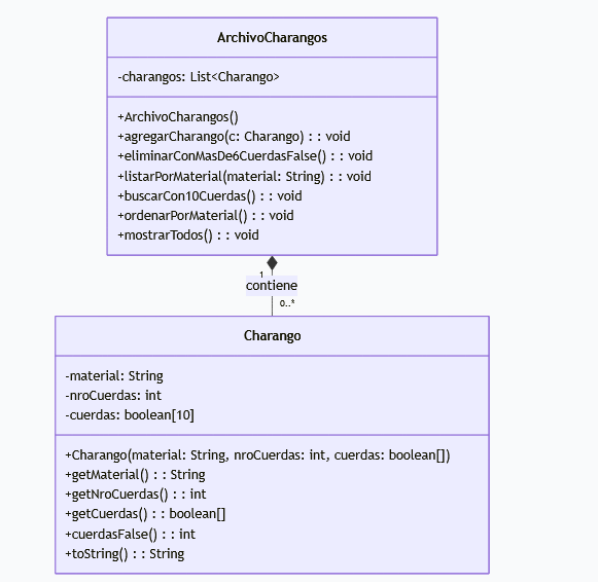
Practica N 3

Persistencia

Nombre: Carlos Angel Morales Cruz

Ejercicio1



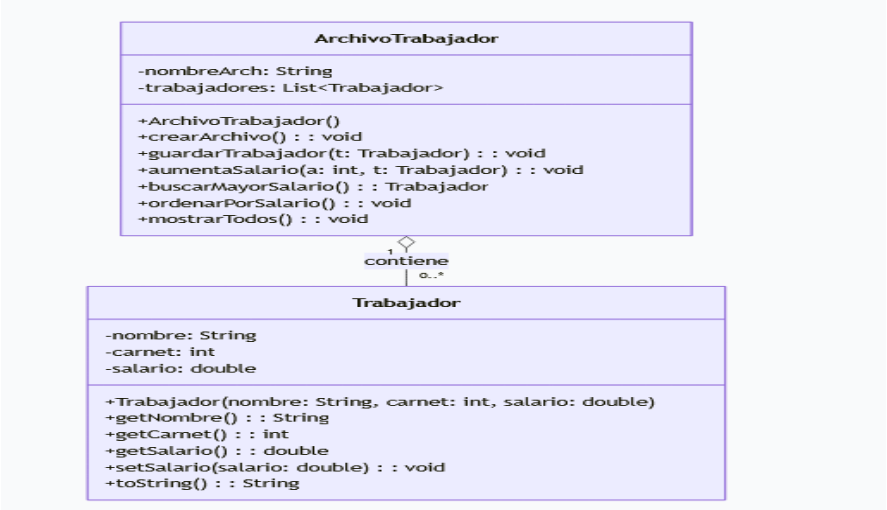

```java
public class Charango implements Serializable {
    private static final long serialVersionUID = 1L;

    private Integer id;
    private String material;
    private int numeroCuerdas;
    private boolean[] cuerdas;
    private LocalDateTime fechaCreacion;

    public Charango() {
        this.cuerdas = new boolean[10];
        this.fechaCreacion = LocalDateTime.now();
    }

    public Charango(String material, int numeroCuerdas) {
        this();
        this.material = material;
        setNumeroCuerdas(numeroCuerdas);
        inicializarCuerdas();
    }

    public Charango(Integer id, String material, int numeroCuerdas, boolean[] cuerdas) {
        this.id = id;
        this.material = material;
        setNumeroCuerdas(numeroCuerdas);
        setCuerdas(cuerdas);
        this.fechaCreacion = LocalDateTime.now();
    }

    private void inicializarCuerdas() {
        for (int i = 0; i < numeroCuerdas && i < 10; i++) {
            cuerdas[i] = true;
        }
    }
```

```java

    public int contarCuerdasFalsas() {
        int contador = 0;
        for (boolean cuerda : cuerdas) {
            if (!cuerda) contador++;
        }
        return contador;
    }

    public boolean tieneMasDe6CuerdasFalsas() {
        return contarCuerdasFalsas() > 6;
    }

    public boolean esDeMaterial(String material) {
        return this.material.equalsIgnoreCase(material);
    }

    public boolean tiene10Cuerdas() {
        return numeroCuerdas == 10;
    }

    public String getRepresentacionCuerdas() {
        StringBuilder sb = new StringBuilder(str: "[");
        for (int i = 0; i < 10; i++) {
            if (i < numeroCuerdas) {
                sb.append(cuerdas[i] ? "■" : "□");
            } else {
                sb.append(str: " ");
            }
        }
        sb.append(str: "]");
        return sb.toString();
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getMaterial() {
        return material;
```

```java
    public void setId(Integer id) {
        this.id = id;
    }

    public String getMaterial() {
        return material;
    }

    public void setMaterial(String material) {
        this.material = material;
    }

    public int getNumeroCuerdas() {
        return numeroCuerdas;
    }

    public void setNumeroCuerdas(int numeroCuerdas) {
        if (numeroCuerdas < 0 || numeroCuerdas > 10) {
            throw new IllegalArgumentException(s: "Número de cuerdas inválido");
        }
        this.numeroCuerdas = numeroCuerdas;
    }

    public boolean[] getCuerdas() {
        return Arrays.copyOf(cuerdas, newLength: 10);
    }

    public void setCuerdas(boolean[] cuerdas) {
        if (cuerdas.length != 10) {
            throw new IllegalArgumentException(s: "Array debe tener 10 elementos");
        }
        this.cuerdas = Arrays.copyOf(cuerdas, newLength: 10);
    }

    public LocalDateTime getFechaCreacion() {
        return fechaCreacion;
    }

    @Override
    public String toString() {
        return String.format(format: "Charango[ID=%d, Material=%s, Cuerdas=%d, Falsas=%d] %s",
                id, material, numeroCuerdas, contarCuerdasFalsas(), getRepresentacionCuerdas());
    }
```

```java
    @Override
    public String toString() {
        return String.format(format: "Charango[ID=%d, Material=%s, Cuerdas=%d, Falsas=%d] %s",
                id, material, numeroCuerdas, contarCuerdasFalsas(), getRepresentacionCuerdas());
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Charango charango = (Charango) obj;
        return id != null && id.equals(charango.id);
    }

    @Override
    public int hashCode() {
        return id != null ? id.hashCode() : 0;
    }
}
```

