1. Some languages are clearly compiled(e.g. C), and some clearly interpreted(e.g. Python). Where does Java fit into this model? How does Java use this model to provide some measure of machine independence? List the types of files you see in a project folder and how they relate to this.

* Java is compiled. But rather than being compiled straight to machine code that a CPU can execute, the Java class files (source files of a project) are compiled to an intermediate binary form called Java bytecode with the file extension .class. This bytecode is then interpreted by the JVM or Java Virtual Machine. The JVM interprets the bytecode into native machine code specific to the type of hardware in use. Any machine that has the JVM installed can interpret and execute the Java bytecode no matter what hardware was used when it was compiled into bytecode, therefore giving Java some measure of machine independence.

1. What is the difference between “==” and equals() methods to compare two String objects?

“==” compares references (memory addresses) and checks identity equality. Equals() method compares values (content equality) Example:

String s1 = “text”; String s2 = s1; String s3 = “more text”; String s4 = “text”;

s2 == s1 //true checks identity equality, they both point to the same address

s1.equals(“text”); //true checks content equality

s1 == s4 //true checks identity equality, String Pool at work

1. Why is a String immutable>what does that mean? Give an example of an operation that demonstrates immutability and how you need to alter the code to work around it.

String is immutable in Java, that means an instance of the String cannot be modified. All information in an instance is initialized when the instance is created and the information cannot be modified. There are advantages of immutable classes. String is designed to be immutable.

* Requirement of String Pool. When a String is created, if the string already exists in the pool, the reference of the existing string will be returned instead of creating a new object. If the String were mutable, then changing one reference would lead to the wrong value for other references.
* Caching Hashcode – hash can be calculated once and cached so it does not have to be calculated everytime it is used, if the String could be modified, this hash would not be able to be cached
* Security - if String is used as a parameter for opening a file or a network connection etc, if it were changeable this could pose a security vulnerability.
* Immutable Objects are thread safe. Can be shared freely among multiple threads eliminating the need to do synchronization.
* Source https://www.programcreek.com/2013/04/why-string-is-immutable-in-java/

1. List the 8 fundamental primitive data types in Java. What are the key differences between the numeric data types on the list? String seems like a notable exception here; describe at least two ways we use strings that demonstrates their non-primitive nature.

* char(16bits), byte(8bits), short(16), int(32), long(64), float(32), double(64), Boolean(8)
* byte can only store values from -127-128; short, int and long can store whole number values of increasingly larger size (up to 2^32 for int) and even larger for long. floats and doubles are different because they can store floating point values
* Strings are not primitive types but we can use them like primitives in some ways. For example: we can use the + operator with Strings just like we can with ints and other numeric types and even chars. Also String is like a primitive when you instantiate it, you do not have to use the new keyword. The syntax for instantiating it is similar to a primitive:
* String str = “text”;
* You do not have to use an import statement. String class is in the standard library.

1. **Inheritance**

**Inheritance**

* 1. Is-A relationship vs. Has-A relationship
  2. Syntax (using extends)
  3. What is inherited and what isn't
* All protected and public Methods from the parent class are inherited
* Fields, constructors, and private methods are not inherited
* A class declared final cannot have anything inherited from it
  1. Method overriding
* Method overriding means there is a method in a subclass with the exact same method signature as a method in a super class and the version of this method in the subclass has its own specific implementation or behavior compared to the version in the superclass.
* Method overriding in one way Java achieves Runtime Polymorphism – the type of the object that invokes the method determines which version of an overridden method is executed
  1. UML class diagram
  2. Why use inheritance?
* Code reuse – use the same fields and methods already defined in another class
* Make global changes to derived classes by making one change in a base class
* Common Data and methods can be encapsulated in a parent class and child classes can just have specific attributes of their own
* Polymorphism allows different behavior depending on the type of objects which invoke methods
  1. Abstract classes, abstract methods
* An abstract class is a class which has even just one abstract method which is a method with no implementation. An abstract class cannot be instantiated. All concrete classes that extend an abstract class must override all abstract methods and provide implementations of the abstract methods.
  1. *Comparable* interface

1. **Polymorphism**
   * Understanding static type vs. dynamic type
   * The ability to substitute an object of a sub-class whenever an object of a superclass is called for
   * Dynamic dispatch
   * Casting, ClassCastException, instanceof
2. **ArrayList & other data structures**
   * what it is
   * valid index range for different operations
   * wrapper classes (Integer, Double, Boolean, etc): autoboxing/unboxing
   * if I give you the specification for methods, you should know how to use them
   * know the basic algorithm for visiting every element in the ArrayList
   * be able to solve problems using an ArrayList
3. **Iterator**
   * what it is, what it's for
   * how to get one
   * how to use it -- understand the methods hasNext() and next()
4. **Recursion**
   * Be able to trace through a recursive method