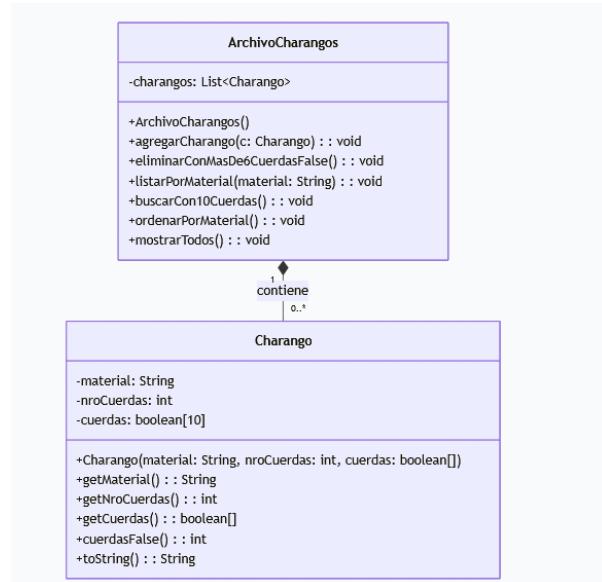


Practica N 3

Persistencia

Nombre: Carlos Angel Morales Cruz

Ejercicio1



```
9  public class Charango implements Serializable {
10      private static final long serialVersionUID = 1L;
11
12      private Integer id;
13      private String material;
14      private int numeroCuerdas;
15      private boolean[] cuerdas;
16      private LocalDateTime fechaCreacion;
17
18      public Charango() {
19          this.cuerdas = new boolean[10];
20          this.fechaCreacion = LocalDateTime.now();
21      }
22
23      public Charango(String material, int numeroCuerdas) {
24          this();
25          this.material = material;
26          setNumeroCuerdas(numeroCuerdas);
27          inicializarCuerdas();
28      }
29
30      public Charango(Integer id, String material, int numeroCuerdas, boolean[] cuerdas) {
31          this.id = id;
32          this.material = material;
33          setNumeroCuerdas(numeroCuerdas);
34          setCuerdas(cuerdas);
35          this.fechaCreacion = LocalDateTime.now();
36      }
37
38      private void inicializarCuerdas() {
39          for (int i = 0; i < numeroCuerdas && i < 10; i++) {
40              cuerdas[i] = true;
41          }
42      }
43  }
```

```
43     public int contarCuerdasFalsas() {
44         int contador = 0;
45         for (boolean cuerda : cuerdas) {
46             if (!cuerda) contador++;
47         }
48         return contador;
49     }
50
51
52     public boolean tieneMasDe6CuerdasFalsas() {
53         return contarCuerdasFalsas() > 6;
54     }
55
56     public boolean esDeMaterial(String material) {
57         return this.material.equalsIgnoreCase(material);
58     }
59
60     public boolean tiene10Cuerdas() {
61         return numeroCuerdas == 10;
62     }
63
64     public String getRepresentacionCuerdas() {
65         StringBuilder sb = new StringBuilder(str: "[" );
66         for (int i = 0; i < 10; i++) {
67             if (i < numeroCuerdas) {
68                 sb.append(cuerdas[i] ? "■" : "□");
69             } else {
70                 sb.append(str: " " );
71             }
72         }
73         sb.append(str: "]");
74         return sb.toString();
75     }
76
77     public Integer getId() {
78         return id;
79     }
80
81     public void setId(Integer id) {
82         this.id = id;
83     }
84
85     public String getMaterial() {
86         return material;
```

```

81     public void setId(Integer id) {
82         this.id = id;
83     }
84
85     public String getMaterial() {
86         return material;
87     }
88
89     public void setMaterial(String material) {
90         this.material = material;
91     }
92
93     public int getNumeroCuerdas() {
94         return numeroCuerdas;
95     }
96
97     public void setNumeroCuerdas(int numeroCuerdas) {
98         if (numeroCuerdas < 0 || numeroCuerdas > 10) {
99             throw new IllegalArgumentException(s: "Número de cuerdas inválido");
100        }
101        this.numeroCuerdas = numeroCuerdas;
102    }
103
104    public boolean[] getCuerdas() {
105        return Arrays.copyOf(cuerdas, newLength: 10);
106    }
107
108    public void setCuerdas(boolean[] cuerdas) {
109        if (cuerdas.length != 10) {
110            throw new IllegalArgumentException(s: "Array debe tener 10 elementos");
111        }
112        this.cuerdas = Arrays.copyOf(cuerdas, newLength: 10);
113    }
114
115    public LocalDateTime getFechaCreacion() {
116        return fechaCreacion;
117    }
118
119    @Override
120    public String toString() {
121        return String.format(format: "Charango[ID=%d, Material=%s, Cuerdas=%d, Falsas=%d] %s",
122                            id, material, numeroCuerdas, contarCuerdasFalsas(), getRepresentacionCuerdas());
123    }
124

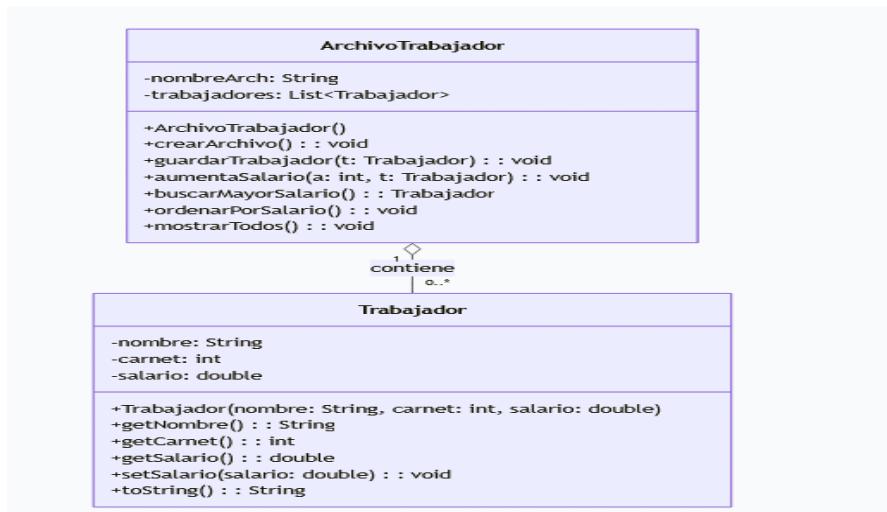
```

```

118
119    @Override
120    public String toString() {
121        return String.format(format: "Charango[ID=%d, Material=%s, Cuerdas=%d, Falsas=%d] %s",
122                            id, material, numeroCuerdas, contarCuerdasFalsas(), getRepresentacionCuerdas());
123    }
124
125    @Override
126    public boolean equals(Object obj) {
127        if (this == obj) return true;
128        if (obj == null || getClass() != obj.getClass()) return false;
129        Charango charango = (Charango) obj;
130        return id != null && id.equals(charango.id);
131    }
132
133    @Override
134    public int hashCode() {
135        return id != null ? id.hashCode() : 0;
136    }
137 }

