# Week 2 – Java Week 2 (2)

Methods and Classes (and other things)

Dr Chris Roadknight

# Break first

```
class BreakDemo2 {
    public static void main(String[] args) {
        boolean found = false;
            first:
            for( int i = 0; i < 3; i++) {
                  second:
                  for(int j = 0; j < 5; j ++ )
                        {
                              System.out.println("second");
                              break first;
                        }
                  }
            third:
            for( int a = 0; a < 7; a++) {
                  System.out.println("third");
            }
      }
}
```

```
C:\Users\z2018044\Documents\JavaLab2>java BreakDemo2
second
third
third
third
third
third
third
third
```

# Break second

```
class BreakDemo2 {
    public static void main(String[] args) {
        boolean found = false;
            first:
            for( int i = 0; i < 3; i++) {
                  second:
                  for(int j = 0; j < 5; j ++ )
                        {
                              System.out.println("second");
                              break second;
                        }
                  }
            third:
            for( int a = 0; a < 7; a++) {
                  System.out.println("third");
            }
      }
}
```

```
C:\Users\z2018044\Documents\JavaLab2>java BreakDemo2
second
second
second
third
third
third
third
third
third
third
third
```

# Break Third

```
C:\Users\z2018044\Documents\JavaLab2>javac BreakDemo2.java
BreakDemo2.java:10: error: undefined label: third
                                        break third;
                                        ^
1 error
```

```
class BreakDemo2 {
    public static void main(String[] args) {
        boolean found = false;
            first:
            for( int i = 0; i < 3; i++) {
                second:
                for(int j = 0; j < 5; j ++ )
                    {
                        System.out.println("second");
                        break third;
                    }
                }
            third:
            for( int a = 0; a < 7; a++) {
                System.out.println("third");
            }
        }
}
```

# Length of an array

- An array is an object
- It has a length instance variable

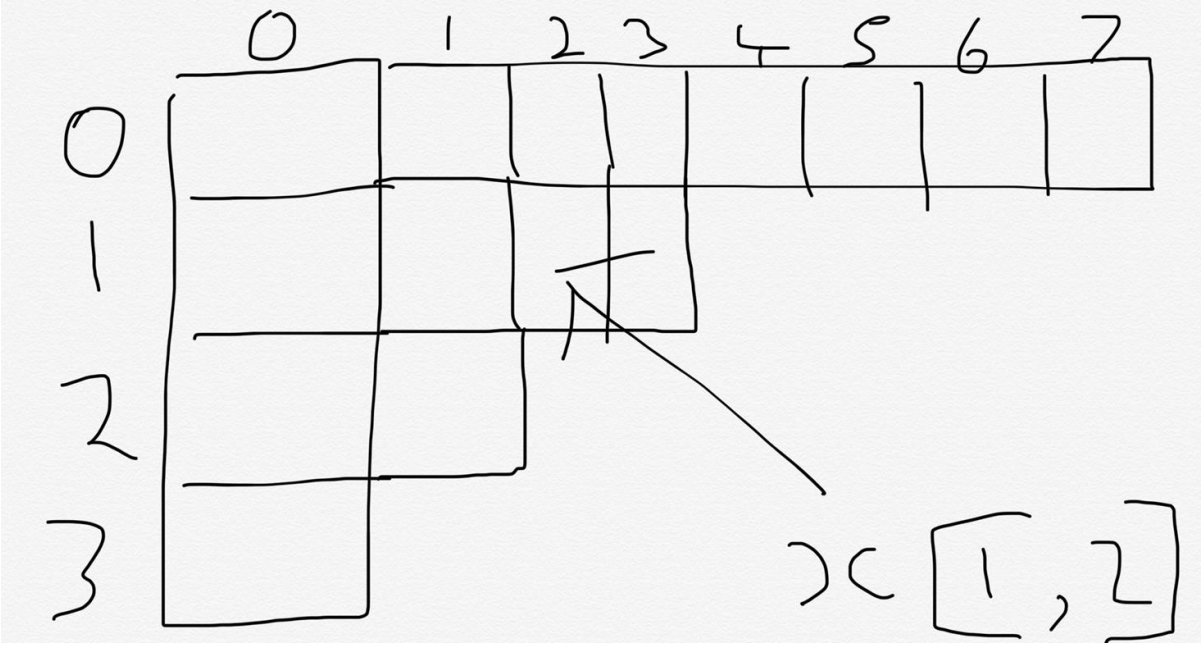
```
int[] testArray;
testArray = new int[5];
testArray[0] = 9;
testArray[1] = 1;
testArray[2] = 1;
X = testArray.length; // 5
```