Ejercicio 2



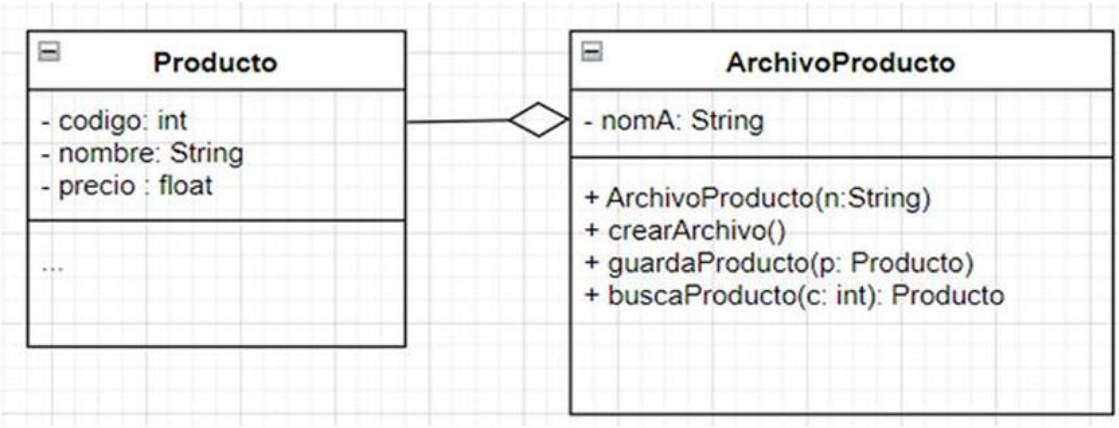

```python
import pickle
import os
from typing import List

class Trabajador:
    def __init__(self, nombre: str, carnet: int, salario: float):
        (method) def aumentar_salario(
            self: Self@Trabajador,
            porcentaje: int
        ) -> None
    def aumentar_salario(self, porcentaje: int):
        self.salario += self.salario * (porcentaje / 100)

    def __str__(self):
        return f"{self.carnet} | {self.nombre:20} | {self.salario:10.2f}"

    def to_dict(self):
        return {'nombre': self.nombre, 'carnet': self.carnet, 'salario': self.salario}

    @classmethod
    def from_dict(cls, data):
        return cls(data['nombre'], data['carnet'], data['salario'])

class ArchivoTrabajador:
    def __init__(self, archivo="trabajadores.dat"):
        self.nombre_arch = archivo
        self.trabajadores = []
        self.cargar()

    def crear_archivo(self):
        try:
            with open(self.nombre_arch, 'wb') as f:
                pickle.dump([], f)
            return True
        except:
            return False
```

```python
    def guardar_trabajador(self, t):
        self.trabajadores.append(t)
        self.guardar()

    def aumenta_salario(self, a, t):
        if t in self.trabajadores:
            t.aumentar_salario(a)
            self.guardar()
            return True
        return False

    def buscar_mayor_salario(self):
        if not self.trabajadores:
            return None
        mayor = self.trabajadores[0]
        for t in self.trabajadores[1:]:
            if t.salario > mayor.salario:
                mayor = t
        return mayor

    def ordenar_por_salario(self):
        return sorted(self.trabajadores, key=lambda x: x.salario)

    def guardar(self):
        try:
            datos = [t.to_dict() for t in self.trabajadores]
            with open(self.nombre_arch, 'wb') as f:
                pickle.dump(datos, f)
        except:
            pass

    def cargar(self):
        try:
            if os.path.exists(self.nombre_arch):
                with open(self.nombre_arch, 'rb') as f:
                    datos = pickle.load(f)
                    self.trabajadores = [Trabajador.from_dict(d) for d in datos]
        except:
            self.trabajadores = []

    def buscar_por_carnet(self, carnet):
        for t in self.trabajadores:
            if t.carnet == carnet:
                return t
        return None

    def eliminar_trabajador(self, carnet):
        for i, t in enumerate(self.trabajadores):
            if t.carnet == carnet:
                del self.trabajadores[i]
                self.guardar()
                return True
        return False
```

```
24  class ArchivoTrabajador:
92      def obtener_todos(self):
93          return self.trabajadores.copy()
94
95      def generar_ejemplos(self):
96          if not self.trabajadores:
97              ejemplos = [
98                  Trabajador("Juan Pérez", 1001, 2500),
99                  Trabajador("María López", 1002, 3200),
100                 Trabajador("Carlos Ruiz", 1003, 1800),
101                 Trabajador("Ana García", 1004, 4100),
102                 Trabajador("Pedro Martínez", 1005, 2900)
103             ]
104             for t in ejemplos:
105                 self.guardar_trabajador(t)
106
107 class SistemaTrabajadores:
108     def __init__(self):
109         self.archivo = ArchivoTrabajador()
110
111     def iniciar(self):
112         while True:
113             print("\n" + "="*50)
114             print("SISTEMA DE TRABAJADORES")
115             print("="*50)
116             print("1. Crear archivo")
117             print("2. Mostrar todos")
118             print("3. Agregar trabajador")
119             print("4. Aumentar salario")
120             print("5. Mayor salario")
121             print("6. Ordenar por salario")
122             print("7. Eliminar trabajador")
123             print("8. Buscar por carnet")
124             print("9. Generar ejemplos")
125             print("0. Salir")
126             print("="*50)
127
128             op = input("Opción: ").strip()
129
130             if op == "0":
131                 print("¡Adiós!")
132                 break
133             elif op == "1":
134                 self.crear_archivo()
135             elif op == "2":
136                 self.mostrar_todos()
137             elif op == "3":
138                 self.agregar_trabajador()
139             elif op == "4":
```

```
            elif op == "4":
                self.aumentar_salario()
            elif op == "5":
                self.mayor_salario()
            elif op == "6":
                self.ordenar_salario()
            elif op == "7":
                self.eliminar_trabajador()
            elif op == "8":
                self.buscar_carnet()
            elif op == "9":
                self.generar_ejemplos()

    def crear_archivo(self):
        if self.archivo.crear_archivo():
            print("✓ Archivo creado")
        else:
            print("X Error al crear archivo")

    def mostrar_todos(self):
        trabajadores = self.archivo.obtener_todos()
        if not trabajadores:
            print("No hay trabajadores")
            return

        print("\n" + "-"*50)
        print(f"{'Carnet':<8} {'Nombre':<20} {'Salario':<10}")
        print("-"*50)
        for t in trabajadores:
            print(t)
        print("-"*50)
        print(f"Total: {len(trabajadores)}")

    def agregar_trabajador(self):
        try:
            nombre = input("Nombre: ").strip()
            if not nombre:
                print("Nombre requerido")
                return

            carnet = int(input("Carnet: "))
            if carnet <= 0:
                print("Carnet debe ser positivo")
                return

            salario = float(input("Salario: "))
```

Ejercicio 3




```python
import pickle
import os

class Producto:
    def __init__(self, codigo, nombre, precio):
        self.codigo = codigo
        self.nombre = nombre
        self.precio = precio

    def __str__(self):
        return f"{self.codigo} | {self.nombre:20} | ${self.precio:10.2f}"

class ArchivoProducto:
    def __init__(self, nomA="productos.dat"):
        self.nomA = nomA
        self.productos = []
        self.cargar()

    def crearArchivo(self):
        try:
            with open(self.nomA, 'wb') as f:
                pickle.dump([], f)
            return True
        except:
            return False

    def guardaProducto(self, p):
        self.productos.append(p)
        self.guardar()

    def buscaProducto(self, c):
        for p in self.productos:
            if p.codigo == c:
                return p
        return None

    def promedioPrecios(self):
        if not self.productos:
            return 0
        total = sum(p.precio for p in self.productos)
        return total / len(self.productos)
```

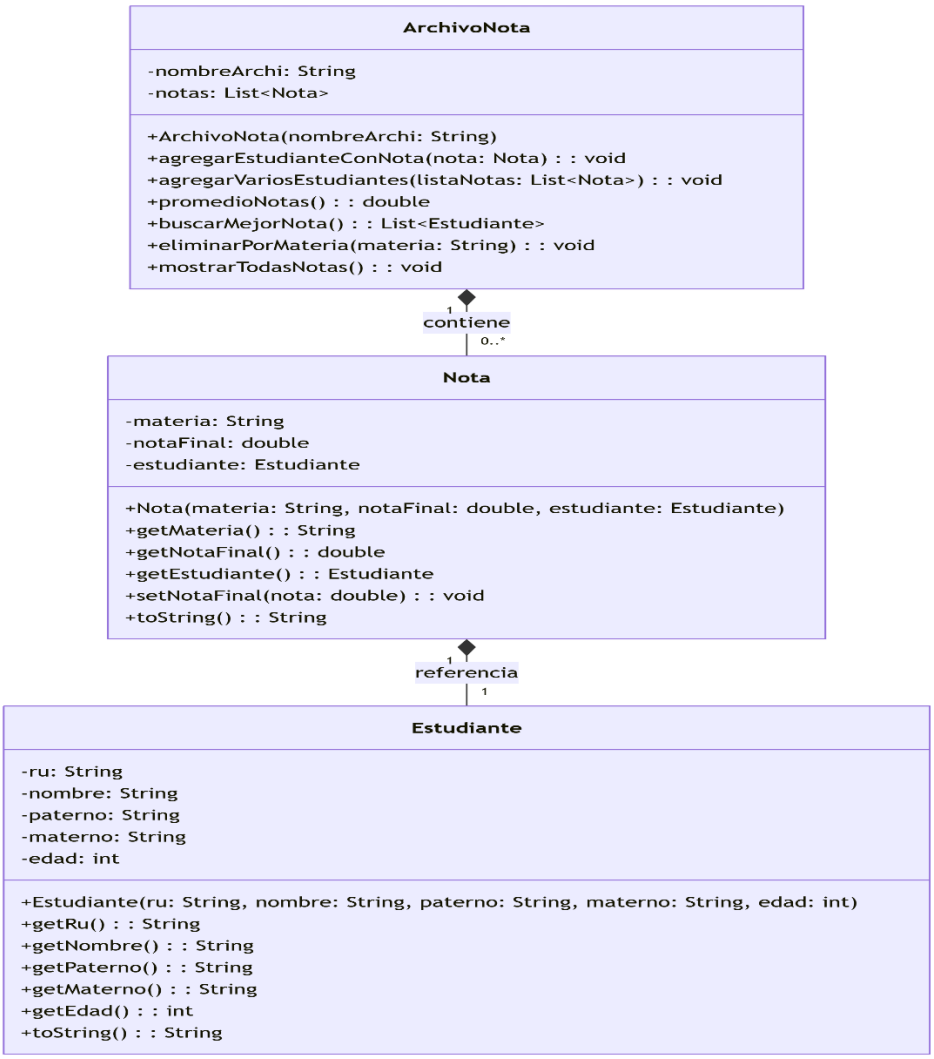
```
        return total / len(self.productos)

    def productoMasCaro(self):
        if not self.productos:
            return None
        return max(self.productos, key=lambda p: p.precio)

    def guardar(self):
        try:
            with open(self.nomA, 'wb') as f:
                pickle.dump(self.productos, f)
        except:
            pass

    def cargar(self):
        try:
            if os.path.exists(self.nomA):
                with open(self.nomA, 'rb') as f:
                    self.productos = pickle.load(f)
        except:
            self.productos = []

    def obtenerTodos(self):
        return self.productos.copy()

    def eliminarProducto(self, codigo):
        for i, p in enumerate(self.productos):
            if p.codigo == codigo:
                del self.productos[i]
                self.guardar()
                return True
        return False

    def generarEjemplos(self):
        if not self.productos:
            ejemplos = [
                Producto(1001, "Laptop", 1200.50),
                Producto(1002, "Mouse", 25.99),
                Producto(1003, "Teclado", 45.75),
                Producto(1004, "Monitor", 350.00),
                Producto(1005, "Tablet", 550.80)
            ]
            for p in ejemplos:
                self.guardaProducto(p)

class SistemaProductos:
```

```python
    def __init__(self):
        self.archivo = ArchivoProducto()

    def iniciar(self):
        while True:
            print("\n" + "="*50)
            print("SISTEMA DE PRODUCTOS")
            print("="*50)
            print("1. Crear archivo")
            print("2. Mostrar todos")
            print("3. Agregar producto")
            print("4. Buscar producto")
            print("5. Promedio precios")
            print("6. Producto más caro")
            print("7. Eliminar producto")
            print("8. Generar ejemplos")
            print("0. Salir")
            print("="*50)

            op = input("Opción: ").strip()

            if op == "0":
                print("¡Adiós!")
                break
            elif op == "1":
                self.crear_archivo()
            elif op == "2":
                self.mostrar_todos()
            elif op == "3":
                self.agregar_producto()
            elif op == "4":
                self.buscar_producto()
            elif op == "5":
                self.promedio_precios()
            elif op == "6":
                self.producto_mas_caro()
            elif op == "7":
                self.eliminar_producto()
            elif op == "8":
                self.generar_ejemplos()

    def crear_archivo(self):
        if self.archivo.crearArchivo():
            print("✓ Archivo creado")
        else:
            print("✗ Error al crear archivo")

    def mostrar todos(self):
```

```python
    def mostrar_todos(self):
        productos = self.archivo.obtenerTodos()
        if not productos:
            print("No hay productos")
            return

        print("\n" + "="*50)
        print(f"{'Código':<10} {'Nombre':<20} {'Precio':<10}")
        print("-"*50)
        for p in productos:
            print(f"{p.codigo:<10} {p.nombre:<20} ${p.precio:<10.2f}")
        print("="*50)
        print(f"Total: {len(productos)} productos")

    def agregar_producto(self):
        try:
            codigo = int(input("Código: "))
            if codigo <= 0:
                print("Código debe ser positivo")
                return

            nombre = input("Nombre: ").strip()
            if not nombre:
                print("Nombre requerido")
                return

            precio = float(input("Precio: "))
            if precio <= 0:
                print("Precio debe ser positivo")
                return

            p = Producto(codigo, nombre, precio)
            self.archivo.guardaProducto(p)
            print("✓ Producto agregado")

        except ValueError:
            print("Entrada inválida")

    def buscar_producto(self):
        try:
            codigo = int(input("Código a buscar: "))
            p = self.archivo.buscaProducto(codigo)

            if p:
                print(f"\nProducto encontrado:")
                print(f"Código: {p.codigo}")
```

```
172     def buscar_producto(self):
178                 print(f"\nProducto encontrado:")
179                 print(f"Código: {p.codigo}")
180                 print(f"Nombre: {p.nombre}")
181                 print(f"Precio: ${p.precio:.2f}")
182             else:
183                 print("Producto no encontrado")
184
185         except ValueError:
186             print("Código inválido")
187
188     def  (variable) promedio: float | Literal[0]
189         promedio = self.archivo.promedioPrecios()
190         print(f"\nPromedio de precios: ${promedio:.2f}")
191
192     def producto_mas_caro(self):
193         p = self.archivo.productoMasCaro()
194         if p:
195             print(f"\nProducto más caro:")
196             print(f"Código: {p.codigo}")
197             print(f"Nombre: {p.nombre}")
198             print(f"Precio: ${p.precio:.2f}")
199         else:
200             print("No hay productos")
201
202     def eliminar_producto(self):
203         try:
204             codigo = int(input("Código a eliminar: "))
205             if self.archivo.eliminarProducto(codigo):
206                 print("✓ Producto eliminado")
207             else:
208                 print("Producto no encontrado")
209         except ValueError:
210             print("Código inválido")
211
212     def generar_ejemplos(self):
213         self.archivo.generarEjemplos()
214         print("✓ Ejemplos generados")
215
216 if __name__ == "__main__":
217     app = SistemaProductos()
218     app.iniciar()
```

