Week 2 – Java Week 2 (1)

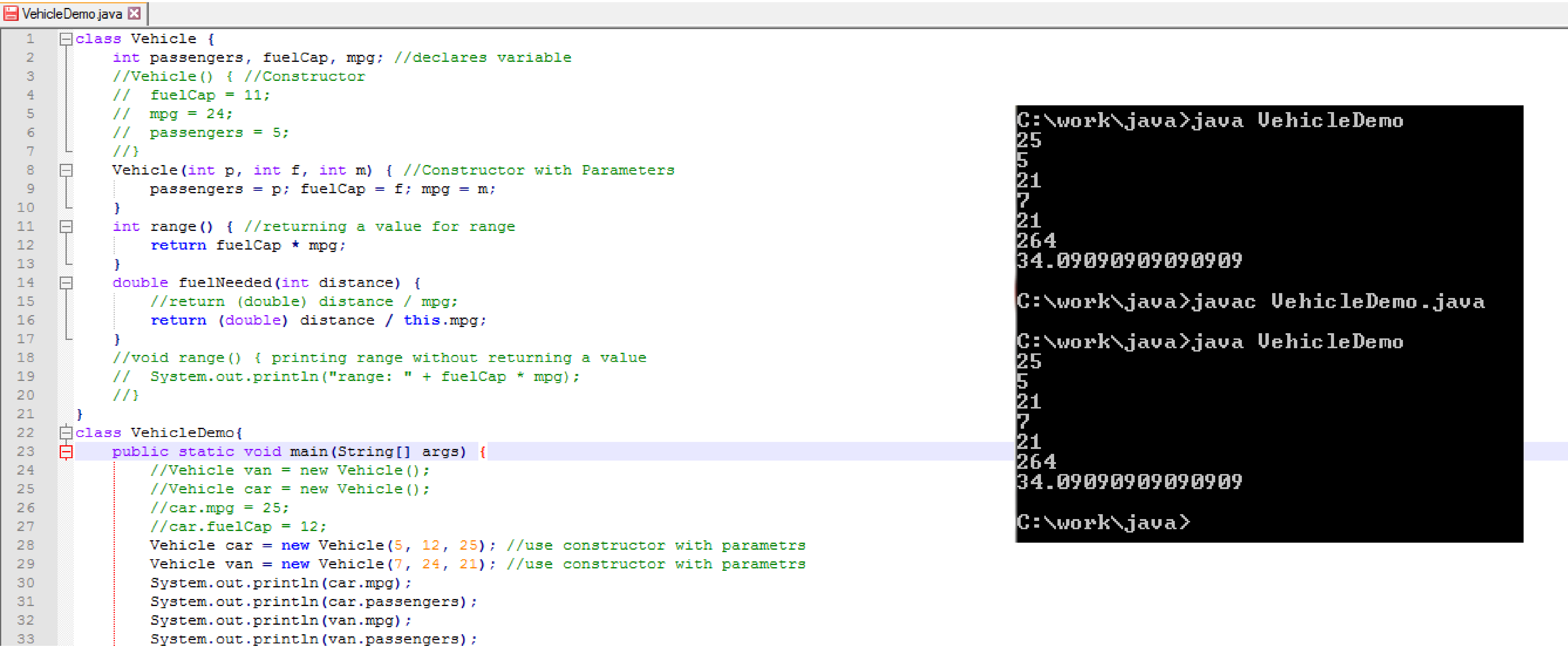
Fundamentals and Introducing Classes, Objects, and Methods

