A: Definitions and Short Answers

i. List the eight primitive data types in Java.

ii. List the non-access modifiers we have seen in class. Describe what each are used for.

iii. Briefly describe the entire process of writing a Java program to running in it from the command line. (Be sure to explain what `javac` and `java` are and what they do.)

iv. In a Java program, why does `main` need to be public?

v. In a Java program, the `main` method is `static`. What would be the consequences if Java did not require this? (i.e., how would you run a program?)

vi. Suppose you have a `final` array of numbers as an attribute. Explain why you can or cannot change the numbers in the array.

vii. Give a concise definition of a method signature. What is the method signature of the `main` method of any program in Java?

viii. What is the difference between a primitive data type and a reference data type in Java?

ix. Briefly describe the difference between a `class attribute` and an `instance attribute`.

x. Briefly describe the difference between a `class method` and an `instance method`.

xi. In the statement `System.out.println("hello, world!");`, explain what `System`, `out` and `println` each are.

xii. Briefly describe the difference between the `stack` and the `heap` in our memory model.

xiii. Briefly describe the four sections of the memory model discussed in class: stack, heap, data segment, code segment.

xiv. What is "`this`" in Java?

xv. Briefly describe the two uses of `this` in a constructor.

xvi. Briefly describe what an `activation record` is and what it is used for.

xvii. Java provides eight primitive wrapper classes. Explain what they are.

xviii. Briefly describe the entire process of adding a file to you local Git repository to updating your GitHub repository to include this file.

xix. When using git, explain the difference between the `commit` and the `push` commands.

xx. Strings are immutable in Java. What does this mean?

xxi. Can a Java class have zero constructors? (Explain yes or no.)

xxii. Given `String s = new String("cat");` and `String t = "cat";`, draw the box & arrow diagram for these two variables.

B: Programming

Create a `Pet` class. Your class must use **encapsulation** (information hiding). Your `Pet` class is not a program. The state and behaviour of the class (and objects) is described as follows:

A `Pet` object will have the following attributes: a name (String), birth year (int), type of pet (String such as "cat", "dog", "eel"), and a list of its children (array of Pet objects).

Each attribute should have a getter method. Provide a setter method only for the list of children (that should take an entire array of Pet objects as input). The birth year must be a constant (i.e., once it is set it must never be changed).

Provide a constructor that takes a name, birth year and type of animal as input and initializes the Pet object. The birth year can be given as 1-digit number (2, 4), a 2-digit number (12, 17, 99, etc) or a 4-digit number. You will store each birth year as a 4-digit number though. For example, 2 means 2002, 13 means 2013 and 1978 means 1978. We won't consider the complicating issues of very old pets.

C: Programming

Create a Java program `UsePet`. Your program must have a method that takes a `Pet` object and a `String` as input and returns `true` if the pet has a child with the name as given by the string, and false otherwise.

Your program must create the following `Pet` object: Fluffy, a cat, who was born in 2012. She has two children: Igor, a cat, who was born in 2013 (who has no children) and Wei, an owl (who was adopted), who was born in 2014 (who has no children). Your program should then ask for user input (using a Scanner object) for a name. Using your method (from above), check if Fluffy has a child with the given name and output an appropriate message. If the user input was "Goran", the output should be one of

`yes, Fluffy has a child named Goran.` or
`no, Fluffy does not have a child named Goran.`

Do not hardcode "fluffy" in the output message. It should print whatever name the `Pet` object in your program has. Your program should work correctly no matter what `Pet` object we initially create. Do not hardcode your solution for the Fluffy pet.

Do not create any `UsePet` objects.

Potentially Useful Things

`java.util.Scanner` has a `next()` method that returns the next `String` token (word) from a given scanner object.

D: Memory Diagrams

Consider the following `main` method

```
Pet p = new Pet("Babak", 9, "eel");
String s = "Babak";
int[] n = {1,3,5,7,9,11};
```

Draw what the memory looks like after all three lines of the main method are executed. Use the box & arrow diagrams. be sure to say what is in the stack, the heap and data segment. Ignore the code segment.