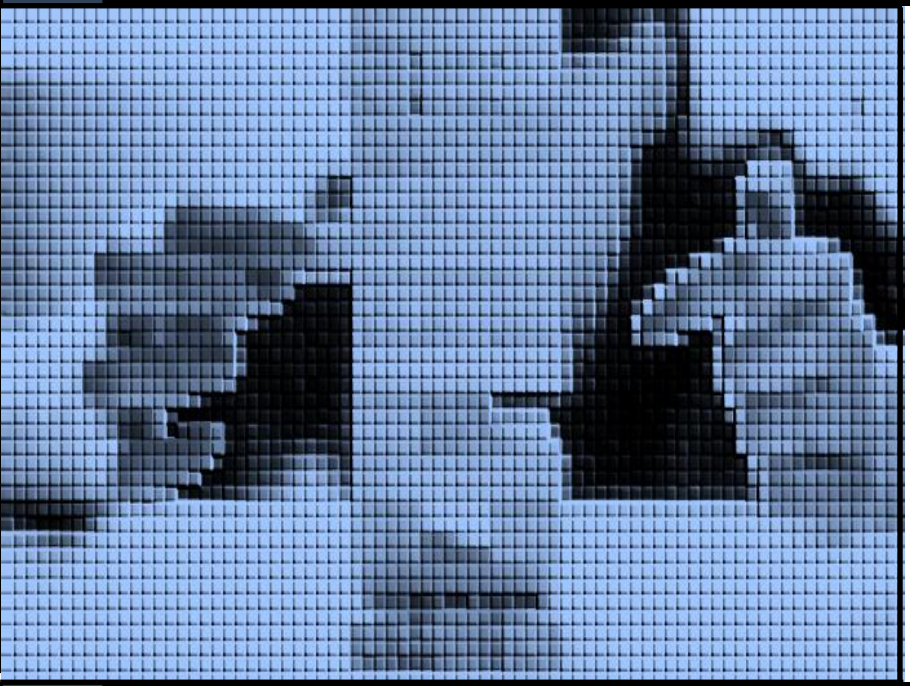
University of BRISTOL

School of Computer Science

Programming in Java



Java Programming – Lecture 05

Arrays

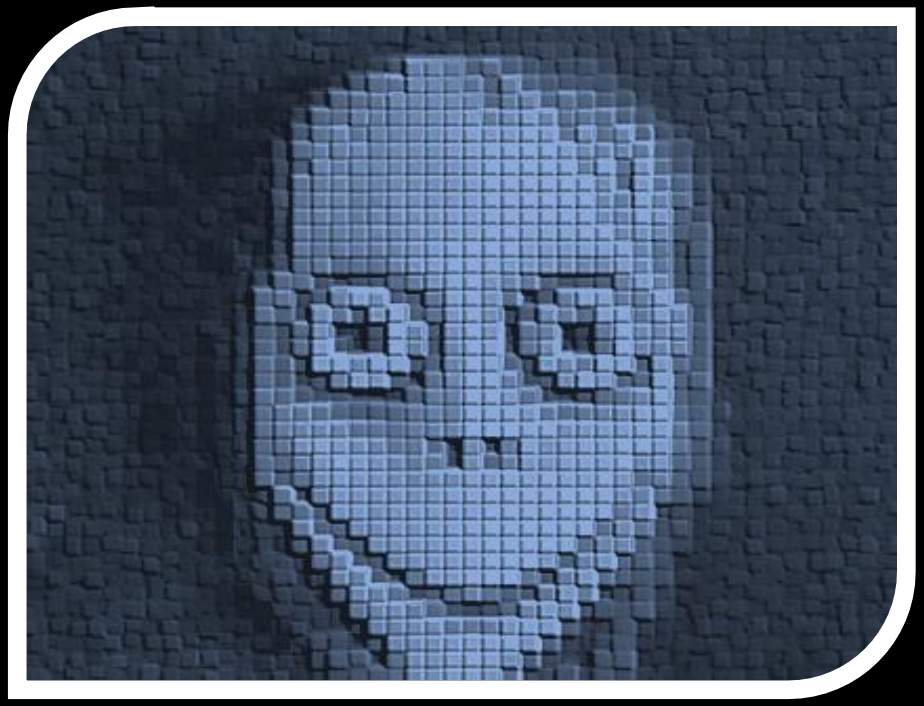
Sion Hannuna and Simon Lock,
based on Tilo Burghardt's C Unit