Dr Chris Roadknight

Textbook slides

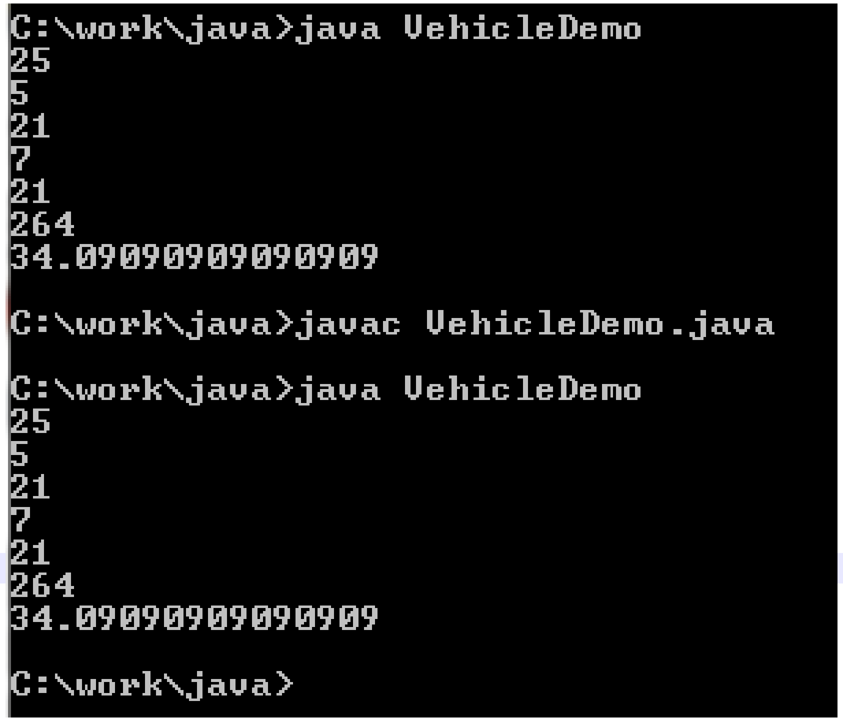
- Chapter 2

Coded Demo



VehicleDemo.java

```java
class Vehicle {
    int passengers, fuelCap, mpg; //declares variable
    //Vehicle() { //Constructor
    //  fuelCap = 11;
    //  mpg = 24;
    //  passengers = 5;
    //}
    Vehicle(int p, int f, int m) { //Constructor with Parameters
        passengers = p; fuelCap = f; mpg = m;
    }
    int range() { //returning a value for range
        return fuelCap * mpg;
    }
    double fuelNeeded(int distance) {
        //return (double) distance / mpg;
        return (double) distance / this.mpg;
    }
    //void range() { printing range without returning a value
    //  System.out.println("range: " + fuelCap * mpg);
    //}
}
class VehicleDemo{
    public static void main(String[] args) {
        //Vehicle van = new Vehicle();
        //Vehicle car = new Vehicle();
        //car.mpg = 25;
        //car.fuelCap = 12;
        Vehicle car = new Vehicle(5, 12, 25); //use constructor with parametrs
        Vehicle van = new Vehicle(7, 24, 21); //use constructor with parametrs
        System.out.println(car.mpg);
        System.out.println(car.passengers);
        System.out.println(van.mpg);
        System.out.println(van.passengers);
```



```
C:\work\java>java VehicleDemo
25
5
21
7
21
264
34.09090909090909

C:\work\java>javac VehicleDemo.java

C:\work\java>java VehicleDemo
25
5
21
7
21
264
34.09090909090909

C:\work\java>
```

Fundamentals

- All objects have a state (Fields) and behavior (Methods)
- You only get a new object in Java when you use ‘new’ to create one (strings are special)
- Fields are usually hidden (encapsulation)
- Methods allow object to object communication
- **Fields** **Instance of Class Vehicle** **Methods**

Fields	Methods
speed	pressBrake
gear	changeGear
colour	respray

Language basics

- Variables
  - Instance variables (non-static fields)
    - Variables within a class but outside any method
    - Initialised when class is instantiated
    - Values unique to each instance of a class but differ between instantiations

speed

  - Class variables (static fields)
    - Variables within a class but outside any method
    - Initialised when class is instantiated
    - Static keyword – Only one copy of this

wheels

  - Local Variables
    - Variables defined inside methods, constructors or blocks
    - Destroyed when we exit the method

i, count

University of Nottingham
UK | CHINA | MALAYSIA

Variable naming convention

- Case sensitive
- Avoid keywords and reserved words
- Must begin with lower case letter
  - (upper case, $ or _ allowed but not conventional)
  - No white space
  - Chinese characters allowed (Unicode)
- Make variables descriptive (therefore self documenting)
- Capitalise first letter of subsequent words
  - `int fuelTankCapacity = 56  //instance variable`
- Constants capitalized
  - `static final int WHEELS = 4  //class variable`

