

```

1 package cl.enmanuelchirinos.pnb.repository;
2
3 import cl.enmanuelchirinos.pnb.model.Usuario;
4 import java.util.List;
5
6 /**
7  * Contrato de operaciones para acceso a datos de Usuario
8  */
9 public interface IUsuarioRepository {
10
11     /**
12      * Busca un usuario por su ID
13      * @param id ID del usuario
14      * @return Usuario encontrado o null
15      */
16     Usuario buscarPorId(int id);
17
18     /**
19      * Busca un usuario por su username
20      * @param username Username a buscar
21      * @return Usuario encontrado o null
22      */
23     Usuario buscarPorUsername(String username);
24
25     /**
26      * Lista todos los usuarios
27      * @return Lista de todos los usuarios
28      */
29     List<Usuario> listarTodos();
30
31     /**
32      * Lista usuarios por rol
33      * @param rol Rol a filtrar (ADMIN, OPERADOR)
34      * @return Lista de usuarios con ese rol
35      */
36     List<Usuario> listarPorRol(String rol);
37
38     /**
39      * Guarda un nuevo usuario
40      * @param usuario Usuario a guardar
41      * @return ID generado
42      */
43     int guardar(Usuario usuario);
44
45     /**
46      * Actualiza un usuario existente
47      * @param usuario Usuario con datos actualizados
48      */
49     void actualizar(Usuario usuario);
50
51     /**
52      * Elimina un usuario por ID
53      * @param id ID del usuario a eliminar
54      */
55     void eliminar(int id);
56
57     /**
58      * Verifica si existe un username
59      * @param username Username a verificar
60      * @return true si existe, false si no
61      */
62     boolean existeUsername(String username);
63
64     /**
65      * Cuenta usuarios activos por rol
66      * @param rol Rol a contar
67      * @return Cantidad de usuarios activos con ese rol
68      */
69     int contarActivosPorRol(String rol);
70 }

```

```

1 package cl.enmanuelchirinos.pnb.repository;
2
3 import cl.enmanuelchirinos.pnb.model.Producto;
4 import java.util.List;
5
6 /**
7  * Contrato de operaciones para acceso a datos de Producto
8  */
9 public interface IProductoRepository {
10
11     /**
12      * Busca un producto por su ID
13      */
14     Producto buscarPorId(int id);
15
16     /**
17      * Lista todos los productos
18      */
19     List<Producto> listarTodos();
20
21     /**
22      * Lista productos activos solamente
23      */
24     List<Producto> listarActivos();
25
26     /**
27      * Lista productos por categoría
28      * @param categoria BEBIDA, SNACK, TIEMPO
29      */
30     List<Producto> listarPorCategoria(String categoria);
31
32     /**
33      * Busca productos por nombre (búsqueda parcial)
34      */
35     List<Producto> buscarPorNombre(String nombre);
36
37     /**
38      * Guarda un nuevo producto
39      */
40     int guardar(Producto producto);
41
42     /**
43      * Actualiza un producto existente
44      */
45     void actualizar(Producto producto);
46
47     /**
48      * Elimina un producto por ID
49      */
50     void eliminar(int id);
51
52     /**
53      * Cambia el estado activo/inactivo de un producto
54      */
55     void cambiarEstado(int id, boolean activo);
56 }

```

```

1 package cl.enmanuelchirinos.pnb.repository;
2
3 import cl.enmanuelchirinos.pnb.model.Venta;
4 import java.time.LocalDate;
5 import java.time.LocalDateTime;
6 import java.util.List;
7
8 /**
9  * Contrato de operaciones para acceso a datos de Venta
10  */
11 public interface IVentaRepository {
12
13     /**
14      * Busca una venta por su ID
15      */
16     Venta buscarPorId(int id);
17
18     /**
19      * Lista todas las ventas
20      */
21     List<Venta> listarTodas();
22
23     /**
24      * Lista ventas por rango de fechas
25      */
26     List<Venta> listarPorRangoFechas(LocalDateTime desde, LocalDateTime hasta);
27
28     /**
29      * Lista ventas del día actual
30      */
31     List<Venta> listarDelDia();
32
33     /**
34      * Lista ventas por usuario
35      */
36     List<Venta> listarPorUsuario(int usuarioId);
37
38     /**
39      * Guarda una nueva venta
40      */
41     int guardar(Venta venta);
42
43     /**
44      * Anula una venta
45      */
46     void anular(int id);
47
48     /**
49      * Calcula total de ventas por rango de fechas
50      */
51     double calcularTotalPorRango(LocalDateTime desde, LocalDateTime hasta);