```

Ejercicio 2



```
1 import pickle
2 import os
3 from typing import List
4
5 class Trabajador:
6     def __init__(self, nombre: str, carnet: int, salario: float):
7         (method) def aumentar_salario(
8             self: Self@Trabajador,
9             porcentaje: int
10        ) -> None
11     def aumentar_salario(self, porcentaje: int):
12         self.salario += self.salario * (porcentaje / 100)
13
14     def __str__(self):
15         return f"{self.carnet} | {self.nombre:20} | {self.salario:10.2f}"
16
17     def to_dict(self):
18         return {'nombre': self.nombre, 'carnet': self.carnet, 'salario': self.salario}
19
20     @classmethod
21     def from_dict(cls, data):
22         return cls(data['nombre'], data['carnet'], data['salario'])
23
24 class ArchivoTrabajador:
25     def __init__(self, archivo="trabajadores.dat"):
26         self.nombre_arch = archivo
27         self.trabajadores = []
28         self.cargar()
29
30     def crear_archivo(self):
31         try:
32             with open(self.nombre_arch, 'wb') as f:
33                 pickle.dump([], f)
34             return True
35         except:
36             return False
37
```

```

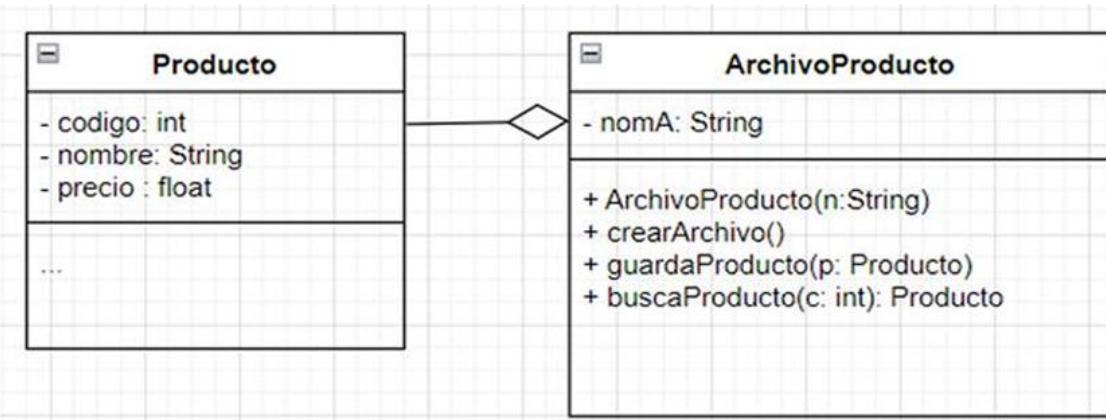
38     def guardar_trabajador(self, t):
39         self.trabajadores.append(t)
40         self.guardar()
41
42     def aumenta_salario(self, a, t):
43         if t in self.trabajadores:
44             t.aumentar_salario(a)
45             self.guardar()
46             return True
47         return False
48
49     def buscar_mayor_salario(self):
50         if not self.trabajadores:
51             return None
52         mayor = self.trabajadores[0]
53         for t in self.trabajadores[1:]:
54             if t.salario > mayor.salario:
55                 mayor = t
56         return mayor
57
58     def ordenar_por_salario(self):
59         return sorted(self.trabajadores, key=lambda x: x.salario)
60
61     def guardar(self):
62         try:
63             datos = [t.to_dict() for t in self.trabajadores]
64             with open(self.nombre_arch, 'wb') as f:
65                 pickle.dump(datos, f)
66         except:
67             pass
68
69     def cargar(self):
70         try:
71             if os.path.exists(self.nombre_arch):
72                 with open(self.nombre_arch, 'rb') as f:
73                     datos = pickle.load(f)
74                     self.trabajadores = [Trabajador.from_dict(d) for d in datos]
75         except:
76             self.trabajadores = []
77
78     def buscar_por_carnet(self, carnet):
79         for t in self.trabajadores:
80             if t.carnet == carnet:
81                 return t
82             ...
83
84         return None
85
86     def eliminar_trabajador(self, carnet):
87         for i, t in enumerate(self.trabajadores):
88             if t.carnet == carnet:
89                 del self.trabajadores[i]
90                 self.guardar()
91                 return True
92
93         return False

```

```
24  class ArchivoTrabajador:
25      def obtener_todos(self):
26          return self.trabajadores.copy()
27
28
29  class SistemaTrabajadores:
30      def __init__(self):
31          self.archivo = ArchivoTrabajador()
32
33
34      def iniciar(self):
35          while True:
36              print("\n" + "*50)
37              print("SISTEMA DE TRABAJADORES")
38              print("*50)
39              print("1. Crear archivo")
40              print("2. Mostrar todos")
41              print("3. Agregar trabajador")
42              print("4. Aumentar salario")
43              print("5. Mayor salario")
44              print("6. Ordenar por salario")
45              print("7. Eliminar trabajador")
46              print("8. Buscar por carnet")
47              print("9. Generar ejemplos")
48              print("0. Salir")
49              print("*50)
50
51              op = input("Opción: ").strip()
52
53              if op == "0":
54                  print("¡Adiós!")
55                  break
56              elif op == "1":
57                  self.crear_archivo()
58              elif op == "2":
59                  self.mostrar_todos()
60              elif op == "3":
61                  self.agregar_trabajador()
62              elif op == "4":
```

```
138             self.agregar_trabajador())
139         elif op == "4":
140             self.aumentar_salario()
141         elif op == "5":
142             self.mayor_salario()
143         elif op == "6":
144             self.ordenar_salario()
145         elif op == "7":
146             self.eliminar_trabajador()
147         elif op == "8":
148             self.buscar_carnet()
149         elif op == "9":
150             self.generar_ejemplos()
151
152     def crear_archivo(self):
153         if self.archivo.crear_archivo():
154             print("✓ Archivo creado")
155         else:
156             print("✗ Error al crear archivo")
157
158     def mostrar.todos(self):
159         trabajadores = self.archivo.obtener.todos()
160         if not trabajadores:
161             print("No hay trabajadores")
162             return
163
164         print("\n" + "-"*50)
165         print(f"{'Carnet':<8} {'Nombre':<20} {'Salario':<10}")
166         print("-"*50)
167         for t in trabajadores:
168             print(t)
169         print("-"*50)
170         print(f"Total: {len(trabajadores)}")
171
172     def agregar_trabajador(self):
173         try:
174             nombre = input("Nombre: ").strip()
175             if not nombre:
176                 print("Nombre requerido")
177                 return
178
179             carnet = int(input("Carnet: "))
180             if carnet <= 0:
181                 print("Carnet debe ser positivo")
182                 return
183
184             salario = float(input("Salario: "))
```

Ejercicio 3



```
1  import pickle
2  import os
3
4  class Producto:
5      def __init__(self, codigo, nombre, precio):
6          self.codigo = codigo
7          self.nombre = nombre
8          self.precio = precio
9
10     def __str__(self):
11         return f"{self.codigo} | {self.nombre:20} | ${self.precio:10.2f}"
12
13 class ArchivoProducto:
14     def __init__(self, nomA="productos.dat"):
15         self.nomA = nomA
16         self.productos = []
17         self.cargar()
18
19     def crearArchivo(self):
20         try:
21             with open(self.nomA, 'wb') as f:
22                 pickle.dump([], f)
23             return True
24         except:
25             return False
26
27     def guardaProducto(self, p):
28         self.productos.append(p)
29         self.guardar()
30
31     def buscaProducto(self, c):
32         for p in self.productos:
33             if p.codigo == c:
34                 return p
35         return None
36
37     def promedioPrecios(self):
38         if not self.productos:
39             return 0
40         total = sum(p.precio for p in self.productos)
41         return total / len(self.productos)
```