Ejercicio 4

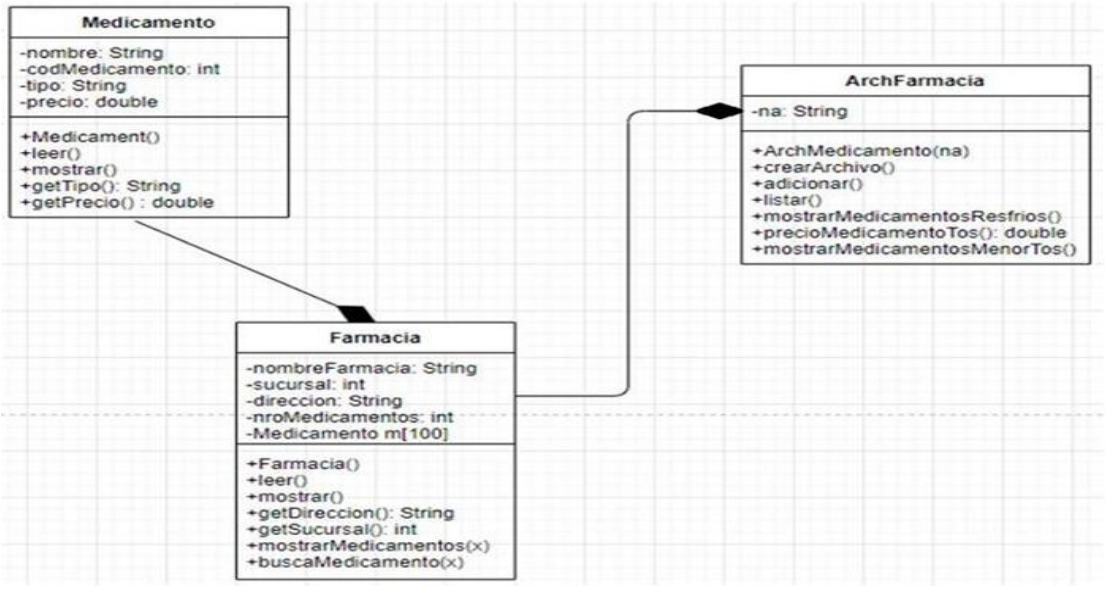



```python
import pickle
import os

class Estudiante:
    def __init__(self, ru, nombre, paterno, materno, edad):
        self.ru = ru
        self.nombre = nombre
        self.paterno = paterno
        self.materno = materno
        self.edad = edad

class Nota:
    def __init__(self, materno, notaFinal, estudiante):
        self.materno = materno
        self.notaFinal = notaFinal
        self.estudiante = estudiante

class ArchiNota:
    def __init__(self, nombreArchi="notas.dat"):
        self.nombreArchi = nombreArchi
        self.notas = []
        self.cargar()
```

(parameter) nombre: Any

```python
    def agregar_estudiantes(self, lista_notas):
        self.notas.extend(lista_notas)
        self.guardar()

    def promedio_notas(self):
        if not self.notas:
            return 0
        total = sum(n.n
        return total / len(self.notas)

    def mejores_estudiantes(self):
        if not self.notas:
            return []
        max_nota = max(n.notaFinal for n in self.notas)
        return [n for n in self.notas if n.notaFinal == max_nota]

    def eliminar_por_materia(self, materia):
        self.notas = [n for n in self.notas if n.materno != materia]
        self.guardar()

    def guardar(self):
        with open(self.nombreArchi, 'wb') as f:
            pickle.dump(self.notas, f)

    def cargar(self):
        if os.path.exists(self.nombreArchi):
            with open(self.nombreArchi, 'rb') as f:
                self.notas = pickle.load(f)

if __name__ == "__main__":
    archi = ArchiNota()
    e1 = Estudiante("123", "Ana", "Lopez", "Perez", 20)
    e2 = Estudiante("124", "Luis", "Garcia", "Martinez", 21)
    n1 = Nota("Matematica", 85, e1)
    n2 = Nota("Fisica", 90, e2)
    n3 = Nota("Matematica", 88, e2)
    archi.agregar_estudiantes([n1, n2, n3])
    print(f"Promedio: {archi.promedio_notas()}")
    mejores = archi.mejores_estudiantes()
    print(f"Mejor(es) estudiante(s): {[(m.estudiante.nombre, m.notaFinal) for m in mejores]}")
    archi.eliminar_por_materia("Matematica")
    print(f"Notas despues de eliminar Matematica: {len(archi.notas)}")
```


