HelloWorld: Java

```java
public class Review {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

- Every .java file you add to a project must have one and only one outermost class with the same name as the file. In the above example the file name is Review.java so our class name is Review. The file extension (.java) isn't part of the class name.

- This required outermost class must have a method called main that is public (java needs to be able to access main to run your program!) and static (static because there will only ever be one instance of main for your entire program). The void keyword simply indicates that the method does not return a value. In other languages your main method can return any type of value but Java specifically requires that the return type of main is void!

- Main must accept one argument called args that is an array of strings (Strings[]). This is because programs can pass something called command line arguments. These allow you to pass information directly to your program through the command line. In Eclipse click run → run configurations → arguments tab and try entering values into the “Program arguments:” window. Then write code to loop through args and print the values! This will clarify how it works.

Datatypes and Variables in Java

```java
public class Primitives {
    public static void main(String[] args) {
        boolean isTrue = (3 == 3);
        boolean alsoTrue = (3 >= 3);
        boolean isFalse = (3 > 3);

        // Strings are technically not primitives even though we
        // treat them as such
        // if a variable can have methods associated with it like .i*/
        String hello = "hello";
        String goodbye = new String("Goodbye"); // see disclaimer below about new String(String);
```
char c = 'c'; // characters are
char a = 'a'; // a single element of a string
char t = 't'; // denoted by a " instead of "'
char h = hello.charAt(0); // charAt gets the 0th index of hello, 'h'

int num = 30; // a valid integer
float fnum = 3.14f; // we specify 'f' to indicate its a float not a double
double dnum = 3.14; // java uses doubles by default for floating point.

• Variables allow us to store information in memory and then reference it at a later point in our program.

• If we do int x; x = 3; System.out.println(x); We are saying “create a variable that only holds integers; set x to the value 3; print out whatever the current value of x is”.

• If we type int x; in our program this is known as a Variable Declaration. We are declaring that this variable exists. When we do x = 3; this is known as a Variable Assignment. Everything to the left of the equal sign in an assignment is always the variable. Everything to the right of the equal sign is always the value we're placing into our variable.

• You ALWAYS have to DECLARE the variables you wish to use before you attempt to use them. If we did int x; z = 7; We would have a compile time error that we attempted to use a variable we haven't created yet. How will Java know what to put in z if you never told it that z exists and that z holds integers?

• You can also do a combination of a Declaration and Assignment in one line like this: int x = 3; This is perfectly valid.

• You cannot use the same name for multiple variables. If I did int x = 3; float x = 9; I would get an error because I tried to redefine x when I already told Java that I was planning on using x as an integer.

• Finally there are multiple different primitive datatypes in Java. We use primitive to illustrate the fact that they differ from Objects (which are created from classes).

• String is not a primitive in Java but it is in most other languages so it is included here. The reason why String is not a primitive is because String has methods associated with it.
  ◦ Basically you can do “Sam”.toUpperCase(); on a string but you could never do 7.isDigit(); or true.isTrue() because the former is a Object while the later two are primitives.

• Notice how we created a string by using the new String() syntax and without. Strings are the only type of Object in java that you can create without using the new Class() syntax. It is
actually wasteful to use it because it uses unnecessary extra memory.

• ints hold whole values like 1, 2, 3, 4, 5;

• floats hold floating point values like 1.2, 3.14, etc.

• doubles also hold floating point values with twice the precision of a float. Use them when you need accuracy over speed.

• booleans can only be true or false. They come in handy when you use conditional statements and loop statements.

Scope:
// SCOPE: the entire class "Scope"
public class Scope {

    private static String scopeMessage = "Hi! I'm a static variable of the data type string!" +
        "I belong to the class scope. The reason why i am \"Static\"" +
        "Is because there is only one instance of me" +
        "I'm private so only methods inside of the Scope class (this class)" +
        "Can access me!";

    // SCOPE: Main
    public static void main(String[] args) {
        int x = 0;
        String mainMessage = "Hi, I'm a string I exist in this method main";

        if (x == 0) {
            // x is accessible here because it was declared in an outerscope.
            x = 5;
        }

        if (true) {
            int z = 3;
            x = 3;
        }

        /* COMPIILATION ERROR:
        * We defined the variable z in the above block so it
CSE 110 Review 4

```java
    * no longer exists once the block is exited */
    System.out.println(z);
    System.out.println(x);

    for (int i=0; i < 5; ++i) {
        System.out.println(i);
    }

    /* COMPIATION ERROR:
    * We defined the variable i, but it only exists in the for block */
    System.out.println(i);

