# Model 1 Hand of Cards

Creating an array of objects is typically a 3-step process:

1. Declare the array

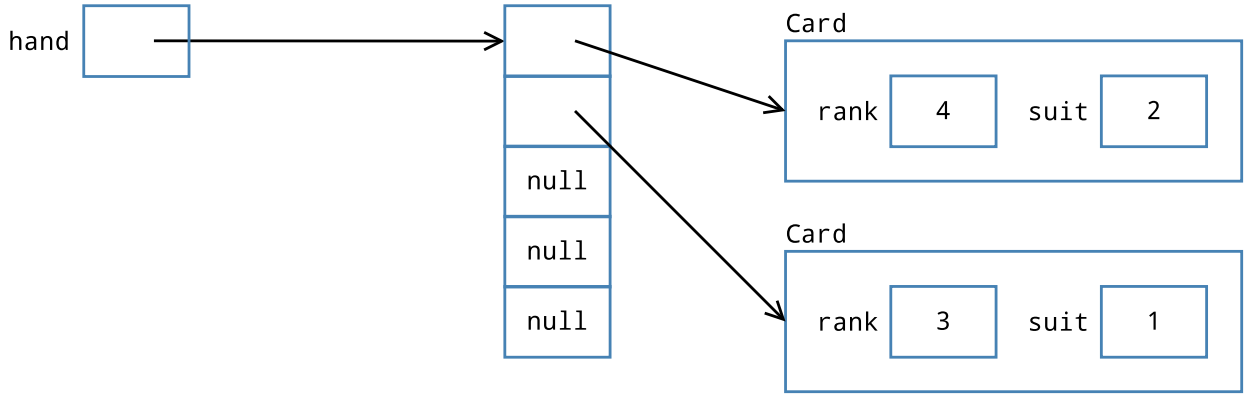
```
Card[] hand;
```

2. Instantiate the array

```
hand = new Card[5];
```

3. Instantiate each object

```
hand[0] = new Card(4, 2);
hand[1] = new Card(3, 1);
```




## Questions (20 min)

**Start time:**

**1**. What is the type of the local variable `hand`? What is the value of `hand` *before* step 2? What is the value of `hand` *after* step 2?

The variable `hand` is an array of `Card` objects. Before step 2, it's uninitialized (i.e., you can't read its value). After step 2, its value is the memory location of the first array element.

**2**. When you create an array (e.g., `new Card[5]`) what is the initial value of each element?

The initial values are automatically set to null (for reference types). For arrays of integers, it's 0; for doubles, it's 0.0; for booleans, it's `false`; for characters, it's the unicode character `'\u0000'`.

**3**. When you construct a new object (e.g., `new Card(4, 2)`) what are the initial values of its attributes (e.g., `this.rank`)?

It depends on the constructor. For the `Card` class, the attributes `rank` and `suit` are initialized from the parameters. If there is no constructor, Java automatically initializes attributes to zero (or equivalent).

*The `new` operator finds a memory location to store an array or object. Java automatically determines how much memory is needed and initializes the contents of the corresponding memory cells to zero. That's why array elements and object attributes have default values, whereas local variables (not allocated with `new`) must be initialized before they are used.*

**4.** Describe in your own words what the following code does. Be sure to explain how the random part works.

```
int index = (int) (Math.random() * hand.length);
hand[index] = null;
```

`Math.random()` returns a value in the range [0, 1). Multiplying that value by `hand.length` and then casting it to an integer gives a value in the range 0..5 inclusive. So this statement randomly sets one of the `Card` references to `null`.

**5.** What is the result of running the loop below? Explain why the if-statement is necessary.

```
for (int i = 0; i < hand.length; i++) {
    if (hand[i] != null) {
        int suit = hand[i].getSuit();
        System.out.println("The suit of #" + i + " is " + Card.SUITS[suit]);
    }
}
```

The loop prints the suits of all cards in the hand. Because some of the cards are `null`, the if-statement prevents `NullPointerException`.

**6.** The ***enhanced for loop*** allows you to iterate the elements of an array. Another name for this structure is the "for each" loop. Rewrite the following example using a standard for loop.

```
String[] days = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
for (String day : days) {
    System.out.println(day + " is a great day!");
}
```

```
for (int i = 0; i < days.length; i++) {
    System.out.println(days[i] + " is a great day!");
}
```

**7.** In contrast to enhanced for loops, what does a standard for loop iterate? Why would it be misleading to name the enhanced for loop variable `i` instead of `day`?

Standard for loops typically iterate indexes; that's why the variable is usually named `i`.

**8.** Rewrite the loop in #5 using an enhanced for loop. Use an appropriate variable name for the `Card` object (i.e., not `i`). For simplicity, you may omit the `System.out.println` line.

```
for (Card card : hand) {
    if (card != null) {
        int suit = card.getSuit();
    }
}
```