“Achieving a goal provides immediate satisfaction; the process of achieving a goal is a lasting pleasure.”

**Evelyn Berezin**
Inventor of the word processor

http://www.computerhistory.org/fellowawards/hall/evelyn-berezin/
Deck of cards

Every card in a standard deck of 52 cards (no jokers) has two properties:

- A suit (hearts, spades, diamonds, clubs)
- A value (ace, two, three, four, five, six, seven, eight, nine, ten, jack, queen, king)
Enumeration in Java

An enumeration, or “enum” for short, is a data type you can create that takes on a fixed number of named values.

```java
enum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}
```
Enumeration example

```java
enum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}

public class Blackjack {
    public static void main(String[] args) {
        System.out.println(Suit.HEARTS);
    }
}
```

**Output:**

HEARTS
Enumeration ordinals

The named values of an enumeration also have corresponding integer values called *ordinals*:

```java
tenum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}

class Blackjack {
    public static void main(String[] args) {
        System.out.print(Suit.HEARTS.ordinal());
        System.out.print(" ");
        System.out.print(Suit.SPADES.ordinal());
        System.out.print(" ");
        System.out.print(Suit.DIAMONDS.ordinal());
        System.out.print(" ");
        System.out.print(Suit.CLUBS.ordinal());
    }
}
```

*Output:*

0 1 2 3
Array of enumeration values

You can get an array of all values in an enumeration:

```java
enum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}

public class Blackjack {
    public static void main(String[] args) {
        Suit[] suits = Suit.values();
        for (int suitIndex = 0; suitIndex < suits.length; suitIndex++) {
            System.out.println(suits[suitIndex]);
        }
    }
}
```

Output:
HEARTS
SPADES
DIAMONDS
CLUBS
Another enumeration for card values

eenum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}

eenum Value {
    ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING
}

class Blackjack {
    public static void main(String[] args) {
        System.out.println(Value.ACE + " of " + Suit.SPADES);
    }
}
How can we represent a deck of cards in Java?

Ideas from the class?
How can we represent a deck of cards in Java?

Some options:

- A 2D array of integers where each element represents some (suit, value) combination
- An array of 52 Card objects, where each Card contains a suit and a value
- A 2D array of Card objects, indexed by suit (row) and value (column)
How can we represent a deck of cards in Java?

Some options:

- A 2D array of integers where each element represents some (suit, value) combination
- An array of 52 Card objects, where each Card contains a suit and a value
- A 2D array of Card objects, indexed by suit (row) and value (column)

Let’s go with this one
To create a 2D array of Card objects:

1. Create a Card class

2. Create a Deck class that contains a 2D array of Card objects
Let’s plan first!

To create a class, we need to identify:

● Its member variables (nouns)
● Its methods (verbs)

For now, let’s not worry about methods.

**Question for the class:**
What are the member variables (nouns) we’d want to store in a Card class?
1. Create a Card class

Member variables:

- suit
- value
1. Create a Card class

```java
class Card {
    public Suit suit;
    public Value value;
}
```
We also need a **constructor** for the class, so we can actually create instances of the Card class.

```java
class Card {
    public Suit suit;
    public Value value;
    
    public Card(Suit suit, Value value) {
        this.suit = suit;
        this.value = value;
    }
}
```

The **constructor** is a **method** that sets the values of a Card object’s **suit** and **value** variables equal the values of parameters passed into the constructor.
Clicker question (not graded today)

What is the output of the following code?

class Number {
    public int num;

    public Number(int num) {
        this.num = num;
    }
}

public class Cs312 {
    public static void main(String[] args) {
        Number[] numbers = new Number[2];
        System.out.println(numbers[0].num);
    }
}

A. 0
B. 1
C. null
D. No output due to syntax error
E. No output due to runtime error
enum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}

denum Value {
    ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN,
    EIGHT, NINE, TEN, JACK, QUEEN, KING
}

class Card {
    public Suit suit;
    public Value value;
    
    public Card(Suit suit, Value value) {
        this.suit = suit;
        this.value = value;
    }
}

class Blackjack {
    public static void main(String[] args) {
        Card card = new Card(Suit.SPADES,
                              Value.ACE);
        System.out.println(
                           "Created a Card object!");
    }
}

Output:
Created a Card object!
2. Create a Deck class containing 2D array of Cards

Let’s plan first!

