Ejercicios de Clase - POO

1- DooBeeDooBeeDo

Enunciado:

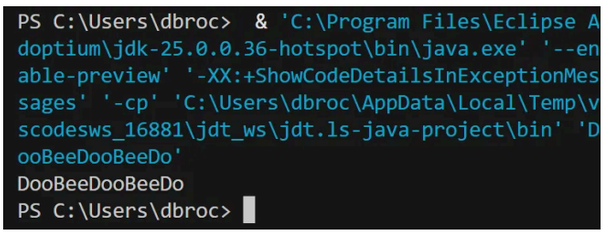
Dada la salida: “DooBeeDooBeeDo” llena el código faltante.

Solución:

Code:

```
public class DooBeeDooBeeDo {
    public static void main(String[] args) {
        int x = 1;
        while(x < 3){
            System.out.print("Doo");
            System.out.print("Bee");
            x = x + 1;
        }
        if (x == 3){
            System.out.print("Do");
        }
    }
}
```

Terminal Output:



2- a-b c-d

Enunciado:

Dada la salida: “`a-b c-d`” reordena el código, para ejecutar un programa en Java que produzca la anterior salida.

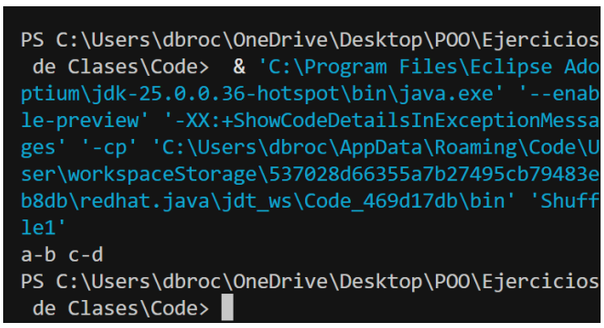
Solución:

Code:

```
public class Shuffle1 {
    public static void main(String[] args) {
        int x = 3;
        while (x > 0) {
            if (x > 2) {
                System.out.print("a");
            }
```

```
        x = x - 1;

        System.out.print("-");

        if (x == 2) {
            System.out.print("b c");
        }

        if (x == 1) {
            System.out.print("d");
            x = x -1;
        }

    }
  }
}
```

Terminal Output:



3- ¿Cuál archivo compilará?

Enunciado:

Cada una de las siguientes líneas de código representa un programa en Java, tu trabajo es ser un compilador y determinar cuál de estos programas compilará. Y si no compilan ¿Cómo lo solucionarías?

Solución:

Code A:

```
public class Exercisea {
   public static void main(String[] args) {
      int x = 1;
      while (x < 10) {
         if (x > 3) {
            System.out.println("big x");
         }
      }
   }
}
```

El anterior programa, a pesar de compilarse, no ejecuta nada en la consola, ya que `x` está definida en `1` lo que permite que el programa cumpla el `while`, pero al no incrementar el valor de `x` en cada paso, no permite cumplir el `if` que está dentro del `while`. La solución está en añadir un contador que permita incrementar el valor de `x` en cada vuelta del ciclo while.

Code B:

```
public static void main(String[] args){
   int x = 5;
   while (x > 1) {
      x = x - 1;
      if (x < 3) {
         System.out.println("small x");
      }
   }
}
```

El anterior código compila y funciona sin tener errores de lógica. A pesar de que le hace falta el `class Exerciseb { }` al inicio.

Code C:

```
class Exercisec {
   int x = 5;
   while (x > 1) {
      x = x - 1;
      if (x < 3) {
         System.out.println("small x");
      }
   }
}
```

En el código anterior hace falta declarar el método principal `public static void main(String[] args)`

4- Diez Botellas Verdes

Enunciado:

Escriba un programa en Java que imprima la letra de la canción “Diez Botellas Verdes”. La canción sigue una estructura repetitiva en la que se comienza con diez botellas colgando en la pared, y en cada iteración, una botella “cae”, reduciendo el número hasta llegar a cero. El programa debe utilizar un bucle para disminuir el número de botellas en cada iteración y condicionales para manejar los cambios en la gramática como “una botella” en lugar de “botellas”.