Java Classes

- You can add functions (usually called Methods) to classes in Java
    - These functions usually use the data in an object
- All executable code in Java MUST be in classes
- Function are defined inside classes (i.e. implementation)
- All executable code is inside a class (No global functions in Java)
- All data is also labelled with access permissions
    - So who can access it
- Data and methods are associated either with:
    - A specific object of that class (normal, unlabelled)
    - The class itself – shared between all objects (static)
        - If associated with the class, you don’t need an object

Keywords

```
abstract continue for new switch assert default goto
package synchronized boolean do if private this
break double implements protected throw byte else
import public throws case enum instanceof return
transient catch extends int short try char final
interface static void class finally long strictfp
volatile const float native super while
```

- Top Tip! - Learning what each of these means takes you a long way in learning the language …also useful for exams
- https://www.geeksforgeeks.org/list-of-all-java-keywords/

Keywords are member dependent

- Eg. Static
- **Static variables -** like a global variable for all other data members of the class, accessed before any object of the class exists, accessed with the class name in which it is defined followed by the dot(.)
- **Static Methods -** A static method can only call other static methods only, can only access static data, accessed with the class name followed by the dot(.)
- **Static class -** The outermost class can not be made static whereas the innermost class can be made static. A static nested class can not access a non-static member of the outer class only static ones.

Primitive Data Types

- byte, short, int, long
  - 8,16,32,64 bit signed two's complement number
  - Long requires the suffix L (eg. long a = 999999999999L)
- float, double
  - 32 and 64 bit floating point numbers
  - float requires f suffix (eg. float a = 2.2f)
  - Never use when precision required (use integers or BigDecimal)
- boolean
  - track true/false conditions
- char
  - 16-bit Unicode character

Datatype	Default value(for fields)
byte	0
short	0
Int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false

Note: local variables different! Accessing an uninitialized local variable will result in a compile-time error.

Object Data Types

- Numbers
  - AtomicInteger, AtomicLong, BigDecimal, BigInteger, Byte, Double, DoubleAccumulator, DoubleAdder, Float, Integer, Long, LongAccumulator, LongAdder, Short
- Strings
  - Object that is an array of char

All have different uses (eg. Float provides (null))

```
char c = 'a';
String s = “a”;
char[] cc = {‘y’,’e’,’s’};
String cc = “yes”
```

Arrays

- Object that holds FIXED number of values (of one type)
- Initially, declare the array

```
byte[] firstArrayBytes;
int[] firstArrayInts;
String[] firstArrayStrings;
```