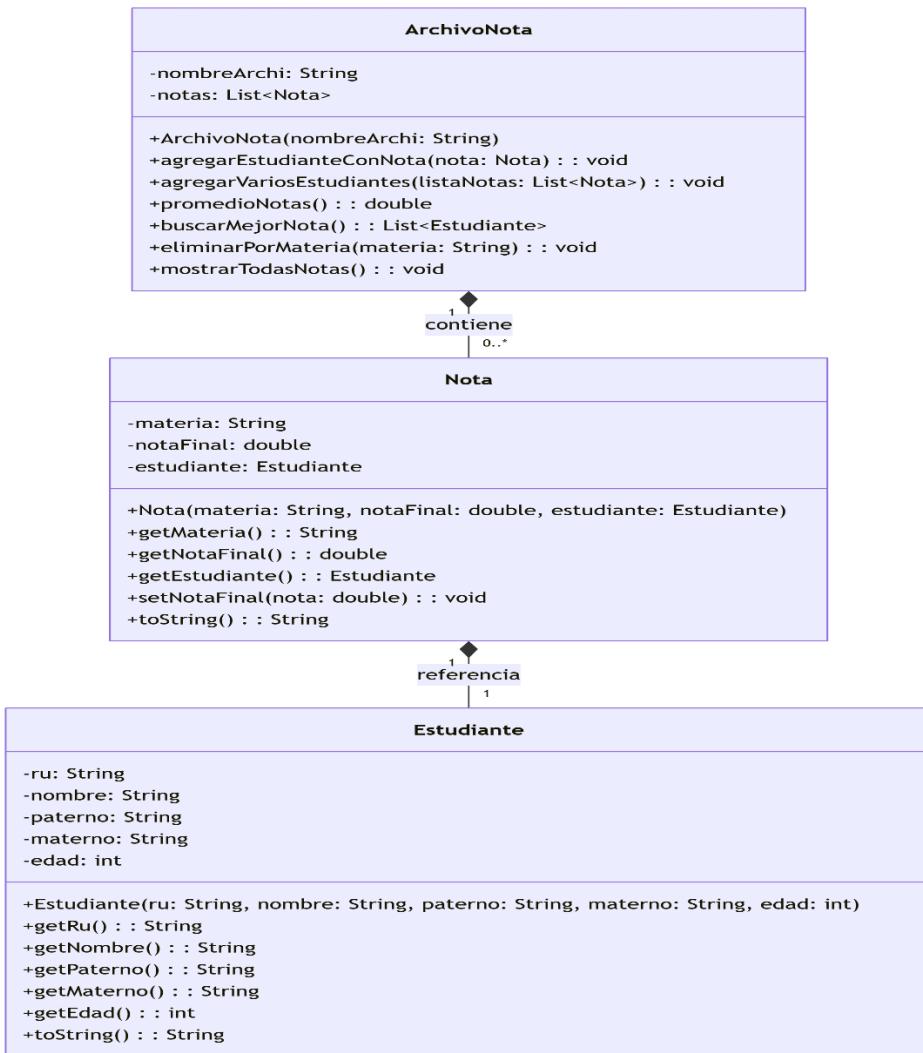
```
41     return total / len(self.productos)
42
43     def productoMasCaro(self):
44         if not self.productos:
45             return None
46         return max(self.productos, key=lambda p: p.precio)
47
48     def guardar(self):
49         try:
50             with open(self.nomA, 'wb') as f:
51                 pickle.dump(self.productos, f)
52         except:
53             pass
54
55     def cargar(self):
56         try:
57             if os.path.exists(self.nomA):
58                 with open(self.nomA, 'rb') as f:
59                     self.productos = pickle.load(f)
60         except:
61             self.productos = []
62
63     def obtenerTodos(self):
64         return self.productos.copy()
65
66     def eliminarProducto(self, codigo):
67         for i, p in enumerate(self.productos):
68             if p.codigo == codigo:
69                 del self.productos[i]
70                 self.guardar()
71                 return True
72         return False
73
74     def generarEjemplos(self):
75         if not self.productos:
76             ejemplos = [
77                 Producto(1001, "Laptop", 1200.50),
78                 Producto(1002, "Mouse", 25.99),
79                 Producto(1003, "Teclado", 45.75),
80                 Producto(1004, "Monitor", 350.00),
81                 Producto(1005, "Tablet", 550.80)
82             ]
83             for p in ejemplos:
84                 self.guardarProducto(p)
85
86 class SistemaProductos:
```

```
87     def __init__(self):
88         self.archivo = ArchivoProducto()
89
90     def iniciar(self):
91         while True:
92             print("\n" + "*50)
93             print("SISTEMA DE PRODUCTOS")
94             print("*50)
95             print("1. Crear archivo")
96             print("2. Mostrar todos")
97             print("3. Agregar producto")
98             print("4. Buscar producto")
99             print("5. Promedio precios")
100            print("6. Producto más caro")
101            print("7. Eliminar producto")
102            print("8. Generar ejemplos")
103            print("0. Salir")
104            print("*50)
105
106            op = input("Opción: ").strip()
107
108            if op == "0":
109                print("¡Adiós!")
110                break
111            elif op == "1":
112                self.crear_archivo()
113            elif op == "2":
114                self.mostrar.todos()
115            elif op == "3":
116                self.agregar_producto()
117            elif op == "4":
118                self.buscar_producto()
119            elif op == "5":
120                self.promedio_precios()
121            elif op == "6":
122                self.producto_mas_caro()
123            elif op == "7":
124                self.eliminar_producto()
125            elif op == "8":
126                self.generar_ejemplos()
127
128        def crear_archivo(self):
129            if self.archivo.crearArchivo():
130                print("✓ Archivo creado")
131            else:
132                print("✗ Error al crear archivo")
133
134        def mostrar.todos(self):
```

```
133
134     def mostrar_todos(self):
135         productos = self.archivo.obtenerTodos()
136         if not productos:
137             print("No hay productos")
138             return
139
140         print("\n" + "*50)
141         print(f'{\'Código\':<10} {\'Nombre\':<20} {\'Precio\':<10}')
142         print("-"*50)
143         for p in productos:
144             print(f'{p.codigo:<10} {p.nombre:<20} ${p.precio:<10.2f}")
145         print("*50)
146         print(f"Total: {len(productos)} productos")
147
148     def agregar_producto(self):
149         try:
150             codigo = int(input("Código: "))
151             if codigo <= 0:
152                 print("Código debe ser positivo")
153                 return
154
155             nombre = input("Nombre: ").strip()
156             if not nombre:
157                 print("Nombre requerido")
158                 return
159
160             precio = float(input("Precio: "))
161             if precio <= 0:
162                 print("Precio debe ser positivo")
163                 return
164
165             p = Producto(codigo, nombre, precio)
166             self.archivo.guardaProducto(p)
167             print("✓ Producto agregado")
168
169         except ValueError:
170             print("Entrada inválida")
171
172     def buscar_producto(self):
173         try:
174             codigo = int(input("Código a buscar: "))
175             p = self.archivo.buscaProducto(codigo)
176
177             if p:
178                 print(f"\nProducto encontrado:")
179                 print(f"Código: {p.codigo}")
```

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

Ejercicio 4



```

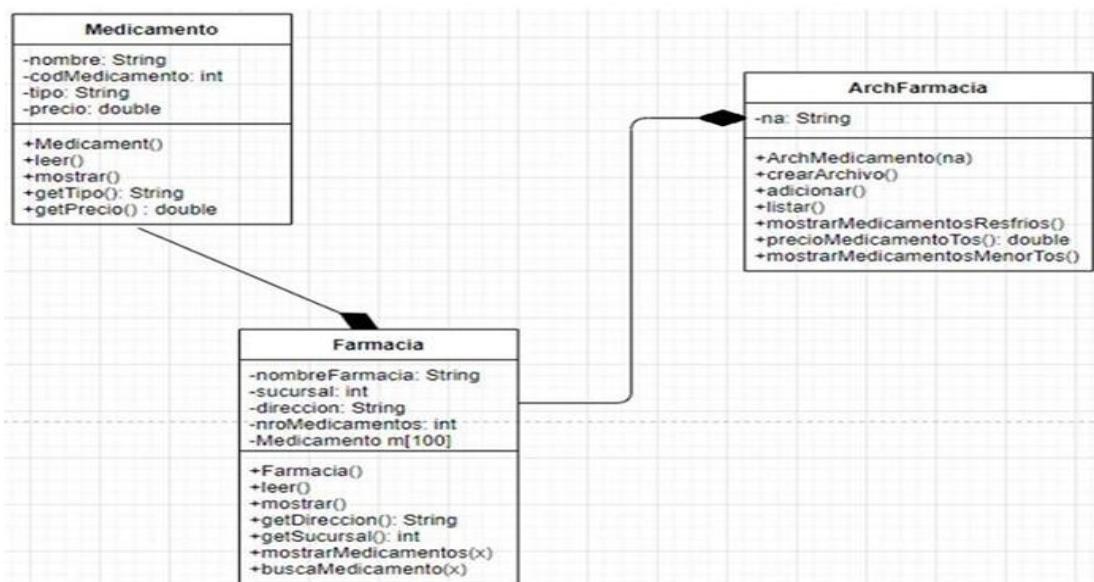
1 import pickle
2 import os
3
4 class Estudiante:
5     def __init__(self, ru, nombre, paterno, materno, edad):
6         self.ru = ru
7         self.nombre = nombre
8         self.paterno = paterno
9         self.materno = materno
10        self.edad = edad
11
12 class Nota:
13     def __init__(self, materno, notaFinal, estudiante):
14         self.materno = materno
15         self.notaFinal = notaFinal
16         self.estudiante = estudiante
17
18 class ArchivoNota:
19     def __init__(self, nombreArchi="notas.dat"):
20         self.nombreArchi = nombreArchi
21         self.notas = []
22         self.cargar()
23
  
```

```

23
24     def agregar_estudiantes(self, lista_notas):
25         self.notas.extend(lista_notas)
26         self.guardar()
27
28     def promedio_notas(self):
29         if not self.notas:
30             return 0
31         total = sum(n.notaFinal for n in self.notas) See Real World Examples From GitHub
32         return total / len(self.notas)
33
34     def mejores_estudiantes(self):
35         if not self.notas:
36             return []
37         max_nota = max(n.notaFinal for n in self.notas)
38         return [n for n in self.notas if n.notaFinal == max_nota]
39
40     def eliminar_por_materia(self, materia):
41         self.notas = [n for n in self.notas if n.materno != materia]
42         self.guardar()
43
44     def guardar(self):
45         with open(self.nombreArchi, 'wb') as f:
46             pickle.dump(self.notas, f)
47
48     def cargar(self):
49         if os.path.exists(self.nombreArchi):
50             with open(self.nombreArchi, 'rb') as f:
51                 self.notas = pickle.load(f)
52
53 if __name__ == "__main__":
54     archi = ArchiNota()
55     e1 = Estudiante("123", "Ana", "Lopez", "Perez", 20)
56     e2 = Estudiante("124", "Luis", "Garcia", "Martinez", 21)
57     n1 = Nota("Matematica", 85, e1)
58     n2 = Nota("Fisica", 90, e2)
59     n3 = Nota("Matematica", 88, e2)
60     archi.agregar_estudiantes([n1, n2, n3])
61     print(f"Promedio: {archi.promedio_notas()}")
62     mejores = archi.mejores_estudiantes()
63     print(f"Mejor(es) estudiante(s): {[ (m.estudiante.nombre, m.notaFinal) for m in mejores]}")
64     archi.eliminar_por_materia("Matematica")
65     print(f"Notas despues de eliminar Matematica: {len(archi.notas)}")

