Arrays

- both parameter and return types can be arrays

```
/**
 * Print a bingo card. Only the uncrossed off numbers are printed.
 *
 * @param card the bingo card to print
 */
public static void printBingoCard(int[] card) {
    for (int i = 0; i < card.length; i++) {
        if (card[i] != -1) {
            System.out.printf(" %2d", card[i]);
        }
    }
    System.out.println();
}
```

```
/**
 * Create a bingo card with n numbers.
 *
 * @param n number of numbers on the card
 * @return a bingo card containing n random numbers
 */
public static int[] createBingoCard(int n) {
    int[] card = new int[n];
    for (int i = 0; i < card.length; i++) {
        card[i] = (int) (Math.random() * 11) + 2;
    }
    return card;
}
```


Subtleties of Parameters and Return Values

- parameters and return values are *passed by value*
  - a copy of the value of the expression is passed
  - for variables, this is a copy of what is in the box
- for primitive types (`int`, `double`, `boolean`, etc), the box contains the actual value
- for all other types (arrays, `String`, all types defined by classes), the box contains the address in memory where the actual value is stored
- this means –
  - assignment to a parameter has only a local effect (the parameter's box is local to the subroutine)
  - assignment to an array slot or invoking a mutable method on an object has a global effect (only the address was copied when the array/object was passed)


```
public static void func1(int a) {
    System.out.println("[func1] before: " + a);
    a = 10;
    System.out.println("[func1] after: " + a);
}
```

```
[main] before: 5
[func1] before: 5
[func1] after: 10

[main] after: 5
```

```
{
    int x = 5;

    System.out.println("[main] before: " + x);
    func1(x);
    scanner.nextLine();
    System.out.println("[main] after: " + x);
}
```

```
public static void func2(double a) {
    System.out.println("[func2] before: " + a);
    a = 10.1;
    System.out.println("[func2] after: " + a);
}
```

```
[main] before: 5.5
[func2] before: 5.5
[func2] after: 10.1

[main] after: 5.5
```

```
{
    double x = 5.5;

    System.out.println("[main] before: " + x);
    func2(x);
    scanner.nextLine();
    System.out.println("[main] after: " + x);
}
```



```
public static void func3(boolean a) {
    System.out.println("[func3] before: " + a);
    a = !a;
    System.out.println("[func3] after: " + a);
}
```

```
[main] before: true
[func3] before: true
[func3] after: false

[main] after: true
```

```
{
    boolean x = true;

    System.out.println("[main] before: " + x);
    func3(x);
    scanner.nextLine();
    System.out.println("[main] after: " + x);
}
```

```
public static void func4(String a) {
    System.out.println("[func4] before: " + a);
    a = "goodbye";
    System.out.println("[func4] after: " + a);
}
```

```
[main] before: hi
[func4] before: hi
[func4] after: goodbye

[main] after: hi
```

```
{
    String x = "hi";

    System.out.println("[main] before: " + x);
    func4(x);
    scanner.nextLine();
    System.out.println("[main] after: " + x);
}
```

```java
public static void func5(int[] a) {
    System.out.print("[func5] before: ");
    printArray(a);
    System.out.println();
    a = new int[] { 10, 9, 8 };
    System.out.print("[func5] after: ");
    printArray(a);
    System.out.println();
}
```

```
[main] before: 1 2 3 4 5
[func5] before: 1 2 3 4 5
[func5] after: 10 9 8

[main] after: 1 2 3 4 5
```

```java
{
    int[] x = { 1, 2, 3, 4, 5 };

    System.out.print("[main] before: ");
    printArray(x);
    System.out.println();
    func5(x);
    scanner.nextLine();
    System.out.print("[main] after: ");
    printArray(x);
    System.out.println();
}
```



```java
public static void func6(int[] a) {
    System.out.print("[func6] before: ");
    printArray(a);
    System.out.println();
    a[0] = 10;
    System.out.print("[func6] after: ");
    printArray(a);
    System.out.println();
}
```

```
[main] before: 1 2 3 4 5
[func6] before: 1 2 3 4 5
[func6] after: 10 2 3 4 5

[main] after: 10 2 3 4 5
```

```java
{
    int[] x = { 1, 2, 3, 4, 5 };

    System.out.print("[main] before: ");
    printArray(x);
    System.out.println();
    func6(x);
    scanner.nextLine();
    System.out.print("[main] after: ");
    printArray(x);
    System.out.println();
}
```


Designing Subroutines

A subroutine's job should be a single complete relatively self-contained task.