- Allocate memory

```
firstArrayInts = new int[3];
```

- Populate array

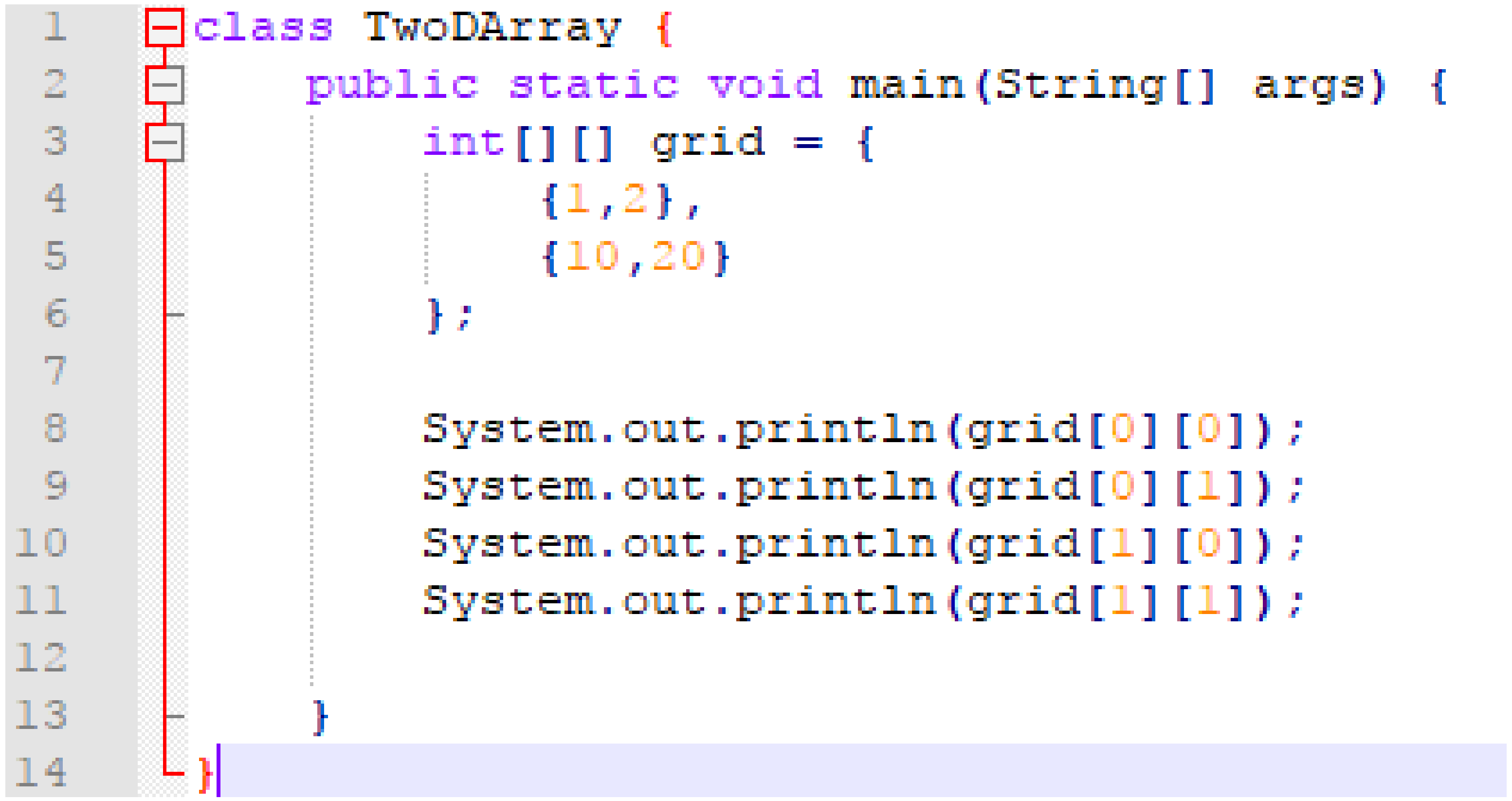
```
firstArrayInts[0] = 9;
firstArrayInts[1] = 1;
firstArrayInts[2] = 1;
```

```
int[] firstArrayInts =
      {9,1,1};
```

int[] intArray = new int[]{
1,2,3,4,5,6,7,8,9,10 };