public domain photo

“ One of my most productive days was throwing away 1,000 lines of code. ”

Ken Thompson (born 1943)
computer science pioneer

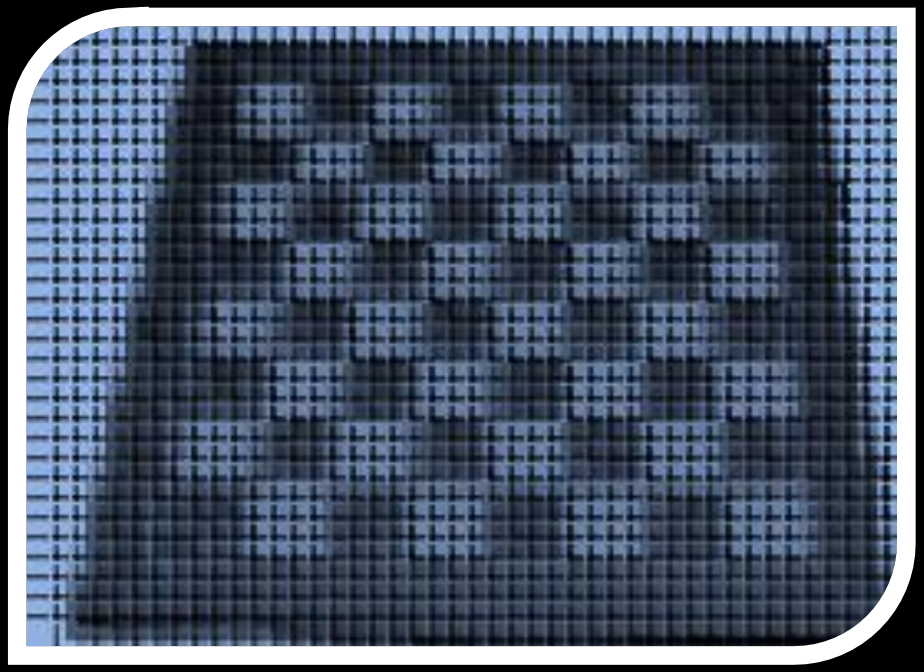
Feature Creep



Feature Creep

- In the previous lecture the advice was to “avoid” quite a number of features, thus adding features to a language is not necessarily a good idea…
- Every single loop type could be rewritten in just `while` loops…
- In fact, one can perform all possible computations with just `while` loops, assignments and expressions…
- However, during the evolution of a language, inevitably features do get added (Java is relatively conservative in this regard).
- This tendency is called *feature creep*, and partly explains why languages go in or out of fashion.

Arrays



Concept of Arrays

- So far, we have declared local variables by giving *each* variable a unique identifier:

```
int a = 10, b = 5, c = 4;
...
```

- A sequence of related variables can, however, be declared as a single *array* giving the number of elements using […]:

```
int[] seq = new int[3] ;  // declare an array of 3 int variables
seq[0] = 10; // initialise first variable
seq[1] = 5;  // initialise second variable
seq[2] = 4;  // initialise third (i.e. last) variable
...
```

- Each individual *array element* is accessible via an index starting from 0.