52
53     /**
54      * Calcula total de ventas del día
55      */
56     double calcularTotalDelDia();
57 }

```

```

1 package cl.enmanuelchirinos.pnb.repository.mock;
2
3 import cl.enmanuelchirinos.pnb.model.Usuario;
4 import cl.enmanuelchirinos.pnb.repository.IUsuarioRepository;
5 import java.util.ArrayList;
6 import java.util.List;
7 import java.util.stream.Collectors;
8
9 /**
10  * Implementación Mock (en memoria) del repositorio de Usuario
11  * Usaremos esto hasta tener la base de datos en Clase 4
12  */
13 public class UsuarioRepositoryMock implements IUsuarioRepository {
14
15     private List<Usuario> usuarios;
16     private int nextId;
17
18     public UsuarioRepositoryMock() {
19         usuarios = new ArrayList<>();
20         nextId = 1;
21         cargarDatosIniciales();
22     }
23
24     private void cargarDatosIniciales() {
25         // Usuarios de ejemplo
26         usuarios.add(new Usuario(nextId++, "admin", "admin123",
27             "Administrador del Sistema", "ADMIN", true));
28         usuarios.add(new Usuario(nextId++, "operador1", "op123",
29             "Juan Pérez", "OPERADOR", true));
30         usuarios.add(new Usuario(nextId++, "operador2", "op456",
31             "María González", "OPERADOR", true));
32         usuarios.add(new Usuario(nextId++, "cajero", "caj123",
33             "Pedro Ramírez", "OPERADOR", false));
34     }
35
36     @Override
37     public Usuario buscarPorId(int id) {
38         return usuarios.stream()
39             .filter(u -> u.getId() == id)
40             .findFirst()
41             .orElse(null);
42     }
43
44     @Override
45     public Usuario buscarPorUsername(String username) {
46         return usuarios.stream()
47             .filter(u -> u.getUsername().equalsIgnoreCase(username))
48             .findFirst()
49             .orElse(null);
50     }
51
52     @Override
53     public List<Usuario> listarTodos() {
54         return new ArrayList<>(usuarios); // Copia defensiva
55     }
56
57     @Override
58     public List<Usuario> listarPorRol(String rol) {
59         return usuarios.stream()
60             .filter(u -> u.getRol().equalsIgnoreCase(rol))
61             .collect(Collectors.toList());
62     }
63
64     @Override
65     public int guardar(Usuario usuario) {
66         usuario.setId(nextId++);
67         usuarios.add(usuario);
68         return usuario.getId();
69     }
70
71     @Override
72     public void actualizar(Usuario usuario) {
73         Usuario existente = buscarPorId(usuario.getId());
74         if (existente != null) {
75             int index = usuarios.indexOf(existente);
76             usuarios.set(index, usuario);
77         }
78     }
79
80     @Override
81     public void eliminar(int id) {
82         usuarios.removeIf(u -> u.getId() == id);
83     }
84
85     @Override
86     public boolean existeUsername(String username) {
87         return usuarios.stream()
88             .anyMatch(u -> u.getUsername().equalsIgnoreCase(username));
89     }
90
91     @Override
92     public int contarActivosPorRol(String rol) {
93         return (int) usuarios.stream()
94             .filter(u -> u.getRol().equalsIgnoreCase(rol))
95             .filter(Usuario::isActivo)
96             .count();
97     }
98 }

```

```

1 package cl.enmanuelchirinos.pnb.repository.mock;
2
3 import cl.enmanuelchirinos.pnb.model.Producto;
4 import cl.enmanuelchirinos.pnb.repository.IProductoRepository;
5 import java.util.ArrayList;
6 import java.util.List;
7 import java.util.stream.Collectors;
8
9 /**
10  * Implementación Mock (en memoria) del repositorio de Producto
11  */
12 public class ProductoRepositoryMock implements IProductoRepository {
13
14     private List<Producto> productos;
15     private int nextId;
16
17     public ProductoRepositoryMock() {
18         productos = new ArrayList<>();
19         nextId = 1;
20         cargarDatosIniciales();
21     }
22
23     private void cargarDatosIniciales() {
24         // Bebidas
25         productos.add(new Producto(nextId++, "Espresso", "BEBIDA", "CAFE", 2500, true));
26         productos.add(new Producto(nextId++, "Cappuccino", "BEBIDA", "CAFE", 3000, true));
27         productos.add(new Producto(nextId++, "Latte", "BEBIDA", "CAFE", 3200, true));
28         productos.add(new Producto(nextId++, "Americano", "BEBIDA", "CAFE", 2800, true));
