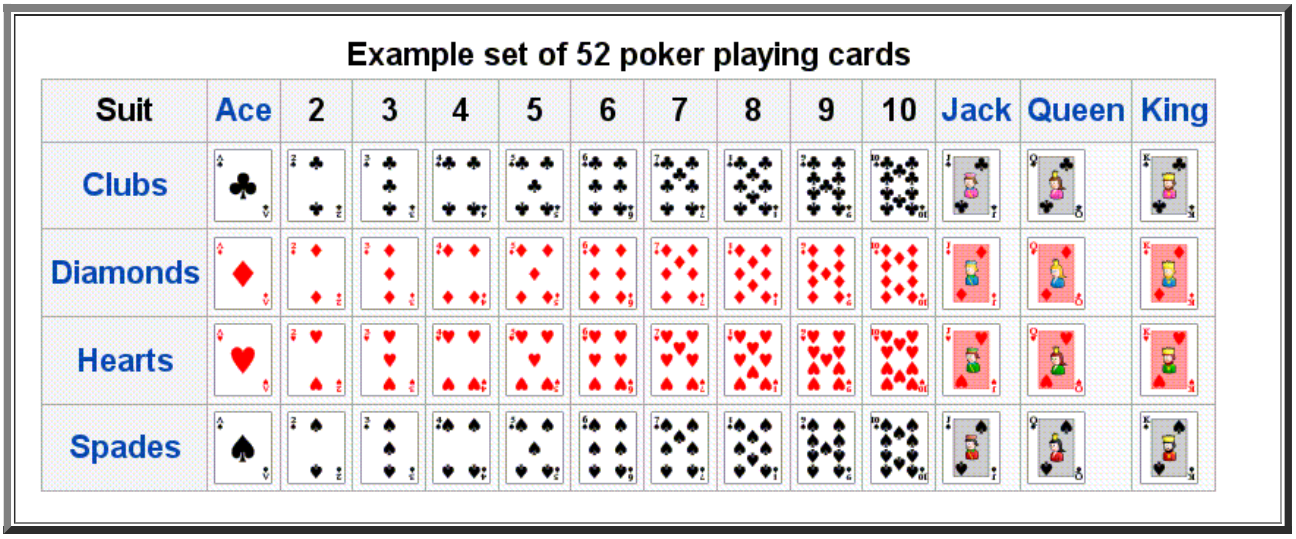
Implementing a *deck* of cards

- **A deck of playing cards**
  - What a deck of playing cards looks like for real:



- ***Representing* a deck of cards**
  - To **represent** a **deck of cards**, we need **52 `Card` objects**
  - A **possible representation**:

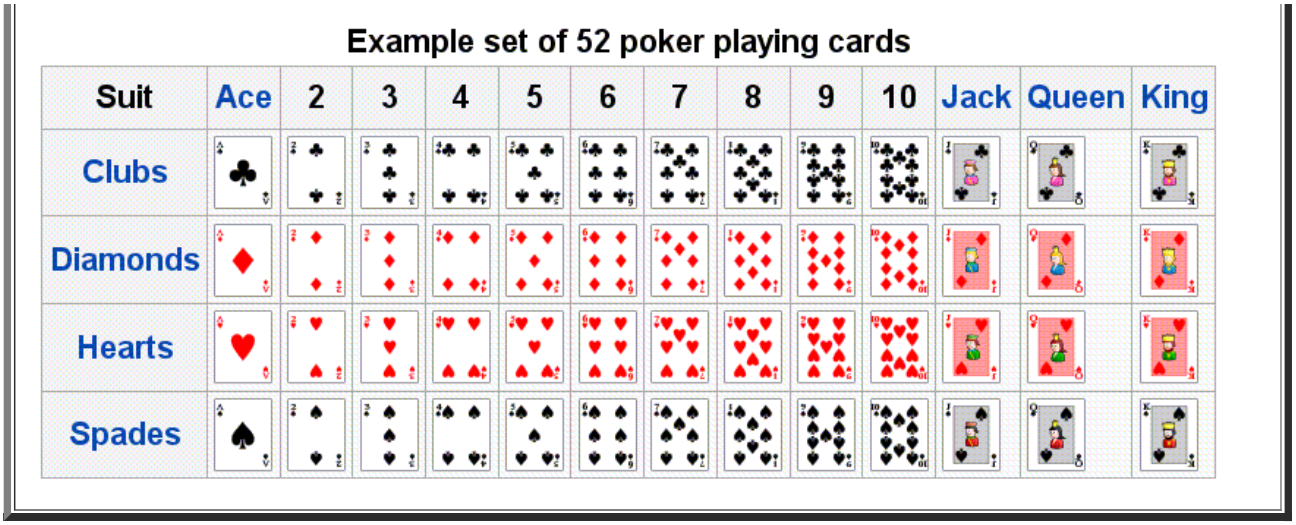
```
public class DeckOfCards
{
   private Card c1;
   private Card c2;
   ...
   private Card c52;
   // 52 Card variables !!!
}
```