- 2d array is an array of arrays (so 2 lengths)

# Irregular Arrays

- Only need to declare leftmost dimension

```
Int[][] x = new int[4][];
x[0] = new int[8];
x[1] = new int[4];
x[2] = new int[2];
x[3] = new int[1];
```



# Strings

- Strings are objects
  - (hence the S)
  - And therefore have methods like

```
length()
toUpperCase()
toLowerCase()
```

- Strings are immutable
  - Need to create a new one every time.
  - ‘unused’ one garbage collected
- Can use subString  Method

```
String s="Haskell is hello";
System.out.println(s.substring(0,14));
```

# Escape characters

- You sometimes need the “”

| Escape character | Result | Description |
|---|---|---|
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

```
class Escape {
    public static void main(String[] args) {
        String Film = new String ("My favorite film is "Let the Right One In" by Tomas Alfredson");
    }
}
```

```
class Escape {
    public static void main(String[] args) {
        String Film = new String ("My favorite film is \"Let the Right One In\" by Tomas Alfredson");
    }
}
```

# Access Modifiers

- Determine if other classes can use a field or invoke a method from this class
- Top level
  - public – visible by all classes everywhere
  - no access modifier (package-private) – own package only (more later)
- Member level
  - private  - Can only be access by its own class
  - protected - own package (as with package-private) and by a subclass of its class in another package (eg extends….more later)

# Access Modifiers, example

```
public class A { public int x; private int y;}
private class B { public int x; public int y;}
public class C {
     public static void main(String[] args) {
          A objectA = new A();
          objectA.x = 3; // legal
          objectA.y = 3; // illegal

          B objectB = new B();
          objectB.x = 3; // illegal
          objectB.y = 3; // illegal

     }
}
```

# Access Modifiers

| Modifier | Class | Package | Subclass | World* | |
|---|---|---|---|---|---|
| public | Access | Access | Access | Access | |
| protected | Access | Access | Access | No Access | |
| No modifier (package-private) | Access | Access | No Access | No Access | |
| private | Access | No Access | No Access | No Access | |

More in the labs….

# Passing Arguments to Methods

- Primitives and references both passed by value using ***call-by-value***
- Value of the argument is **copied into** the parameter of the method..
- …no pointers

- If the copied argument is primitive then changes made do not effect the original argument

- If the copied argument is a reference type then changes made do effect the object, because the **reference to** the object is copied

# Reminder

- **Example:**
  - **In the Vehicle class:**

```
double fuelNeeded(int distance) {
    return (double) distance / mpg;
}
```

Parameter

  - **In VehicleDemo's main( ) method:**

```
System.out.println(car.fuelNeeded(750));
    // prints "30.0"
```

Argument

# Passing Primitives

```
C:\Users\z2018044\Documents\JavaLab2>java TestPass
4
4
3
4
4
```

```
1  class Data {
2      void make3(int x) {
3          System.out.println(x); //(2) prints 4
4          x = 3;
5          System.out.println(x); // (3) prints 3
6      }
7  }
8  class TestPass {
9      public static void main(String[] args) {
10         Data d = new Data();
11         int x = 4;
12         System.out.println(x); // (1) prints 4
13         d.make3(x);
14         System.out.println(x); // (4) prints 4
15         System.out.println(x); // (5) prints 4
16     }
17 }
```

# Passing references

```
C:\Users\z2018044\Documents\JavaLab2>java RefTest
steps 1 or 3 =7
2 =7
steps 1 or 3 =10
4 =10
```



```
1  class Data {
2      public int x;
3      void addTo(Data d) { // adds this x to d.x
4          d.x = d.x + this.x; //d.x is 4 .. this object is dl
5          System.out.println("steps 1 or 3 =" +d.x); //(1) prints 7(3) prints 10
6      }
7  }
8  class RefTest {
9      public static void main(String[] args) {
10         Data dl = new Data(), d2 = new Data();
11         dl.x = 3; d2.x = 4;
12         dl.addTo(d2);
13         System.out.println("2 ="+d2.x); // (2) prints 7 d2 has permanently changed
14         dl.addTo(d2);
15         System.out.println("4 =" +d2.x); // (4) prints 10
16     }
17 }
18
```