29         productos.add(new Producto(nextId++, "Coca-Cola", "BEBIDA", "GASEOSA", 1500, true));
30         productos.add(new Producto(nextId++, "Sprite", "BEBIDA", "GASEOSA", 1500, true));
31
32         // Snacks
33         productos.add(new Producto(nextId++, "Brownie", "SNACK", "POSTRE", 2000, true));
34         productos.add(new Producto(nextId++, "Cheesecake", "SNACK", "POSTRE", 2500, true));
35         productos.add(new Producto(nextId++, "Papas Fritas", "SNACK", "SALADO", 1800, true));
36         productos.add(new Producto(nextId++, "Nachos", "SNACK", "SALADO", 2200, true));
37
38         // Tiempo de Arcade
39         productos.add(new Producto(nextId++, "15 minutos", "TIEMPO", "ARCADE", 1500, true));
40         productos.add(new Producto(nextId++, "30 minutos", "TIEMPO", "ARCADE", 2500, true));
41         productos.add(new Producto(nextId++, "1 hora", "TIEMPO", "ARCADE", 4000, true));
42         productos.add(new Producto(nextId++, "2 horas", "TIEMPO", "ARCADE", 7000, true));
43     }
44
45     @Override
46     public Producto buscarPorId(int id) {
47         return productos.stream()
48             .filter(p -> p.getId() == id)
49             .findFirst()
50             .orElse(null);
51     }
52
53     @Override
54     public List<Producto> listarTodos() {
55         return new ArrayList<>(productos);
56     }
57
58     @Override
59     public List<Producto> listarActivos() {
60         return productos.stream()
61             .filter(Producto::isActive)
62             .collect(Collectors.toList());
63     }
64
65     @Override
66     public List<Producto> listarPorCategoria(String categoria) {
67         return productos.stream()
68             .filter(p -> p.getCategoria().equalsIgnoreCase(categoria))
69             .collect(Collectors.toList());
70     }
71
72     @Override
73     public List<Producto> buscarPorNombre(String nombre) {
74         String nombreLower = nombre.toLowerCase();
75         return productos.stream()
76             .filter(p -> p.getNombre().toLowerCase().contains(nombreLower))
77             .collect(Collectors.toList());
78     }
79
80     @Override
81     public int guardar(Producto producto) {
82         producto.setId(nextId++);
83         productos.add(producto);
84         return producto.getId();
85     }
86
87     @Override
88     public void actualizar(Producto producto) {
89         Producto existente = buscarPorId(producto.getId());
90         if (existente != null) {
91             int index = productos.indexOf(existente);
92             productos.set(index, producto);
93         }
94     }
95
96     @Override
97     public void eliminar(int id) {
98         productos.removeIf(p -> p.getId() == id);
99     }
100
101     @Override
102     public void cambiarEstado(int id, boolean activo) {
103         Producto producto = buscarPorId(id);
104         if (producto != null) {
105             producto.setActivo(activo);
106         }
107     }
108 }

```

```

1 package cl.enmanuelchirinos.pnb.repository.mock;
2
3 import cl.enmanuelchirinos.pnb.model.Venta;
4 import cl.enmanuelchirinos.pnb.repository.IVentaRepository;
5 import java.time.LocalDate;
6 import java.time.LocalDateTime;
7 import java.util.ArrayList;
8 import java.util.List;
9 import java.util.stream.Collectors;
10
11 /**
12  * Implementación Mock (en memoria) del repositorio de Venta
13  */
14 public class VentaRepositoryMock implements IVentaRepository {
15
16     private List<Venta> ventas;
17     private int nextId;
18
19     public VentaRepositoryMock() {
20         ventas = new ArrayList<>();
21         nextId = 1;
22         cargarDatosIniciales();
23     }
24
25     private void cargarDatosIniciales() {
26         // Ventas de ejemplo del día actual
27         LocalDateTime ahora = LocalDateTime.now();
28
29         ventas.add(new Venta(nextId++, ahora.minusHours(5), 1, "admin", 7500, "ACTIVA"));
30         ventas.add(new Venta(nextId++, ahora.minusHours(4), 2, "operador1", 5000, "ACTIVA"));
31         ventas.add(new Venta(nextId++, ahora.minusHours(3), 1, "admin", 12000, "ACTIVA"));
32         ventas.add(new Venta(nextId++, ahora.minusHours(2), 3, "operador2", 3500, "ACTIVA"));
33         ventas.add(new Venta(nextId++, ahora.minusHours(1), 2, "operador1", 8500, "ANULADA"));