```

Ejercicio 5



```
medo > * Medicamento.py > ...
1 import pickle
2 import os
3
4 class Medicamento:
5     def __init__(self, nombre="", codMedicamento=0, tipo="", precio=0.0):
6         self.nombre = nombre
7         self.codMedicamento = codMedicamento
8         self.tipo = tipo
9         self.precio = precio
10    def leer(self):
11        self.nombre = input("Nombre: ")
12        self.codMedicamento = int(input("Codigo: "))
13        self.tipo = input("Tipo: ")
14        self.precio = float(input("Precio: "))
15    def mostrar(self):
16        print(f"{self.nombre} {self.codMedicamento} {self.tipo} {self.precio}")
17    def getTipo(self):
18        return self.tipo
19    def getPrecio(self):
20        return self.precio
21
22 class Farmacia:
23     def __init__(self):
24         self.nombreFarmacia = ""
25         self.sucursal = 0
26         self.direction = ""
27         self.moMedicamentos = 0
28         self.m = [Medicamento() for _ in range(100)]
29     def leer(self):
30         self.nombreFarmacia = input("Nombre farmacia: ")
31         self.sucursal = int(input("Sucursal: "))
32         self.direction = input("Direccion: ")
33         self.moMedicamentos = int(input("Cant medicamentos: "))
34         for i in range(self.moMedicamentos):
35             self.m[i].leer()
36     def mostrar(self):
37         print(f"{self.nombreFarmacia} {self.sucursal} {self.direction}")
38         for i in range(self.moMedicamentos):
39             self.m[i].mostrar()
40     def getDireccion(self):
41         return self.direction
42     def getSucursal(self):
43         return self.sucursal
44     def mostrarMedicamentos(self, x):
45         for i in range(self.moMedicamentos):
46             if self.m[i].getTipo() == x:
47                 self.m[i].mostrar()
48     def buscaMedicamento(self, x):
49         for i in range(self.moMedicamentos):
50             if self.m[i].nombre == x:
51                 return self.m[i]
52         return None
53
54 class ArchFarmacia:
55     def __init__(self, na="farmacias.dat"):
56         self.na = na
57         self.farmacias = []
58         self.cargar()
59     def crearArchivo(self):
```

```

59     def crearArchivo(self):
60         n = int(input("Cant farmacias: "))
61         for _ in range(n):
62             f = Farmacia()
63             f.leer()
64             self.farmacias.append(f)
65             self.guardar()
66     def adicionar(self):
67         f = Farmacia()
68         f.leer()
69         self.farmacias.append(f)
70         self.guardar()
71     def listar(self):
72         for f in self.farmacias:
73             f.mostrar()
74     def mostrarMedicamentosResfrios(self):
75         for f in self.farmacias:
76             f.mostrarMedicamentos("Resfrio")
77     def predoMedicamentoTos(self):
78         precio_total = 0
79         cantidad = 0
80         for f in self.farmacias:
81             for i in range(f.moMedicamentos):
82                 if f.m[i].getTipo() == "Tos":
83                     precio_total += f.m[i].getPrecio()
84                     cantidad += 1
85         return precio_total / cantidad if cantidad > 0 else 0
86     def mostrarMedicamentosMenorTos(self):
87         promedio = self.predoMedicamentoTos()
88         for f in self.farmacias:
89             for i in range(f.moMedicamentos):
90                 if f.m[i].getTipo() == "Tos" and f.m[i].getPrecio() < promedio:
91                     f.m[i].mostrar()
92     def a(self, sucursal_x):
93         for f in self.farmacias:
94             if f.getSucursal() == sucursal_x:
95                 f.mostrarMedicamentos("Tos")
96     def b(self):
97         for f in self.farmacias:
98             for i in range(f.moMedicamentos):
99                 if f.m[i].nombre == "Tapsin":
100                     print(f"{f.getSucursal()} {f.getDireccion()}")
101     def c(self, tipo):
102         for f in self.farmacias:
103             f.mostrarMedicamentos(tipo)
104     def d(self):
105         self.farmacias.sort(key=lambda x: x.getDireccion())
106         self.guardar()
107         self.listar()
108     def e(self, tipo, sucursal_y, sucursal_z):
109         farmacia_y = next((f for f in self.farmacias if f.getSucursal() == sucursal_y), None)
110         farmacia_z = next((f for f in self.farmacias if f.getSucursal() == sucursal_z), None)
111         if not farmacia_y or not farmacia_z:
112             return
113         movidos = []
114         i = 0
115         while i < farmacia_y.moMedicamentos:
116             if farmacia_y.m[i].getTipo() == tipo:

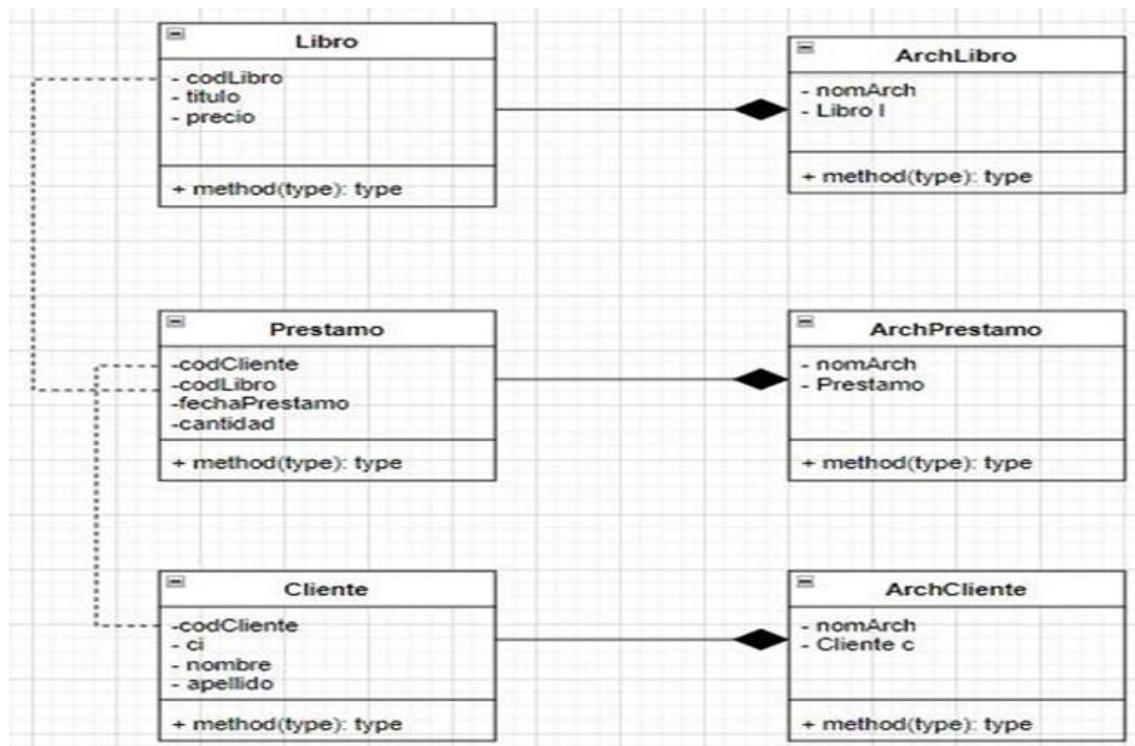
```

```

115     while i < farmacia_y.moMedicamentos:
116         if farmacia_y.m[i].getTipo() == tipo:
117             movidos.append(farmacia_y.m[i])
118             for j in range(i, farmacia_y.moMedicamentos - 1):
119                 farmacia_y.m[j] = farmacia_y.m[j + 1]
120             farmacia_y.moMedicamentos -= 1
121         else:
122             i += 1
123     for med in movidos:
124         if farmacia_z.moMedicamentos < 100:
125             farmacia_z.m[farmacia_z.moMedicamentos] = med
126             farmacia_z.moMedicamentos += 1
127     self.guardar()
128 def guardar(self):
129     with open(self.na, 'wb') as f:
130         pickle.dump(self.farmacias, f)
131 def cargar(self):
132     if os.path.exists(self.na):
133         with open(self.na, 'rb') as f:
134             self.farmacias = pickle.load(f)
135
136 if __name__ == "__main__":
137     arch = ArchFarmacia()
138     arch.crearArchivo()
139     arch.a(1)
140     arch.b()
141     arch.c("Resfrio")
142     arch.d()
143     arch.e("Tos", 1, 2)