- if you can't state the task briefly and without a lot of use of “and”, it is probably not a single task
- if the task involves changing many variables used elsewhere, it is probably not a complete or self-contained task
  - return values provide only a limited way for a subroutine to affect its caller
  - the number of parameters is not limited, but there is such a thing as too many

Think of a subroutine as a friend who can do a task for you. If explaining what the task is takes too long, it is too big of a job for one friend. If you have to be too involved in what the friend is doing, you might as well do the task yourself.

Designing Subroutines

In general, a static subroutine should do one (and only one) of the following –

- obtain input
  - print a prompt as needed
  - return the input obtained
- produce output
  - print rather than return

input and output limits a subroutine to particular applications – values can only come from a particular source, specific formatting

subroutines simplify organization

- compute something
  - return the result rather than print
  - all necessary values should come via parameters, not input
- change the state of its parameters
  - all necessary values should come via parameters, not input
  - no return value or printing

parameters and return values allow the caller to determine where values come from and how they are used

subroutines are reusable

```java
/**
 * Get the number of players.
 *
 * @return the number of players (>= 1)
 */
public static int getNumPlayers() {
    Scanner input = new Scanner(System.in);
    for (; true;) {
        System.out.print("how many players?  ");
        int numplayers = input.nextInt();
        if (numplayers >= 1) {
            return numplayers;
        }
        System.out.println("number of players must be at least 1");
    }
}
```

- ★ input
- output
- compute
- change params

```java
/**
 * Create a bingo card with n numbers.
 *
 * @param n number of numbers on the card
 * @return a bingo card containing n random numbers
 */
public static int[] createBingoCard(int n) {
    int[] card = new int[n];
    for (int i = 0; i < card.length; i++) {
        card[i] = (int) (Math.random() * 11) + 2;
    }
    return card;
}
```

- input
- output
- ★ compute
- change params


```java
/**
 * Print a bingo card. Only the uncrossed off numbers are printed.
 *
 * @param card the bingo card to print
 */
public static void printBingoCard(int[] card) {
    for (int i = 0; i < card.length; i++) {
        if (card[i] != -1) {
            System.out.printf(" %2d", card[i]);
        }
    }
    System.out.println();
}
```

- input
- ★ output
- compute
- change params

```java
/**
 * Print all of the bingo cards.
 *
 * @param cards cards to print
 */
public static void printBingoCards(int[][] cards) {
    for (int player = 0; player < cards.length; player++) {
        System.out.print("player " + player + ":");
        printBingoCard(cards[player]);
    }
}
```

- input
- ★ output
- compute
- change params


```java
/**
 * Cross off one occurrence of number, if any, in the card.
 *
 * @param card   bingo card
 * @param number the number to cross off
 */
public static void crossOff(int[] card, int number) {
    for (int i = 0; i < card.length; i++) {
        if (card[i] == number) {
            card[i] = -1;
            break;
        }
    }
}
```

- input
- output
- compute
- ★ change params

```java
/**
 * Roll a pair of 6-sided dice.
 *
 * @return the sum of the values rolled
 */
public static int rollDice() {
    int die1 = (int) (Math.random() * 6) + 1;
    int die2 = (int) (Math.random() * 6) + 1;
    return die1 + die2;
}
```

- input
- output
- ★ compute
- change params


```java
/**
 * How many uncrossed off numbers are still on the card?
 *
 * @param card the bingo card
 * @return the number of not-yet-crossed-off numbers
 */
public static int numRemaining(int[] card) {
    int numleft = 0;
    for (int i = 0; i < card.length; i++) {
        if (card[i] != -1) {
            numleft++;
        }
    }
    return numleft;
}
```

- input
- output
- ★ compute
- change params