  - A ***much better* representation** is to use an **array** of **`Card` objects**:

```
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**.

- **Constructor method(s) for a deck of cards**
  - We will have a **constructor** method to create a **deck** of cards containing:



- The **constructor method**:

```
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 crate 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
                                                       // position i
   }
}
```

Explanation:

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

- So we will **create 4×13 = 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[]`.

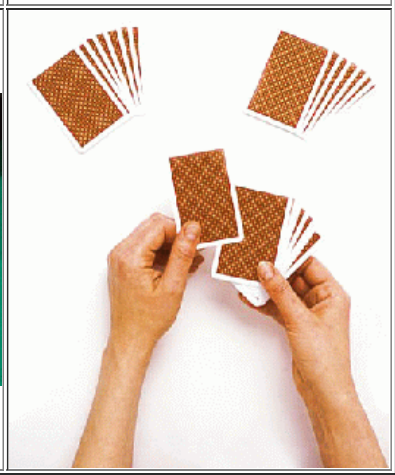
    We will return **13 cards** on **1 line**

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

  - The `toString()` method:

```
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**:

Shuffle up and...	**Deal** !!!
	

- **Simulating *"dealing cards"* from a deck of cards**
  - **Fact:**

    - A **computer program** ***cannot*** **deal cards** --- at least **not** ***physically***

  - **Simulation:**

    - In a **computer simulation**, we ***only*** aim to **achieve** the ***same result***

    What is the **result** of **dealing some cards**:

Result = **information** about **what cards has been dealt**

We can **represent** this **hand** (= a **collection of cards**) in a **computer program** with the following **5 *simulated* card objects (= information)**:

```
3h  6c  8h  Js  Jh
```

- **Furthermore:**

  - When a **card** is ***dealt* from the deck (of cards)**, the ***same* card cannot be dealt *again*** !!!

How do we **simulate** this ???

Answer:

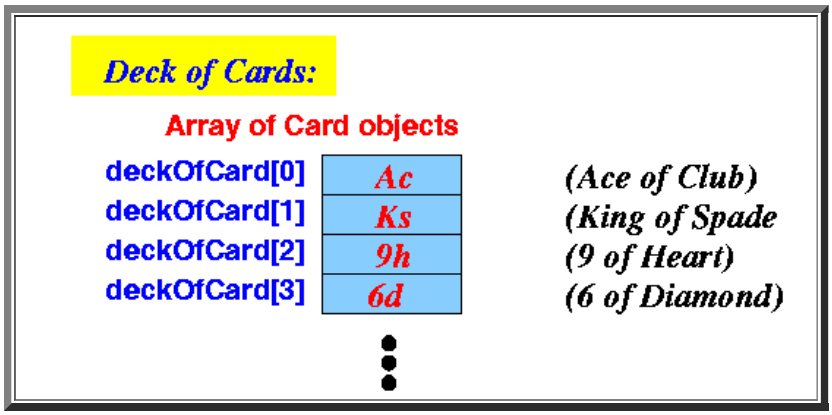
  - ***More* information** is necessary....