34         ventas.add(new Venta(nextId++, ahora.minusMinutes(30), 1, "admin", 6000, "ACTIVA"));
35     }
36
37     @Override
38     public Venta buscarPorId(int id) {
39         return ventas.stream()
40             .filter(v -> v.getId() == id)
41             .findFirst()
42             .orElse(null);
43     }
44
45     @Override
46     public List<Venta> listarTodas() {
47         return new ArrayList<>(ventas);
48     }
49
50     @Override
51     public List<Venta> listarPorRangoFechas(LocalDateTime desde, LocalDateTime hasta) {
52         return ventas.stream()
53             .filter(v -> !v.getFechaHora().isBefore(desde) &&
54                 !v.getFechaHora().isAfter(hasta))
55             .collect(Collectors.toList());
56     }
57
58     @Override
59     public List<Venta> listarDelDia() {
60         LocalDateTime inicioDia = LocalDate.now().atStartOfDay();
61         LocalDateTime finDia = inicioDia.plusDays(1).minusSeconds(1);
62         return listarPorRangoFechas(inicioDia, finDia);
63     }
64
65     @Override
66     public List<Venta> listarPorUsuario(int usuarioId) {
67         return ventas.stream()
68             .filter(v -> v.getUsuarioId() == usuarioId)
69             .collect(Collectors.toList());
70     }
71
72     @Override
73     public int guardar(Venta venta) {
74         venta.setId(nextId++);
75         venta.setFechaHora(LocalDateTime.now());
76         venta.setEstado("ACTIVA");
77         ventas.add(venta);
78         return venta.getId();
79     }
80
81     @Override
82     public void anular(int id) {
83         Venta venta = buscarPorId(id);
84         if (venta != null) {
85             venta.setEstado("ANULADA");
86         }
87     }
88
89     @Override
90     public double calcularTotalPorRango(LocalDateTime desde, LocalDateTime hasta) {
91         return listarPorRangoFechas(desde, hasta).stream()
92             .filter(v -> "ACTIVA".equals(v.getEstado()))
93             .mapToDouble(Venta::getTotal)
94             .sum();
95     }
96
97     @Override
98     public double calcularTotalDelDia() {
99         return listarDelDia().stream()
100             .filter(v -> "ACTIVA".equals(v.getEstado()))
101             .mapToDouble(Venta::getTotal)
102             .sum();
103     }
104 }

```

```

1 package cl.emmanuelchirinos.pmb.service;
2
3 import cl.emmanuelchirinos.pmb.model.usuario;
4 import cl.emmanuelchirinos.pmb.repository.UsuarioRepository;
5 import java.util.List;
6 import java.util.stream.Collectors;
7
8 /**
9  * Servicio de lógica de negocio para usuario
10  * Contiene validaciones y reglas de negocio
11  */
12 public class UsuarioService {
13
14     private final UsuarioRepository repository;
15
16     /**
17      * Constructor con inyección de dependencias
18      */
19     public UsuarioService(UsuarioRepository repository) {
20         this.repository = repository;
21     }
22
23     /**
24      * Autentica un usuario
25      * @throws RuntimeException si las credenciales son inválidas
26      */
27     public Usuario autentica(String username, String password) {
28         // Validaciones básicas
29         if (username == null || username.trim().isEmpty()) {
30             throw new IllegalArgumentException("El username es obligatorio");
31         }
32         if (password == null || password.trim().isEmpty()) {
33             throw new IllegalArgumentException("La contraseña es obligatoria");
34         }
35
36         // Buscar usuario
37         Usuario usuario = repository.buscarPorUsername(username.trim());
38
39         if (usuario == null) {
40             throw new RuntimeException("Credenciales inválidas");
41         }
42
43         if (!usuario.isActivo()) {
44             throw new RuntimeException("Usuario inactivo. Contacta al administrador");
45         }
46
47         // Verificar contraseña (por ahora sin hash)
48         if (!usuario.getPassword().equals(password)) {
49             throw new RuntimeException("Credenciales inválidas");
50         }
51
52         return usuario;
53     }
54
55     /**
56      * Crea un nuevo usuario
57      */
58     public void crear(String username, String password, String nombreCompleto, String rol) {
59         // Validaciones
60         validarDatosUsuario(username, password, nombreCompleto, rol);
61
62         // Verificar que no exista el username