Multidimensional Arrays

- Use 2 or more sets of brackets



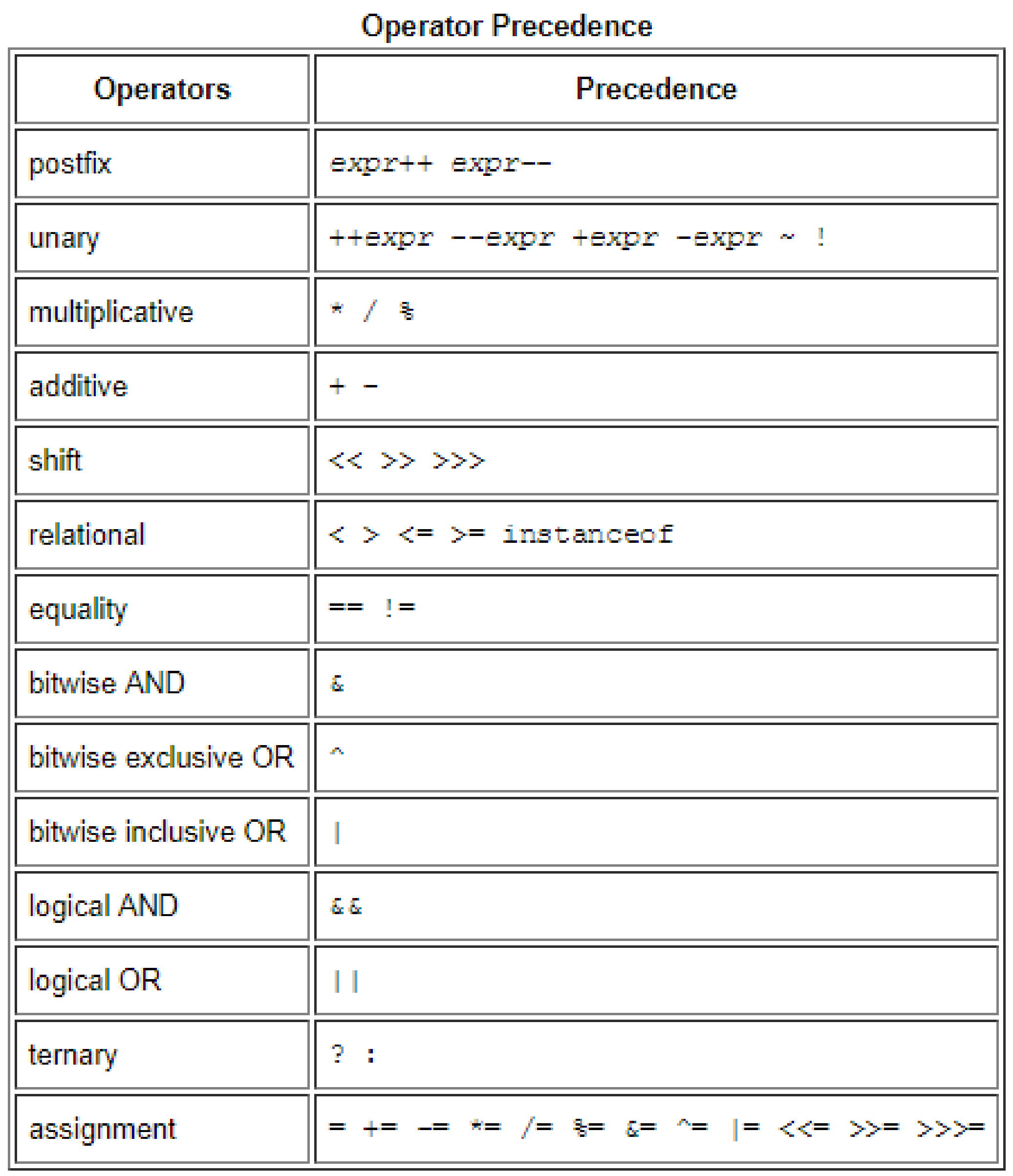
```
class TwoDArray {
    public static void main(String[] args) {
        int[][] grid = {
            {1,2},
            {10,20}
        };

        System.out.println(grid[0][0]);
        System.out.println(grid[0][1]);
        System.out.println(grid[1][0]);
        System.out.println(grid[1][1]);

    }
}
```

Operators

- Mathamatical
- Logical
- Bitwise
- Assignment
- Misc



Operator Precedence

Operators	Precedence
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Decision making

- Decisions
  - if-then, if-then-else, switch
- Looping
  - for, while, do-while
  - labelled for, enhanced for
- Branching
  - break, continue, return

Decisions -Switch

```
public class Season {
    public static void main(String[] args) {
        int season = 2;
        String seasonString;
        switch (season) {
            case 1:  seasonString = "Spring";
                     break;
            case 2:  seasonString = "Summer";
                     break;
            case 3:  seasonString = "Autumn (Winter is Coming)";
                     break;
            case 4:  seasonString = "Winter";
                     break;
            default: seasonString = "not a season";
                     break;
        }
        System.out.println(seasonString);
    }
}
```

Looping

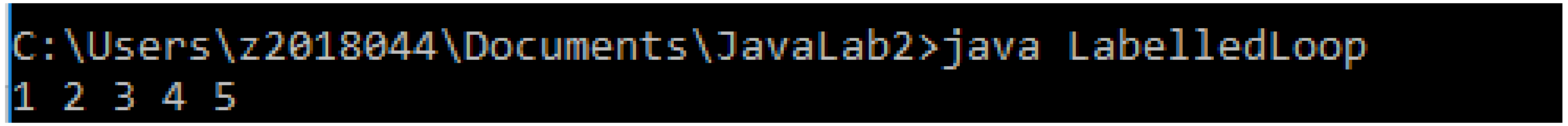
- Condition checking
  - While (may not enter body)
  - Do while (enters body at least once)

- Iterate through finite set of values
  - For

- If you need to calculate something to know if to continue looping, use while.