- **Recall** that we has **stored** a **deck of cards** as an ***array* of `Card` objects**:

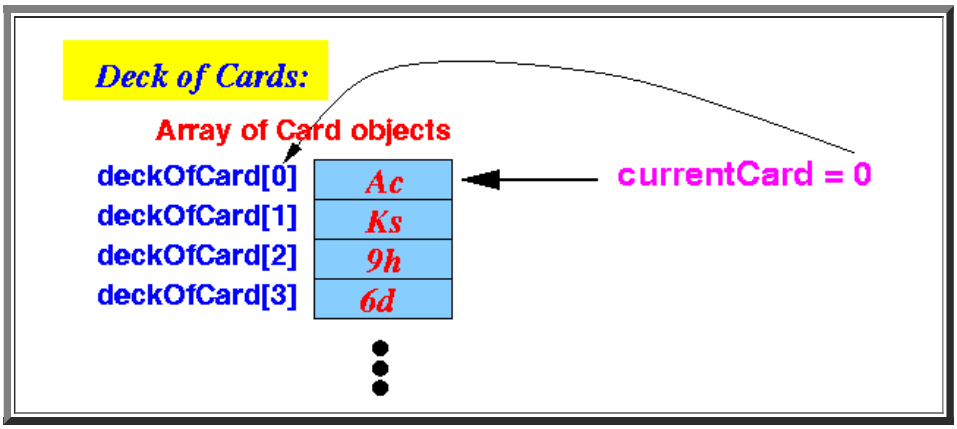
  - The **definition** of the **`DeckOfCards` class**

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

  - Depicted **graphically** (make things clearer):



We can **simulate *"dealing"* a card** from a **deck** using an **"current card" index variable**:



- **How to *simulate*** "dealing a card":

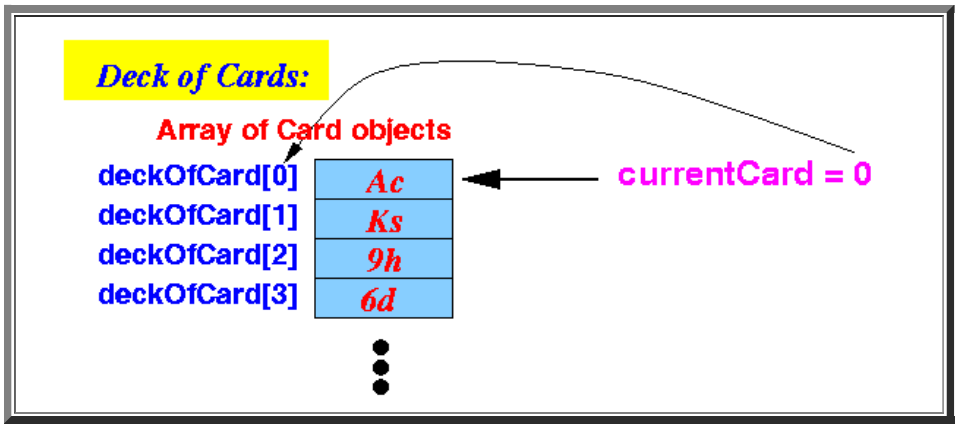
  - **Initial state**: (a fresh deck)

    - Suppose **no card** has been dealt yet.

      The **next card** that will be dealt is the **card** store at the **variable** `deckOfCard[0]`

      We initilize: `currentCard = 0`

    Graphical depiction of the **initial state**:



  - **Deal a card:**

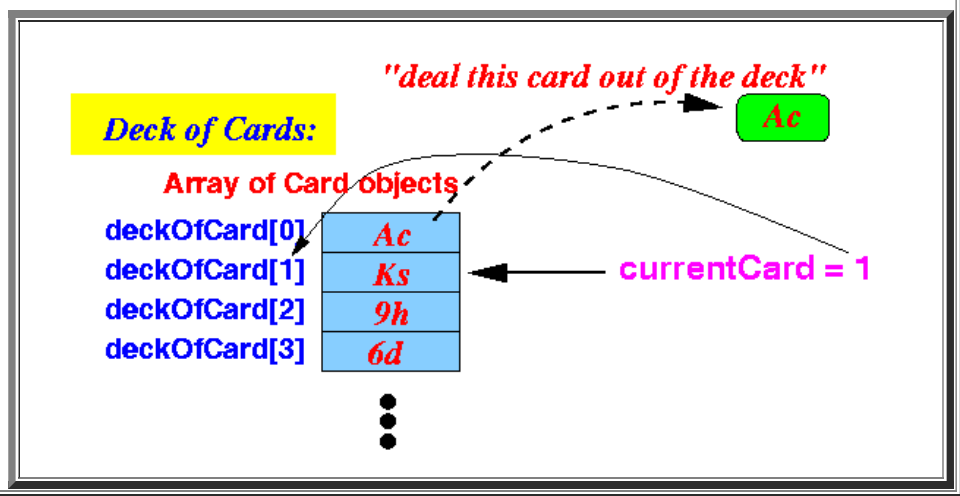
    - **First**, we **return** the **Card object** stored at the `currentCard` **position**