To create a class, we need to identify:

● Its member variables (nouns)
● Its methods (verbs)

For now, let’s not worry about methods.

**Question for the class:**
What are the member variables (nouns) we’d want to store in a Deck class?
2. Create a Deck class containing 2D array of Cards

Only one member variable (noun) needed:
- A 2D array of Cards, indexed by suit (row) and value (column)

class Deck {
    public Card[][][] deck;
}

2. Create a Deck class containing 2D array of Cards

And we need a constructor for the Deck class too.

What should the constructor method do for the Deck class?

The job of a constructor is mostly to initialize/instantiate all of its member variables.

The Deck class has only one member variable: Card[][] deck

How would we initialize this 2D array of Cards?
Initializing a 2D array of Cards

1. Initialize the whole array first, like you’d do for any 2D array

   ```java
   deck = new Card[something][something];
   ```

2. Then instantiate each Card that’s in the array
Initializing a 2D array of Cards

1. Initialize the whole array first, like you’d do for any 2D array
   
   ```
   deck = new Card[something][something];
   ```

2. Then instantiate each Card that’s in the array

   Let’s do Step 1 first!
enum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}

d enum Value {
    ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING
}

class Card {
    public Suit suit;
    public Value value;

    public Card(Suit suit, Value value) {
        this.suit = suit;
        this.value = value;
    }
}

class Deck {
    public Card[][] deck;

    public Deck() { 
        deck = new Card[something][something];
    }
}
Initializing the 2D array of Cards

We want the 2D array to have:
Initializing the 2D array of Cards

We want the 2D array to have:
- the rows correspond to suits
**Initializing the 2D array of Cards**

We want the 2D array to have:
- the rows correspond to suits
- the columns correspond to the values
Initializing the 2D array of Cards

We want the 2D array to have:
- **4 rows**
- **13 columns**

<p>| | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<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></td>
</tr>
<tr>
<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></td>
</tr>
<tr>
<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></td>
</tr>
</tbody>
</table>

The cards are arranged in rows and columns as shown in the diagram.
So, back to our original question....

```java
enum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}

enum Value {
    ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN,
    EIGHT, NINE, TEN, JACK, QUEEN, KING
}

class Card {
    public Suit suit;
    public Value value;

    public Card(Suit suit, Value value) {
        this.suit = suit;
        this.value = value;
    }
}

class Deck {
    public Card[][] deck;

    public Deck() {
        deck = new Card[something][something];
    }
}
```

What should these be?
So, back to our original question….

```java
enum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}

enum Value {
    ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING
}

class Card {
    public Suit suit;
    public Value value;

    public Card(Suit suit, Value value) {
        this.suit = suit;
        this.value = value;
    }
}
```

```java
class Deck {
    public Card[][] deck;

    public Deck() {
        deck = new Card[something][something];
    }
}
```

But let’s not use magic numbers….