```

Ejercicio 6



```
1 import pickle
2 import os
3 from datetime import datetime
4
5 class Libro:
6     def __init__(self, codLibro=0, titulo="", precio=0.0):
7         self.codLibro = codLibro
8         self.titulo = titulo
9         self.precio = precio
10    def leer(self):
11        self.codLibro = int(input("Cod libro: "))
12        self.titulo = input("Titulo: ")
13        self.precio = float(input("Precio: "))
14    def mostrar(self):
15        print(f"{self.codLibro} {self.titulo} {self.precio}")
16
17 class Prestamo:
18     def __init__(self, codCliente=0, codLibro=0, fechaPrestamo="", cantidad=0):
19         self.codCliente = codCliente
20         self.codLibro = codLibro
21         self.fechaPrestamo = fechaPrestamo
22         self.cantidad = cantidad
23    def leer(self):
24        self.codCliente = int(input("Cod cliente: "))
25        self.codLibro = int(input("Cod libro: "))
26        self.fechaPrestamo = input("Fecha: ")
27        self.cantidad = int(input("Cantidad: "))
28    def mostrar(self):
29        print(f"{self.codCliente} {self.codLibro} {self.fechaPrestamo} {self.cantidad}")
30
31 class Cliente:
32     def __init__(self, codCliente=0, ci="", nombre="", apellido ""):
33         self.codCliente = codCliente
34         self.ci = ci
35         self.nombre = nombre
36         self.apellido = apellido
37    def leer(self):
38        self.codCliente = int(input("Cod cliente: "))
39        self.ci = input("CI: ")
40        self.nombre = input("Nombre: ")
41        self.apellido = input("Apellido: ")
42    def mostrar(self):
43        print(f"{self.codCliente} {self.ci} {self.nombre} {self.apellido}")
44
45 class ArchLibro:
46     def __init__(self, nomArch="libros.dat"):
47         self.nomArch = nomArch
48         self.libros = []
49         self.cargar()
50     def crearArchivo(self):
51         n = int(input("Cant libros: "))
52         for _ in range(n):
53             l = Libro()
54             l.leer()
55             self.libros.append(l)
56         self.guardar()
57     def adicionar(self):
58         l = Libro()
59         l.leer()
```

```

45  class ArchLibro:
46      def adicionar(self):
47          self.libros.append(l)
48          self.guardar()
49      def listar(self):
50          for l in self.libros:
51              l.mostrar()
52      def a(self, x, y):
53          for l in self.libros:
54              if x <= l.precio <= y:
55                  l.mostrar()
56      def c(self, archPrestamo):
57          prestamos_cod = set(p.codLibro for p in archPrestamo.prestamos)
58          for l in self.libros:
59              if l.codLibro not in prestamos_cod:
60                  l.mostrar()
61      def guardar(self):
62          with open(self.nomArch, 'wb') as f:
63              pickle.dump(self.libros, f)
64      def cargar(self):
65          if os.path.exists(self.nomArch):
66              with open(self.nomArch, 'rb') as f:
67                  self.libros = pickle.load(f)
68
69  class ArchPrestamo:
70      def __init__(self, nomArch="prestamos.dat"):
71          self.nomArch = nomArch
72          self.prestamos = []
73          self.cargar()
74      def crearArchivo(self):
75          n = int(input("Cant prestamos: "))
76          for _ in range(n):
77              p = Prestamo()
78              p.leer()
79              self.prestamos.append(p)
80          self.guardar()
81      def adicionar(self):
82          p = Prestamo()
83          p.leer()
84          self.prestamos.append(p)
85          self.guardar()
86      def listar(self):
87          for p in self.prestamos:
88              p.mostrar()
89      def b(self, codLibro, archLibro):
90          libro = next((l for l in archLibro.libros if l.codLibro == codLibro), None)
91          if libro:
92              total = sum(p.cantidad * libro.precio for p in self.prestamos if p.codLibro == codLibro)
93              print(f"Ingreso total: {total}")
94      def d(self, codLibro, archCliente):
95          clientes_cod = set(p.codCliente for p in self.prestamos if p.codLibro == codLibro)
96          for c in archCliente.clientes:
97              if c.codCliente in clientes_cod:
98                  c.mostrar()
99      def e(self):
100         conteo = {}
101         for p in self.prestamos:
102             conteo[p.codLibro] = conteo.get(p.codLibro, 0) + p.cantidad
103         if conteo:
104             ...

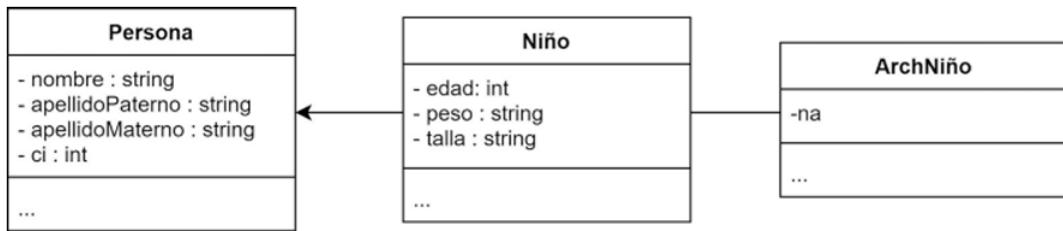
```

```

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



```

1 v import pickle
2   import os
3   from datetime import datetime
4
5 v class Libro:
6 v     def __init__(self, codLibro=0, titulo="", precio=0.0):
7       self.codLibro = codLibro
8       self.titulo = titulo
9       self.precio = precio
10 v    def leer(self):
11      self.codLibro = int(input("Cod libro: "))
12      self.titulo = input("Titulo: ")
13      self.precio = float(input("Precio: "))
14 v    def mostrar(self):
15      print(f"{self.codLibro} {self.titulo} {self.precio}")
16
17 v class Prestamo:
18 v     def __init__(self, codCliente=0, codLibro=0, fechaPrestamo="", cantidad=0):
19       self.codCliente = codCliente
20       self.codLibro = codLibro
21       self.fechaPrestamo = fechaPrestamo
22       self.cantidad = cantidad
23 v     def leer(self):
24       self.codCliente = int(input("Cod cliente: "))
25       self.codLibro = int(input("Cod libro: "))
26       self.fechaPrestamo = input("Fecha: ")
27       self.cantidad = int(input("Cantidad: "))
28 v     def mostrar(self):
29       print(f"{self.codCliente} {self.codLibro} {self.fechaPrestamo} {self.cantidad}")
30
31 v class Cliente:
32 v     def __init__(self, codCliente=0, ci="", nombre="", apellido ""):
33       self.codCliente = codCliente
34       self.ci = ci
35       self.nombre = nombre
36       self.apellido = apellido
37 v     def leer(self):
38       self.codCliente = int(input("Cod cliente: "))
39       self.ci = input("CI: ")
40       self.nombre = input("Nombre: ")
41       self.apellido = input("Apellido: ")
42 v     def mostrar(self):
43       print(f"{self.codCliente} {self.ci} {self.nombre} {self.apellido}")
44
45 v class ArchLibro:
46 v     def __init__(self, nomArch="libros.dat"):
47       self.nomArch = nomArch
48       self.libros = []
49       self.cargar()
50 v     def crearArchivo(self):
51       ...
  
```