    /* This is fine though. We can "redeclare" i because it only existed
    * in the above for loop, not in the current scope or any other parent scope */
    int i;
    for (i=0; i < 5; ++i) {
        System.out.println(i);
    }

    /* Now it's time to go to the foobar function() */
    foobar();

    public static void foobar() {

        scopeMessage = "Hi, I'm foobar(), I'm changing the scope message
        because I can." +  
                       "I have access to it because its in the outerscope";

        /* COMPIATION ERROR:
        * We can't access the variable mainMessage because it only exists
        * within the main function. We don't have access to it! */
        mainMessage = "Hello";

    }
```

• Scope refers to the visibility of variables at different points in your program.

• In Java variables are scoped to different blocks in the code. A good general rule is that anything that goes inbetween { and } can be considered a new block of code and thus a
new scope.

- Methods, If, For, While, and Classes all represent different variable scopes.
- Variables defined in outer blocks of code can be accessed by inner blocks but not vice versa.

IO with Scanners
Input.java:

```java
import java.util.Scanner;
public class Input {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int num = s.nextInt();
        double fraction = s.nextDouble();
        String line = s.nextLine();
        String word = s.next();
    }
}
```

- Scanner is a class that allows us to get the users input and use it to influence our programs.
- Always remember to import the required class at the beginning of your program! Java won't know what the scanner class is until you tell it where it is and that you intend to use it by using the import keyword.
- To use a Scanner you have to first create a new scanner by using the keyword new Scanner(System.in); The new keyword is basically asking the system to allocate space in memory for a new instance of a given class. We always have to pass System.in into the Scanner when we first create it otherwise our scanner object wouldn't know where to get the input from! When we do new Scanner(System.in) or new String(“Sam”) or create a new instance of any class we are telling the system to create a new instance of a given class and we're giving the class's constructor a variable (System.in or “Sam”) to help instruct it so that it knows how to construct our given instance! More on this later if you're confused!
- Finally we have to assign our new instance of a scanner that we created with new Scanner(System.in) to a value so that we can use it throughout the rest of the program! Here we take the new instance and assign it to a variable s of the type Scanner.
- Finally to get the user input we need to use our Scanner (called s). s (and all other objects of the type Scanner) has a number of methods associated with it that allow us to do so. These include:
  - s.nextInt() ← Waits for the user to type an integer value and press the enter key. Then it
returns the integer value they typed. It doesn't matter how many spaces the user types before or after the number as long as their input is a valid number.

- `s.nextDouble()` ← waits for the user to type a double value and press enter. Then returns the double they typed. Again spaces do not matter.
- `s.nextLine();` ← Allows the user to type an entire line and returns the entire line as a String;
- `s.next()` ← Regardless of how many characters the user types it only gets the first WORD the user types in and returns this as a String. If I typed “programming is fun” the s.next() would only return “programming” and stop at the first space.

- These are the primary functions you'll need to know but there are many more associated with Scanner

Conditional Statements

**Conditions.java:**

```java
public class Conditions {
    public static void main(String[] args) {

        /*
        boolean algebra guide
        AND:
            true  && true   = true;
            true  && false  = false;
            false && true   = false;
            false && false = false;
        OR:
            true  || true  = true;
            true  || false = true;
            false || true  = true;
            false || false = false;
        */

        int x = 3;