Ejercicio 5

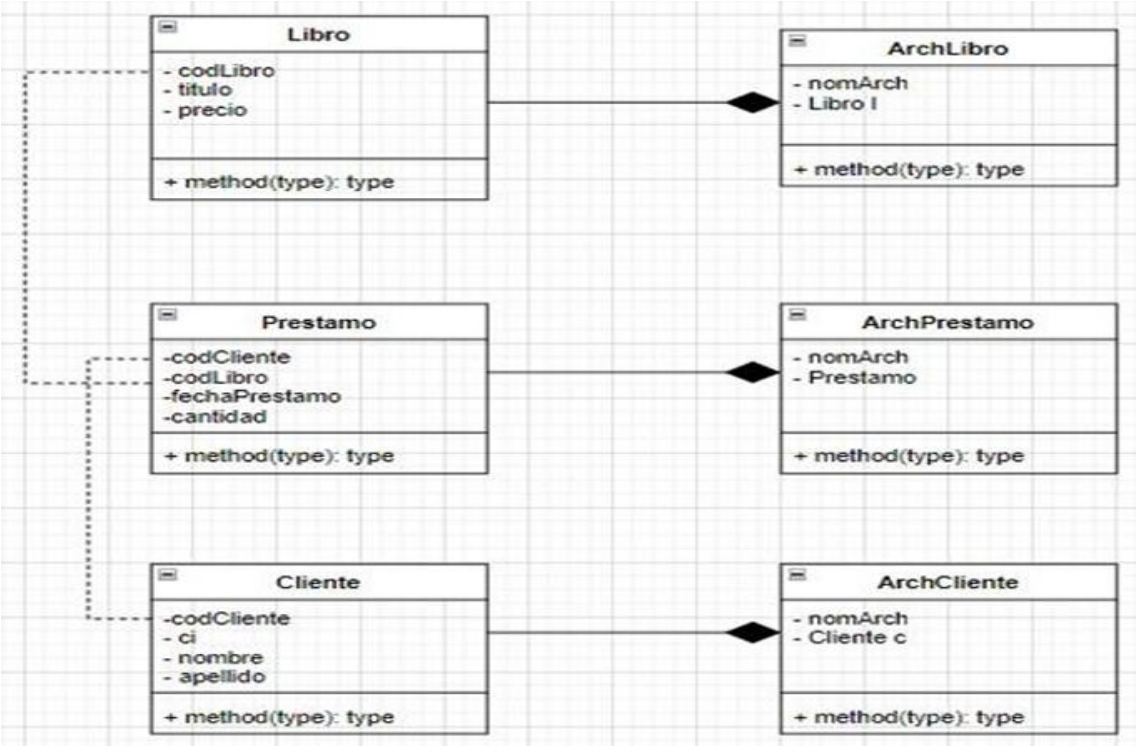


```python
import pickle
import os

class Medicamento:
    def __init__(self, nombre="", codMedicamento=0, tipo="", precio=0.0):
        self.nombre = nombre
        self.codMedicamento = codMedicamento
        self.tipo = tipo
        self.precio = precio
    def leer(self):
        self.nombre = input("Nombre: ")
        self.codMedicamento = int(input("Codigo: "))
        self.tipo = input("Tipo: ")
        self.precio = float(input("Precio: "))
    def mostrar(self):
        print(f"{self.nombre} {self.codMedicamento} {self.tipo} {self.precio}")
    def getTipo(self):
        return self.tipo
    def getPrecio(self):
        return self.precio

class Farmacia:
    def __init__(self):
        self.nombreFarmacia = ""
        self.sucursal = 0
        self.direction = ""
        self.moMedicamentos = 0
        self.m = [Medicamento() for _ in range(100)]
    def leer(self):
        self.nombreFarmacia = input("Nombre farmacia: ")
        self.sucursal = int(input("Sucursal: "))
        self.direction = input("Direccion: ")
        self.moMedicamentos = int(input("Cant medicamentos: "))
        for i in range(self.moMedicamentos):
            self.m[i].leer()
    def mostrar(self):
        print(f"{self.nombreFarmacia} {self.sucursal} {self.direction}")
        for i in range(self.moMedicamentos):
            self.m[i].mostrar()
    def getDireccion(self):
        return self.direction
    def getSucursal(self):
        return self.sucursal
    def mostrarMedicamentos(self, x):
        for i in range(self.moMedicamentos):
            if self.m[i].getTipo() == x:
                self.m[i].mostrar()
    def buscaMedicamento(self, x):
        for i in range(self.moMedicamentos):
            if self.m[i].nombre == x:
                return self.m[i]
        return None

class ArchFarmacia:
    def __init__(self, na="farmacias.dat"):
        self.na = na
        self.farmacias = []
        self.cargar()
    def crearArchivo(self):
```

```python
    def crearArchivo(self):
        n = int(input("Cant farmacias: "))
        for _ in range(n):
            f = Farmacia()
            f.leer()
            self.farmacias.append(f)
        self.guardar()
    def adicionar(self):
        f = Farmacia()
        f.leer()
        self.farmacias.append(f)
        self.guardar()
    def listar(self):
        for f in self.farmacias:
            f.mostrar()
    def mostrarMedicamentosResfrios(self):
        for f in self.farmacias:
            f.mostrarMedicamentos("Resfrio")
    def predoMedicamentoTos(self):
        precio_total = 0
        cantidad = 0
        for f in self.farmacias:
            for i in range(f.moMedicamentos):
                if f.m[i].getTipo() == "Tos":
                    precio_total += f.m[i].getPrecio()
                    cantidad += 1
        return precio_total / cantidad if cantidad > 0 else 0
    def mostrarMedicamentosMenorTos(self):
        promedio = self.predoMedicamentoTos()
        for f in self.farmacias:
            for i in range(f.moMedicamentos):
                if f.m[i].getTipo() == "Tos" and f.m[i].getPrecio() < promedio:
                    f.m[i].mostrar()
    def a(self, sucursal_x):
        for f in self.farmacias:
            if f.getSucursal() == sucursal_x:
                f.mostrarMedicamentos("Tos")
    def b(self):
        for f in self.farmacias:
            for i in range(f.moMedicamentos):
                if f.m[i].nombre == "Tapsin":
                    print(f"{f.getSucursal()} {f.getDireccion()}")
    def c(self, tipo):
        for f in self.farmacias:
            f.mostrarMedicamentos(tipo)
    def d(self):
        self.farmacias.sort(key=lambda x: x.getDireccion())
        self.guardar()
        self.listar()
    def e(self, tipo, sucursal_y, sucursal_z):
        farmacia_y = next((f for f in self.farmacias if f.getSucursal() == sucursal_y), None)
        farmacia_z = next((f for f in self.farmacias if f.getSucursal() == sucursal_z), None)
        if not farmacia_y or not farmacia_z:
            return
        movidos = []
        i = 0
        while i < farmacia_y.moMedicamentos:
            if farmacia_y.m[i].getTipo() == tipo:
```

```
        while i < farmacia_y.moMedicamentos:
            if farmacia_y.m[i].getTipo() == tipo:
                movidos.append(farmacia_y.m[i])
                for j in range(i, farmacia_y.moMedicamentos - 1):
                    farmacia_y.m[j] = farmacia_y.m[j + 1]
                farmacia_y.moMedicamentos -= 1
            else:
                i += 1
        for med in movidos:
            if farmacia_z.moMedicamentos < 100:
                farmacia_z.m[farmacia_z.moMedicamentos] = med
                farmacia_z.moMedicamentos += 1
        self.guardar()
    def guardar(self):
        with open(self.na, 'wb') as f:
            pickle.dump(self.farmacias, f)
    def cargar(self):
        if os.path.exists(self.na):
            with open(self.na, 'rb') as f:
                self.farmacias = pickle.load(f)

if __name__ == "__main__":
    arch = ArchFarmacia()
    arch.crearArchivo()
    arch.a(1)
    arch.b()
    arch.c("Resfrio")
    arch.d()
    arch.e("Tos", 1, 2)
```

