Implementing *playing cards*
• Data fields?
- **Properties** of a **playing card**:
  - Each **playing card** has a **suit**
    - Spade, Heart, Club or Diamond
  - Each **playing card** has a **rank**
    - A, 2, 3, ..., Q, K, A

  The **Ace** has 2 ranks, depending on the game played and how it is used in a hand....

- **Note**:
  - We will **not** use a **Joker** in our **playing card**
  - We will play **poker** with our **deck of cards** and this games does not use **Jokers**
  - We can **easily** add a **Joker** card with a **special value** (like 99) and add the card to the deck.
Representing playing cards

- **We first need a code** to represent the suit and rank of a playing card

- **Code** for the **suit** of a **playing card**:

  ```
  - 1 = Diamond
  - 2 = Club
  - 3 = Heart
  - 4 = Spade
  ```

  (I choose this code, because in **most games**, Spade ranks **higher** than Heart, Heart ranks **higher** than Club, and Club ranks **higher** than Diamond)

- **Code** for the **rank** of a **playing card**:

  ```
  - 2 = 2
  - 3 = 3
  - 4 = 4
  - ...
  - 10 = 10
  - 11 = J
  - 12 = Q
  - 13 = K
  - 14 = A
  ```

- **(Initial) Data structure** used to **represent** a **(one) playing card**:

  ```java
  public class Card {
      private byte cardSuit;
      private byte cardRank;
  }
  ```
Examples:

- `cardSuit = 1` and `cardRank = 4`
- `cardSuit = 2` and `cardRank = 9`
- `cardSuit = 3` and `cardRank = 13`
- `cardSuit = 4` and `cardRank = 14`
• **Storing the encoding method inside the Card class**

  - We can **add** the following **card suit** and **card value encoding information** inside the **Card class** to help with **decoding** the **card suit and card rank**:

    ```java
    public class Card {
        private static final String[] Suit = { "s", "d", "c", "h", "s"};
        private static final String[] Rank = { "a", "2", "3", "4", "5", "6", "7", "8", "9", "10", "j", "q", "k", "a"};

        private byte cardSuit;
        private byte cardRank;
    }
    ```

  - **How to use** the information in the arrays **Suit[]** and **Rank[]**:

    - **Suit[0]** is not used ("*" means error)
    - **Suit[1]** contains the string "d" (for Diamond)
    - **Suit[2]** contains the string "c" (for Club)
    - **Suit[3]** contains the string "h" (for Heart)
    - **Suit[4]** contains the string "s" (for Spade)
    - **Rank[2]** contains the string "2"
    - ...
    - **Rank[10]** contains the string "10"
    - **Rank[11]** contains the string "j"
    - **Rank[12]** contains the string "Q"
    - **Rank[13]** contains the string "K"
    - **Rank[14]** contains the string "A"

  So:

    ```java
    Suit[ cardSuit ]  gives us the suit information of the card
    Rank[ cardRank ]  gives us the rank information of the card
    ```
- `cardsuit = 1` and `cardrank = 4`
  
  
- `cardsuit = 2` and `cardrank = 9`
  
  
- `cardsuit = 3` and `cardrank = 13`
  
  
- `cardsuit = 4` and `cardrank = 14`
  
Further information encoding:

- We can improve the readability by defining some symbolic constants that describe the suit information:

```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;

    private static final String[] Suit = { "\x00", "d", "c", "h", "s" };
    private static final String[] Rank = { "\x00", "\x00", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A" };

    private byte cardSuit;
    private byte cardRank;
}
```

With these symbolic constants, we don’t have to remember the facts that: 4 = Spade, 3 = Heart, and so on...
• Constructors?
Purpose of constructor methods

- Create a playing card

Information necessary to create a playing card:
- The suit of the card
- The rank of the card

A useful constructor:

```java
public Card ( int suit, int rank )
{
    initial the instance variables "cardSuit" and "cardRank"
    to represent this play card.
}
```

Is the default constructor useful?

- Is there a special card in the deck that you would want to create by default?
  - Not likely
  - So: omit it...

Is the copy constructor useful?

- Would you want to copy an existing card (in a deck of card)?
  - Not likely, because each card is unique
  - So: omit it...
Constructor Method for 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;

    private static final String[] Suit = {"\"", "d", "c", "h", "s"};
    private static final String[] Rank = {"\"", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"};

    private byte cardSuit;
    private byte cardRank;

    /* -----------------------------------------------
     * Constructor
     * Example usage:
     * Card x = new Card( Card.SPADE, 1 ); // Ace of Spade
     * Card x = new Card( Card.HEART, 11 ); // Jack of Heart
     * ----------------------------------------------- */

    public Card( int suit, int rank ) {
        if ( rank == 1 (Ace) )
            cardRank = 14; // Give Ace the rank 14
        else
            cardRank = (byte) rank;

        cardSuit = (byte) suit;
    }
}
```
Remember that the private variables (methods) are inaccessible by program statements located outside the Card class.

So to your program, the Card class looks like THIS:

```java
public class Card {
    public Card( int suit, int rank ) {
        if ( rank == 1 (Ace) )
            cardRank = 14; // Give Ace the rank 14
        else
            cardRank = (byte) rank;
        cardSuit = (byte) suit;
    }
}
```

In other words:

- With the definition of the Card class that we have written so far, a Java program can only create a card object.
- A Java program cannot do any other operations on a card object !!!
• Methods?
Operations:

- Get the suit of a playing card as a number (for computational purposes)
- Get the suit of a playing card as a String (for printing purposes)
- Get the rank of a playing card as a number (for computational purposes)
- Get the rank of a playing card as a String (for printing purposes)

Note:

- We do not need to change the suit or the rank of a playing card
  
  So... we do not need to implement any update operations
public int suit()
{
    return ( cardSuit );  // This is a shorthand for:
    //    this.cardSuit
}

public String suitStr()
{
    return ( Suit[ cardSuit ] );  // This is a shorthand for:
    //    this.Suit[ this.cardSuit ]
}

public int rank()
{
    return ( cardRank );
}

public String rankStr()
{
    return ( Rank[ cardRank ] );
}
• Printing a Playing Card

  o Programming advice:

    - **Always** include the `toString()` method in a class
    
    This will **allow** the Java Program to **print the object variable** in a **format** that is **tailored for that object**

  o Printing a **Playing card**

    - We will print a **playing card** as

      ```java
      public String toString()
      {
        return ( Rank[ cardRank ] + Suit[ cardSuit ] );
      }
      ```

        **Example:**

        - `5s` = 5 of spade
        - `4h` = 4 of heart
        - `Kd` = K of diamond
        - `Qc` = Q of club
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 store 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 \((x, \text{Card.CLUB} \text{ and } 1)\) 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;
}
```

RAM memory:
- `main`
- `4000`
- `cardSuit`
- `cardRank`
- `suit this`
- `rank`
- `1`
- `2 (= Card.CLUB)`
- `4000`
- `r`
- `x`
- `4000`
The assignment `cardRank = 14` (is short hand 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();
    ...}
```

```
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);
}
```
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)`