```

50     def crearArchivo(self):
51         n = int(input("Cant libros: "))
52         for _ in range(n):
53             l = Libro()
54             l.leer()
55             self.libros.append(l)
56         self.guardar()
57     def adicionar(self):
58         l = Libro()
59         l.leer()
60         self.libros.append(l)
61         self.guardar()
62     def listar(self):
63         for l in self.libros:
64             l.mostrar()
65     def a(self, x, y):
66         for l in self.libros:
67             if x <= l.precio <= y:
68                 l.mostrar()
69     def c(self, archPrestamo):
70         prestamos_cod = set(p.codLibro for p in archPrestamo.prestamos)
71         for l in self.libros:
72             if l.codLibro not in prestamos_cod:
73                 l.mostrar()
74     def guardar(self):
75         with open(self.nomArch, 'wb') as f:
76             pickle.dump(self.libros, f)
77     def cargar(self):
78         if os.path.exists(self.nomArch):
79             with open(self.nomArch, 'rb') as f:
80                 self.libros = pickle.load(f)
81
82 class ArchPrestamo:
83     def __init__(self, nomArch="prestamos.dat"):
84         self.nomArch = nomArch
85         self.prestamos = []
86         self.cargar()
87     def crearArchivo(self):
88         n = int(input("Cant prestamos: "))
89         for _ in range(n):
90             p = Prestamo()
91             p.leer()
92             self.prestamos.append(p)
93         self.guardar()
94     def adicionar(self):
95         p = Prestamo()
96         p.leer()
97         self.prestamos.append(p)
98         self.guardar()
99     def listar(self):
100        for p in self.prestamos:
101            p.mostrar()
102    def b(self, codLibro, archLibro):
103        libro = next((l for l in archLibro.libros if l.codLibro == codLibro), None)
104        if libro:
105            total = sum(p.cantidad * libro.precio for p in self.prestamos if p.codLibro == codLibro)
106            print(f"Ingreso total: {total}")
107    def d(self, codLibro, archCliente):
108        clientes_cod = set(p.codCliente for p in self.prestamos if p.codLibro == codLibro)
109        for c in archCliente.clientes:
110            if c.codCliente in clientes_cod:
111                c.mostrar()
112    def e(self):
113        conteo = {}
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:

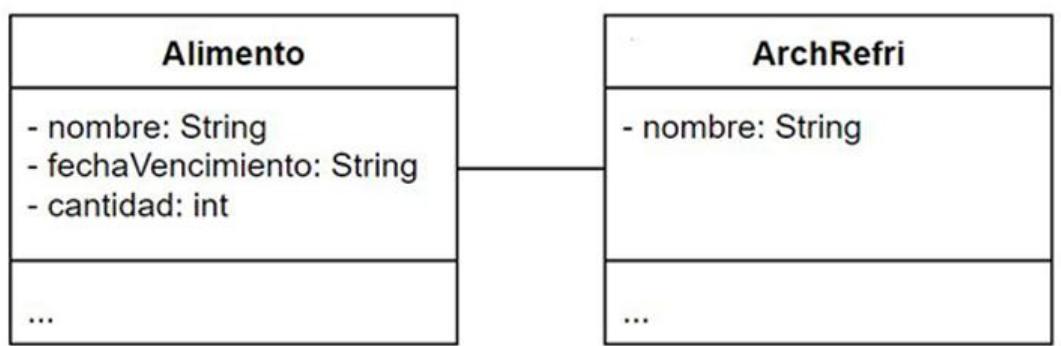
```

```

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()
171     archPrestamo.f()

```

Ejercicio 8



```

1  import pickle
2  import os
3  from datetime import datetime
4
5  class Alimento:
6      def __init__(self, nombre="", fecha_vencimiento=None, cantidad=0):
7          self.nombre = nombre
8          self.fecha_vencimiento = fecha_vencimiento
9          self.cantidad = cantidad
10     def leer(self):
11         self.nombre = input("Nombre: ")
12         self.fecha_vencimiento = input("Fecha vencimiento (dd/mm/yyyy): ")
13         self.cantidad = int(input("Cantidad: "))
14     def mostrar(self):
15         print(f"{self.nombre} {self.fecha_vencimiento} {self.cantidad}")
16     def esta_vencido(self):
17         try:
18             fecha_venc = datetime.strptime(self.fecha_vencimiento, "%d/%m/%Y")
19             return fecha_venc < datetime.now()
20         except:
21             return False
22     def caduca_anteriores_a(self, fecha_x):
23         try:
24             fecha_venc = datetime.strptime(self.fecha_vencimiento, "%d/%m/%Y")
25             fecha_comp = datetime.strptime(fecha_x, "%d/%m/%Y")
26             return fecha_venc < fecha_comp
27         except:
28             return False
29
30 class ArchRefri:
31     def __init__(self, nombre="refri.dat"):
32         self.nombre = nombre
33         self.alimentos = []
34         self.cargar()
35     def crearArchivo(self):
36         n = int(input("Cant alimentos: "))
37         for _ in range(n):
38             a = Alimento()
39             a.leer()
40             self.alimentos.append(a)
41         self.guardar()
42     def modificar_por_nombre(self, nombre):
43         for a in self.alimentos:
44             if a.nombre == nombre:
45                 print("Nuevos datos:")
46                 a.leer()
47                 self.guardar()
48                 return
49             print("No encontrado")
50     def eliminar_por_nombre(self, nombre):
51         self.alimentos = [a for a in self.alimentos if a.nombre != nombre]
52         self.guardar()
53     def b(self, fecha_x):
54         for a in self.alimentos:
55             if a.caduca_anteriores_a(fecha_x):
56                 a.mostrar()
57     def c(self):
58         self.alimentos = [a for a in self.alimentos if a.cantidad > 0]
59         self.guardar()

```

(function) `def input(prompt: object = "", /)`
 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.

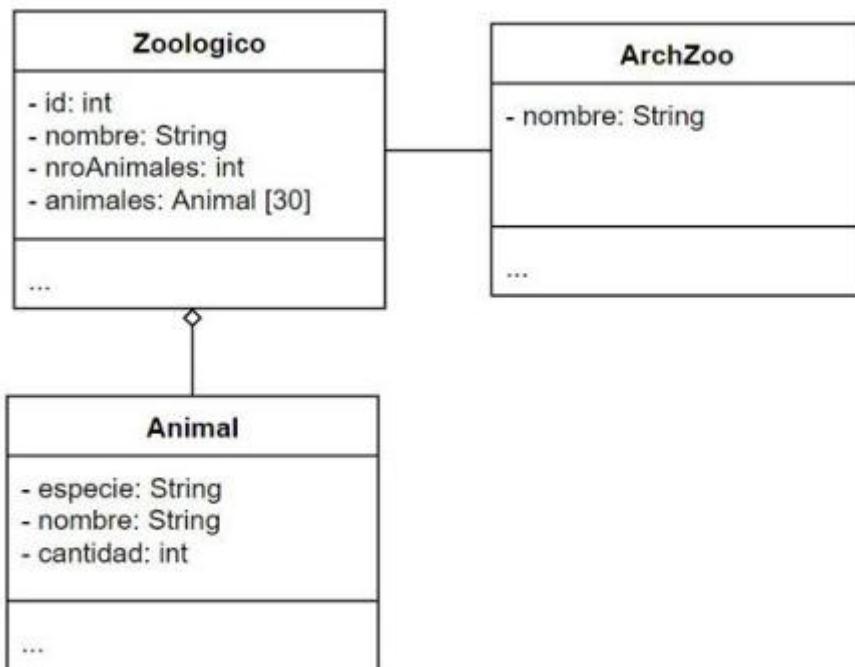
```
1  import pickle
2  import os
3  from datetime import datetime
4
5  class Alimento:
6      def __init__(self, nombre="", fecha_vencimiento="", cantidad=0):
7          self.nombre = nombre
8          self.fecha_vencimiento = fecha_vencimiento
9          self.cantidad = cantidad
10     def leer(self):
11         self.nombre = input("Nombre: ")
12         self.fecha_vencimiento = input("Fecha vencimiento (dd/mm/yyyy): ")
13         self.cantidad = int(input("Cantidad: "))
14     def mostrar(self):
15         print(f"{self.nombre} {self.fecha_vencimiento} {self.cantidad}")
16     def esta_vencido(self):
17         try:
18             fecha_venc = datetime.strptime(self.fecha_vencimiento, "%d/%m/%Y")
19             return fecha_venc < datetime.now()
20         except:
21             return False
22     def caduca_anter_de(self, fecha_x):
23         try:
24             fecha_venc = datetime.strptime(self.fecha_vencimiento, "%d/%m/%Y")
25             fecha_comp = datetime.strptime(fecha_x, "%d/%m/%Y")
26             return fecha_venc < fecha_comp
27         except:
28             return False
29
30 class ArchRefri:
31     def __init__(self, nombre="refri.dat"):
32         self.nombre = nombre
33         self.alimentos = []
34         self.cargar()
35     def crearArchivo(self):
36         n = int(input("Cant alimentos: "))
37         for _ in range(n):
38             a = Alimento()
39             a.leer()
40             self.alimentos.append(a)
41         self.guardar()
42     def modificar_por_nombre(self, nombre):
43         for a in self.alimentos:
44             if a.nombre == nombre:
45                 print("Nuevos datos:")
46                 a.leer()
47                 self.guardar()
48                 return
```