Ejercicio 6

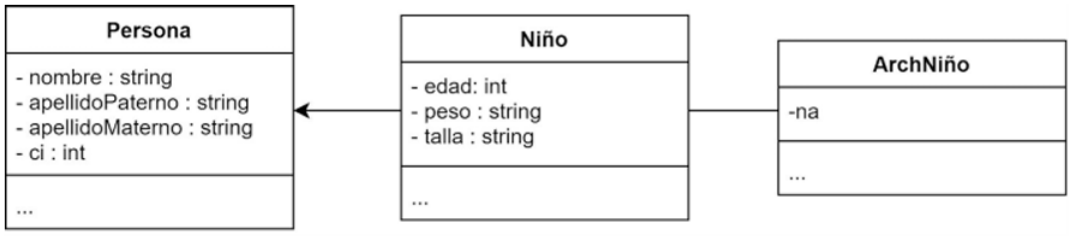


```python
import pickle
import os
from datetime import datetime

class Libro:
    def __init__(self, codLibro=0, titulo="", precio=0.0):
        self.codLibro = codLibro
        self.titulo = titulo
        self.precio = precio
    def leer(self):
        self.codLibro = int(input("Cod libro: "))
        self.titulo = input("Titulo: ")
        self.precio = float(input("Precio: "))
    def mostrar(self):
        print(f"{self.codLibro} {self.titulo} {self.precio}")

class Prestamo:                                                    (parameter) cantidad: int
    def __init__(self, codCliente=0, codLibro=0, fechaPrestamo="", cantidad=0):
        self.codCliente = codCliente
        self.codLibro = codLibro
        self.fechaPrestamo = fechaPrestamo
        self.cantidad = cantidad
    def leer(self):
        self.codCliente = int(input("Cod cliente: "))
        self.codLibro = int(input("Cod libro: "))
        self.fechaPrestamo = input("Fecha: ")
        self.cantidad = int(input("Cantidad: "))
    def mostrar(self):
        print(f"{self.codCliente} {self.codLibro} {self.fechaPrestamo} {self.cantidad}")

class Cliente:
    def __init__(self, codCliente=0, ci="", nombre="", apellido=""):
        self.codCliente = codCliente
        self.ci = ci
        self.nombre = nombre
        self.apellido = apellido
    def leer(self):
        self.codCliente = int(input("Cod cliente: "))
        self.ci = input("CI: ")
        self.nombre = input("Nombre: ")
        self.apellido = input("Apellido: ")
    def mostrar(self):
        print(f"{self.codCliente} {self.ci} {self.nombre} {self.apellido}")

class ArchLibro:
    def __init__(self, nomArch="libros.dat"):
        self.nomArch = nomArch
        self.libros = []
        self.cargar()
    def crearArchivo(self):
        n = int(input("Cant libros: "))
        for _ in range(n):
            l = Libro()
            l.leer()
            self.libros.append(l)
        self.guardar()
    def adicionar(self):
        l = Libro()
        l.leer()
```

```
class ArchLibro:
    def adicionar(self):
        self.libros.append(l)
        self.guardar()
    def listar(self):
        for l in self.libros:
            l.mostrar()
    def a(self, x, y):
        for l in self.libros:
            if x <= l.precio <= y:
                l.mostrar()
    def c(self, archPrestamo):
        prestamos_cod = set(p.codLibro for p in archPrestamo.prestamos)
        for l in self.libros:
            if l.codLibro not in prestamos_cod:
                l.mostrar()
    def guardar(self):
        with open(self.nomArch, 'wb') as f:
            pickle.dump(self.libros, f)
    def cargar(self):
        if os.path.exists(self.nomArch):
            with open(self.nomArch, 'rb') as f:
                self.libros = pickle.load(f)

class ArchPrestamo:
    def __init__(self, nomArch="prestamos.dat"):
        self.nomArch = nomArch
        self.prestamos = []
        self.cargar()
    def crearArchivo(self):
        n = int(input("Cant prestamos: "))
        for _ in range(n):
            p = Prestamo()
            p.leer()
            self.prestamos.append(p)
        self.guardar()
    def adicionar(self):
        p = Prestamo()
        p.leer()
        self.prestamos.append(p)
        self.guardar()
    def listar(self):
        for p in self.prestamos:
            p.mostrar()
    def b(self, codLibro, archLibro):
        libro = next((l for l in archLibro.libros if l.codLibro == codLibro), None)
        if libro:
            total = sum(p.cantidad * libro.precio for p in self.prestamos if p.codLibro == codLibro)
            print(f"Ingreso total: {total}")
    def d(self, codLibro, archCliente):
        clientes_cod = set(p.codCliente for p in self.prestamos if p.codLibro == codLibro)
        for c in archCliente.clientes:
            if c.codCliente in clientes_cod:
                c.mostrar()
    def e(self):
        conteo = {}
        for p in self.prestamos:
            conteo[p.codLibro] = conteo.get(p.codLibro, 0) + p.cantidad
        if conteo:
```

```
82  class ArchPrestamo:
112     def e(self):
114         for p in self.prestamos:
115             conteo[p.codLibro] = conteo.get(p.codLibro, 0) + p.cantidad
116         if conteo:
117             max_libro = max(conteo, key=conteo.get)
118             print(f"Libro mas prestado: {max_libro}")
119     def f(self):
120         conteo = {}
121         for p in self.prestamos:
122             conteo[p.codCliente] = conteo.get(p.codCliente, 0) + 1
123         if conteo:
124             max_cliente = max(conteo, key=conteo.get)
125             print(f"Cliente con mas prestamos: {max_cliente}")
126     def guardar(self):
127         with open(self.nomArch, 'wb') as f:
128             pickle.dump(self.prestamos, f)
129     def cargar(self):
130         if os.path.exists(self.nomArch):
131             with open(self.nomArch, 'rb') as f:
132                 self.prestamos = pickle.load(f)
133
134 class ArchCliente:
135     def __init__(self, nomArch="clientes.dat"):
136         self.nomArch = nomArch
137         self.clientes = []
138         self.cargar()
139     def crearArchivo(self):
140         n = int(input("Cant clientes: "))
141         for _ in range(n):
142             c = Cliente()
143             c.leer()
144             self.clientes.append(c)
145         self.guardar()
146     def adicionar(self):
147         c = Cliente()
148         c.leer()
149         self.clientes.append(c)
150         self.guardar()
151     def listar(self):
152         for c in self.clientes:
153             c.mostrar()
154     def guardar(self):
155         with open(self.nomArch, 'wb') as f:
156             pickle.dump(self.clientes, f)
157     def cargar(self):
158         if os.path.exists(self.nomArch):
159             with open(self.nomArch, 'rb') as f:
160                 self.clientes = pickle.load(f)
161
162 if __name__ == "__main__":
163     archLibro = ArchLibro()
164     archPrestamo = ArchPrestamo()
165     archCliente = ArchCliente()
166     archLibro.a(10, 50)
167     archPrestamo.b(101, archLibro)
168     archLibro.c(archPrestamo)
169     archPrestamo.d(101, archCliente)
170     archPrestamo.e()
```

Ejercicio 7



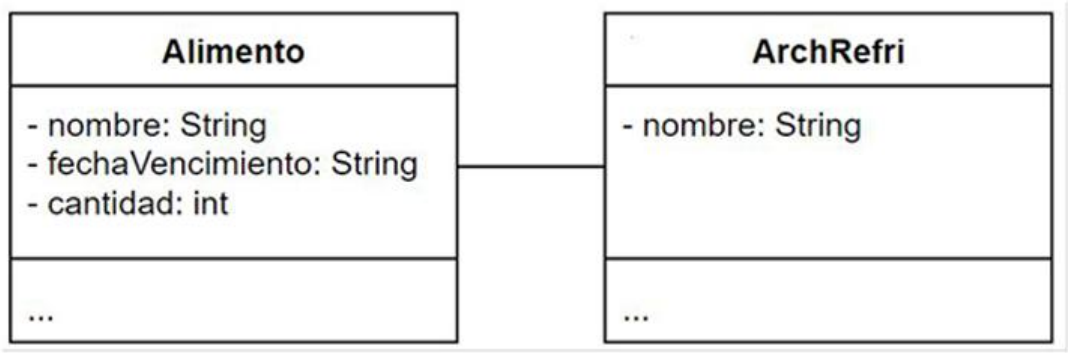

```python
import pickle
import os
from datetime import datetime

class Libro:
    def __init__(self, codLibro=0, titulo="", precio=0.0):
        self.codLibro = codLibro
        self.titulo = titulo
        self.precio = precio
    def leer(self):
        self.codLibro = int(input("Cod libro: "))
        self.titulo = input("Titulo: ")
        self.precio = float(input("Precio: "))
    def mostrar(self):
        print(f"{self.codLibro} {self.titulo} {self.precio}")

class Prestamo:
    def __init__(self, codCliente=0, codLibro=0, fechaPrestamo="", cantidad=0):
        self.codCliente = codCliente
        self.codLibro = codLibro
        self.fechaPrestamo = fechaPrestamo
        self.cantidad = cantidad
    def leer(self):
        self.codCliente = int(input("Cod cliente: "))
        self.codLibro = int(input("Cod libro: "))
        self.fechaPrestamo = input("Fecha: ")
        self.cantidad = int(input("Cantidad: "))
    def mostrar(self):
        print(f"{self.codCliente} {self.codLibro} {self.fechaPrestamo} {self.cantidad}")

class Cliente:
    def __init__(self, codCliente=0, ci="", nombre="", apellido=""):
        self.codCliente = codCliente
        self.ci = ci
        self.nombre = nombre
        self.apellido = apellido
    def leer(self):
        self.codCliente = int(input("Cod cliente: "))
        self.ci = input("CI: ")
        self.nombre = input("Nombre: ")
        self.apellido = input("Apellido: ")
    def mostrar(self):
        print(f"{self.codCliente} {self.ci} {self.nombre} {self.apellido}")

class ArchLibro:
    def __init__(self, nomArch="libros.dat"):
        self.nomArch = nomArch
        self.libros = []
        self.cargar()
    def crearArchivo(self):
```

```
    def crearArchivo(self):
        n = int(input("Cant libros: "))
        for _ in range(n):
            l = Libro()
            l.leer()
            self.libros.append(l)
        self.guardar()
    def adicionar(self):
        l = Libro()
        l.leer()
        self.libros.append(l)
        self.guardar()
    def listar(self):
        for l in self.libros:
            l.mostrar()
    def a(self, x, y):
        for l in self.libros:
            if x <= l.precio <= y:
                l.mostrar()
    def c(self, archPrestamo):
        prestamos_cod = set(p.codLibro for p in archPrestamo.prestamos)
        for l in self.libros:
            if l.codLibro not in prestamos_cod:
                l.mostrar()
    def guardar(self):
        with open(self.nomArch, 'wb') as f:
            pickle.dump(self.libros, f)
    def cargar(self):
        if os.path.exists(self.nomArch):
            with open(self.nomArch, 'rb') as f:
                self.libros = pickle.load(f)

class ArchPrestamo:
    def __init__(self, nomArch="prestamos.dat"):
        self.nomArch = nomArch
        self.prestamos = []
        self.cargar()
    def crearArchivo(self):
        n = int(input("Cant prestamos: "))
        for _ in range(n):
            p = Prestamo()
            p.leer()
            self.prestamos.append(p)
        self.guardar()
    def adicionar(self):
        p = Prestamo()
        p.leer()
        self.prestamos.append(p)
        self.guardar()
    def listar(self):
        for p in self.prestamos:
            p.mostrar()
    def b(self, codLibro, archLibro):
        libro = next((l for l in archLibro.libros if l.codLibro == codLibro), None)
        if libro:
            total = sum(p.cantidad * libro.precio for p in self.prestamos if p.codLibro == codLibro)
            print(f"Ingreso total: {total}")
    def d(self, codLibro, archCliente):
```

```
    def d(self, codLibro, archCliente):
        clientes_cod = set(p.codCliente for p in self.prestamos if p.codLibro == codLibro)
        for c in archCliente.clientes:
            if c.codCliente in clientes_cod:
                c.mostrar()
    def e(self):
        conteo = {}
        for p in self.prestamos:
            conteo[p.codLibro] = conteo.get(p.codLibro, 0) + p.cantidad
        if conteo:
            max_libro = max(conteo, key=conteo.get)
            print(f"Libro mas prestado: {max_libro}")
    def f(self):
        conteo = {}
        for p in self.prestamos:
```

```
    def f(self):
        conteo = {}
        for p in self.prestamos:
            conteo[p.codCliente] = conteo.get(p.codCliente, 0) + 1
        if conteo:
            max_cliente = max(conteo, key=conteo.get)
            print(f"Cliente con mas prestamos: {max_cliente}")
    def guardar(self):
        with open(self.nomArch, 'wb') as f:
            pickle.dump(self.prestamos, f)
    def cargar(self):
        if os.path.exists(self.nomArch):
            with open (variable) prestamos: Any
                self.prestamos = pickle.load(f)

class ArchCliente:
    def __init__(self, nomArch="clientes.dat"):
        self.nomArch = nomArch
        self.clientes = []
        self.cargar()
    def crearArchivo(self):
        n = int(input("Cant clientes: "))
        for _ in range(n):
            c = Cliente()
            c.leer()
            self.clientes.append(c)
        self.guardar()
    def adicionar(self):
        c = Cliente()
        c.leer()
        self.clientes.append(c)
        self.guardar()
    def listar(self):
        for c in self.clientes:
            c.mostrar()
    def guardar(self):
        with open(self.nomArch, 'wb') as f:
            pickle.dump(self.clientes, f)
    def cargar(self):
        if os.path.exists(self.nomArch):
            with open(self.nomArch, 'rb') as f:
                self.clientes = pickle.load(f)

if __name__ == "__main__":
    archLibro = ArchLibro()
    archPrestamo = ArchPrestamo()
    archCliente = ArchCliente()
    archLibro.a(10, 50)
    archPrestamo.b(101, archLibro)
    archLibro.c(archPrestamo)
    archPrestamo.d(101, archCliente)
    archPrestamo.e()
    archPrestamo.f()
```

