Name:_______________________________ **Partner:** _________________________

Python Activity 65: Java – Object Oriented Programming Review

This activity helps us review all the ***Object Oriented Programming*** concepts that we've covered in this class, so far! We can use an exploration of ***Java*** to better understand these concepts as they apply to Python.

Learning Objectives
Students will be able to:
Content:

- Define the OOP concepts of ***abstraction, inheritance, encapsulation,*** and ***polymorphism***
- Connect the OOP concepts to Python code
- Describe the differences in syntax between Python & Java ***classes***

Process:

- Write Java code equivalents of Python code using ***classes, attributes,*** and ***methods***

Prior Knowledge

- Concepts: OOP, Python, Java return, Java methods, primitive types

Concept Model:

CM1. Match the *Object-Oriented Programming principle* on the left, with its corresponding explanation on the right:

Abstraction	The bundling of data, along with the methods that operate on that data, into a single unit.
Inheritance	The ability for one object/class to take on the states, behaviors, and functionality of another (parent) object/class.
Encapsulation	Using a single type entity (method, operator) to represent different types in different scenarios (e.g., operator or method overloading).
Polymorphism	Hide unnecessary details from the programmer/user.

What is an example of data *abstraction* in Python: ________________________________

Why might *abstraction* be useful?

____

What is an example of *inheritance* in Python: ________________________________

Why might *inheritance* be useful?

____

What is an example of *encapsulation* in Python: ________________________________

Why might *encapsulation* be useful?

____

What is an example of *polymorphism* in Python: ____________________________

Why might *polymorphism* be useful?

____

CM2. For the statements about methods & functions below, circle whether they apply to Python, Java, or both:

a. Always defined within a class. Python&Java Methods Python functions

b. Stand-alone logical blocks of code that are defined outside of a class.
Python&Java Methods Python functions

c. Are called using dot notation on a specific instance of the containing class.
Python&Java Methods Python functions

d. Once defined, be called from anywhere in the program (by importing if in a separate module).
Python&Java Methods Python functions

e. Its definition specifies parameters that must be passed explicitly, if they are passed at all.
Python&Java Methods Python functions

f. Can optionally manipulate parameters. Python&Java Methods Python functions

g. May perform an action (e.g., print or modify), and/or return a value (or implicitly return nothing).
Python&Java Methods Python functions

h. Can operate on the attributes/instance variables that are defined within the containing class.
Python&Java Methods Python functions

CM3. Match the Java *scope keyword* on the left, with its corresponding explanation on the right:

`private`	Methods/variables are not accessible from outside of the containing class.
`protected`	Methods/variables can be freely used outside of the class.
`public`	Methods/variables should only be accessed by subclasses.

How do we indicate *private* scope variables/methods in Python? ____________________

Why might we want to scope something as *private*?

____

How do we indicate *protected* scope variables/methods in Python? __________________

Why might we want to scope something as *protected*?

____

How do we indicate *public* scope variables/methods in Python? ____________________

Why might we want to scope something as *public*?

____

Critical Thinking Questions:

CLASSES – METHODS

1. The table below contains an example of a Java class with two methods:

TestClass.java

```
public class TestClass {
    public String sayHi(String name) {
        return "Hello " + name;
    }

    public static void main (String args[]) {
        TestClass test = new TestClass();

        System.out.println(test.sayHi("CS134"));
    }
}
```

a. Write the Python version of the code above:

b. Underline the *method header* in the Java code above, as well as in your Python version. What class does this *method* belong to?

____

What is a *method*?

____

What is the difference between a *function* and a *method* (in Python)?

____

FYI: Java does not have ***classless functions*** like Python does!

c. Circle the *object instance* in the Java code above, as well as in your Python version. What class is this an *instance of*?

____

What is an *object instance*?

____

d. Place a star next to the *method invocation/calling* in the Java code above, as well as your Python version.
How do we know the method is being called?

____

How does invoking/calling a *method* versus a *function* (in Python) differ?

____

e. What are the *parameters* in this Java & Python code?

____

There are two main ways that Java *method parameters* differ from Python *method parameters*. What are they?
______________________________and______________________________

f. How does *Python* know what code to run when we run it as a script?

____

How might *Java* know what code to run when we run it as a script?