# Method overloading

- Two methods in a class can share a name, as long as they have different parameter declarations.

```
class Overload {
     void display() {
          System.out.println("<nothing>");
     }
     void display(int x) {
          System.out.println(x);
     }
}
```

# Constructor overloading

```
1  class MyClass{
2      int x;
3      MyClass(){
4          System.out.println("inside MyClass().");
5          x=0;
6      }
7      MyClass(int i){
8          System.out.println("inside MyClass(int).");
9          x=i;
10     }
11     MyClass(double d){
12         System.out.println("inside MyClass(double).");
13         x=(int)d;
14     }
15     MyClass(int i, int j){
16         System.out.println("inside MyClass(int, int).");
17         x=i*j;
18     }
19 }
```

Construct 4 objects in different ways with the same name

# Overloading constructo

```
C:\Users\z2018044\Documents\JavaLab2>java OverloadCons
inside MyClass().
inside MyClass(int).
inside MyClass(double).
inside MyClass(int, int).
t1.x: 0
t2.x: 88
t3.x: 17
t4.x: 8
```

- Different constructors called based on parameters in **new**

```
21  class OverloadCons{
22      public static void main(String[] args) {
23          MyClass t1 = new MyClass();
24          MyClass t2 = new MyClass(88);
25          MyClass t3 = new MyClass(17.23);
26          MyClass t4 = new MyClass(2,4);
27          System.out.println("t1.x: " + t1.x);
28          System.out.println("t2.x: " + t2.x);
29          System.out.println("t3.x: " + t3.x);
30          System.out.println("t4.x: " + t4.x);
31      }
32  }
```

# Recursion

- Generally, functions that call themselves but with a base case that satisfies the condition
- So in java this is Methods that call themselves
- This means a task of doing a process n times is broken down into:
  - Doing a job once
  - Then doing the other n-1 processes
    - N-1 processes can be broken down to:
      - Doing a process once
      - Doing the other n-2 processes
        - N-2 processes can be broken down to: ……..

# Recursion example

```
class StarDrawer{
void drawStars(int n){
    if (n==1) //base case
        System.out.print("*");
    else {
        System.out.print("*");
        drawStars(n-1);
    }
}
}

class StarDrawingDemo{
    public static void main(String[] args) {
        StarDrawer drawer = new StarDrawer();
            drawer.drawStars(8);
            System.out.println();
    }
}
```

```
C:\Users\z2018044\Documents\JavaLab2>java StarDrawingDemo
********
```

Note: Usually, base case is checking for a feature of the data not a simple number (eg. Length of an array)

# Recursion

- Each call of the method has its own copies of the method's variables that are kept on a stack
- Not always better than iterative  (in java… never*…..Haskell ??)
- Each method has an overhead, stack overflows occur quickly
- If too many methods are called this may impact on performance

(depends on resources)

- This kind of investigation is part of other modules…Compilers, Algorithms etc

# Static variable

```
C:\Users\z2018044\Documents\JavaLab2>java StatC
s1 3 4
s2 3 4
s1 5 6
s2 3 6
```

- If you declare a variable **`static`**, it is a variable shared by all instances of the class.

```
1  class StatC {
2      int x = 3; // instance variable, not static
3      static int y = 4; // static variable
4
5      public static void main(String[] args) {
6          StatC s1 = new StatC(), s2 = new StatC();
7          System.out.println("s1 " + s1.x + " " + s1.y);
8          System.out.println("s2 " + s2.x + " " + s2.y);
9          s1.x = 5; // changes s1's x, not s2's x
10         s1.y = 6; // changes y for both s1 and s2
11         System.out.println("s1 " + s1.x + " " + s1.y);
12         System.out.println("s2 " + s2.x + " " + s2.y);
13     }
14 }
15
```

# Static methods

If you declare a method static, it can be called independently of any object through its class name.

```
class StatMethod {
	static void statMeth() {
		System.out.println("Called");
	}
}
class Test {
	public static void main(String[] args) {
	StatMethod.statMeth(); // prints "Called‟ no need for an object!
	}
}
```

# Inner classes

- A class declared within another class is called an inner class.
- Its scope is the **enclosing** class.
- It has access to all the variables and methods of the **enclosing** class.

# Example of Inner class

```
class Outer {
     int x = 5;
     class Inner {
          void changeX() { x = 3; } // change Outer's x
     }
     void adjust() {
          Inner inn = new Inner();
          inn.changeX(); // Outer's x is now 3
     }
}
```