Ejercicio 8


Alimento
- nombre: String
- fechaVencimiento: String
- cantidad: int
...
ArchRefri
- nombre: String
...

```python
import pickle
import os
from datetime import datetime

class Alimento:
    def __init__(self, nombre="",
        self.nombre = nombre
        self.fecha_vencimiento =
        self.cantidad = cantidad
    def leer(self):
        self.nombre = input("Nomb
        self.fecha_vencimiento = input("Fecha vencimiento (dd/mm/yyyy): ")
        self.cantidad = int(input("Cantidad: "))
    def mostrar(self):
        print(f"{self.nombre} {self.fecha_vencimiento} {self.cantidad}")
    def esta_vencido(self):
        try:
            fecha_venc = datetime.strptime(self.fecha_vencimiento, "%d/%m/%Y")
            return fecha_venc < datetime.now()
        except:
            return False
    def caduca_antes_de(self, fecha_x):
        try:
            fecha_venc = datetime.strptime(self.fecha_vencimiento, "%d/%m/%Y")
            fecha_comp = datetime.strptime(fecha_x, "%d/%m/%Y")
            return fecha_venc < fecha_comp
        except:
            return False

class ArchRefri:
    def __init__(self, nombre="refri.dat"):
        self.nombre = nombre
        self.alimentos = []
        self.cargar()
    def crearArchivo(self):
        n = int(input("Cant alimentos: "))
        for _ in range(n):
            a = Alimento()
            a.leer()
            self.alimentos.append(a)
        self.guardar()
    def modificar_por_nombre(self, nombre):
        for a in self.alimentos:
            if a.nombre == nombre:
                print("Nuevos datos:")
                a.leer()
                self.guardar()
                return
        print("No encontrado")
    def eliminar_por_nombre(self, nombre):
        self.alimentos = [a for a in self.alimentos if a.nombre != nombre]
        self.guardar()
    def b(self, fecha_x):
        for a in self.alimentos:
            if a.caduca_antes_de(fecha_x):
                a.mostrar()
    def c(self):
        self.alimentos = [a for a in self.alimentos if a.cantidad > 0]
        self.guardar()
```

```
(function) def input(
    prompt: object = "",
    /
) -> str
```

Read a string from standard input. The trailing newline is stripped.

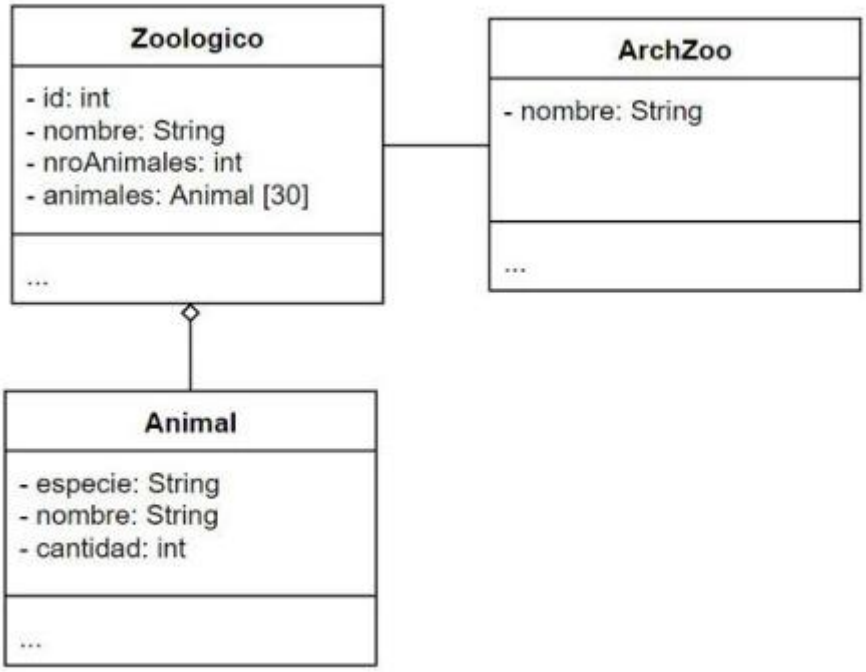
The prompt string, if given, is printed to standard output without a trailing newline before reading input.

If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError. On *nix systems, readline is used if available.

```python
import pickle
import os
from datetime import datetime

class Alimento:
    def __init__(self, nombre="", fecha_vencimiento="", cantidad=0):
        self.nombre = nombre
        self.fecha_vencimiento = fecha_vencimiento
        self.cantidad = cantidad
    def leer(self):
        self.nombre = input("Nombre: ")
        self.fecha_vencimiento = input("Fecha vencimiento (dd/mm/yyyy): ")
        self.cantidad = int(input("Cantidad: "))
    def mostrar(self):
        print(f"{self.nombre} {self.fecha_vencimiento} {self.cantidad}")
    def esta_vencido(self):
        try:
            fecha_venc = datetime.strptime(self.fecha_vencimiento, "%d/%m/%Y")
            return fecha_venc < datetime.now()
        except:
            return False
    def caduca_antes_de(self, fecha_x):
        try:
            fecha_venc = datetime.strptime(self.fecha_vencimiento, "%d/%m/%Y")
            fecha_comp = datetime.strptime(fecha_x, "%d/%m/%Y")
            return fecha_venc < fecha_comp
        except:
            return False

class ArchRefri:
    def __init__(self, nombre="refri.dat"):
        self.nombre = nombre
        self.alimentos = []
        self.cargar()
    def crearArchivo(self):
        n = int(input("Cant alimentos: "))
        for _ in range(n):
            a = Alimento()
            a.leer()
            self.alimentos.append(a)
        self.guardar()
    def modificar_por_nombre(self, nombre):
        for a in self.alimentos:
            if a.nombre == nombre:
                print("Nuevos datos:")
                a.leer()
                self.guardar()
                return
```

```python
class ArchRefri:
    def crearArchivo(self):
        n = int(input("Cant alimentos: "))
        for _ in range(n):
            a = Alimento()
            a.leer()
            self.alimentos.append(a)
        self.guardar()
    def modificar_por_nombre(self, nombre):
        for a in self.alimentos:
            if a.nombre == nombre:
                print("Nuevos datos:")
                a.leer()
                self.guardar()
                return
        print("No encontrado")
    def eliminar_por_nombre(self, nombre):
        self.alimentos = [a for a in self.alimentos if a.nombre != nombre]
        self.guardar()
    def b(self, fecha_x):
        for a in self.alimentos:
            if a.caduca_antes_de(fecha_x):
                a.mostrar()
    def c(self):
        self.alimentos = [a for a in self.alimentos if a.cantidad > 0]
        self.guardar()
    def d(self):
        for a in self.alimentos:
            if a.esta_vencido():
                a.mostrar()
    def e(self):
        if not self.alimentos:
            return
        max_alimento = max(self.alimentos, key=lambda x: x.cantidad)
        max_alimento.mostrar()
    def listar(self):
        for a in self.alimentos:
            a.mostrar()
    def guardar(self):
        with open(self.nombre, 'wb') as f:
            pickle.dump(self.alimentos, f)
    def cargar(self):
        if os.path.exists(self.nombre):
            with open(self.nombre, 'rb') as f:
                self.alimentos = pickle.load(f)

if __name__ == "__main__":
    arch = ArchRefri()
    arch.crearArchivo()
    arch.modificar_por_nombre("Leche")
    arch.eliminar_por_nombre("Pan")
    arch.b("01/01/2024")
    arch.c()
    arch.d()
    arch.e()
```

