```java
import java.util.Vector;

public class Intro {
       String note = "Labs may not be connected to what's explained in the lecture, First two
       weeks are about Java, we start by learning how it relates to C++ (makes it easier).";
       String also = "Requirement next week.";
}

public class HelloWorld {
       // Instance variables, can include access levels (package private if not
       // specified).
       String S= "The default access modifier (if not private, protected or public is specifie
       d) is package private.";
       String Q= "which means that it may be accessed from any other class in the same package
        as the declaring class.";
       public float lostNumber;
       // Same applies to instance methods.
       // Any java program requires at least one class with the main method.
       public static void main(String[] args) {
       System.out.println("Hello, World"); //System is a final class, out is an instance of Pr
       intStream type which is a public and static member variable of the System class, PrintI
       n() is is public method present in all instances of PrintStream (e.g. out). //Geeks.
       System.out.println("Goodbye, World");
       //This the main, write code here.
}
}

public class JavaFacts {
       String firstFact = "//Java is an object-
       oriented language, secure and highly portable (runs on a virtual machine)";
       String platformIndependence = " Java is translated from the native machine instruction
       set in runtime, it runs under control of JVM (Java virtual machine) so any operating sy
       stem compatible with JVM can run it";
       String threads = "We'll learn in detail about threads (Good advantage in Java)";
       String Robustness = "Robust and Secure, it's harder to shoot your self in the foot with
        Java than with C++, offers more memory protection";
}

public class CompilationAndExecution {
       String Compilation = " Java code is writtein in an .java file, which is compiled by Jav
       aC into a .class file";
       String Execution = " The.class file contains Java bytecodes which are platform-
       independent machine instructions, the .class file is then run by JVM ";
       String bytecodes = "In runtime bytecodes are interepreted to the native instruction set
        based on the current platform";
       String howJVMWorks = " JVM Class loader loads all required classes, JVM verifier checks
        for illegal byte codes>> JVM memory manager releases memory back to the OS, the proces
       s in which it manages dereferenced objects is called garbage collection";
       String toRunJava = "Need a Java Runtime Environment (JRE), contains JVM, class librarie
       s and other supporting files (needed to run programs) ";
```

```
      String toCodeJava = "Need JDK (Jave Development Kit) >> Tools to develop Java programs
      (to code and compile )";
}

public class DifferencesToCpp { // Check:
      // http://ee402.eeng.dcu.ie/introduction/chapter-5---introduction-to-java/5-6---
      differences-and-similarities-between-c-and-java
      // https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B
      String Pointers = "No pointers in Java (Restrict pointers in later versions), also ever
      ything besides primitive data types is an object in Java. ";
      String Operators = " Operators are not overridable";
      String Garbage = "Implements garbage collection";

}

public class NamingVariables {
      String $_VarNames = "Start with letter or $ or _, other characters also include digits,
       choose meaningful names and use camel case, make sure to sidestep reserved keywords";
      String NamingMethods = "Method Names: Camel Case (Classes even first word is capital)";
      String TemporayVariables = "Temporary variables: c, d, e (characters) and i, j, k, m, n
       (integers)";
      String Consts = "Const. descriptive, capital case.";

}

public class DataTypes {
      byte lost = 4; // Types similar co C++, but C++ doesn't have this.
      String BTW = "Java also has what's known as wrapper classes for primitive data types, i
      .e. object versions of each";
      String wrapperClasses = "Usually, make the first letter of the primitive data type capi
      tal";
      // They help if we'd like to modify some variable inside a method, among other
      // reasons:
      // https://www.geeksforgeeks.org/wrapper-classes-java/
      String Casting;
      // Widening is automatic:
      int smallVal = 5;
      long largeCont = smallVal;
      float generalVal = largeCont;
      // If we have two variables a and b then
      String SE_Fact = "Whenever we are performing any arithmetic operation between two varia
      ble a & b the result is always, max(int, type of a, type of b)"; // SE=Stack
      // Exchange
      // (StackOverflow)
      // Adding two bytes doesn't generally fit in one byte, but even if they do Java
      // doesn't care, give that an int.
      // Can then cast to byte again if seen fit by the programmer:
      byte a = 10;
      byte b = 20;
      byte c = (byte) (a + b); // Downcasting is explicit!
```

```java
      // Source: StackOverflow
}

public class LogicalOperators {
      String SE_Logic = "in C++, the operators &, | and ^(XOR) are purely bitwise operators.