Labelled for loops

```
class LabelledLoop
    {
        public static void main(String args[])
        {
            int i,j;
            loop1:   for(i=1;i<=10;i++)
            {
                for(j=1;j<=10;j++)   //second loop
                {
                    System.out.print(j + " ");
                    if(j==5)
                        break loop1;      //Statement 1
                }
            }
        }
    }
```

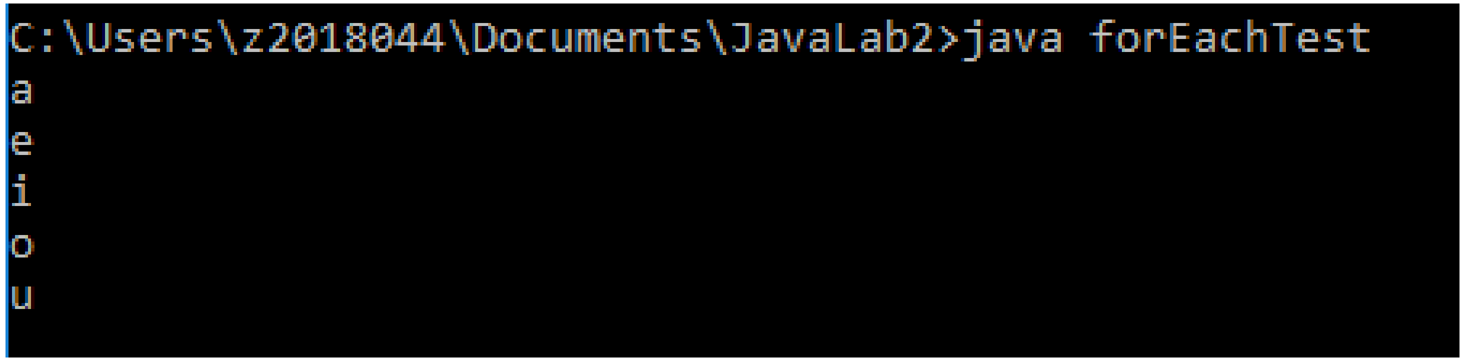


```
C:\Users\z2018044\Documents\JavaLab2>java LabelledLoop
1 2 3 4 5
```

Enhance for loop (For each)

- Just makes coding array access easier

```
class forEachTest {
    public static void main(String[] args) {
        char[] vowels = {'a', 'e', 'i', 'o', 'u'};
        for (char item: vowels) {
            System.out.println(item);
        }
    }
}
```



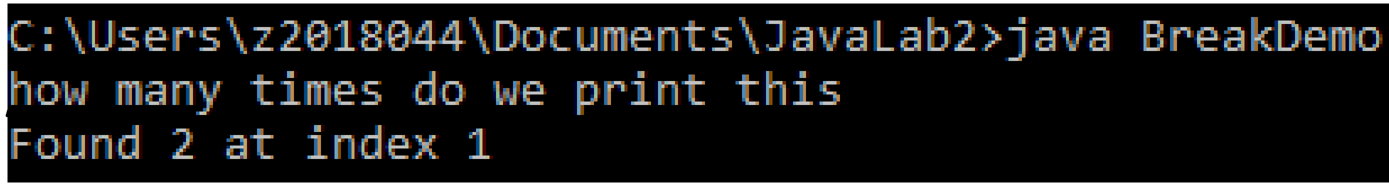
```
C:\Users\z2018044\Documents\JavaLab2>java forEachTest
a
e
i
o
u
```

Break?????????

- Unlabelled break
  - Switch has an unlabeled break (not required but convention)
  - Control flow transfers to the next code block
- Labelled break
  - Add a label
  - When we break, return to that label