      (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:



- **The *updated* definition of the `DeckOfCards` class**

  - We need to **add** one more piece of **information** (namely: `currentCard`) to **simulate** ***dealing cards*** from a deck:

```
public class DeckOfCards
{
   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...
   }

   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";
      }
      return ( s );
   }
}
```

- OK, we are now **ready** to **implement** the **desired *operations*** on a **deck of cards**

  Let's **shuffle up** and **deal** !!!!

- ***Dealing a card* from the deck**
  - **Name of the "deal a card" method:**

    - Let's **pick** this name: `deal`

  - **Input parameters:**

    - **Dealing the next card** does not require any **additional information**

      Therefore: **no input parameters** necessary

  - **Output value:**

    - **Dealing the next card** must return the **current card** in the **deck**

      Therefore: the method **deal** must return a `Card` **object**

  - Therefore, the **header** of the `deal` method is as follows:

```
// No parameters
// Returns a "Card" object
public Card deal()
{
   ....
}
```

  - What must the `deal` method **do**:

- From the **above discussion** (see: **click here**):
  - **First**, we **return** the **Card object** stored at the `currentCard` **position**
  - **Then**, we **advance** the `currentCard` **position** to the ***next* Card object**

The `deal()` method:

```
public class DeckOfCards
{
   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;
      }
   }

   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";
      }
      return ( s );
   }
}
```

- **Note:**
  - In the `deal()` method, we make use of the fact that `currentCard++` **evaluates** to the ***old* value**
  - So the `Card` **object** that is **return** is the one **pointed to by the *old* value** of `currentCard` ***before*** the **increment operation** !!!

- ***Shuffling* a deck of cards**
  - The **effect** of ***shuffling* a deck of cards**:

- The **order** of the **cards** in the deck becomes **random**
- After **shuffling**, we start **dealing card** from the **top of the deck**

  (I.e., `currentCard` is **reset** to **0 (zero)**).

- The **classic solution** (a **well-known** ***trick*** in Computer Science) used to ***shuffle objects*** **stored 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**:

```
1   2   3   4   5   6   7   8   9   10
```

- Pick **2 elements** ***randomly*** and **exchange them**:

```
1   2   3   4   5   6   7   8   9   10

Result:
1   2   9   4   5   6   7   8   3   10
```

- Pick **2 elements** ***randomly*** and **exchange them**:

```
1   2   9   4   5   6   7   8   3   10

Result:
1   2   9   4   7   6   5   8   3   10
```

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

- **The `shuffle` method**
  - **Name** the method... let's pick this name:

```
shuffle
```

  - **Input parameters:**

    - We can tell the `shuffle` method the **number pair exchanges** that it needs to perform.

      Therefore: **parameter *n*** = **number of "exchanges" performed** (i.e., **how "long" it needs to shuffle**)

  - **Output value:**

    - **Shuffling a deck of cards** does ***not*** **return any value**

      (It only has **effect** of the **state of the cards in the deck**)

      Therefore: the method **shuffle** return a `void` **type**

  - Therefore, the **header** of the `shuffle` method is as follows:

```
// Input: n = # exchange operations performed
// Returns nothing....
public void shuffle(int n)
{
   ....
}
```

- What must the `deal` method **do**:

  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];
   }
}
```

- We need to solve **2 problems**:

  - **How to** pick a **random** ***integer*** **number** between **0 .. 51**.
  - **How to exchange** 2 cards in an array.

- **Picking** a **random** ***integer*** **number** between **[0 .. 51]**

  - The **method** `Math.random()` returns a **random number** betweem **[0 .. 1)**:

```
static double random()

    Returns a double value with a positive sign,
    greater than or equal to 0.0 and less than 1.0.