Ejercicio 9

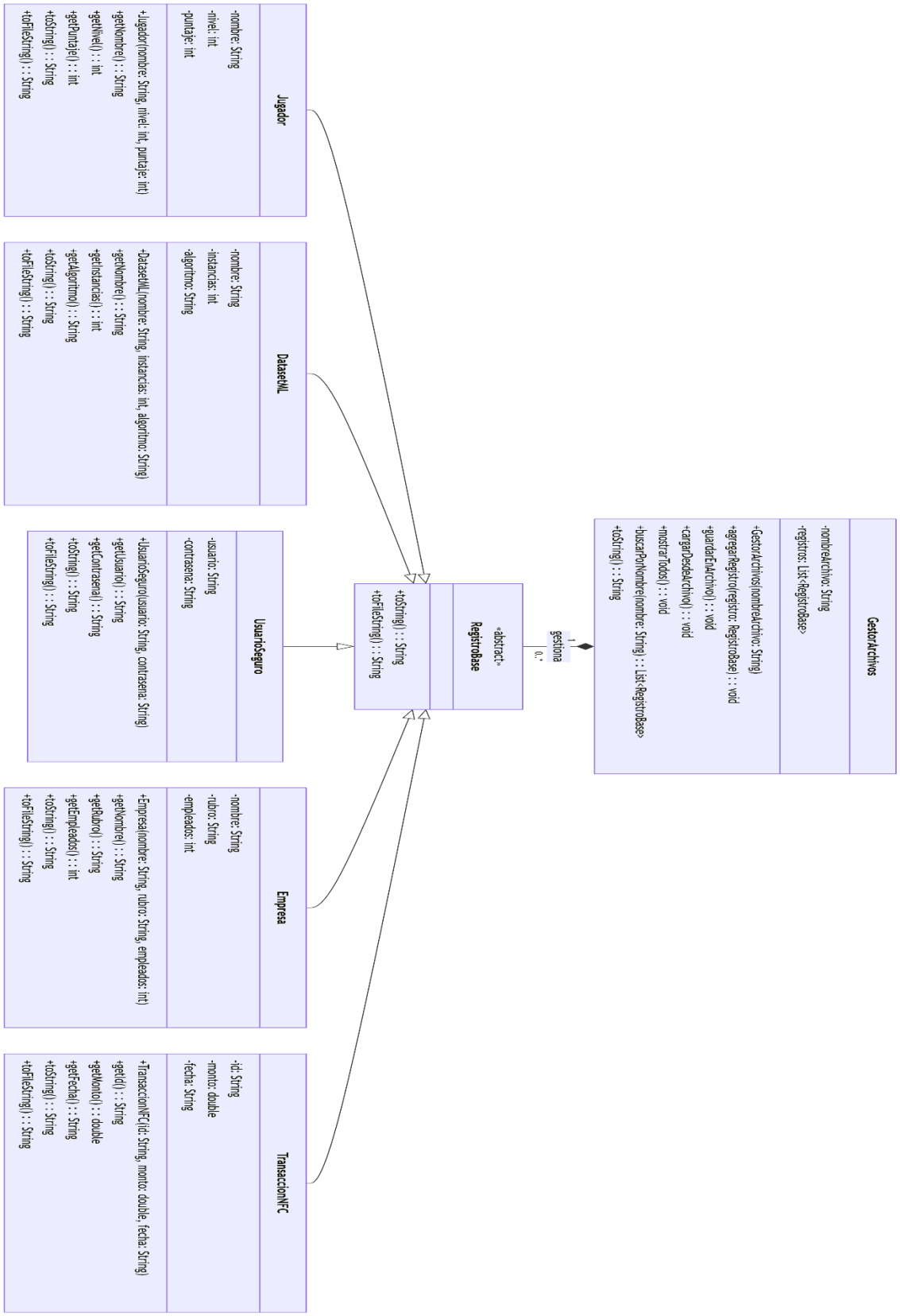



```python
import pickle
import os

class Animal:
    def __init__(self, especie="", nombre="", cantidad=0):
        self.especie = especie
        self.nombre = nombre
        self.cantidad = cantidad
    def leer(self):
        self.especie = input("Especie: ")
        self.nombre = input("Nombre: ")
        self.cantidad = int(input("Cantidad: "))
    def mostrar(self):
        print(f"{self.especie} {self.nombre} {self.cantidad}")

class Zoologico:
    def __init__(self, id=0, nombre="", nroAnimales=0):
        self.id = id
        self. (variable) nroAnimales: int
        self.nroAnimales = nroAnimales
        self.animales = [Animal() for _ in range(30)]
    def leer(self):
        self.id = int(input("ID: "))
        self.nombre = input("Nombre: ")
        self.nroAnimales = int(input("Nro animales: "))
        for i in range(self.nroAnimales):
            self.animales[i].leer()
    def mostrar(self):
        print(f"{self.id} {self.nombre} {self.nroAnimales}")
        for i in range(self.nroAnimales):
            self.animales[i].mostrar()
    def variedad_animales(self):
        especies = set()
        for i in range(self.nroAnimales):
            especies.add(self.animales[i].especie)
        return len(especies)
    def esta_vacio(self):
        return self.nroAnimales == 0
    def animales_especie_x(self, especie_x):
        for i in range(self.nroAnimales):
            if self.animales[i].especie == especie_x:
                self.animales[i].mostrar()
```

```
39      def animales_especie_x(self, especie_x):
42                  self.animales[i].mostrar()
43      def agregar_animal(self, animal):
44          if self.nroAnimales < 30:
45              self.animales[self.nroAnimales] = animal
46              self.nroAnimales += 1
47      def quitar_animales_especie(self, especie_x):
48          i = 0
49          removidos = []
50          while i < self.nroAnimales:
51              if self.animales[i].especie == especie_x:
52                  removidos.append(self.animales[i])
53                  for j in range(i, self.nroAnimales - 1):
54                      self.animales[j] = self.animales[j + 1]
55                  self.nroAnimales -= 1
56              else:
57                  i += 1
58          return removidos
59
60  class ArchZoo:
61      def __init__(self, nombre="zoologicos.dat"):
62          self.nombre = nombre
63          self.zoos = []
64          self.cargar()
65      def crearArchivo(self):
66          n = int(input("Cant zoologicos: "))
67          for _ in range(n):
68              z = Zoologico()
69              z.leer()
70              self.zoos.append(z)
71          self.guardar()
72      def modificar(self, id_zoo):
73          for z in self.zoos:
74              if z.id == id_zoo:
75                  z.leer()
76                  self.guardar()
77                  return
78          print("No encontrado")
79      def eliminar(self, id_zoo):
80          self.zoos = [z for z in self.zoos if z.id != id_zoo]
81          self.guardar()
82      def b(self):
83          if not self.zoos:
84              return
85          max_variedad = max(z.variedad_animales() for z in self.zoos)
86          for z in self.zoos:
87              if z.variedad_animales() == max_variedad:
88                  z.mostrar()
89      def c(self):
90          vacios = [z for z in self.zoos if z.esta_vacio()]
91          for z in vacios:
92              z.mostrar()
93          self.zoos = [z for z in self.zoos if not z.esta_vacio()]
94          self.guardar()
95      def d(self, especie_x):
96          for z in self.zoos:
97              z.animales_especie_x(especie_x)
98      def e(self, id_x, id_y, especie_x):
```

```python
    def e(self, id_x, id_y, especie_x):
        zoo_x = next((z for z in self.zoos if z.id == id_x), None)
        zoo_y = next((z for z in self.zoos if z.id == id_y), None)
        if not zoo_x or not zoo_y:
            return
        animales_mover = zoo_x.quitar_animales_especie(especie_x)
        for animal in animales_mover:
            zoo_y.agregar_animal(animal)
        self.guardar()
    def listar(self):
        for z in self.zoos:
            z.mostrar()
    def guardar(self):
        with open(self.nombre, 'wb') as f:
            pickle.dump(self.zoos, f)
    def cargar(self):
        if os.path.exists(self.nombre):
            with open(self.nombre, 'rb') as f:
                self.zoos = pickle.load(f)

if __name__ == "__main__":
    arch = ArchZoo()
    arch.crearArchivo()
    arch.modificar(1)
    arch.eliminar(2)
    arch.b()
    arch.c()
    arch.d("Leon")
    arch.e(1, 3, "Tigre")
```