63         if (repository.existsByUsername(username)) {
64             throw new RuntimeException("El username '" + username + "' ya existe");
65         }
66
67         // Crear usuario
68         Usuario usuario = new Usuario();
69         usuario.setUsername(username.trim().toLowerCase());
70         usuario.setPassword(password); // TODO: Hash en Clase S
71         usuario.setNombreCompleto(nombreCompleto.trim());
72         usuario.setRol(rol);
73         usuario.setActivo(true);
74
75         repository.guardar(usuario);
76     }
77
78     /**
79      * Actualiza un usuario existente
80      */
81     public void actualizar(int id, String username, String password,
82                           String nombreCompleto, String rol, boolean activo) {
83         // Validaciones
84         validarDatosUsuario(username, password, nombreCompleto, rol);
85
86         // Buscar usuario existente
87         Usuario usuario = repository.buscarPorId(id);
88         if (usuario == null) {
89             throw new RuntimeException("Usuario no encontrado");
90         }
91
92         // Verificar username único (si cambió)
93         if (!usuario.getUsername().equals(username) &&
94             repository.existsByUsername(username)) {
95             throw new RuntimeException("El username '" + username + "' ya existe");
96         }
97
98         // Actualizar datos
99         usuario.setUsername(username.trim().toLowerCase());
100         if (password != null && !password.trim().isEmpty()) {
101             usuario.setPassword(password);
102         }
103         usuario.setNombreCompleto(nombreCompleto.trim());
104         usuario.setRol(rol);
105         usuario.setActivo(activo);
106
107         repository.actualizar(usuario);
108     }
109
110     /**
111      * Elimina un usuario
112      * Regla de negocio: No se puede eliminar el último ADMIN activo
113      */
114     public void eliminar(int id) {
115         Usuario usuario = repository.buscarPorId(id);
116         if (usuario == null) {
117             throw new RuntimeException("Usuario no encontrado");
118         }
119
120         // Validar que no sea el último admin activo
121         if (usuario.getRol().equals("ADMIN") && usuario.isActivo()) {
122             int adminsActivos = repository.contarActivosPorRol("ADMIN");
123             if (adminsActivos == 1) {
124                 throw new RuntimeException(
125                     "No se puede eliminar el último administrador activo");
126             }
127         }
128
129         repository.eliminar(id);
130     }
131
132     /**
133      * Lista todos los usuarios
134      */
135     public List<Usuario> listarTodos() {
136         return repository.listarTodos();
137     }
138
139     /**
140      * Lista usuarios activos
141      */
142     public List<Usuario> listarActivos() {
143         return repository.listarTodos().stream()
144             .filter(Usuario::isActivo)
145             .collect(Collectors.toList());
146     }
147
148     /**
149      * Busca usuarios por texto (username o nombre)
150      */
151     public List<Usuario> buscar(String texto) {
152         if (texto == null || texto.trim().isEmpty()) {
153             return listarTodos();
154         }
155
156         String textoLower = texto.toLowerCase();
157         return repository.listarTodos().stream()
158             .filter(u -> u.getUsername().toLowerCase().contains(textoLower) ||
159                 u.getNombreCompleto().toLowerCase().contains(textoLower))
160             .collect(Collectors.toList());
161     }
162
163     /**
164      * Validaciones comunes de usuario
165      */
166     private void validarDatosUsuario(String username, String password,
167                                     String nombreCompleto, String rol) {
168         if (username == null || username.trim().isEmpty()) {
169             throw new IllegalArgumentException("El username es obligatorio");
170         }
171         if (username.trim().length() < 4) {
172             throw new IllegalArgumentException(
173                 "El username debe tener al menos 4 caracteres");
174         }
175
176         if (password == null || password.trim().isEmpty()) {
177             throw new IllegalArgumentException("La contraseña es obligatoria");
178         }
179         if (password.length() < 6) {