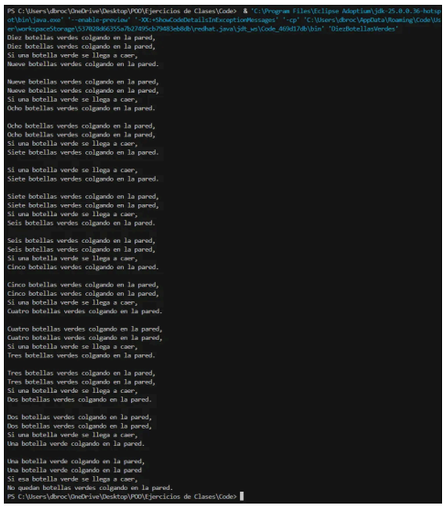
Solución:

Code:

```
public class DiezBotellasVerdes {
   public static void main(String[] args) {
      String botellasPlural = " botellas verdes colgando en la pared";
      String botellasSingular = " botella verde colgando en la pared";
      String[] numeros = {"Diez", "Nueve", "Ocho", "Siete", "Seis", "Cinco", "Cuatro", "Tres",
"Dos", "Una"};
      for (int i = 0; i < numeros.length;) {
         while (i <= 8){
            System.out.println(numeros[i] + botellasPlural + ",\n" + numeros[i] + botellasPlural
+ ",");
            System.out.println("Si una botella verde se llega a caer,");
            i++;
```

```
            if (i == 9){
                System.out.println(numeros[i] + botellasSingular + ".\n");
            }
            else{
                System.out.println(numeros[i] + botellasPlural + ".\n");
            }
        }
        if (i == 9){
            System.out.println(numeros[i] + botellasSingular + ",\n" + numeros[i] + botellasSin
gular);
            System.out.println("Si esa botella verde se llega a caer,\nNo quedan" + botellasPlur
al + ".");
            i++;
        }
      }
   }
}
```

Terminal Output:



5- Una noche de películas en Netflix

Enunciado:

- Mariana llega a casa un viernes en la noche y decide usar Netflix para relajarse. En la pantalla principal aparece su perfil con su nombre y una lista personalizada de recomendaciones. Ella observa varias opciones: series que ha visto recientemente, películas nuevas y títulos guardados en su lista.
- Decide iniciar la reproducción de una película. Antes de que empiece, puede ver informacion básica como el título, la duración, la categoría y quien la dirigió. Una vez comienza, tiene la posibilidad de detenerla, continuarla más tarde o adelantarla algunos minutos.
- Mientras la película avanza, la plataforma registra cuánto tiempo ha visto y, al terminar, le permite dejar una calificacion o marcar si le gusto. Automaticamente, esa acción influye en las recomendaciones futuras.
- Por otro lado, su hermano usa el mismo servicio con un perfil diferente. A pesar de que acceden desde la misma cuenta, cada uno tiene su propio historial de visualización y su lista de series favoritas.
- Finalmente, al día siguiente, Mariana recibe en su celular una notificación de Netflix con sugerencias de nuevas series similares a las que vio recientemente.

A partir de esta descripción:

1. Identificar al menos **tres objetos** presentes en el caso.
2. Proponer **algunos características** que cada objeto debería tener.
3. Proponer **algunas acciones** que esos objetos pueden realizar.

Solución:

- Objeto 1: Perfil
  - Características del objeto perfil:
    - Nombre de perfil
    - Historial de reproducción
    - Calificación de contenido
  - Acciones que puede realizar el objeto Perfil (Métodos)
    - Eliminar Perfil
    - Crear Perfil
    - Cambiar avatar
    - Activar modo niños
- Objeto 2: Contenido
  - Características del objeto contenido:
    - Título
    - Tipo de contenido
    - Categoría de Contenido
    - Duración
    - Reparto (Créditos)
  - Acciones que puede realizar el objeto Contenido (Métodos)
    - Reproducir
    - Calificar
    - Finalizar
- Objeto 3: Cuenta
  - Características del objeto Cuenta
    - Mail
    - Número de teléfono
    - Método de pago
  - Acciones que puede realizar el objeto Cuenta (Métodos)
    - Crear cuenta
    - Cobrar Tarjeta(?)
    - Canjear Tarjeta Regalo (?)

