

Revision: arrays and multidimensional arrays

- We can have arrays of array references, just like we can have arrays of primitive types.
- This is usually used for creating multidimensional arrays.
- Whether or not an array is multidimensional, we use the new operator or the array initializer to create an array.
- The array is stored in a place separated from other variables. We use a reference variable to store a reference to an array so that we can access it.
- Arrays need not be deallocated: they will be destroyed by the Java bytecode interpreter some time after the last reference to it is destroyed. ("Garbage collection")

0911B L17-1

Passing arrays in methods

- We have already seen that we can pass an array into a method by passing its reference. The method can then modify the array using that reference.
- We can also return an array reference from a method. This is usually the way we can use to give an object back from the method to the caller. Example:

```
public static void main(String[] args) {
    char [][] board = createBoard(8);
    ...
}

public static char[][] createBoard(int size) {
    char [][] b = new char[size][size];
    for (int i = 0; i < size; ++i)
        for (int j = 0; j < size; ++j)
            b[i][j] = '.';
    return b;
}
```

0911B L17-2

Why we use method arguments and return values?

- Beginning programmers usually don't understand why we want to use method arguments and return values, if they can always use class variables.
- The good thing about local variable is that you know exactly who is able to access them (i.e. read its value, write new values to it).
- It is usually possible to understand a method completely by just looking at a single method if it only uses local variables. Everything is isolated to one method only.
- If you use class variables instead, you will have to keep track on what is happening on the class variable throughout the whole program to understand the program: a much more difficult task!

0911B L17-3

Illustration

Calling a() in the program on the left results in infinite loop. Can you see it?

```
static int i;
public static void a() {
    for (i=0; i<10; ++i)
        b();
}
public static void b() {
    for (i=0; i<5; ++i) {
        System.out.print(i);
    }
    System.out.println();
}

public static void a() {
    for (int i=0; i<10; ++i)
        b();
}
public static void b() {
    for (int i=0; i<5; ++i) {
        System.out.print(i);
    }
    System.out.println();
}
```

It is very tiring to trace all the methods. It is much better to use local variable whenever it is possible.

0911B L17-4

Length of array

- If a method receives an array argument, or get an array from a method call, it might not know the array size.
- Given a reference `arr` to an array, we can ask for its size by `arr.length`. This gives you the number of elements in the array.
- For example, the following method reverse an int array:

```
// Reverse an array. Works for array of any size.
public static void reverseArray(int[] arr) {
    int left = 0, right = arr.length-1;
    while (left < right) {
        int temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp;
        ++left;
        --right;
    }
}
```

0911B L17-5

The arguments of main

- Finally it is time for us to understand what is the argument of the main method: it is an array of String's.
- To run a program, you can type something like "java MyProg foo bar ...". Then "foo", "bar", ... appears in the arguments of main.
- Thus you can have an alternative method to get input, rather than reading it from `StdIn()`. Example:

```
// Print out the arguments given to main
public class PrintArgs {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; ++i)
            System.out.println(args[i]);
    }
}
```

0911B L17-6