180             throw new IllegalArgumentException(
181                 "La contraseña debe tener al menos 6 caracteres");
182         }
183
184         if (nombreCompleto == null || nombreCompleto.trim().isEmpty()) {
185             throw new IllegalArgumentException("El nombre completo es obligatorio");
186         }
187
188         if (rol == null || rol.trim().isEmpty()) {
189             throw new IllegalArgumentException("El rol es obligatorio");
190         }
191         if (!rol.equals("ADMIN") && !rol.equals("OPERADOR")) {
192             throw new IllegalArgumentException(
193                 "El rol debe ser ADMIN u OPERADOR");
194         }
195     }
196 }

```

```

1 package cl.enmanuelchirinos.pnb.service;
2
3 import cl.enmanuelchirinos.pnb.model.Producto;
4 import cl.enmanuelchirinos.pnb.repository.IProductoRepository;
5 import java.util.List;
6
7 /**
8  * Servicio de Lógica de negocio para Producto
9  */
10 public class ProductoService {
11
12     private final IProductoRepository repository;
13
14     public ProductoService(IProductoRepository repository) {
15         this.repository = repository;
16     }
17
18     /**
19      * Crea un nuevo producto
20      */
21     public void crear(String nombre, String categoria, String tipo, double precio) {
22         // Validaciones
23         validarDatosProducto(nombre, categoria, tipo, precio);
24
25         // Crear producto
26         Producto producto = new Producto();
27         producto.setNombre(nombre.trim());
28         producto.setCategoria(categoria);
29         producto.setTipo(tipo);
30         producto.setPrecio(precio);
31         producto.setActivo(true);
32
33         repository.guardar(producto);
34     }
35
36     /**
37      * Actualiza un producto existente
38      */
39     public void actualizar(int id, String nombre, String categoria,
40         String tipo, double precio, boolean activo) {
41         // Validaciones
42         validarDatosProducto(nombre, categoria, tipo, precio);
43
44         // Buscar producto
45         Producto producto = repository.buscarPorId(id);
46         if (producto == null) {
47             throw new RuntimeException("Producto no encontrado");
48         }
49
50         // Actualizar
51         producto.setNombre(nombre.trim());
52         producto.setCategoria(categoria);
53         producto.setTipo(tipo);
54         producto.setPrecio(precio);
55         producto.setActivo(activo);
56
57         repository.actualizar(producto);
58     }
59
60     /**
61      * Elimina un producto
62      */
63     public void eliminar(int id) {
64         Producto producto = repository.buscarPorId(id);
65         if (producto == null) {
66             throw new RuntimeException("Producto no encontrado");
67         }
68
69         repository.eliminar(id);
70     }
71
72     /**
73      * Cambia el estado de un producto
74      */
75     public void cambiarEstado(int id) {
76         Producto producto = repository.buscarPorId(id);
77         if (producto == null) {
78             throw new RuntimeException("Producto no encontrado");
79         }
80
81         repository.cambiarEstado(id, !producto.isActivo());
82     }
83
84     /**
85      * Lista todos los productos
86      */
87     public List<Producto> listarTodos() {
88         return repository.listarTodos();
89     }
90
91     /**
92      * Lista productos activos
93      */
94     public List<Producto> listarActivos() {
95         return repository.listarActivos();
96     }
97
98     /**
99      * Lista productos por categoria
100     */
101     public List<Producto> listarPorCategoria(String categoria) {
102         return repository.listarPorCategoria(categoria);
103     }
104
105     /**
106      * Busca productos por nombre
107     */
108     public List<Producto> buscarPorNombre(String nombre) {
109         return repository.buscarPorNombre(nombre);
110     }
111
112     /**
113      * Validaciones de producto
114     */
115     private void validarDatosProducto(String nombre, String categoria,
116         String tipo, double precio) {
117         if (nombre == null || nombre.trim().isEmpty()) {
118             throw new IllegalArgumentException("El nombre es obligatorio");
119         }
120
121         if (categoria == null || categoria.trim().isEmpty()) {