6- Biblioteca con Libros

Enunciado:

- Una biblioteca universitaria necesita un sistema sencillo para registrar sus libros.

- Cada libro debe tener un título y un autor, además de poder indicar si está disponible o prestado.
- Cuando un usuario solicita un libro:
  - Si está disponible, el sistema debe marcarlo como prestado.
  - Si ya estaba prestado, el sistema debe mostrar un mensaje informando que no puede volver a prestarse.
- Cuando el usuario devuelve un libro:
  - El sistema debe marcarlo como disponible otra vez.

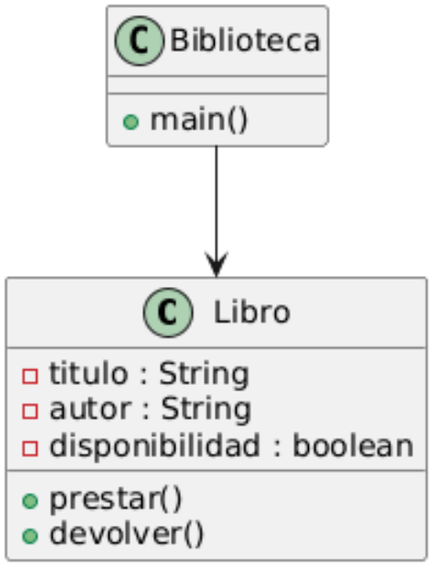
Tu tarea

1. Diseña una clase Libro que represente un libro.
   - Define los atributos necesarios para guardar su información.
   - Crea un método inicial para inicializar el título y el autor.
2. Implementa los métodos que permitan:
   - Prestar un libro.
   - Devolver un libro.
3. En un programa principal (main):
   - Crea un libro con un título y un autor.
   - Intenta prestarlo dos veces seguidas y observa qué ocurre.
   - Luego, devuélvelo y verifica que queda disponible.

Solución:

UML:

```
@startuml

class Libro {
   - titulo : String
   - autor : String
   - disponibilidad : boolean

   + prestar()
   + devolver()
}

class Biblioteca {

   + main()
}
Biblioteca --> Libro

@enduml
```