What if we wanted this code to work for a reduced deck with a different number of Values (like games using only A, K, Q, J, 10, 9)?
Remember, the values() method for an enum type?

```java
enum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}

enum Value {
    ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN,
    EIGHT, NINE, TEN, JACK, QUEEN, KING
}

class Card {
    public Suit suit;
    public Value value;

    public Card(Suit suit, Value value) {
        this.suit = suit;
        this.value = value;
    }
}

class Deck {
    public Card[][] deck;
    public static final int NUM_SUITES = Suit.values().length;
    public static final int NUM_VALUES = Value.values().length;

    public Deck() {
        deck = new Card[NUM_SUITES][NUM_VALUES];
    }
}
```
Initializing a 2D array of Cards

1. Initialize the whole array first, like you’d do for any 2D array

   ```java
   deck = new Card[something][something];
   ```

2. Then instantiate each Card that’s in the array
Instantiate each Card in the 2D array

class Deck {
    public Card[][] deck;
    public static final int NUM_SUITS = Suit.values().length;
    public static final int NUM_VALUES = Value.values().length;

    public Deck() {
        deck = new Card[NUM_SUITS][NUM_VALUES];

        for (int suitIndex = 0; suitIndex < NUM_SUITS; suitIndex++) {
            for (int valueIndex = 0; valueIndex < NUM_VALUES; valueIndex++) {
                deck[suitIndex][valueIndex] = new Card(suit, value);
            }
        }
    }
}
class Deck {
    public Card[][] deck;
    public static final int NUM_SUITS = Suit.values().length;
    public static final int NUM_VALUES = Value.values().length;

    public Deck() {
        deck = new Card[NUM_SUITS][NUM_VALUES];

        for (int suitIndex = 0; suitIndex < NUM_SUITS; suitIndex++) {
            for (int valueIndex = 0; valueIndex < NUM_VALUES; valueIndex++) {
                deck[suitIndex][valueIndex] = new Card( suit, value);
            }
        }
    }
}
Instantiate each Card in the 2D array

class Deck {
    public Card[][] deck;

    public static final int NUM_SUITS = Suit.values().length;
    public static final int NUM_VALUES = Value.values().length;

    public Deck() {
        deck = new Card[NUM_SUITS][NUM_VALUES];

        Suit[] suits = Suit.values();
        Value[] values = Value.values();
        for (int suitIndex = 0; suitIndex < NUM_SUITS; suitIndex++) {
            Suit suit = suits[suitIndex];
            for (int valueIndex = 0; valueIndex < NUM_VALUES; valueIndex++) {
                Value value = values[valueIndex];
                deck[suitIndex][valueIndex] = new Card(suit, value);
            }
        }
    }
}
Okay, we’ve finished the Deck class. Now what?

Well, what do you do when you have a deck of cards?

Play cards!

Let’s see if we can build some pieces of code that would help us make a Java program to play a game of, say, Blackjack.
Why do we shuffle cards?

So we can draw cards at random!
public class Blackjack {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Card randomCard = deck.drawRandomCard();
    }
}
Method to draw (return) a random Card

class Deck {
    public Card[][] deck;
    public static final int NUM_SUITS = Suit.values().length;
    public static final int NUM_VALUES = Value.values().length;

    public Deck() {
        ...
    }

    public Card drawRandomCard() {
        Random random = new Random();
        int randomSuitIndex = random.nextInt(NUM_SUITS);
        int randomValueIndex = random.nextInt(NUM_VALUES);
        return deck[randomSuitIndex][randomValueIndex];
    }
}
What card did we draw?

```java
public class Blackjack {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Card randomCard = deck.drawRandomCard();
        // But now how do we see what card we drew?
    }
}
```
public class Blackjack {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Card randomCard = deck.drawRandomCard();
        // But now how do we see what card we drew?  
        // Maybe we could print it out somehow...
        // But how do we print out a Card’s suit and value?
    }
}
class Card {
    public Suit suit;
    public Value value;

    public Card(Suit suit, Value value) {
        this.suit = suit;
        this.value = value;
    }

    public String toString() {
        return value + " of " + suit;
    }
}
We can call toString() to print the Card’s suit & value

```java
public class Blackjack {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Card randomCard = deck.drawRandomCard();
        System.out.println(randomCard.toString());
    }
}
```
public class Blackjack {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Card randomCard = deck.drawRandomCard();
        System.out.println(randomCard.toString());
    }
}
....you can remove .toString() from print statements!

```java
public class Blackjack {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Card randomCard = deck.drawRandomCard();
        System.out.println(randomCard);
    }
}
```

.toString() is automatically called on objects that aren’t Strings!
What is the output of the following code?

class Number {
    public int num;

    public Number(int num) {
        this.num = num;
    }

    public String toString() {
        return "" + num / 2;
    }
}

public class Cs312 {
    public static void main(String[] args) {
        Number number = new Number(1);
        System.out.println(number);
    }
}
A hand

During a card game, the cards you’re holding in your hand are called a hand.