____

Classes – Attributes

2. The table below contains a Java & Python implementation of our LinkedList class:

LinkedList.java, linkedlist.py

```java
public class LinkedList {
  private String value;
  private LinkedList rest;

  public LinkedList(String val) {
    this.value = val;
    this.rest = null;
  }

  public LinkedList(String val,
                    LinkedList other) {
    this.value = val;
    this.rest = other;
  }

  public String getValue() {
    return this.value;
  }

  public LinkedList getRest() {
    return this.rest;
  }

  public void setValue(String v) {
    this.value = v;
  }
```

```python
class LinkedList:


    def __init__(self, value=None, rest=None):
        self._value = value
        self._rest = rest







    def get_value(self):
        return self._value

    def get_rest(self):
        return self._rest

    def set_value(self, val):
        self._value = val
```

a. Underline where we declare the *class attributes* in the Java code above, as well as in the Python version.
Are these *attributes* private, protected, or public? How do you know?

c. In our Java code, we have two *constructors*, whereas in Python we can only have one of an equivalent method. What might the comparable method be? *Hint:* Constructors *construct* a new instance.

d. What are the *getter* methods in our Java & Python code?

Why do we call them *getter* or *accessor* methods?

e. What are the *setter* methods in our Java & Python code?

Why do we call them *setter* or *mutator* methods?

CLASSES – STRING REPRESENTATION

3. The table below continues our example from the previous question:

LinkedList.java(continued)

```java
private String toStringHelper(){
  // Comment:
  if (this.getRest() == null) {
    return this.getValue();
  } else { // Comment:
    return this.getValue() + ", " + this.getRest().toStringHelper();
  }
}

public String toString() {
  // Comment:
  return "[" + this.toStringHelper() + "]";
}
```

a. Fill in the in-line comments in the above code, explaining what the line(s) below it does.

b. What do the methods in this example code do?

c. How did we write code to create a LinkedList object in *Python* and then print a string version of the object?

Write a line of code to create an instance of a LinkedList object in *Java* and then call a method above to print the string version of the instance:

How might we call *Python's* version of the above method *implicitly?*

FYI: Java's ***toString()*** method is also called *implicitly* when the object instance is in a `System.out.println(..)` statement.

Why are *methods that convert instances to strings* useful?

____

CLASSES – COMPARING OBJECTS

4. The table below continues our example from the previous question:

LinkedList.java(continued)

```
public boolean equals(LinkedList other) {
  if (this.getRest() == null && other.getRest() == null) {
      return true;
  } else if (this.getRest() != null && other.getRest() != null) {
      boolean val = this.getValue().equals(other.getValue());
      boolean r = this.getRest().equals(other.getRest());
      return val && r;
  } else {
    return false;
  }
}
```

a. Write the Python version of the code above:

```
def __eq__(self, other):
    # If both lists are empty
    if self._rest is None and other.get_rest() is None:
        return self._value == other.get_value()
    elif self._rest is not None and other.get_rest() is not None :
        return self._value == other.get_value() and self._rest ==
other.get_rest()

    # If we reach here, then one of the lists is empty and other is not
    else:
        return False
```

b. What does this method do?

c. Write example *Python* code to use this method:

Write example *Java* code to use this method:

How might we call *Python's* version of the above method *implicitly?*

In Python, when might we use the method we implemented in (a), and when might we use the `is` operator*?* Why?

FYI: In Java the ***.equals(..)*** method is comparable to Python's ***__eq__(..)*** method. And Java's **==** is comparable to Python's ***is*** operator. In Java, we typically use ***.equals(..)*** to compare anything other than *primitive types*.

In Java, when might we use the method in the example code, and when might we use the == operator?

Classes – Other Useful Methods

5. The table below continues our example from the previous question:

LinkedList.java(continued)

```java
public int length() {
  if (this.getRest() == null && this.getValue() == null) {
    return 0;
  } else if (this.getRest() == null) {
    return 1;
  } else {
    return 1 + this.getRest().length();
  }
}

public boolean contains(String search) {
  if (this.getValue().equals(search)){
    return true;
  } else if (this.getRest() == null) {
    return false;
  } else {
    return this.getRest().contains(search);
  }
}
```

a. Write the Python version of the code above:

```python
# len() function calls __len__() method
# slightly updated version accounts for empty list
def __len__(self):
    # base case: i'm an empty list
    if self._rest is None and self._value is None:
        return 0
    # i am the last item
    elif self._rest is None and self._value is not None:
        return 1
    else:
        # same as return 1 + self._rest.__len__()
        return 1 + len(self._rest)

# in operator calls __contains__() method
def __contains__(self, val):
    if self._value == val:
        return True
    elif self._rest is None:
        return False
    else:
        # same as calling self.__contains__(val)
        return val in self._rest
```

b. What do these methods do?

____

c. Write example *Python* code to use this method:

Write example *Java* code to use this method:

How might we call *Python's* version of the above methods *implicitly?*

____

FYI: In Java there isn't an equivalent way to *implicitly* call the ***length(..)*** and ***contains(..)*** methods.

d. What are *special methods* in Python*?*

____

From what we've seen so far, does Java have *special methods*?

____

FYI: Java does not support ***operator overloading*** (i.e., redefining common operations like + or []), but it does support ***method overloading*** (i.e., same method, different parameters).