Code:

```
import java.util.ArrayList;
import java.util.Scanner;

class Libro {
```

```
    String titulo;
    String autor;
    boolean disponibilidad;

    public Libro(String titulo, String autor) {
        this.titulo = titulo;
        this.autor = autor;
        this.disponibilidad = true;
    }

    public void prestar() {
        if (disponibilidad) {
            disponibilidad = false;
            System.out.println("El libro ha sido prestado.");
        } else {
            System.out.println("El libro ya está prestado. No puede prestarse nuevamente.");
        }
    }

    public void devolver() {
        if (!disponibilidad) {
            disponibilidad = true;
            System.out.println("El libro ha sido devuelto y ahora está disponible.");
        } else {
            System.out.println("El libro ya estaba disponible.");
        }
    }
}

public class Biblioteca {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ArrayList<Libro> libros = new ArrayList<>();
        libros.add(new Libro("Cien años de soledad", "Gabriel Garcia Marquez"));
        libros.add(new Libro("El viejo y el mar", "Ernest Hemingway"));
        libros.add(new Libro("1984", "George Orwell"));

        int opcion;

        do {
            System.out.println("\nBienvenido al programa de gestión de biblioteca, para salir, intr
oduzca '9'");
            System.out.println("\nMENÚ PRINCIPAL");
            System.out.println("1. Agregar libro");
            System.out.println("2. Mostrar libros");
            System.out.print("Elige una opción: ");
            opcion = scanner.nextInt();
            scanner.nextLine();

            switch (opcion) {
                case 1:
                    System.out.print("Ingrese el título del libro: ");
                    String titulo = scanner.nextLine();
                    System.out.print("Ingrese el autor del libro: ");
                    String autor = scanner.nextLine();
```

```
            libros.add(new Libro(titulo, autor));
            System.out.println("Libro agregado correctamente.");
            break;

        case 2:
            System.out.println("\nLISTA DE LIBROS");

            for (int i = 0; i < libros.size(); i++) {
                Libro l = libros.get(i);

                String estado;
                if (l.disponibilidad) {
                    estado = "DISPONIBLE";
                } else {
                    estado = "PRESTADO";
                }

                System.out.println((i + 1) + ". " + l.titulo +
                        ", Autor: " + l.autor +
                        " (" + estado + ")");
            }

            System.out.println("\n¿Qué desea hacer?");
            System.out.println("1. Prestar un libro");
            System.out.println("2. Devolver un libro");
            System.out.print("Opción: ");

            int accion = scanner.nextInt();

        switch (accion) {
            case 1:
                {
                    System.out.print("Ingrese el número del libro que desea prestar: ");
                    int num = scanner.nextInt();
                    if (num < 1 || num > libros.size()) {
                        System.out.println("Selección inválida.");
                    } else {
                        Libro libro = libros.get(num - 1);

                        if (libro.disponibilidad) {
                            libro.prestar();
                        } else {
                            System.out.println("Ese libro NO está disponible para préstamo.");
                        }
                    }   break;
                }
            case 2:
                {
                    System.out.print("Ingrese el número del libro que desea devolver: ");
                    int num = scanner.nextInt();
                    if (num < 1 || num > libros.size()) {
                        System.out.println("Selección inválida.");
                    } else {
                        Libro libro = libros.get(num - 1);

                        if (!libro.disponibilidad) {
```

```
                        libro.devolver();
                    } else {
                        System.out.println("Ese libro ya está disponible, no se puede devolve
r.");
                    }
                }   break;
            }
            default:
                System.out.println("Opción inválida.");
                break;
        }

            break;


        default:
            if (opcion != 9){
                System.out.println("Opción inválida.");}
      }

    } while (opcion != 9);

    scanner.close();
  }
}
```

7- Videojuego Simple

Enunciado:

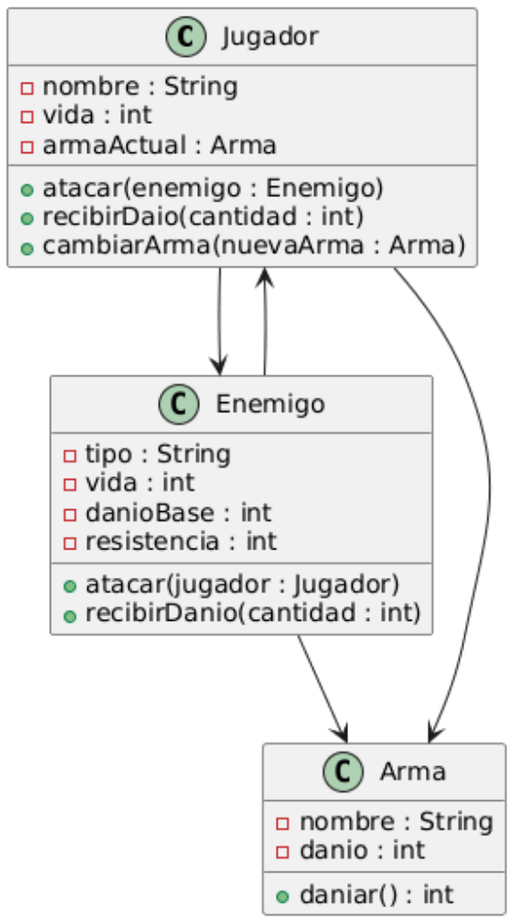
Reto para la próxima clase: Modelar un videojuego simple

- Imagina que eres parte del equipo que está diseñando un videojuego de aventura.
  En el juego, un jugador recorre un escenario en el que se encuentra con diferentes enemigos.
  El jugador puede portar una arma para defenderse, y cada vez que ataca o recibe daño, el estado del juego cambia.
- Los enemigos tienen distintas formas de reaccionar: algunos son más débiles, otros más resistentes.
  El jugador, a su vez, puede mejorar sus armas o cambiarlas durante la partida.
- El sistema debe ser capaz de representar estas interacciones básicas entre jugador, enemigo y arma, de manera que podamos simular combates sencillos.
  Tu tarea
  - Identifica los objetos principales en este escenario.
  - Piensa qué información debería tener cada objeto (por ejemplo, qué características los diferencian).
  - Propón qué acciones puede realizar cada uno dentro del juego.
  - Lleva tus ideas a la próxima clase para discutirlas en grupo y construir juntos un modelo en Java.