122             throw new IllegalArgumentException("La categoria es obligatoria");
123         }
124         if (!categoria.equals("BEBIDA") && !categoria.equals("SNACK") &&
125             !categoria.equals("TIEMPO")) {
126             throw new IllegalArgumentException(
127                 "Categoria invalida. Debe ser BEBIDA, SNACK o TIEMPO");
128         }
129
130         if (tipo == null || tipo.trim().isEmpty()) {
131             throw new IllegalArgumentException("El tipo es obligatorio");
132         }
133
134         if (precio <= 0) {
135             throw new IllegalArgumentException("El precio debe ser mayor a 0");
136         }
137     }
138 }

```

```

1 package cl.enmanuelchirinos.pnb.service;
2
3 import cl.enmanuelchirinos.pnb.model.Venta;
4 import cl.enmanuelchirinos.pnb.repository.IVentaRepository;
5 import java.time.LocalDateTime;
6 import java.util.List;
7
8 /**
9  * Servicio de Lógica de negocio para Venta
10  */
11 public class VentaService {
12
13     private final IVentaRepository repository;
14
15     public VentaService(IVentaRepository repository) {
16         this.repository = repository;
17     }
18
19     /**
20      * Registra una nueva venta
21      */
22     public int registrarVenta(int usuarioId, String usuarioNombre, double total) {
23         // Validaciones
24         if (usuarioId <= 0) {
25             throw new IllegalArgumentException("Usuario inválido");
26         }
27         if (usuarioNombre == null || usuarioNombre.trim().isEmpty()) {
28             throw new IllegalArgumentException("Nombre de usuario es obligatorio");
29         }
30         if (total <= 0) {
31             throw new IllegalArgumentException("El total debe ser mayor a 0");
32         }
33
34         // Crear venta
35         Venta venta = new Venta();
36         venta.setUsuarioId(usuarioId);
37         venta.setUsuarioNombre(usuarioNombre);
38         venta.setTotal(total);
39
40         return repository.guardar(venta);
41     }
42
43     /**
44      * Anula una venta
45      */
46     public void anularVenta(int id) {
47         Venta venta = repository.buscarPorId(id);
48         if (venta == null) {
49             throw new RuntimeException("Venta no encontrada");
50         }
51
52         if ("ANULADA".equals(venta.getEstado())) {
53             throw new RuntimeException("La venta ya está anulada");
54         }
55
56         repository.anular(id);
57     }
58
59     /**
60      * Lista todas las ventas
61      */
62     public List<Venta> listarTodas() {
63         return repository.listarTodas();
64     }
65
66     /**
67      * Lista ventas del día
68      */
69     public List<Venta> listarDelDia() {
70         return repository.listarDelDia();
71     }
72
73     /**
74      * Lista ventas por rango de fechas
75      */
76     public List<Venta> listarPorRango(LocalDateTime desde, LocalDateTime hasta) {
77         if (desde == null || hasta == null) {
78             throw new IllegalArgumentException("Las fechas son obligatorias");
79         }
80         if (desde.isAfter(hasta)) {
81             throw new IllegalArgumentException("La fecha desde debe ser anterior a la fecha hasta");
82         }
83
84         return repository.listarPorRangoFechas(desde, hasta);
85     }
86
87     /**
88      * Lista ventas por usuario
89      */
90     public List<Venta> listarPorUsuario(int usuarioId) {
91         if (usuarioId <= 0) {
92             throw new IllegalArgumentException("Usuario inválido");
93         }
94
95         return repository.listarPorUsuario(usuarioId);
96     }
97
98     /**
99      * Calcula el total de ventas del día
100     */
101     public double calcularTotalDelDia() {
102         return repository.calcularTotalDelDia();
103     }
104
105     /**
106      * Calcula el total de ventas por rango
107     */
108     public double calcularTotalPorRango(LocalDateTime desde, LocalDateTime hasta) {
109         if (desde == null || hasta == null) {
110             throw new IllegalArgumentException("Las fechas son obligatorias");
111         }
112         if (desde.isAfter(hasta)) {
113             throw new IllegalArgumentException("La fecha desde debe ser anterior a la fecha hasta");
114         }
115
116         return repository.calcularTotalPorRango(desde, hasta);
117     }
118 }

```

```

1 package cl.enmanuelchirinos.pnb.controller;
2
