Practica 1:
Repaso de Objetos y sus ideas fundamentales

Ejercicio 1

Dado el siguiente código en Java:

```java
class Persona {
    private String nombre;
    private String apellido;
    private String edad;

    public String nombreYApellido() {
        return nombre() + " " + this.apellido();
    }

    public void cumplir() {
        this.edad = this.edad + 1;
    }
}

class Main {
    private String nombre;
    private String apellido;
    private String edad;

    public static void main(String[] args) {
        Persona p1 = new Persona();
        Persona p2 = new Persona();
    }
}
```

Se pide que responda:

1. ¿Cuántas clases se definen en el código?
2. ¿Cuántos objetos se definen?
3. ¿Cuáles son los atributos de la clase Persona?
4. ¿Cuáles son las interfaces de la clase Persona?
5. ¿Cuál es la superclase de Persona?

Ejercicio 2

Dado el siguiente código en Python:

```
class Vehiculo:
    def __init__(self):
        self._cantidadDeRuedas = 0
        self.kmsRecorridos = 0

    def recorrer(self, kms):
        self.kmsRecorridos += kms

class Motocicleta:
    def __init__(self):
        super().__init__()
        self._cantidadDeRuedas = 2

    def wheelie(self):
        self._cantidadDeRuedas = 1

    def unwheelie(self):
        self._cantidadDeRuedas = 2

class Auto:
    def __init__(self):
        super().__init__()
        self._cantidadDeRuedas = 4
        self._pasajeros = 1

    def pasajeros(self, cant):
        self._pasajeros = cant

tutu = Auto()
tutu._pasajeros = 5
tutu.recorrer(100)

motito = Motocicleta()
motito.wheelie()
motito.recorrer(10)
motito.unwheelie()
```

Se pide que responda:

6. ¿Cuántas instancias se están creando?
7. ¿Se respeta el encapsulamiento en todos los casos?
8. ¿Cuál es el estado final de “tutu”?
9. ¿Cuál es el estado final de “motito”?

Ejercicio 3

En esta oportunidad vamos a implementar lo siguiente utilizando Java. Cualquier versión de Java es adecuada, pero optamos por mantenernos en el uso de características de Java 6, incluso si usa una JDK más moderna.

a. Vamos a modelar a nuestra queridísima golondrina, Pepita. Sabemos que Pepita, como toda ave, sabe volar una cierta cantidad de kilómetros y comer una cantidad de gramos de comida. Cada vez que come, pepita repone energía (a razón de una unidad por gramo ingerido) y cada vez que vuela, gasta energía (a razón de 3 unidades por cada kilómetro recorrido). Además, cuando un ave nace, es bien sabido que inicia su ciclo de vida con 2 unidades de energía.

   Lo que queremos es un programa donde hagamos comer a pepita 5 gramos de comida, luego la hagamos volar 1 kilómetro, y después la hagamos comer 20 gramos y volar otros 2 kilómetros. Pepita debe saber decirnos cuánta energía tiene al final del día.

b. Nuestro modelo es simple, pero hay restricciones obvias que tal vez no hayamos tenido en cuenta. Sí un ave quiere volar pero no tiene energía, entonces debería ocurrir una excepción.

c. Además de todo lo realizado, vamos a querer preguntar a un ave la distancia recorrida. La distancia recorrida para cualquier ave tiene que ver con cuántos kilómetros ha volado, y se mide en 1 por kilómetro.

d. Como otras golondrinas, Pepita gusta de pescar. Cuando atrapa un pez, se lo come, y aumenta su energía en 10. Sin embargo, solo atrapa un pez una de cada 10 veces que intenta pescar (de forma aleatoria), por lo que no siempre es una tarea fructífera. Cada vez que intenta pescar, consume dos unidades de energía. Pepón, otra golondrina, es el mejor pescador, y solo le cuesta una unidad de energía por cada vez que pesca. Por supuesto, cuando una golondrina no tiene suficiente energía, ocurre una excepción. El pescar, también implica recorrer una distancia, a razón de 1 por cada vez que se pesca.