Ejercicio 10



```java
package Diez;
import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class CobrosNFC {
    private List<Transaccion> transacciones;
    private static final String ARCHIVO = "cobros_nfc.txt";

    public CobrosNFC() {
        transacciones = new ArrayList<>();
        cargarTransacciones();
    }

    public void agregarTransaccion(Transaccion transaccion) {
        transacciones.add(transaccion);
        guardarTransacciones();
    }

    public void mostrarTransacciones() {
        for (Transaccion t : transacciones) {
            System.out.println(t);
        }
    }

    public Transaccion buscarPorID(String id) {
        for (Transaccion t : transacciones) {
            if (t.getId().equalsIgnoreCase(id)) {
                return t;
            }
        }
        return null;
    }

    private void guardarTransacciones() {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(ARCHIVO))) {
            for (Transaccion t : transacciones) {
                writer.write(t.getId() + "," + t.getMonto() + "," + t.getFecha());
                writer.newLine();
            }
        } catch (IOException e) {
            System.err.println("Error al guardar transacciones: " + e.getMessage());
        }
    }

    private void cargarTransacciones() {
        File archivo = new File(ARCHIVO);
        if (!archivo.exists()) {
            return;
        }

        try (BufferedReader reader = new BufferedReader(new FileReader(ARCHIVO))) {
            String linea;
            while ((linea = reader.readLine()) != null) {
                String[] datos = linea.split(regex: ",");
                if (datos.length == 3) {
                    String id = datos[0];
                    double monto = Double.parseDouble(datos[1]);
                    String fecha = datos[2];
                    transacciones.add(new Transaccion(id, monto, fecha));
                }
            }
        } catch (IOException e) {
            System.err.println("Error al cargar transacciones: " + e.getMessage());
        }
    }
}
```

```java
package Diez;

public class Dispositivo {
    private String mac;
    private String nombre;
    private int velocidad;

    public Dispositivo(String mac, String nombre, int velocidad) {
        this.mac = mac;
        this.nombre = nombre;
        this.velocidad = velocidad;
    }

    public String getMac() {
        return mac;
    }

    public String getNombre() {
        return nombre;
    }

    public int getVelocidad() {
        return velocidad;
    }

    @Override
    public String toString() {
        return mac + " - " + nombre + " - " + velocidad + " Mbps";
    }
}
```

```java
package Diez;
import java.util.Scanner;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        RedWifi redWifi = new RedWifi();
        CobrosNFC cobrosNFC = new CobrosNFC();

        while (true) {
            System.out.println(x: "\n=== MENÚ PRINCIPAL ===");
            System.out.println(x: "1. Gestión de Red WiFi");
            System.out.println(x: "2. Gestión de Cobros NFC");
            System.out.println(x: "3. Salir");
            System.out.print(s: "Seleccione una opción: ");

            int opcionPrincipal = scanner.nextInt();
            scanner.nextLine();

            switch (opcionPrincipal) {
                case 1 -> gestionRedWifi(scanner, redWifi);
                case 2 -> gestionCobrosNFC(scanner, cobrosNFC);
                case 3 -> {
                    System.out.println(x: "Saliendo...");
                    scanner.close();
                    System.exit(status: 0);
                }
                default -> System.out.println(x: "Opción no válida");
            }
        }
    }
```

```
    }

    private static void gestionRedWifi(Scanner scanner, RedWifi redWifi) {
        while (true) {
            System.out.println(x: "\n=== GESTIÓN DE RED WiFi ===");
            System.out.println(x: "1. Agregar dispositivo");
            System.out.println(x: "2. Mostrar todos los dispositivos");
            System.out.println(x: "3. Buscar dispositivo por MAC");
            System.out.println(x: "4. Volver al menú principal");
            System.out.print(s: "Seleccione una opción: ");

            int opcion = scanner.nextInt();
            scanner.nextLine();

            switch (opcion) {
                case 1 -> {
                    System.out.print(s: "Ingrese dirección MAC: ");
                    String mac = scanner.nextLine();
                    System.out.print(s: "Ingrese nombre del dispositivo: ");
                    String nombre = scanner.nextLine();
                    System.out.print(s: "Ingrese velocidad de conexión (Mbps): ");
                    int velocidad = scanner.nextInt();
                    scanner.nextLine();

                    Dispositivo dispositivo = new Dispositivo(mac, nombre, velocidad);
                    redWifi.agregarDispositivo(dispositivo);
                    System.out.println(x: "Dispositivo agregado correctamente.");
                }
                case 2 -> {
                    System.out.println(x: "\n=== DISPOSITIVOS CONECTADOS ===");
                    redWifi.mostrarDispositivos();
                }
                case 3 -> {
                    System.out.print(s: "Ingrese dirección MAC a buscar: ");
                    String macBuscar = scanner.nextLine();
                    Dispositivo encontrado = redWifi.buscarPorMAC(macBuscar);
                    if (encontrado != null) {
                        System.out.println("Dispositivo encontrado: " + encontrado);
                    } else {
                        System.out.println(x: "No se encontró dispositivo con esa MAC.");
                    }
                }
                case 4 -> {
                    return;
                }
                default -> System.out.println(x: "Opción no válida");
            }
        }
    }

    private static void gestionCobrosNFC(Scanner scanner, CobrosNFC cobrosNFC) {
        while (true) {
            System.out.println(x: "\n=== GESTIÓN DE COBROS NFC ===");
            System.out.println(x: "1. Agregar transacción");
            System.out.println(x: "2. Mostrar todas las transacciones");
            System.out.println(x: "3. Buscar transacción por ID");
            System.out.println(x: "4. Volver al menú principal");
            System.out.print(s: "Seleccione una opción: ");

            int opcion = scanner.nextInt();
            scanner.nextLine();

            switch (opcion) {
                case 1 -> {
                    System.out.print(s: "Ingrese ID de transacción: ");
                    String id = scanner.nextLine();
                    System.out.print(s: "Ingrese monto: ");
                    double monto = scanner.nextDouble();
                    scanner.nextLine();
                    System.out.print(s: "Ingrese fecha (dd/mm/aaaa): ");
```

```
100                     System.out.print(s: "Ingrese fecha (dd/mm/aaaa): ");
101                     String fecha = scanner.nextLine();
102
103                     Transaccion transaccion = new Transaccion(id, monto, fecha);
104                     cobrosNFC.agregarTransaccion(transaccion);
105                     System.out.println(x: "Transacción agregada correctamente.");
106                 }
107                 case 2 -> {
108                     System.out.println(x: "\n=== TODAS LAS TRANSACCIONES ===");
109                     cobrosNFC.mostrarTransacciones();
110                 }
111                 case 3 -> {
112                     System.out.print(s: "Ingrese ID a buscar: ");
113                     String idBuscar = scanner.nextLine();
114                     Transaccion encontrada = cobrosNFC.buscarPorID(idBuscar);
115                     if (encontrada != null) {
116                         System.out.println("Transacción encontrada: " + encontrada);
117                     } else {
118                         System.out.println(x: "No se encontró transacción con ese ID.");
119                     }
120                 }
121                 case 4 -> {
122                     return;
123                 }
124                 default -> System.out.println(x: "Opción no válida");
125             }
126         }
127     }
128 }
129
```

```
1  package Diez;
2  // Archivo: RedWifi.java ···
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class RedWifi {
8      private List<Dispositivo> dispositivos;
9      private static final String ARCHIVO = "dispositivos_wifi.txt";
10
11     publi
12         void Diez.RedWifi.cargarDispositivos()
13         cargarDispositivos();
14     }
15
16     public void agregarDispositivo(Dispositivo dispositivo) {
17         dispositivos.add(dispositivo);
18         guardarDispositivos();
19     }
20
21     public void mostrarDispositivos() {
22         for (Dispositivo d : dispositivos) {
23             System.out.println(d);
24         }
25     }
26
27     public Dispositivo buscarPorMAC(String mac) {
28         for (Dispositivo d : dispositivos) {
29             if (d.getMac().equalsIgnoreCase(mac)) {
30                 return d;
31             }
32         }
33         return null;
```

```
        return null;
    }

    private void guardarDispositivos() {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(ARCHIVO))) {
            for (Dispositivo d : dispositivos) {
                writer.write(d.getMac() + "," + d.getNombre() + "," + d.getVelocidad());
                writer.newLine();
            }
        } catch (IOException e) {
            System.err.println("Error al guardar dispositivos: " + e.getMessage());
        }
    }

    private void cargarDispositivos() {
        File archivo = new File(ARCHIVO);
        if (!archivo.exists()) {
            return;
        }

        try (BufferedReader reader = new BufferedReader(new FileReader(ARCHIVO))) {
            String linea;
            while ((linea = reader.readLine()) != null) {
                String[] datos = linea.split(regex: ",");
                if (datos.length == 3) {
                    String mac = datos[0];
                    String nombre = datos[1];
                    int velocidad = Integer.parseInt(datos[2]);
                    dispositivos.add(new Dispositivo(mac, nombre, velocidad));
                }
            }
        } catch (IOException e) {
            System.err.println("Error al cargar dispositivos: " + e.getMessage());
        }
    }
}
```

Diez > J Transaccion.java > ...

```
package Diez;

public class Transaccion {
    private String id;
    private double monto;
    private String fecha;

    public Transaccion(String id, double monto, String fecha) {
        this.id = id;
        this.monto = monto;
        this.fecha = fecha;
    }

    public String getId() {
        return id;
    }

    public double getMonto() {
        return monto;
    }

    public String getFecha() {
        return fecha;
    }

    @Override
    public String toString() {
        return "ID: " + id + " - Monto: $" + monto + " - Fecha: " + fecha;
    }
}
```