If we’re going to write a Java program for playing a card game, we need to create another class to represent the concept of a hand of cards.

What is a hand?

It’s just a bunch of cards.
Let's plan!

To create a class, we need to identify:
- Its member variables (nouns)
- Its methods (verbs)

For now, let's not worry about methods.

**Question for the class:**
What is/are the member variable(s) (nouns) we’d want to store in a Hand class?
An array of Cards!

class Hand {
    public Card[] cards;
}

How do we implement the Hand class’ constructor?

Let’s think about how we’d like the Hand class to be used:

● In a real card game like Blackjack, we start out empty-handed.
● Then we add cards to our hand one at a time.
How do we implement the `Hand` class’ constructor?

Let’s think about how we’d like the `Hand` class to be used:
- In a real card game like Blackjack, we start out empty-handed.
- Then we add cards to our hand one at a time.

In Java, that might look like this:

```java
public class Blackjack {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Hand hand = new Hand();
        hand.addCard(deck.drawRandomCard());
        hand.addCard(deck.drawRandomCard());
        hand.addCard(deck.drawRandomCard());
    }
}
```
How do we implement the Hand class’ constructor?

Let’s think about how we’d like the Hand class to be used:

- In a real card game like Blackjack, we start out empty-handed.
- Then we add cards to our hand one at a time.

In Java, that might look like this:

```java
public class Blackjack {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Hand hand = new Hand();
        hand.addCard(deck.drawRandomCard());
        hand.addCard(deck.drawRandomCard());
        hand.addCard(deck.drawRandomCard());
    }
}
```

(We might draw the same Card more than once with this approach, but let’s ignore that for now.)
How do we implement the Hand class’ constructor?

```java
class Hand {
    public Card[] cards;

    public Hand() {
        cards = new Card[0];  // start with zero Cards
    }

    public void addCard(Card card) {
        // Then, add Cards one at a time.
    }
}
```

Many ways to do this; let’s choose one approach, which is to start with an array of 0 Cards, and then add Cards to the Hand one at a time.
Okay, now let’s fill in the addCard() method!

Plan first!

How would we add a new card to the Hand class’s cards array?
Can we just do this?

class Hand {
    public Card[] cards;

    public Hand() {
        cards = new Card[0];  // start with zero Cards
    }

    public void addCard(Card card) {
        cards[cards.length] = card;
    }
}
Can we just do this?

class Hand {
    public Card[] cards;

    public Hand() {
        cards = new Card[0]; // start with zero Cards
    }

    public void addCard(Card card) {
        cards[cards.length] = card;
    }
}

Nope! ArrayIndexOutOfBoundsException!
We need to make the array bigger first, and then add the new Card to it!
So…. can we just make the array bigger like this?

class Hand {
    public Card[] cards;

    public Hand() {
        cards = new Card[0];  // start with zero Cards
    }

    public void addCard(Card card) {
        cards = new Card[cards.length + 1];
        cards[cards.length - 1] = card;
    }
}

Uh-oh! Now we’ve completely wiped out the old Card(s) that may have been stored in the cards array before we resized it!
Sigh, okay, how do we store the old Cards in the cards array before we resize it?
class Hand {
    public Card[] cards;

    public Hand() {
        cards = new Card[0]; // start with zero Cards
    }

    public void addCard(Card card) {
        Card[] existingCards = cards;
        cards = new Card[cards.length + 1];
        // Copy all existingCards into cards...
        cards[existingCards.length] = card;
    }
}