(See Java's API doc: click here)
```

  - Therefore:

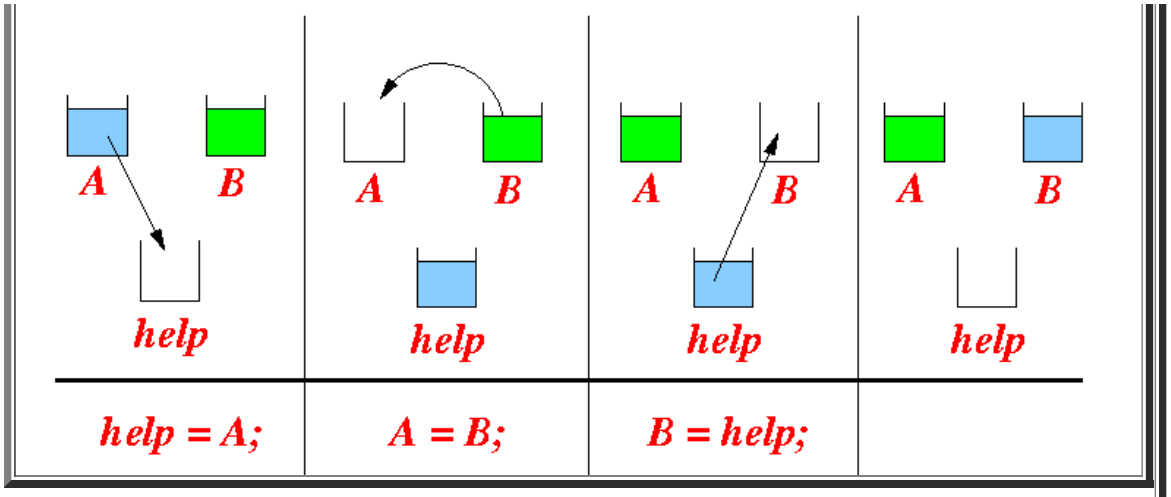
```
        0.0   ≤   Math.random()   <   1.0

  ⇒   0.0   ≤  52×Math.random() <  52.0

  ⇒   0   ≤  (int) (52×Math.random()) <  52
```

    The **integers** that are **≥ 0** and **< 52** are:

    - **0, 1, 2, 3, ..., 50, 51**

- **Exchanging 2 elements** in an **array**:

  - We **have seen** this **problem before** in the **Selection Sort Algorithm** --- See: **click here**
  - The **classic algorithm** used to **exchange the values of 2 variables** is the **3-way exchange algorithm**:



- **Exchange:** `deckOfCard[i]` and `deckOfCard[j]`

```
Card help;

help = deckOfCard[i];
deckOfCard[i] = deckOfCard[j];
deckOfCard[j] = help;
```

- 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
}
```

- **The `DeckOfCard` Class**

  - This is the **complete** definition of the `DeckOfCard` **class**

    It can **simulate** a **deck** of **52 playing cards**:

    - You can **shuffle** the deck of cards
    - You can **deal** a card to a player.

    (A **deck of "computer cards"** can only perform these **2 operations** !!!)

  - The `DeckOfCards` class definition:

```
/* ------------------------------------------------------
```

```
   Deck: a deck of cards
   ---------------------------------------------------- */

public class DeckOfCards
{
   public static final int NCARDS = 52;

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

   public DeckOfCards( )    // Constructor
   {
      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;
   }

   /* --------------------------------
      shuffle(n): shuffle the deck
      --------------------------------- */
   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
   }
   /* ------------------------------------------
      deal(): deal deckOfCards[currentCard] out
      ------------------------------------------ */
   public Card deal()
   {
      if ( currentCard < NCARDS )
      {
         return ( deckOfCards[ currentCard++ ] );
      }
      else
      {
         System.out.println("Out of cards error");
         return ( null );  // Error;
      }
   }

   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";
      }
      return ( s );
   }
}
```

- In the next web page, we will **use** the `DeckOfCards` class and deal a **poker hand (5 cards)**.