r = x.suit();
    ...
}
```
The `new Card` operator first create a `Card object` and returns a `object reference` to the local variable `x`:

```java
public static void main(String[] args) {
    Card x;
    int r;
    x = new Card(Card.CLUB, 1);
    r = x.suit();
    ...
}
```
Immediately, the execution will **invoke the constructor using** the newly created object.

```java
public static void main(String[] args) {
    Card x;
    int r;
    x = new Card(Card.CLUB, 1);
    r = x.suit();
}
...
```

**Constructor!**

```java
public Card(int suit, int rank) {
    if (rank == 1)
        cardRank = 14;
    else
        cardRank = (byte) rank;
    cardSuit = (byte) suit;
}
```
The parameters \( x, \text{Card.CLUB} \) and \( 1 \) are passed to the constructor method:

```java
public static void main(String[] args) {
    Card x;
    int r;
    x = new Card( Card.CLUB, 1);
    r = x.suit();
}
```

```java
public Card(int suit, int rank) {
    if ( rank == 1 )
        cardRank = 14;
    else
        cardRank = (byte) rank;
    cardSuit = (byte) suit;
}
```
The assignment `cardRank = 14` is shorthand for `this.cardRank = 14`, will store the value 14 into the instance variable `cardRank`.

```java
public static void main(String[] args)
{
    Card x;
    int r;
    x = new Card(Card.CLUB, 1);
    r = x.suit();
    ...
}
```

Constructor!

```java
public Card(int suit, int rank)
{
    if (rank == 1)
        cardRank = 14;
    else
        cardRank = (byte) rank;
    cardsuit = (byte) suit;
}
```
The assignment `cardSuit = suit` (is short hand for `this.cardSuit = suit`), will store the value 2 into the instance variable `cardSuit`.

```java
public static void main(String[] args) {
    Card x;
    int r;
    x = new Card(Card.CLUB, 1);
    r = x.suit();
    ...
}
```

Constructor!

```java
public Card(int suit, int rank) {
    if (rank == 1) {
        cardRank = 14;
    } else {
        cardRank = (byte) rank;
    }
    cardSuit = (byte) suit;
}
```
When the constructor exits, we have initialized the newly created object:

```java
public static void main(String[] args) {
    Card x;
    int r;
    x = new Card(Card.CLUB, 1);
    r = x.suit();
    ...
}
```

**RAM memory**

```
4000
```

```
cardSuit 2
```

```
cardRank 14
```

```
r
```

```
x 4000
```

```
public Card(int suit, int rank) {
    if (rank == 1)
        cardRank = 14;
    else
        cardRank = (byte) rank;
    cardSuit = (byte) suit;
}
Invoking of the instance method `x.suit()` is executed as follows:

The invocation will pass `x` as the implicit parameter `this`:

```java
public static void main(String[ ] args) {
    Card x;
    int r;
    x = new Card( Card.CLUB, 1);
    r = x.suit();
    ...
}
```

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

RAM memory:

- `main`
- `cardSuit` 2
- `cardRank` 14
- `this` 4000
- `r` 4000
- `x` 4000
- The **instance method** will now use the *implicit parameter* `this` to return the `cardSuit (= 2)` value:

```java
public static void main(String[] args) {
    Card x;
    int r;
    x = new Card(Card.CLUB, 1);
    r = x.suit();
    ...
}

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

because `return(cardSuit)` is the short hand for `return(this.cardSuit)`
Implementing a **deck** of cards

<table>
<thead>
<tr>
<th>Suit</th>
<th>Ace</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>
<th>Jack</th>
<th>Queen</th>
<th>King</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clubs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diamonds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hearts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spades</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example set of 52 poker playing cards
• Representing a deck of cards

  ○ To represent a deck of cards, we need 52 card objects

  ○ A possible representation:

    ```java
    public class DeckOfCards
    {
        private Card c1;
        private Card c2;
        ...
        private Card c52;
        // 52 Card variables !!!
    }
    ```

  ○ A much better representation is to use an array of card objects:

    ```java
    public class DeckOfCards
    {
        private Card[] deckOfCards;       // Used to store all 52 cards
    }
    ```

    We can use a constructor method to initialize the deck of card to contain the right cards.
The constructor method:

```java
public class DeckOfCards
{
    public static final int NCARDS = 52;
    private Card[] deckOfCards; // Contains all 52 cards

    /* -----------------------------------------------
    The constructor method: make 52 cards in a deck
    ----------------------------------------------- */
    public DeckOfCards()
    {
        /* -------------------------------
        First: create the array
        ----------------------------------------------- */
        deckOfCards = new Card[NCARDS]; // Very important !!
        // We must create the array first

        /* -----------------------------------------------
        Next: initialize all 52 card objects in the newly created array
        ----------------------------------------------- */
        int i = 0;
        for (int suit = Card.DIAMOND; suit <= Card.SPADE; suit++)
            for (int rank = 1; rank <= 13; rank++)
                deckOfCards[i++] = new Card(suit, rank); // Put card in
    }
}
```

Explanation:

- The variable `suit` will go through the values `Card.DIAMOND (= 1)` uptl and including `Card.SPADE (= 4)`
- The variable `rank` will go through the values 1 up to and including 13

So we will create $4 \times 13 = 52$ cards

- The variable `i` is incremented by 1 so each new card will be stored in a different array element `deckOfCards[i].`
• 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><img src="image" alt="Shuffling cards" /></td>
<td><img src="image" alt="Dealing cards" /></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" -> **Ac**

    - `currentCard = 1`
```java
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 in an array is the following algorithm:

  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>
<tbody>
<tr>
<td>Result:</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</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>
<tbody>
<tr>
<td>Result:</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

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

```java
// 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:

```java
/* -----------------------------------------
   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);
    }
}
```