```

30  class ArchRefri:
31      def crearArchivo(self):
32          n = int(input("Cant alimentos: "))
33          for _ in range(n):
34              a = Alimento()
35              a.leer()
36              self.alimentos.append(a)
37          self.guardar()
38      def modificar_por_nombre(self, nombre):
39          for a in self.alimentos:
40              if a.nombre == nombre:
41                  print("Nuevos datos:")
42                  a.leer()
43                  self.guardar()
44                  return
45              print("No encontrado")
46      def eliminar_por_nombre(self, nombre):
47          self.alimentos = [a for a in self.alimentos if a.nombre != nombre]
48          self.guardar()
49      def b(self, fecha_x):
50          for a in self.alimentos:
51              if a.caduca_anter_de(fecha_x):
52                  a.mostrar()
53      def c(self):
54          self.alimentos = [a for a in self.alimentos if a.cantidad > 0]
55          self.guardar()
56      def d(self):
57          for a in self.alimentos:
58              if a.esta_vencido():
59                  a.mostrar()
60      def e(self):
61          if not self.alimentos:
62              return
63          max_alimento = max(self.alimentos, key=lambda x: x.cantidad)
64          max_alimento.mostrar()
65      def listar(self):
66          for a in self.alimentos:
67              a.mostrar()
68      def guardar(self):
69          with open(self.nombre, 'wb') as f:
70              pickle.dump(self.alimentos, f)
71      def cargar(self):
72          if os.path.exists(self.nombre):
73              with open(self.nombre, 'rb') as f:
74                  self.alimentos = pickle.load(f)
75
76
77
78
79
80  if __name__ == "__main__":
81      arch = ArchRefri()
82      arch.crearArchivo()
83      arch.modificar_por_nombre("Leche")
84      arch.eliminar_por_nombre("Pan")
85      arch.b("01/01/2024")
86      arch.c()
87      arch.d()
88      arch.e()

```

Ejercicio 9



```
1 import pickle
2 import os
3
4 class Animal:
5     def __init__(self, especie="", nombre="", cantidad=0):
6         self.especie = especie
7         self.nombre = nombre
8         self.cantidad = cantidad
9     def leer(self):
10        self.especie = input("Especie: ")
11        self.nombre = input("Nombre: ")
12        self.cantidad = int(input("Cantidad: "))
13    def mostrar(self):
14        print(f"{self.especie} {self.nombre} {self.cantidad}")
15
16 class Zoologico:
17     def __init__(self, id=0, nombre="", nroAnimales=0):
18         self.id = id
19         self.(variable) nroAnimales: int
20         self.nroAnimales = nroAnimales
21         self.animales = [Animal() for _ in range(30)]
22     def leer(self):
23        self.id = int(input("ID: "))
24        self.nombre = input("Nombre: ")
25        self.nroAnimales = int(input("Nro animales: "))
26        for i in range(self.nroAnimales):
27            self.animales[i].leer()
28    def mostrar(self):
29        print(f"{self.id} {self.nombre} {self.nroAnimales}")
30        for i in range(self.nroAnimales):
31            self.animales[i].mostrar()
32    def variedad_animales(self):
33        especies = set()
34        for i in range(self.nroAnimales):
35            especies.add(self.animales[i].especie)
36        return len(especies)
37    def esta_vacio(self):
38        return self.nroAnimales == 0
39    def animales_especie_x(self, especie_x):
40        for i in range(self.nroAnimales):
41            if self.animales[i].especie == especie_x:
42                self.animales[i].mostrar()
```

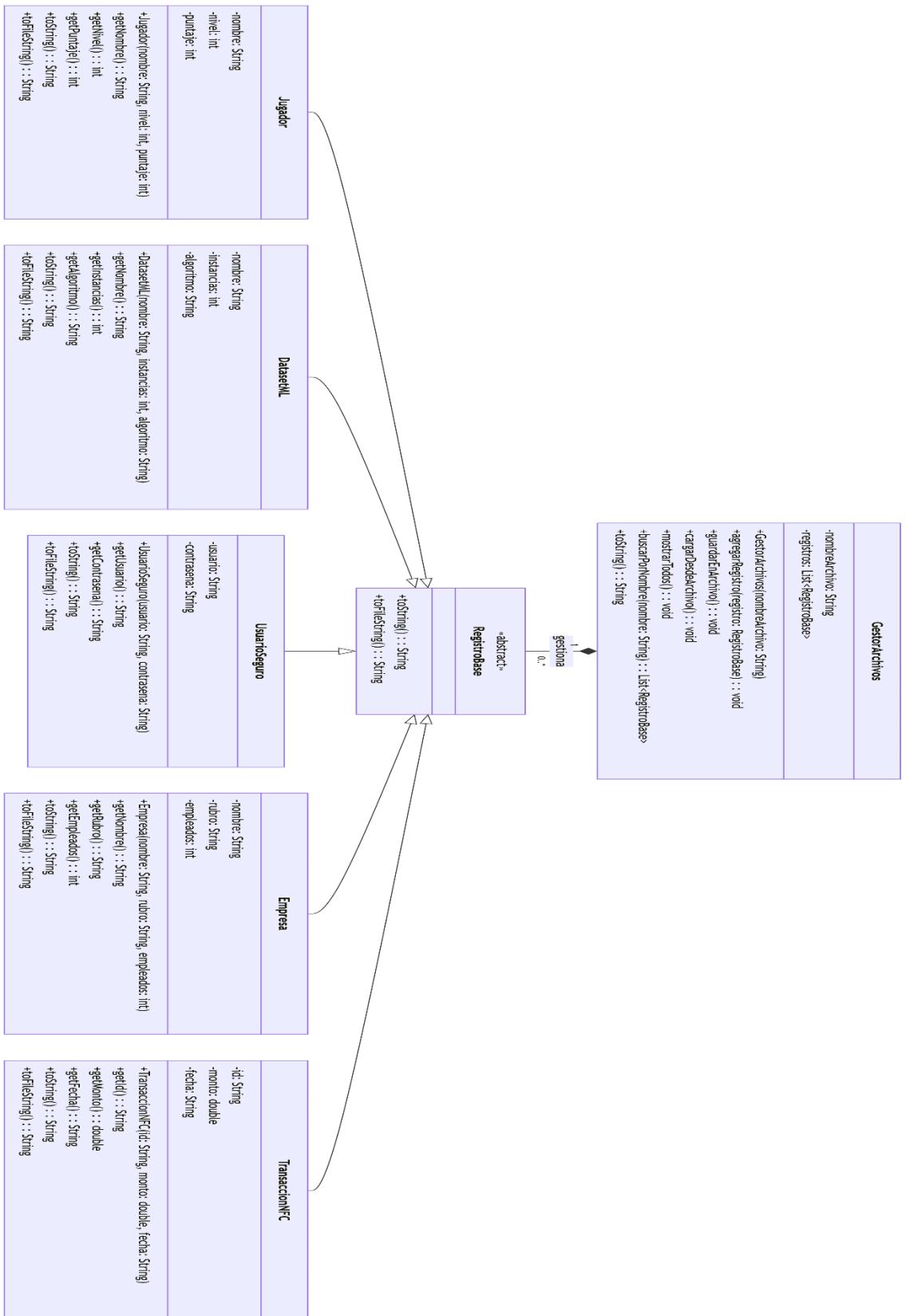
```

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):

```

```
98     def e(self, id_x, id_y, especie_x):
99         zoo_x = next((z for z in self.zoos if z.id == id_x), None)
100        zoo_y = next((z for z in self.zoos if z.id == id_y), None)
101        if not zoo_x or not zoo_y:
102            return
103        animales_mover = zoo_x.quitar_animales_especie(especie_x)
104        for animal in animales_mover:
105            zoo_y.agregar_animal(animal)
106        self.guardar()
107    def listar(self):
108        for z in self.zoos:
109            z.mostrar()
110    def guardar(self):
111        with open(self.nombre, 'wb') as f:
112            pickle.dump(self.zoos, f)
113    def cargar(self):
114        if os.path.exists(self.nombre):
115            with open(self.nombre, 'rb') as f:
116                self.zoos = pickle.load(f)
117
118 if __name__ == "__main__":
119     arch = ArchZoo()
120     arch.crearArchivo()
121     arch.modificar(1)
122     arch.eliminar(2)
123     arch.b()
124     arch.c()
125     arch.d("Leon")
126     arch.e(1, 3, "Tigre")
```

Ejercicio 10



