## More Java…
Arrays, methods and parameter passing

**IMPORTANT:**
Read Chap 7-11 of How to think like a…

Lecture 4

### Arrays

- Arrays are used to store and manipulate several variables of the same type
- **Array X:**
  - To access the i-th element:
    ```java
i = 5;
X[i] = 2;
int a = X[i] + 1;
```
  - Note: first element is at index 0.
  - `X.length` is the number of elements in X (here, it’s 13)
  - Java makes sure you don’t write outside arrays:
    - `ArrayIndexOutOfBoundsException` gets thrown if you try

### Arrays in Java

- **Declaration:**
  ```java
double[] a;   // a will be array of double.
int[] x,y;    // x and y will both be arrays of ints
```
- **Allocation**
  ```java
  a = new double[10];
x = new int[5];
```

### Methods

- **Method (a.k.a. function, procedure, or routine):**
  - Piece of code that carries a specific computation
  - Can be called (executed) from anywhere in the code (if they are public)
  - Can take one or more parameters (arguments) as input
  - Can return a value (or an array, or any object)

  ```java
  public static float square( float x ) {
    float s = x*x;
    return s;
  }
  ```
- **Local variables:**
  - Variables declared inside a method (e.g. `s`).
  - They are discarded after the method finishes being executed.

### Multi-dimensional arrays

- Arrays can have more than one dimension:
  ```java
double matrix [][] = new double[10][10];
// initialize the matrix to zero
for ( int i=0; i<10; i++) {
  for ( int j=0; j<10; j++) {
    matrix[ i ][ j ] = 0;
  }
}
// make it an identity matrix
for ( int i=0; i<10; i++) matrix[ i ][ i ]=1;
```
public class courseID {
    // prints welcoming statement. Takes no arguments. Returns nothing
    public static void printWelcome() {
        System.out.println("Welcome to COMP 250");
    }
    // prints welcoming statement for the given courseID. Returns nothing
    public static void printWelcome(int courseID) {
        System.out.println("Welcome to COMP " + courseID);
    }
    // returns the letter grade for the given percent grade
    public static char getGradeFromPercent(double percent) {
        char grade;
        if (percent >= 0.8) grade = 'A';
        if (percent >= 0.7 && percent < 0.8) grade = 'B';
        if (percent < 0.7) grade = 'C';
        return grade;
    }
    public static void main(String args[]) {
        printWelcome();
        printWelcome(203);
        System.out.println("The grade is " + g);
    }
}

public class fakeAssign {
    public static int main(int a) { // compilation error:
        a = 2; // only on copies of the original a and b
        System.out.println("After fakeAssign a: " + a + " b: " + b);
    }
    // returns the letter grade for the given percent grade
    public static double circleArea(double r) {
        // just to see what happens return r;
        r = -1;
        System.out.println("After stupid a: " + a + " b: " + b);
        System.out.println("Again after stupid a: " + a + " b: " + b);
    }
    // prints welcoming statement. Takes no arguments. Returns nothing
    public static void printWelcome() {
        System.out.println("Welcome to COMP 250");
    }
    // prints welcoming statement for the given courseID. Returns nothing
    public static void printWelcome(int courseID) {
        System.out.println("Welcome to COMP " + courseID);
    }
    // returns the letter grade for the given percent grade
    public static char getGradeFromPercent(double percent) {
        char grade;
        if (percent >= 0.8) grade = 'A';
        if (percent >= 0.7 && percent < 0.8) grade = 'B';
        if (percent < 0.7) grade = 'C';
        return grade;
    }
    public static void main(String args[]) {
        printWelcome();
        printWelcome(203);
        System.out.println("The grade is " + g);
    }
}

Why are methods useful?
• Code re-use: a method can be called (executed) as often as we want, from anywhere in the program. No need to duplicate code.
• Encapsulation: Allows to think of a piece of code as a black box with a well-defined function. Users don’t need to know how the method works, only what the method does: what are its arguments, what does it return.
• Makes program much easier to design, understand and debug

The truth about parameter passing
• What happens when a method is called?
  1. The flow of execution of the code calling the method is interrupted.
  2. If the methods takes some arguments, these arguments are allocated in memory (stack). They are initialized with the value of the arguments provided by the caller.
  3. If variables are declared within the method, they are also put on the stack.
  4. The code of the method is executed. This may include calling other methods.
  5. When the code of the method has been executed, it may return a value to the caller. All local variables and arguments created on the stack are discarded.
• Summary: Parameters are passed by value
  – The method called receives a copy of the parameters passed
  – Since it is working a copy, the method can’t “change the original”
  – But watch out with arrays and non-primitive types...

```java
public static void main(String args[]) {
    System.out.println("Radius: " + area);
    System.out.println("Area: " + area);
}
```
Parameter passing with arrays

```java
static void changeArray(int a[]) {
    System.out.println("First, a[0] is " + a[0]);
    a[0]=2;
    System.out.println("Then, a[0] is " + a[0]);
    a = new int[2];
    a[0]=3;
    System.out.println("Then, a[0] is " + a[0]);
}
```

```java
public static void main(String args[]) {
    int[] array;
    array = new int[3];
    array[0] = 1;
    changeArray(array);
    System.out.println("Finally, array[0] is " + array[0] );
}
```

Strings

- Strings store sequences of characters
- Strings behave just like arrays (but they’re more than that)

```java
String s;  // s is a reference to a String. Currently, it’s a null String
String s = "Hello";
char c = s.charAt(1);  // c is ‘e’
int l = s.length();  // l is 5
String t = s.substring(1, 3);  // t is a new string with "el"
```

- Complete description of String operations:
  - http://docs.oracle.com/javase/1.5.0/docs/api/

Input/Output

- Java has a large number of ways to read in and write out data. We will use only the most basic.
- To import IO libraries, start your code with:
  ```java
  import java.io.*;  // this should be the first line of your code
  ```
- To read data from keyboard:
  ```java
  BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));
  System.out.println("Enter your name:");
  String name = keyboard.readLine();    // reads one line from the keyboard
  System.out.println("Enter your age:");
  String ageString = keyboard.readLine();
  int age = Integer.parseInt(ageString); // convert the string into an integer
  keyboard.close();                                  // close the stream when we are done
  ```
- To write data from file named "myOutput.txt":
  ```java
  BufferedWriter myFile = new BufferedWriter(new FileWriter("myOutput.txt"));
  String line="Hello my friends!";
  myFile.writeLine(line);
  ...
  myFile.close();
  ```
- Good tutorial on IO:
- Full documentation:
  - http://docs.oracle.com/javase/1.5.0/docs/api/

Input

- To read data from file named "myFile.txt":
  ```java
  BufferedReader myFile = new BufferedReader(new FileReader("myFile.txt"));
  String line = myFile.readLine();
  ```

Output

- To write data from file named "myOutput.txt":
  ```java
  BufferedWriter myFile = new BufferedWriter(new FileWriter("myOutput.txt"));
  String line="Hello my friends!";
  myFile.writeLine(line);
  ...
  myFile.close();
  ```
- Good tutorial on IO:
- Full documentation:
  - http://docs.oracle.com/javase/1.5.0/docs/api/