Solución:

UML:

```
@startuml
class Jugador {
    - nombre : String
    - vida : int
    - armaActual : Arma
    + atacar(enemigo : Enemigo)
    + recibirDaio(cantidad : int)
    + cambiarArma(nuevaArma : Arma)
}
class Enemigo {
    - tipo : String
    - vida : int
    - danioBase : int
    - resistencia : int
    + atacar(jugador : Jugador)
    + recibirDanio(cantidad : int)
}
class Arma {
    - nombre : String
    - danio : int
    + daniar() : int
}
Jugador --> Arma
Jugador --> Enemigo
Enemigo --> Jugador
Enemigo --> Arma
@enduml
```



8- Empleado de una empresa de tecnología

Enunciado:

- Una empresa de desarrollo de software necesita llevar un registro de sus empleados.
  Cada **Empleado** debe contar con información básica como el **nombre completo**, la **fecha de contratación** y el **cargo** que ocupa dentro de la organización.
- En algunos casos, basta con registrar solo el nombre del empleado (por ejemplo, cuando es un nuevo practicante y aún no tiene un cargo asignado).
  En otros casos, se requiere registrar nombre y cargo, dejando la fecha de contratación vacía porque todavía no ha sido formalizado el ingreso.
  Finalmente, cuando la información está completa, se deben guardar los tres datos: nombre, fecha de contratación y cargo.
- La empresa también necesita crear copias de un empleado ya existente, con toda su información, para realizar simulaciones internas en su sistema de pruebas.

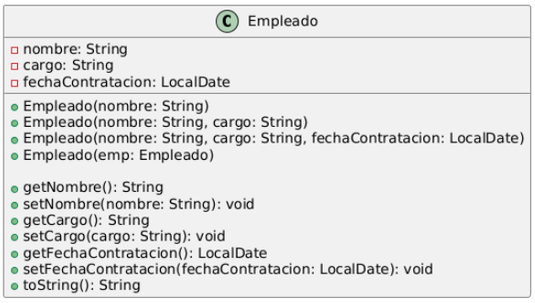
1. Diseña la clase **Empleado** con los atributos adecuados.
2. Implementa al menos **tres constructores**:
   - Uno que reciba **todos los atributos**.
   - Uno que solo reciba el **nombre**.
   - Uno que reciba **nombre y cargo**.

3. Implementa un **constructor copia** que permita crear un nuevo objeto *Empleado* a partir de otro.
4. Crea un programa de prueba *(main)* que instancie varios empleados usando los diferentes constructores.
5. Genera el **diagrama UML de la clase Empleado**, mostrando atributos, constructores y visibilidad.

Solución:

UML:

```
@startuml
class Empleado {
    - nombre: String
    - cargo: String
    - fechaContratacion: LocalDate

    + Empleado(nombre: String)
    + Empleado(nombre: String, cargo: Stri
ng)
    + Empleado(nombre: String, cargo: Stri
ng, fechaContratacion: LocalDate)
    + Empleado(emp: Empleado)

    + getNombre(): String
    + setNombre(nombre: String): void
    + getCargo(): String
    + setCargo(cargo: String): void
    + getFechaContratacion(): LocalDate
    + setFechaContratacion(fechaContrata
cion: LocalDate): void
    + toString(): String
}
@enduml
```