```
File > CobrosNFC.java / 77 lines
  1 package Diez;
  2 import java.io.*;
  3 import java.util.ArrayList;
  4 import java.util.List;
  5
  6 public class CobrosNFC {
  7     private List<Transaccion> transacciones;
  8     private static final String ARCHIVO = "cobros_nfc.txt";
  9
 10    public CobrosNFC() {
 11        transacciones = new ArrayList<>();
 12        cargarTransacciones();
 13    }
 14
 15    public void agregarTransaccion(Transaccion transaccion) {
 16        transacciones.add(transaccion);
 17        guardarTransacciones();
 18    }
 19
 20    public void mostrarTransacciones() {
 21        for (Transaccion t : transacciones) {
 22            System.out.println(t);
 23        }
 24    }
 25
 26    public Transaccion buscarPorID(String id) {
 27        for (Transaccion t : transacciones) {
 28            if (t.getId().equalsIgnoreCase(id)) {
 29                return t;
 30            }
 31        }
 32        return null;
 33    }
 34
 35    private void guardarTransacciones() {
 36        try (BufferedWriter writer = new BufferedWriter(new FileWriter(ARCHIVO))) {
 37            for (Transaccion t : transacciones) {
 38                writer.write(t.getId() + "," + t.getMonto() + "," + t.getFecha());
 39                writer.newLine();
 40            }
 41        } catch (IOException e) {
 42            System.err.println("Error al guardar transacciones: " + e.getMessage());
 43        }
 44    }
 45
 46    private void cargarTransacciones() {
 47        File archivo = new File(ARCHIVO);
 48        if (!archivo.exists()) {
 49            return;
 50        }
 51
 52        try (BufferedReader reader = new BufferedReader(new FileReader(ARCHIVO))) {
 53            String linea;
 54            while ((linea = reader.readLine()) != null) {
 55                String[] datos = linea.split(regex: ",");
 56                if (datos.length == 3) {
 57                    String id = datos[0];
 58                    double monto = Double.parseDouble(datos[1]);
 59                    String fecha = datos[2];
 60                    transacciones.add(new Transaccion(id, monto, fecha));
 61                }
 62            }
 63        } catch (IOException e) {
 64            System.err.println("Error al cargar transacciones: " + e.getMessage());
 65        }
 66    }
 67 }
```

```
nez > Dispositivo.java > ...
1 package Diez;
2
3 public class Dispositivo {
4     private String mac;
5     private String nombre;
6     private int velocidad;
7
8     public Dispositivo(String mac, String nombre, int velocidad) {
9         this.mac = mac;
10        this.nombre = nombre;
11        this.velocidad = velocidad;
12    }
13
14    public String getMac() {
15        return mac;
16    }
17
18    public String getNombre() {
19        return nombre;
20    }
21
22    public int getVelocidad() {
23        return velocidad;
24    }
25
26    @Override
27    public String toString() {
28        return mac + " - " + nombre + " - " + velocidad + " Mbps";
29    }
30 }
```

```
1 package Diez;
2 import java.util.Scanner;
3
4 public class Main {
5     Run | Debug
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8         RedWifi redWifi = new RedWifi();
9         CobrosNFC cobrosNFC = new CobrosNFC();
10
11         while (true) {
12             System.out.println(x: "\n==> MENÚ PRINCIPAL ==>");
13             System.out.println(x: "1. Gestión de Red WiFi");
14             System.out.println(x: "2. Gestión de Cobros NFC");
15             System.out.println(x: "3. Salir");
16             System.out.print(s: "Seleccione una opción: ");
17
18             int opcionPrincipal = scanner.nextInt();
19             scanner.nextLine();
20
21             switch (opcionPrincipal) {
22                 case 1 -> gestionRedWifi(scanner, redWifi);
23                 case 2 -> gestionCobrosNFC(scanner, cobrosNFC);
24                 case 3 -> {
25                     System.out.println(x: "Saliendo...");
26                     scanner.close();
27                     System.exit(status: 0);
28                 }
29             }
30         }
31     }
```

```

31     }
32
33     private static void gestionRedWifi(Scanner scanner, RedWifi redWifi) {
34         while (true) {
35             System.out.println("n==> GESTIÓN DE RED WiFi ==");
36             System.out.println("1. Agregar dispositivo");
37             System.out.println("2. Mostrar todos los dispositivos");
38             System.out.println("3. Buscar dispositivo por MAC");
39             System.out.println("4. Volver al menú principal");
40             System.out.print(s: "Seleccione una opción: ");
41
42             int opcion = scanner.nextInt();
43             scanner.nextLine();
44
45             switch (opcion) {
46                 case 1 -> {
47                     System.out.print(s: "Ingrese dirección MAC: ");
48                     String mac = scanner.nextLine();
49                     System.out.print(s: "Ingrese nombre del dispositivo: ");
50                     String nombre = scanner.nextLine();
51                     System.out.print(s: "Ingrese velocidad de conexión (Mbps): ");
52                     int velocidad = scanner.nextInt();
53                     scanner.nextLine();
54
55                     Dispositivo dispositivo = new Dispositivo(mac, nombre, velocidad);
56                     redWifi.agregarDispositivo(dispositivo);
57                     System.out.println("n==> Dispositivo agregado correctamente.");
58                 }
59                 case 2 -> {
60                     System.out.println("n==> DISPOSITIVOS CONECTADOS ==");
61                     redWifi.mostrarDispositivos();
62                 }
63                 case 3 -> {
64                     System.out.print(s: "Ingrese dirección MAC a buscar: ");
65                     String macBuscar = scanner.nextLine();
66                     Dispositivo encontrado = redWifi.buscarPorMAC(macBuscar);
67                     if (encontrado != null) {
68                         System.out.println("Dispositivo encontrado: " + encontrado);
69                     } else {
70                         System.out.println("No se encontró dispositivo con esa MAC.");
71                     }
72                 }
73                 case 4 -> {
74                     return;
75                 }
76                 default -> System.out.println("Opción no válida");
77             }
78         }
79     }
80
81     private static void gestionCobrosNFC(Scanner scanner, CobrosNFC cobrosNFC) {
82         while (true) {
83             System.out.println("n==> GESTIÓN DE COBROS NFC ==");
84             System.out.println("1. Agregar transacción");
85             System.out.println("2. Mostrar todas las transacciones");
86             System.out.println("3. Buscar transacción por ID");
87             System.out.println("4. Volver al menú principal");
88             System.out.print(s: "Seleccione una opción: ");
89
90             int opcion = scanner.nextInt();
91             scanner.nextLine();
92
93             switch (opcion) {
94                 case 1 -> {
95                     System.out.print(s: "Ingrese ID de transacción: ");
96                     String id = scanner.nextLine();
97                     System.out.print(s: "Ingrese monto: ");
98                     double monto = scanner.nextDouble();
99                     scanner.nextLine();
100                    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 }
```

```
1 package Diez;
2 > // Archivo: RedWifi.java...
3 import java.util.ArrayList;
4 import java.util.List;
5
6
7 public class RedWifi {
8     private List<Dispositivo> dispositivos;
9     private static final String ARCHIVO = "dispositivos_wifi.txt";
10
11     public void cargarDispositivos() {
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;
34     }
35 }
```

```

33     return null;
34 }
35
36 private void guardarDispositivos() {
37     try (BufferedWriter writer = new BufferedWriter(new FileWriter(ARCHIVO))) {
38         for (Dispositivo d : dispositivos) {
39             writer.write(d.getMac() + "," + d.getNombre() + "," + d.getVelocidad());
40             writer.newLine();
41         }
42     } catch (IOException e) {
43         System.err.println("Error al guardar dispositivos: " + e.getMessage());
44     }
45 }
46
47 private void cargarDispositivos() {
48     File archivo = new File(ARCHIVO);
49     if (!archivo.exists()) {
50         return;
51     }
52
53     try (BufferedReader reader = new BufferedReader(new FileReader(ARCHIVO))) {
54         String linea;
55         while ((linea = reader.readLine()) != null) {
56             String[] datos = linea.split(regex: ",");
57             if (datos.length == 3) {
58                 String mac = datos[0];
59                 String nombre = datos[1];
60                 int velocidad = Integer.parseInt(datos[2]);
61                 dispositivos.add(new Dispositivo(mac, nombre, velocidad));
62             }
63         }
64     } catch (IOException e) {
65         System.err.println("Error al cargar dispositivos: " + e.getMessage());
66     }
67 }
68 }

```

```

Diez > J Transaccion.java > ...
1 package Diez;
2
3 public class Transaccion {
4     private String id;
5     private double monto;
6     private String fecha;
7
8     public Transaccion(String id, double monto, String fecha) {
9         this.id = id;
10        this.monto = monto;
11        this.fecha = fecha;
12    }
13
14    public String getId() {
15        return id;
16    }
17
18    public double getMonto() {
19        return monto;
20    }
21
22    public String getFecha() {
23        return fecha;
24    }
25
26    @Override
27    public String toString() {
28        return "ID: " + id + " - Monto: $" + monto + " - Fecha: " + fecha;
29    }
30 }

```

