Java Lecture Outline:

1. Syntax: curly brackets, semicolon at the end of each line, strongly typed
2. Basics:
   1. Primitives: int, boolean, long, double, float, char,
      1. int i = 0;
   2. Special Objects: String, array
      1. String a = “lsjdsk”;
      2. int[] arr = {9, 3, 4}; or int[] arr = new int[3];
   3. Then of course there are normal Objects, the main focus of what we’re going to be talking about.
   4. undeclared variables = null
3. Methods: <visibility> <static> <return type> name (<type> arg1, …) {…}
   1. Visibility: public/private
   2. Static/not static (will explain when we get to classes)
   3. Return type could be any of the types mentioned above or if it doesn’t return anything, void
   4. Each of the arguments also have their types listed
4. Main method: In Java, when the JVM(what runs your code, basically) runs your code, it will run your Main method first. That’s why you should generally have one Main method. (If you have more than one main method, this is also doable but more advanced):

public static void main(String[] args){

System.out.println(“Hello world”);

}

1. Creating an object (every object is a class but not every class is an object):
   1. Constructors: <visibility> className(args){}
      1. Default constructor if you don’t put anything = public Rectangle() {}
      2. You can create many different constructors, with different parameters.

public Rectangle(int width, int height) {…}

public Rectangle(int width) {…}

* 1. Explain this, dot notation, static variables…

Class Rect:

int height;

int width;

public Rectangle(int width, int height){

this.width = width;

this.height = height

}

public int getArea(){

return this.width\*this.height;

}

public static boolean doesIntersect(Rectangle rect1, Rectangle rect2){

return true;

}

public boolean doMagic(){  
 return this.magic();   
}  
  
private boolean magic(){  
 return true;  
}

* 1. Getters-Setters: If you wanted your width and height to be unchangeable, for example, you would make them private, and create get method for them. (mark each method above as a g or s or c)
  2. Static methods and variables: we can also create static methods and variables in our objects. The use for this will be explained in next step.

1. Using an object in a class – Once you define your object, other classes can create this *object (this is called creating an instance of an object), call their public methods and access their public variables.*

*Rectangle r = new Rectangle(10, 5);*

*int area = r.getArea();*

*You can have as many instances as you want for an object, and each of these object will have their own instance variables. Which is why the keyword “this” is used. The exception to this are the static variables. Static variables are shared among all instances of an object. Similarly, the static methods are also shared. So either can be called by doing:*

*Rectangle.MAX\_WIDTH;*

*Rectangle.staticMethod()*

*Instead of calling r.width or r.Method(), as we would on a non-static method.*

1. Generics (need the type of collection specified):

ArrayList<Integer> intarray= new ArrayList<Integer>  
HashMap<String, ArrayList<Integer>>

Because Generic objects need objects to be declared, all of the primitives also have object counterparts: Integer, Boolean…

1. Inheritance:
   1. You get all public methods and variables from your parent.
   2. In order to create the relation you say:

Public class Person{ String name;}

public class Student extends Person {String school;}

public class CSStudent extends Student {private String major = “CS”; String year;}

* 1. super – keyword refers to the super class (a.k.a “this” class’ parent) In the examples above, Person is the super class of Student, which is the super class of CSStudent.

1. Interface-abstract (really important, the difference is asked in a lot of interviews):

Their general purpose is to give a guideline for what to develop in order to have a class of that type.

* 1. Interface, is an outline of what methods you want a class to have. None of the methods can have a body.
  2. Abstract, on the other hand, can have both method stubs and implemented methods.

These are used to write Comparable(interface), EventListener(interface), Collection(interface), Iterable(interface)…

Abstract classes are the ones that usually start with Abstract, i.e. AbstractList.

When you implement an interface or extend an abstract class, you have to implement the method stubs they have.