Initialisation and Size Inference

- *Constant-length* arrays can be initialised compactly:
- For fixed-length initialisation only, the compiler can work out the length of the array using *inference*:

```
int [] seq = {10, 5, 4};  // declare and initialise
```

Bugs

- Indexing from `0` is usually helpful, but it is still easy to make mistakes, especially ones like this:

```
...
  int[] seq = new int[3] ;
  seq[3] = 137; // major bug here!
...
```

- The element `seq[3]` doesn't exist, so java will throw an exception when you run this code

Java Runtime Exceptions for Arrays

- An ArrayIndexOutOfBoundsException occurs when your program tries to access an array element with an illegal index.

```
public class OutOfBounds {
   public static void main(String[] args) {
      int[] seq = {10, 5, 4};
      int outOfBoundsIndex = 1000000000; // index larger than array length
      System.out.println(seq[outOfBoundsIndex]); // access outside array bounds
   }
}
```

- This program will likely throw an **ArrayIndexOutOfBoundsException** during run-time (when the program is executed and stops with an error).
- Compilers will not generally be able to avoid such exceptions, as an array index may be unknown at compile-time (e.g., read in from the user).

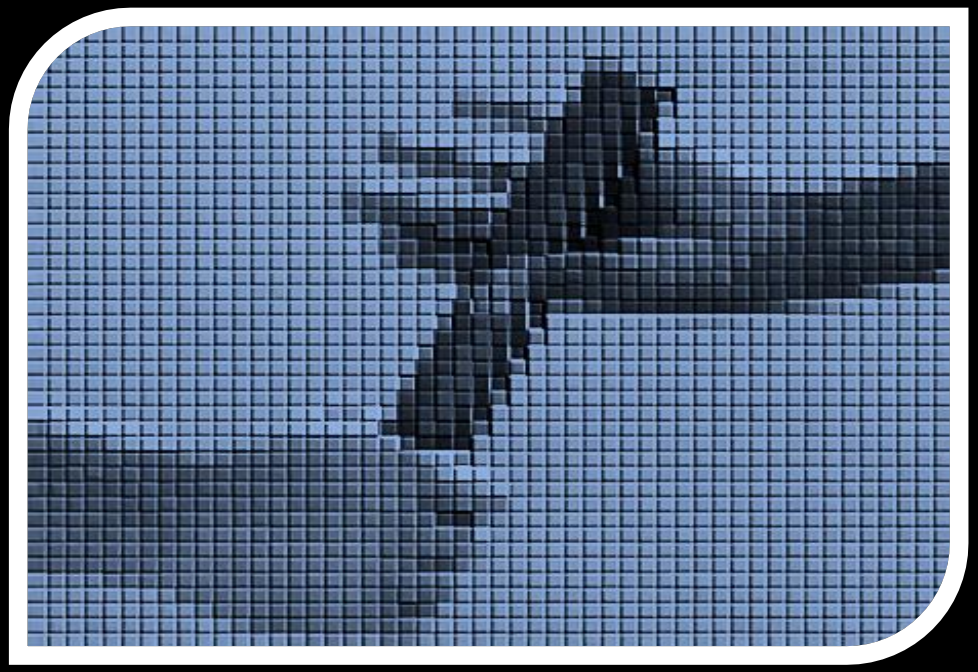
Variable-length Arrays

- The length of an array can be an `int` *variable* (in C11):

```
int calculate(int noElements) {
  int seq[noElements]; // declare array of length noElements
  ...
}
```

- This variable can even be read in from the user or a file.
- However, the length of an array *can't change* after it is declared (it only varies between function-calls or program-runs).