This just creates another “house” (reference) to the existing cards array’s elements, so the garbage collector doesn’t throw those Cards away. It is NOT a copy of the cards array.
Okay. How do we copy `existingCards → cards`?

class Hand {
    public Card[] cards;

    public Hand() {
        cards = new Card[0]; // start with zero Cards
    }

    public void addCard(Card card) {
        Card[] existingCards = cards;
        cards = new Card[cards.length + 1];
        // Copy all `existingCards` into `cards`...
        cards[existingCards.length] = card;
    }
}
Okay. How do we copy `existingCards → cards`?

class Hand {
    public Card[] cards;

    public Hand() {
        cards = new Card[0];  // start with zero Cards
    }

    public void addCard(Card card) {
        Card[] existingCards = cards;
        cards = new Card[cards.length + 1];
        for (int cardIndex = 0; cardIndex < existingCards.length; cardIndex++) {
            cards[cardIndex] = existingCards[cardIndex];
        }
        cards[existingCards.length] = card;
    }
}
Now we have Card, Deck, and Hand classes!

Now we can proceed to the next task: calculate the value of a Hand in Blackjack!

From Wikipedia's Blackjack article:
- The value of cards two through ten is their pip value (2 through 10).
- Face cards (Jack, Queen, and King) are all worth ten.
- Aces can be worth one or eleven.
- A hand's value is the sum of the card values.
- The objective is to get as close to 21 points as you can without going over.

So, we first need a method to calculate the value of a single Card in Blackjack. Then, we can write a method that calculates the value of a whole Hand by summing the values of the individual Cards in that Hand.
The value of a single Card in Blackjack

- Aces can be worth one or eleven. *(Let’s ignore 1 for now.)*
- Face cards (Jack, Queen, and King) are all worth ten.
- The value of cards two through ten is their pip value (2 through 10).

class Card {

    ...

    public int blackjackValue() {
        if (value == Value.ACE) {
            return 11;
        }

        if (value == Value.JACK || value == Value.QUEEN || value == Value.KING) {
            return 10;
        }

        return value.ordinal() + 1;
    }
}
Let’s test our new method **NOW!!**

```java
public class Blackjack {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Card card = deck.drawRandomCard();
        System.out.println("Card is " + card);
        System.out.println("Blackjack value is " + card.blackjackValue());
    }
}
```

Run this test program several times. You’ll get different outputs most of the time since you’re picking a card at random.

**Output:**
Card is SEVEN of HEARTS
Blackjack value is 7
Now we can calculate the value of a Hand

class Hand {
    public Card[] cards;

    public Hand() {
        cards = new Card[0];  // initially, start with 0 cards
    }

    public void addCard(Card card) {
        Card[] existingCards = cards;
        cards = new Card[cards.length + 1];
        for (int cardIndex = 0; cardIndex < existingCards.length; cardIndex++) {
            cards[cardIndex] = existingCards[cardIndex];
        } cards[existingCards.length] = card;
    }

    public int blackjackValue() {
        int sum = 0;
        for (int cardIndex = 0; cardIndex < cards.length; cardIndex++) {
            sum += cards[cardIndex].blackjackValue();
        }
        return sum;
    }
}
A (too) simple algorithm for playing Blackjack

```java
public class Blackjack {
    public static void main(String[] args) {
        Deck deck = new Deck();
        Hand hand = new Hand();
        while (hand.blackjackValue() < 21) {
            Card card = deck.drawRandomCard();
            System.out.println("Drew " + card);
            hand.addCard(card);
            System.out.println("Hand value is now " + hand.blackjackValue());
        }
    }
}
```

Possible output:
Drew QUEEN of SPADES
Hand value is now 10
Drew ACE of CLUBS
Hand value is now 21