      In Java, they can be bitwise or logical operators, depending on the context.";
      // This means that they can be used in conditions.
      int x = 4;
      // if( x != null && x.equals("*BINGO*") )
      {
      String SE_thenHowAreTheAndsDifferent = "'&' performs both tests, while '&&' only perfor
      ms the 2nd test if the first is also true. This is known as shortcircuiting and may be
      considered as an optimization (&&). This is especially useful in guarding against nulln
      ess(NullPointerException).";
      // We don't want the second check if the first is false (nullness otherwise)
      }
}

public class Stringos {
      // The string class represents all strings in Java.
      // String objects are read-only.
      // So if we try to change the content of a string, it points to a new location
      // in memory with the new string rather than changing its old value.
      void Concatenate() {
      String full = "Nether";
      String part = "Grasp";
      full += part; // "Nether" on its own still exists in the memory, to change the content
      of the
      // string it's better to use a different class (StringBuffer).
      }
      // Some useful methods in String CLass (also check slides):
      // https://image.slidesharecdn.com/is-6615-4415kavitaganesanv2-150205145659-conversion-
      gate01/95/introduction-to-java-strings-by-kavita-ganesan-20-638.jpg?cb=1437513229

      void Compare() {
      String Cat = "Thanos";
      String Kitty = "Sleeps";
      if (Cat.equalsIgnoreCase(Kitty)) // .equals doesn't ignore case, don't use ==, it only
      checks if the references
      // of both the objects are the same.
      {
      // Do something.
      }
}

void produceStringsFromOtherObjects() {
      String needToKnow = "Every class in java is child of Object class either directly or in
      directly. Object class contains toString() method.";
      String Also = "We can use toString() method to get string representation of an object."
      ;
```

```
        String Andthis = "This occurs implicitly when we try to print the Object reference (toS
        tring is implicitly invoked)";
        String Overriding = "It can be overriden (which is preferable) and in this case our ver
        sion would be the one invoked instead";
}

void StringsFromPrimitives() {
        // If it's a primitive type, .valueOf can help
        int S = 2;
        String Sleepy = String.valueOf(S); // S is now converted into string.
        // The
        System.out.println();
        // Is overloaded for boolean, char, int, long, float, double, char, String,
        // Object (through toString) types

}

void PrimitivesFromStrings() {
        // Each primitive type has a wrapper classes (talked about earlier), the wrapper
        // classes provide methods to convert the string to a primitive.
        // e.g.
        String amount = "50";
        int x = Integer.parseInt(amount);
        // same with floats using FLoat.parseFloat(...)
}

void ModifyingStringContent() {
        // Should use StringBuffer instead of String
        String firstFact = "String class is immutable. StringBuffer class is mutable. String is
         slow and consumes more memory when you concate many strings";
        String Because = " because every time it creates a new instance. StringBuffer is fast a
        nd consumes less memory when you cancat strings."; // Tutorials
        // point.
        // Reversing a string:
        }

        public String reverseIt(String S) {
        StringBuffer Stringo = new StringBuffer();
        int n = S.length();
        for (int i = 0; i < n; i++)
        Stringo.append(s.chatAt(n - i - 1));

        return Stringo.toString();
        }

}

// For loops are similiar to C++, perhaps this generally applies to loops and if
// conditions.
```

```java
public class BreakContinue {
        void Loops() {
        byte x = 1;
        outer: while (x == 0) // give label name to a loop
        {
        if (x != 1) {
        break outer; // The labelled break specifies the loop to exit (here continue from x--)
        }
        x++;
        }
        x--;
        }
        // Labeled continue is similar, to restart the loop with the label.
}


public class Array
{
        void createAnArrayOfPrimitives()
        {
        //1. Delcare:
        int Energies[];
        //int [] energies; also works.

        //2. Create object:
        Energies= new int [6]; //Can be done inline with the previous step.

        //3. Optionally Initilize:
        Energies[5]=12;
        //Can also do:
        int Powers={1,2,3,4,5};  //Array constants can be only used in initializers.

}

void makeItMultidimensional()
{
        //Similiar to C++
        int [][] Matrix= new int [10][10];
        Matrix[5][5]=12;
        }

        void makeItResizable()
        {
        //The length of an array is fixed once it is created and elements cannot be added or re
        moved prior to its creation. (Techie Delight)
        //However, the vector class allows resizing with any objects of any type:
        //import java.util.*;
        Vector <Integer> longlist= new Vector(10);
        int x=3;
        longlist.addElement(x);         //push_back
        longlist.insertElementAt(x, 4); //insert at the 4th index
        longlist.removeElementAt(4);    //takes the index
        //can also do: firstElement(), elementAt(index), indexOf(value), size() and more.



        }

    }
```