e. Bombón es una paloma. las palomas, también son aves, y por tanto vuelan y comen. Pero además les gusta defecar en cualquier estatua, busto o monumento que encuentren en la ciudad. Cada vez que va al baño, la paloma decrementa su energía en 1. Sí preguntamos la distancia recorrida luego de ir al baño, esta habrá incrementado en 1, ya que ir al baño conlleva moverse.

f. Las siguientes preguntas son para reflexionar. Intente responderlas pensando en su modelo, y reflexione sí la respuesta lo satisface.

   - ¿Cuáles son las clases que necesito?
   - ¿Tengo una clase por personaje o una clase por tipo de ave?
   - ¿Qué es más conveniente? ¿Por qué?
   - ¿Qué representación se ha utilizado para las entidades? ¿Cuales son los atributos?

- ¿Se le ocurre otra representación que permita realizar las mismas operaciones?
- ¿Quién accede a qué cosa?
- ¿Puede el main acceder a la energía del ave?
- ¿De qué forma?
- ¿Quién tiene la responsabilidad de imprimir el resultado?
- ¿Cuando pongo una restricción?
- ¿Dónde la pongo?
- ¿Y sí tengo más de una cosa que consume energía?

g. Queremos agregar a la aplicación un logger, que vaya dándonos información acerca de cada evento que ocurre en la misma. El logger es el mismo para todo el sistema, y simplemente debería saber responder a “showInfo”, “showWarn” y “showError”, en donde muestra en pantalla un mensaje que comienza con “INFO: “, “WARN: “ o “ERROR: “ según corresponda. Ojo, no hay que usar una biblioteca de logging ni nada parecido, sino que creamos la nuestra.

h. El logger debería poder ser configurado al arranque en uno de tres modos, INFO, WARN o ERROR. Sí está configurado como INFO, todo mensaje debe ser mostrado, sí está configurado como WARN, solo los mensajes de showWarn y showError deberían mostrarse, y sí está configurado como ERROR, solo los mensajes de showError deberían mostrarse.

i. Haga que las aves tengan ahora siempre un nombre, y loggen como info todas las acciones que realizan. También debería loguearse como error en aquellos casos en los que ocurra alguno.

j. Responda ahora las siguientes preguntas:

   - ¿Hay algún patrón de diseño involucrado en la construcción del logger? ¿Cuál/es?
   - ¿Qué beneficios tiene conocer y poder aplicar un patrón de diseño?

k. Además de las aves, tenemos a Twinkle, la mariposa. Las mariposas saben volar y comer, pero su comportamiento es diferente al de las aves. Las mariposas no están preocupadas por la energía, cuando comen, solo engordan, a razón de un gramo por cada 5 gramos de comida ingerida. Por otro lado cuando vuelan, solo mantienen registro de la distancia recorrida.

l. Queremos poner en una lista a Twinkle, Pepita, Pepón y a Bombón, en ese orden. Luego, queremos pedirle a cada animal que:

   - Coma 20 gramos de alimento.
   - Vuele 2 kilómetros.
   - Coma 10 gramos de alimento.
   - Vuele otros 3 kilómetros.
   - Nos diga cada uno qué distancia recorrió en total.

   Escriba un main que realice estas acciones.

Ejercicio 4

Vuelva a implementar el código anterior ahora en Python. Piense en qué cosas cambian con respecto a Java.

Ejercicio 5

Piense y compare el código escrito en Java y en Python.

- ¿Qué cambia entre uno y otro?
- ¿Se necesita escribir exactamente el mismo código?
- ¿Hay cosas que puedo evitar en uno y son obligatorios en otro?
- ¿Hay alguna limitación en cuanto a la visibilidad?
- ¿Hay alguna característica interesante con respecto al encapsulamiento?
- Los patrones, ¿Se implementan igual?