Passing Arrays



The Average Program

```java
import java.util.Scanner;
public class AverageCalculator {

   // Calculate average of numbers
   public static double avg(int n, double[] numbers) {
      double sum = 0;
      for (int i = 0; i < n; i++) {
         sum += numbers[i];
      }
      return sum / n;
   }

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.println("How many numbers to average:");
      int n = scanner.nextInt();
      double[] numbers = new double[n];
      for (int i = 0; i < n; i++) {
         System.out.printf("Enter number no %d:\n", i + 1);
         numbers[i] = scanner.nextDouble();
      }
      System.out.printf("Average is: %f\n", avg(n, numbers));
      scanner.close();
   }
}
```

AverageCalculator.java

- Arrays can be passed to a function as an argument.
- However, we need to hand over the array length separately.
- The argument notation in the example only works if `n` is before `numbers[n]` in the argument list.
- Also common is the use of `numbers[]` as argument, because the compiler doesn't need to know the length of items.

Pass-by-Value

- So far, ordinary (non-array) variables have been passed to functions *by value*.
- That means, the variable (or constant or calculated expression) is *copied* into the argument variable, and this copy is available for the scope of the function only.
- Any changes made to the argument variable have *no effect* on the original (outside the function's scope):

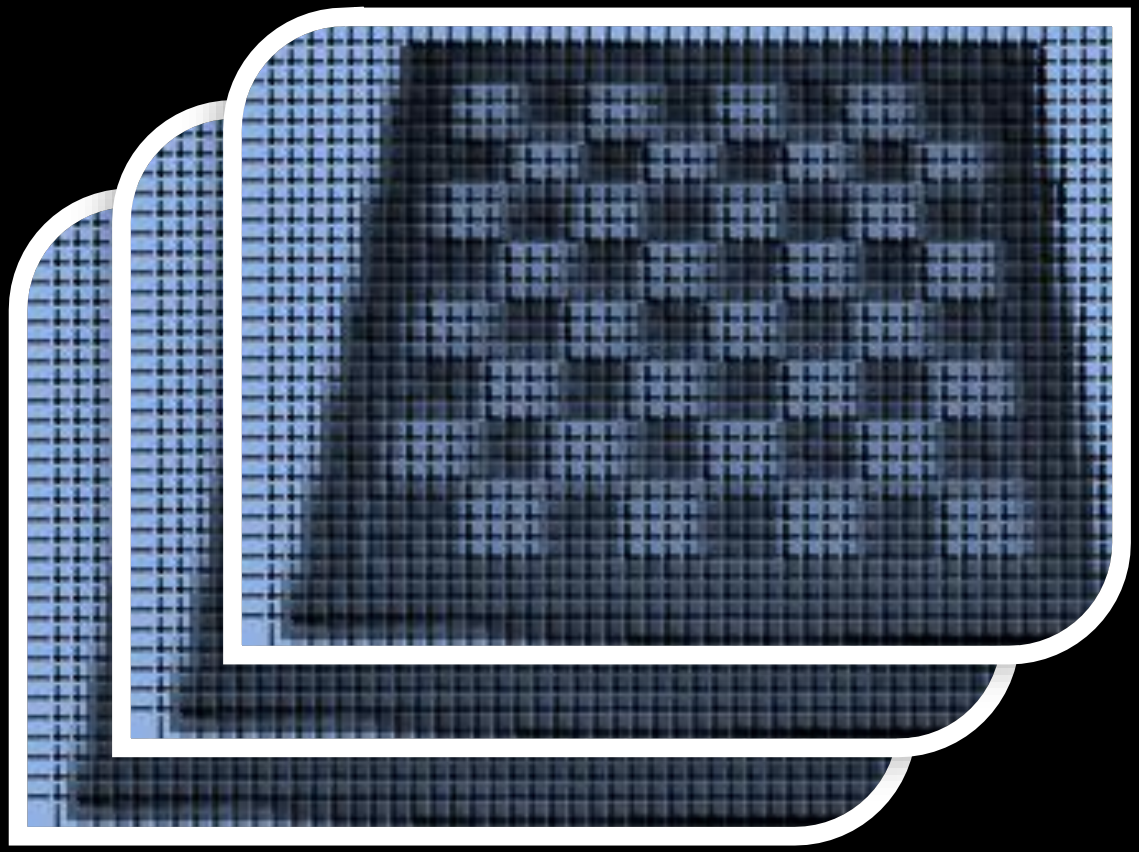
```
public static int square(int x) { x = x * x; return x; }
...
int x = 7, s = square(x);
System.Out.printf("%d %d\n", x, s);//square has no effect
on this x
...
```

