Java Review – Breaking The Surface

- The Way Java Works
- Code Structure in Java
- Anatomy of a class
- Writing a class with main()
- Looping & Conditional Branching

The Way Java Works

1. Source: source document that uses an established protocol (in our case the Java language). Saved as a .java file.

2. Compiler: source document is run through the source code compiler. The compiler checks for errors and won't compile until it's satisfied everything will run correctly.

The Way Java Works

3. Output (code): The compiler creates a new document coded into java bytecode (.class file). Any device capable of running Java will be able to translate the file into something it can run. The compiled bytecode is platform indepenedent.

4. Virtual Machines: virtual machines (JVM) are software installed inside electronic gadgets that read and run the bytecode.

Code Structure in Java

What goes in a source file?

- A source code file (with the .java extension) holds one **class** definition. The class represents a piece of your program, although a very tiny application might need just a single class. The class must go within a pair of curly braces.

Source File

```
public class Dog {




}
```

CLASS

Code Structure in Java

What goes in a class?

- A class has one or more ***methods.*** In the Dog class, the ***bark*** method will hold instructions for how the Dog should bark. Your methods mus be declared *inside* a class (in other words, within the curly braces of the class).

Class File

```
public class Dog {
        void bark ( ) {



        }
}
```

METHOD

Code Structure in Java

What goes in a method?

- Within the curly braces of a method, write your instructions for how that method should be performed. Method *code* is basically a set of statements, and for now you can think of a method kind of like a function or prodcedure.

Method

```
public class Dog {
        void bark ( ) {
                Statment1;
                Statment2;
        }
}
```

STATEMENTS

Anatomy of a Class

- When the JVM starts running, it looks for the class you give it at the command line. Then it starts lookin g for a specifically-written method that looks exactly like this

```
    public static void main (String [ ] args) {
                // your code goes here
}
```

Anatomy of a class

- Next, the JVM runs everything between the curly braces { } of your main method. Every java application has to have at least one **class**, and at least one **main** method (not one main per *class*; just one main per *application.*)

Writing a class with a main

- In Java, everything goes in a **class**. You'll type your source code file (with a .java extension), then compile it into a new class file (with a .class extension). When you run your program, you're really running a *class*.
- Running a program means telling the JVM to “Load the class, then start executing its main() method. Keep running until all the code in main() is finished.”

Writing a class with a main

- The **main( )** method is where your program starts running.
- No matter how big your program is (in other words no matter how many *classes* your program uses), there's got to be a **main( )** method to get the ball rolling.

What can you say in the main method?

- Once you're inside the main (or *any* method), the fun begins. You can say all the normal things that you say in most programming languages to ***make the computer do something***.

What can you say in the main method?

- Your code can tell the JVM to:

1. **do something:**

   Statements: declarations, assignments, method calls, etc.

```
int x = 3;

String name = “Dirk”;

x = x * 17;

System.out.print(“x is “ + x);

double d = Math.random( );
```

What can you say in the main method?

- Your code can tell the JVM to:

2. **do something again and again**

Loops: for and while

```
 while (x > 12) {
          x = x – 1;
 }
 I for (int x = 0; x < 10; x = x + 1) {
          System.out.print(“x is now “ + x);
}
```

What can you say in the main method?

- Your code can tell the JVM to:

3. **do something under this condition**

Branching: if/else tests

```
l if (x == 10) {
        System.out.print(“x must be 10”);
} else {System.out.print(“x isn't 10”);
}
```

Looping and looping and...

- Java has three standard looping constructs: *while, do-while,* and *for*.
- As long as some condition is true, you do everything inside the loop *block*. The loop block is bounded by a pair of curly braces, so whatever you want to repeat needs to be inside that block.

Looping and looping and...

- The key to a loop is the *conditional test*. In java a conditional test is an expression that results in a *boolean* value – in other words, something that ies either ***true*** or ***false***.
- You can do a simple boolean test by checking the value of a variable, using a *comparison operator* including:

< (less than)

> (greater than)

== (equality) (yes that's *two* equals signs)

Example of a while loop

```
public class Loop {
        public static void main (String [ ] args) {
                int x = 1;
                System.out.println(“Before the loop”);
                I while (x < 4) {
                System.out.println(“In the loop”);
                System.out.println(“Value of x is “ + x);
                I x = x + 1;
        }
        System.out.println(“This is after the loop”);
    }
  }
```

Conditional Branching

- In Java an *if* test is basically the same as the boolean test in a *while* loop – except instead of sayng, “*while* the condition is true...”, you'll say, “*if* the condition is true...”

Conditional Branching

```
class ifTest {
        public static void main (String [ ] args) {
                int x = 3;
                if (x == 3) {
                        System.out.println("x must be 3");
                }
                System.out.println("This runs no matter
what");
        }
    }
```

Conditional Branching

- The code from the previous slide executes the line “x must be 3” only if the condition (*x* is equal to 3) is true. Regardless of whether it's true, the line that prints, “This runs no matter what” will run. So depending on the value of *x*, either one statement or two will print out.

Conditional Branching

- We can add an *else* to the condition:

```
class ifTest2 {
    public static void main (String [ ] args) {
        int x = 2;
        if (x == 3) {
            System.out.println(“x must be 3”);
        } else {
            System.out.println(“x is NOT 3”);
        }
        System.out.println(“This runs no matter what”);
    }
}
```

Bonus: Syntax Fun

- Each statement must end in a semicolon.

```
x = x + 1;
```

- A single-line comment beings with two forward slashes.

```
//this is a single-line comment
```

- Most white space doesn't matter.

```
X            =            3   ;
```

Bonus: Syntax Fun

- Variables are declared with a **name** and **type**

```
int weight;
//type: int, name: weight
```

- Classes and methods must be defined within a pair of curly braces.

```
public void go( ) {
    //your code goes here
}
```

Bonus: Quiz

- Statements end in a ______?
- Code blocks are defined by a pair of ______?
- Declare a variable with a _____ and _____?
- The **assignment** operator is ___ equal sign
- The **equals** operator is ___ equals signs
- A _____ loop runs everything within its block (defined by curly braces) as long as the *conditional test* is ______?

Bonus: Quiz

- If the conditional test is _____, the *while* loop code block won't run, and execution will move down to the code immediately after the loop block.
- Put a boolean test inside ______?

while …. { }

Exercises

Code Challenge

- BeerSong: Create an app that perform the classic children's travel song “99 bottles of beer.” We need a class with a *main()*, an *int* and a *String* variable, a *while* loop, and an *if* test.