Code:

```
import java.time.LocalDate;
class Empleado {
    private String nombre;
    private String cargo;
    private LocalDate fechaContratacion;
    public Empleado(String nombre) {
        this.nombre = nombre;
    }
    public Empleado(String nombre, String cargo) {
        this.nombre = nombre;
        this.cargo = cargo;
    }
    public Empleado(String nombre, String cargo, LocalDate fechaContratacion) {
        this.nombre = nombre;
        this.cargo = cargo;
        this.fechaContratacion = fechaContratacion;
    }
```

```
    public Empleado(Empleado emp) {
        this.nombre = emp.nombre;
        this.cargo = emp.cargo;
        this.fechaContratacion = emp.fechaContratacion;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getCargo() {
        return cargo;
    }
    public void setCargo(String cargo) {
        this.cargo = cargo;
    }
    public LocalDate getFechaContratacion() {
        return fechaContratacion;
    }
    public void setFechaContratacion(LocalDate fechaContratacion) {
        this.fechaContratacion = fechaContratacion;
    }
    @Override
    public String toString() {
    return "Empleado: \nNombre = " + nombre + "\nCargo = " + cargo + "\nFecha de Contrata
cion = " + fechaContratacion + "\n";
    }
}
public class RegistroEmpleados {
    public static void main(String[] args) {
        Empleado emp1 = new Empleado("Carlos");
        System.out.println(emp1);
        Empleado emp2 = new Empleado("Ana", "Gerente");
        System.out.println(emp2);
        Empleado emp3 = new Empleado("Luis", "Analista", LocalDate.of(2025, 05, 10));
        System.out.println(emp3);
        Empleado emp4 = new Empleado(emp3);
        System.out.println(emp4);
    }
}
```

9- Caso de estudio - Registro de estudiantes

Enunciado:

- En la universidad se está implementando un sistema sencillo para llevar el control de los estudiantes que participan en diferentes cursos. Actualmente, los registros se realizan de manera manual en hojas de cálculo, lo que genera errores y duplicidad de información.
- El nuevo sistema permitirá que un administrador o un profesor realicen el registro de los estudiantes de forma más organizada y segura.
- Cuando un usuario autorizado accede al sistema, debe poder registrar un nuevo estudiante, ingresando información básica: nombre completo, edad y número de identificación. El

sistema validará que el número de identificación no se repita, de modo que no existan duplicados.

- Si todos los datos son correctos, el sistema creará un “Estudiante” con la información ingresada y lo almacenará en una lista temporal en memoria. Finalmente, se mostrará un mensaje de confirmación para asegurar que el registro fue exitoso.
- En caso de que el usuario intente registrar un estudiante con un número de identificación ya existente, el sistema deberá mostrar un mensaje de error y solicitar un nuevo número de identificación válido.
  Actividad

1. A partir de este escenario, identifiquen:
   - Actores primarios y secundarios.
   - Precondiciones.
   - Flujo principal de eventos.
   - Posibles flujos alternativos.
2. Completen la especificación del caso de uso CU-001: Registrar estudiante siguiendo el formato entregado en clase.

Identificador
Breve descripción
Actores Primarios
Actores Secundarios
Precondiciones
Flujo principal
Postcondiciones
Flujos alternativos

Solución:

Identificador	CU-001: Registrar estudiante
Breve descripción	Permite registrar un nuevo estudiante ingresando nombre, edad e identificación, validando que no exista duplicado.
Actores Primarios	- Administrador - Profesor
Actores Secundarios	Sistema de Registro
Precondiciones	- El usuario debe estar autenticado. - Debe existir una lista disponible para almacenar estudiantes.
Flujo principal	1- El usuario selecciona “Registrar estudiante”. 2- El sistema solicita datos. 3- El usuario ingresa nombre, edad e identificación. 4- El sistema valida datos y verifica que el ID no exista. 5- El sistema registra al estudiante. 6- El sistema reporta el registro exitoso.
Postcondiciones	El estudiante debe quedar registrado sin duplicidad de identificación.
Flujos alternativos	**Si hay ID duplicado:** El sistema notifica error y solicita un nuevo ID. **Si hay Campos incompletos:** Muestra error y pide volver a ingresar datos. **Si la Edad es inválida:** Muestra error y solicita corregir la edad.