Pass-by-Reference

- Array arguments, however, are passed *by reference*.
- That means the argument variable is made to *refer to* the original array, because copying a large array would be expensive.
- That means that if a function changes any of the elements of the array, those *changes are seen* in the caller's original array (outside the function's scope):

```
public static int square(int x[]) { x[0] = x[0] * x[0]; return x[0]; }
...
int [] x = { 7 };
int s = square(x);
System.out.printf("%d %d\n", x[0], s);  // square has changed x[0]
...
```

Arrays of Arrays

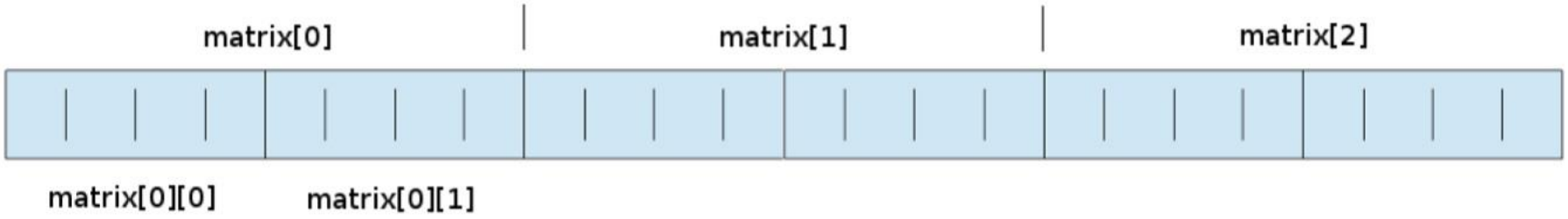


Arrays of Arrays

- A two-dimensional array (or 2D matrix) in C is just an array of arrays:

```
int [][] matrix = {{1, 4}, {5, 3}, {9, 2}};
System.out.printf("bottom right element: %d\n", matrix[2][1]);
...
```

- Arrays are stored sequentially in memory, thus, our 2D matrix is a sequential array of sequential arrays:



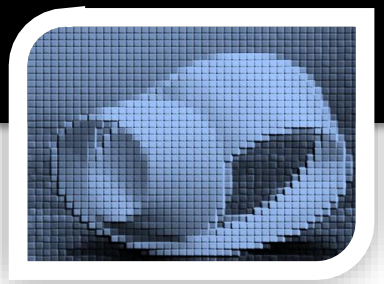
Printing a Matrix

- When passing a multi-dimensional array, the compiler *does* need to know the length of each dimension (except the first), so handing over the dimensions as separate parameters is essential:

```
public static void print(int h, int w, int[][] matrix) {
   for (int r = 0; r < h; r++) {
      for (int c = 0; c < w; c++) {
         if (c > 0) System.out.print(" ");
         System.out.print(matrix[r][c]);
      }
      System.out.println();
   }
}
```

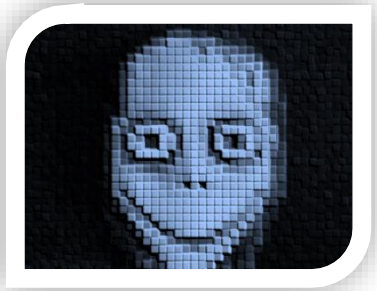
MatrixPrinter.java

Lecture 06 Summary



IN THIS LECTURE WE COVERED:

- *feature creep, arrays, runtime exceptions, passing arrays, pass-by-value, pass-by-reference, arrays of arrays*



AFTER THIS LECTURE:

- In your own time: recap the concepts of this lecture and compile, run and comprehend all its programs.