Unidad 1: Introducción a la Programación Orientada a Objetos.

- Introducción a la POO
- Clases y Objetos

Guía de ejercicios:

1. Se tiene el siguiente código incompleto para la definición de una patente de un automotor. Complete las líneas incompletas para lograr una definición correcta (suponga que los argumentos son válidos).

```
public class Patente {
   String letrasInicioPatente;
   int numeroPatente;
   String letrasFinPatente;

   1. public ___________() {
         letrasInicioPatente = "";
         numeroPatente = 0;
         letrasFinPatente = "";
   }

   2. public Patente(_____________,_____________,_____________) {
         this.letrasInicioPatente = letrasInicioPatente;
         this.numeroPatente = numeroPatente;
         this.letrasFinPatente = letrasFinPatente;
   }

   public int getNumeroPatente() {
         3. ___________ numeroPatente;
   }

   public void setNumeroPatente(int numeroPatente) {
         this.numeroPatente = numeroPatente;
   }

   4. public __________ getLetrasInicioPatente() {
         return letrasInicioPatente;
   }

   5. public __________ getLetrasFinPatente() {
         return letrasFinPatente;
   }

   public void setLetrasInicioPatente(String letrasInicioPatente) {
       6. _________.letrasInicioPatente = letrasInicioPatente;
   }

   public void setLetrasFinPatente(String letrasFinPatente) {
       7. _________.letrasFinPatente = letrasFinPatente;
   }

}
```

2. Al código del ejercicio anterior se le realizaron modificaciones para validar los argumentos de los métodos **setxxx**. Ahora se deberá controlar que el atributo `numeroPatente` asuma valores entre cero y mil y que el atributo `letrasInicioPatente` y `letrasFinPatente` tenga una longitud de 2. Complete el siguiente código para lograr lo esperado.

```
public class Patente {

   String letrasInicioPatente;
   int numeroPatente;
   String letrasFinPatente;


   public Patente() {
         letrasInicioPatente = "AA";
         numeroPatente = 111;
         letrasFinPatente = "AA";
   }


   public int getNumeroPatente() {
       return numeroPatente;
   }


   public void setNumeroPatente(int numeroPatente) {
         1. if (____________________ && (numeroPatente < 1000))
         2.     ____________________________________________
            else
                System.out.println("Número de Patente Inválido");
   }


   public String getLetrasInicioPatente() {
       return letrasInicioPatente;
   }


   public String getLetrasFinPatente() {
       return letrasFinPatente;
   }


   public void setLetrasInicioPatente(String letrasPatente) {
       3. if (_____________________________________)
              this.letrasInicioPatente = letrasPatente;
          else
       4.     __________.out.println("Letras de Patente Inválida");
   }


   public boolean validarLongitudPatente(String unasletrasPatente) {
       5. return unasletrasPatente.length()___________;
   }

}
```

3.

a) Se desea modelar ecuaciones lineales de una variable independiente (rectas). Una recta se define por dos por dos puntos. Cada punto es un par de la forma (x,y) donde dichos valores del par son del tipo primitivo **float**. Se le solicita que defina la clase Punto y la clase Recta.

Clase Punto

Constructor

- ✓ `Punto(float x,float y)`
  Crea una instancia de un punto con las coordenadas argumentos.

Métodos

- ✓ `getX(): float`
  Retornar el valor de abcisa.

- ✓ `getY(): float`
  Retornar el valor de ordenada.

- ✓ `setX(float nuevoValor) : void`
  Establece un nuevo valor de abcisa.

- ✓ `setY(float nuevoValor) : void`
  Establece un nuevo valor de ordenada.

Clase Recta

Constructores

- ✓ `Recta(Punto p1, Punto p2)`
  Crea una instancia de Recta con los puntos argumentos.

- ✓ `Recta()`
  Crea una instancia de la recta identidad (y(x) = x)

Métodos:

- ✓ `pendiente(): float`
  Retornar la pendiente de la recta.

- ✓ `paralelas(Recta otraRecta): boolean`
  Recta *true* en caso que la recta argumento sea paralela a la recta receptora del mensaje, *false* caso contrario.

b) Defina la clase **`UsaRecta`** para realizar una aplicación de consola. El método main deberá tener la siguiente secuencia de instrucciones:

- ✓ Crear el punto P1(1,1)
- ✓ Crear el punto P2(2,2).
- ✓ Crear la recta R1 con los puntos P1 y P2.
- ✓ Mostrar por pantalla la pendiente de la recta R1.
- ✓ Crear el punto P3(3,3).
- ✓ Crear la recta R2 con los puntos P2 y P3.

4. Haz una clase llamada **Password** que cumple con las siguientes condiciones:

- Que tenga los atributos longitud y texto.
- Los constructores serán los siguientes:
    - Un constructor por defecto.
    - Un constructor con la contraseña que nosotros le pasamos.
- Los métodos que implementa serán:
    - *esFuerte()*: devuelve un booleano que indica si el texto de la password es fuerte (true) o no (false). Para que sea fuerte debe tener más de 2 mayúsculas, más de 1 minúscula y más de 5 números.
    - métodos getter y setter para el texto de la contraseña.
    - método getter para la longitud.
    - Si lo consideran necesario se pueden incorporar más métodos, privados o públicos.

Además, crea una clase ejecutable con el nombre que vos definas y que realice lo siguiente:

- Crear un array de N passwords, siendo N un valor cargado por teclado.
- Crea un bucle que recorra el arreglo para crear un objeto en cada una de sus posiciones y que:
    - pida por teclado la longitud mínima de las contraseñas.
    - pida el ingreso del texto de cada contraseña por teclado.
        - Si la contraseña no tiene la longitud mínima, se vuelve a solicitar hasta que cumpla con la restricción. Sólo se crean objetos cuando la contraseña tiene la longitud mínima.
        - Luego de creado el objeto, se mostrará un mensaje informativo, indicando si la contraseña es fuerte o no.