Unlabelled Break Example

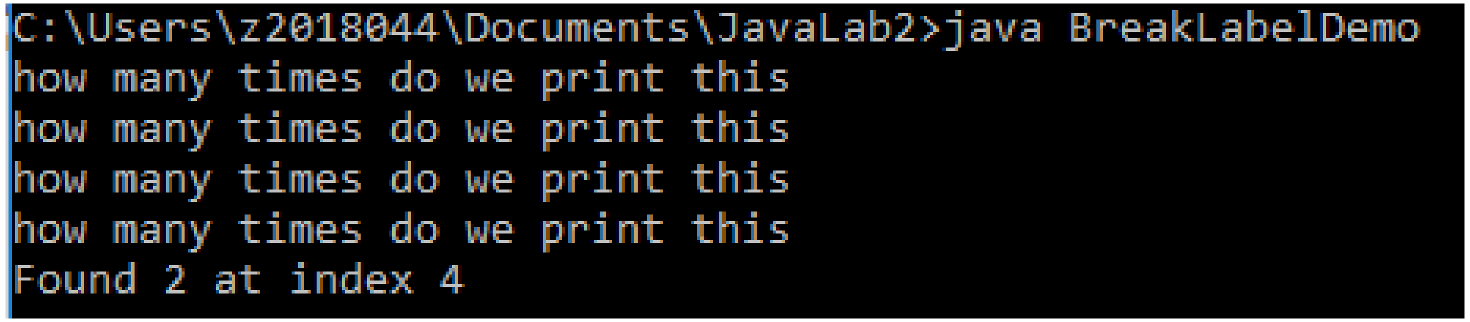
```
class BreakDemo {
    public static void main(String
        int[] arrayOfInts = { 1, 2
        int target = 2;
        int i;
        boolean found = false;
        for (i = 0; i < arrayOfInts.length; i++) {
            if (arrayOfInts[i] == target) {
                found = true;
                break;
            }
            System.out.println("how many times do we print this");
        }
        if (found) {
            System.out.println("Found " + target + " at index " + i);
        } else {
            System.out.println(target + " not in the array");
        }
    }
}
```



```
C:\Users\z2018044\Documents\JavaLab2>java BreakDemo
how many times do we print this
Found 2 at index 1
```

Labelled Break Example

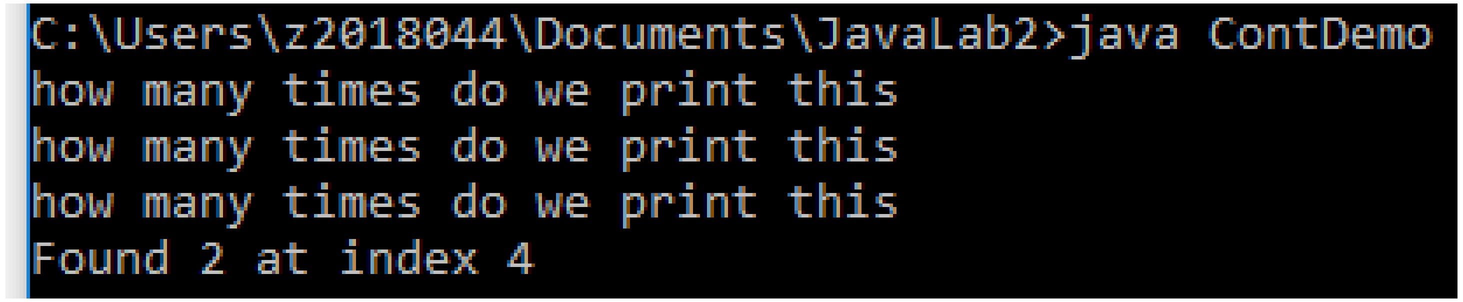
```
class BreakDemo {
    public static void main(String[] args) {
        int[] arrayOfInts = { 1, 2, 3, 4};
        int target = 2;
        int i;
        boolean found = false;
        for (i = 0; i < arrayOfInts.length; i++) {
            breakLabel: if (arrayOfInts[i] == target) {
                found = true;
                break breakLabel;
            }
            System.out.println("how many times do we print this");
        }
        if (found) {
            System.out.println("Found " + target + " at index " + i);
        } else {
            System.out.println(target + " not in the array");
        }
    }
}
```



```
C:\Users\z2018044\Documents\JavaLab2>java BreakLabelDemo
how many times do we print this
how many times do we print this
how many times do we print this
how many times do we print this
Found 2 at index 4
```

Continue
skips the current iteration of a loop

```
class BreakDemo {
    public static void main(String[] args) {
        int[] arrayOfInts = { 1, 2, 3, 4};
        int target = 2;
        int i;
        boolean found = false;
        for (i = 0; i < arrayOfInts.length; i++) {
            if (arrayOfInts[i] == target) {
                found = true;
                continue;
            }
            System.out.println("how many times do we print this");
        }
        if (found) {
            System.out.println("Found " + target + " at index " + i);
        } else {
            System.out.println(target + " not in the array");
        }
    }
}
```



```
C:\Users\z2018044\Documents\JavaLab2>java ContDemo
how many times do we print this
how many times do we print this
how many times do we print this
Found 2 at index 4
```

return

- Exits from current method
- Returns from where method was invoked

- Fundamental to OOP
  - More later