        if (x > 2) {   // evaluates as if (false)
            System.out.println("This isn't true!");
        }
    }
}
```

System.out.println("Java won't enter this block");
System.out.println("These three statements won't be printed.");
}

else if (x == 3 && x == 0) {  // evaluates as if (true && false)
    System.out.println("x can't be 3 AND 0 at the same time");
    System.out.println("Java won't enter this block");
    System.out.println("These three statements won't be printed.");
}

else if (x == 3 || x == 0) {  // evaluates as if (true || false)
    System.out.println("Yay! The above statement evaluated as True. Java is going to print me!");
}

boolean state = true;

if (state == false) {
    System.out.println("This won't be printed");
}

else if (!state) {
    System.out.println("The ! is the logical negation operator");
    System.out.println("It takes the truthiness of the variable or expression and inverts it");
    System.out.println("!true -> false, !false -> true");
    System.out.println("state is true, so negating it will make it false and this block won't print at all!");
}

else {
    System.out.println("Else by itself is the default fallthrough case.");
    System.out.println("If the initial if condition is false and all of the (optional)\nelse if statements are also false then this will execute");
    System.out.println("in this example this block will print!");
}

int z = 8;

/* Prints “ONE” */
switch (z) {
    case 0: System.out.println("None"); break;
    case 1: System.out.println("Of"); break;
    case 2: System.out.println("These"); break;
If statements allow us to branch our programs flow of execution and perform certain operations if certain variables and expressions evaluate to true or false.

If you only wanted to only let a user who is $\geq 18$ use your program you could get their age using a scanner and check if they're 18 using if (age < 18) {System.out.println(“Come back when you're older!”);} }

If statements ultimately evaluate to true or false.

The interior block of a if statement will only happen IF the condition evaluates as true if (true) {System.out.println(“I get printed

You can check multiple conditions in 1 if statement by using the AND ($&$) and (OR) operators. See the Boolean algebra guide in the example.

Else if precedes the initial if statement and allows you to check if the first if is false another different condition.

Else must be the final statement in the if / else if / else block and is optional.

You cannot have an else without an if. You cannot have a else if without an initial if. You can have as many else if statements as you want. Else if and else statements are optional.

IN a nutshell:

- If (cond) {
- /* optional else if */
- /* optional else */
Looping:

```java
class Looping {
    public static void main(String[] args) {
        /* This loop will print [0,10) i.e. excluding 10 */
        for (int i=0; i < 10; ++i) {
            System.out.println(i);
        }
        System.out.println("----------");
        /* This will print [0, 10] i.e. including 10 */
        for (int i=0; i <= 10; ++i) {
            System.out.println(i);
        }
        System.out.println("----------");
        /* this is the same thing as the first for loop */
        int i=0;
        while (i < 10) {
            System.out.println(i);
            i++;
        }
        /* this is the same thing as the second for loop */
        i=0;
        while (i <= 10) {
            System.out.println(i);
            ++i;
        }
        /* NOTE: Don't write code like this! */
        int c=0;
        while (true) {
            while (true) {
                c += 1;
                if (c == 5)
                    break; // break will always break out of the first for loop it encounters
            // the (inner loop)
            }
            break; // this will break out of the outer loop
        }
    }
}
```
System.out.println(c); // will print 5

int x = 0;
do {
    System.out.println(x); // the do-while loop always executes at least once.
} while (x != 0);

Basic OOP:

public class Cat {

    // these are normally private but the purpose of this demo we'll make them public
    public String name;
    public int age;
    public int weight;

    // the constructor for a class is always public and then
    // the name of the class. Constructors don't return anything
    // but notice that we OMIT the keyword "void"
    public Cat(String name, int age, int weight) {

        // this is just a way to disambiguate between the variables
        // that are passed into the method and the actual instance
        // variables above
        // if we didn't include this how would we know
        // which refers to which?
        // refer to http://stackoverflow.com/questions/3728062/what-is-the-meaning-of-this-in-java
        // for more info

        this.name = name;
        this.age = age;
        this.weight = weight;
    }

    /* Nested class, don't worry about this for now */
    public static class Dog {

        public static String name;
        public static int age;
    }
public static void main(String[] args) {

    /* The Dog class is static, that means that
    * there is only ONE instance of a Dog
    * in the entire program. We can access
    * static methods / attributes by using the ClassName.Attribute
    * that we want. Also note that we CANNOT instantiate a new Dog
    * by using the new Dog() syntax. This is because the class is
    * static */
    Dog.name = "Sam";
    Dog.age = 17;

    /* The cat class is not static, that means we can
    create new Objects of the type Cat by calling the constructor
    with the attributes we want to place in the cat */

    Cat alice = new Cat("Alice", 19, 30);
    Cat paul = new Cat("Paul", 14, 42);
    Cat kate = new Cat("Kate", 8, 23);

    // each cat now has its own name, age, and weight

    // lets say that all the cats and our single dog in the
    // program had a birthday and we want to increase their age by 1,
    // we can do so
    // like this:
    //
    System.out.printf("The animals ages are: %d, %d, %d, %d\n",
                        Dog.age, alice.age, paul.age, kate.age);

    Dog.age += 1;
    alice.age += 1;
    paul.age += 1;
    kate.age += 1;

    System.out.printf("The animals ages are now: %d, %d, %d, %d\n",
                        Dog.age, alice.age, paul.age, kate.age);

    }
}
Helpful Resources

- http://learnxinyminutes.com/docs/java/
- https://blog.udemy.com/java-concepts-explained/