3 import cl.enmanuelchirinos.pnb.model.Usuario;
4 import cl.enmanuelchirinos.pnb.service.UsuarioService;
5 import java.util.List;
6
7 /**
8  * Controlador para coordinar operaciones de Usuario
9  * Actúa como intermediario entre la vista y el servicio
10 */
11 public class UsuarioController {
12
13     private final UsuarioService service;
14
15     /**
16      * Constructor con inyección de dependencias
17      */
18     public UsuarioController(UsuarioService service) {
19         this.service = service;
20     }
21
22     /**
23      * Autentica un usuario
24      */
25     public Usuario autenticar(String username, String password) {
26         return service.autenticar(username, password);
27     }
28
29     /**
30      * Crea un nuevo usuario
31      */
32     public void crearUsuario(String username, String password,
33                             String nombreCompleto, String rol) {
34         service.crear(username, password, nombreCompleto, rol);
35     }
36
37     /**
38      * Actualiza un usuario existente
39      */
40     public void actualizarUsuario(int id, String username, String password,
41                                   String nombreCompleto, String rol, boolean activo) {
42         service.actualizar(id, username, password, nombreCompleto, rol, activo);
43     }
44
45     /**
46      * Elimina un usuario
47      */
48     public void eliminarUsuario(int id) {
49         service.eliminar(id);
50     }
51
52     /**
53      * Lista todos los usuarios
54      */
55     public List<Usuario> listarTodos() {
56         return service.listarTodos();
57     }
58
59     /**
60      * Lista usuarios activos
61      */
62     public List<Usuario> listarActivos() {
63         return service.listarActivos();
64     }
65
66     /**
67      * Busca usuarios por texto
68      */
69     public List<Usuario> buscar(String texto) {
70         return service.buscar(texto);
71     }
72 }

```

```

1 package cl.enmanuelchirinos.pnb.controller;
2
3 import cl.enmanuelchirinos.pnb.model.Producto;
4 import cl.enmanuelchirinos.pnb.service.ProductoService;
5 import java.util.List;
6
7 /**
8  * Controlador para coordinar operaciones de Producto
9  */
10 public class ProductoController {
11
12     private final ProductoService service;
13
14     public ProductoController(ProductoService service) {
15         this.service = service;
16     }
17
18     /**
19      * Crea un nuevo producto
20      */
21     public void crearProducto(String nombre, String categoria, String tipo, double precio) {
22         service.crear(nombre, categoria, tipo, precio);
23     }
24
25     /**
26      * Actualiza un producto existente
27      */
28     public void actualizarProducto(int id, String nombre, String categoria,
29                                     String tipo, double precio, boolean activo) {
30         service.actualizar(id, nombre, categoria, tipo, precio, activo);
31     }
32
33     /**
34      * Elimina un producto
35      */
36     public void eliminarProducto(int id) {
37         service.eliminar(id);
38     }
39
40     /**
41      * Cambia el estado de un producto
42      */
43     public void cambiarEstadoProducto(int id) {
44         service.cambiarEstado(id);
45     }
46
47     /**
48      * Lista todos los productos
49      */
50     public List<Producto> listarTodos() {
51         return service.listarTodos();
52     }
53
54     /**
55      * Lista productos activos
56      */
57     public List<Producto> listarActivos() {
58         return service.listarActivos();
59     }
60
61     /**
62      * Lista productos por categoría
63      */
64     public List<Producto> listarPorCategoria(String categoria) {
65         return service.listarPorCategoria(categoria);
66     }
67
68     /**
69      * Busca productos por nombre
70      */
71     public List<Producto> buscarPorNombre(String nombre) {
72         return service.buscarPorNombre(nombre);
73     }
74 }

```

```

1 package cl.enmanuelchirinos.pnb.controller;
2
3 import cl.enmanuelchirinos.pnb.model.Venta;
4 import cl.enmanuelchirinos.pnb.service.VentaService;
5 import java.time.LocalDateTime;
6 import java.util.List;
7
8 /**
9  * Controlador para coordinar operaciones de Venta
10  */
11 public class VentaController {
12
13     private final VentaService service;
14
15     public VentaController(VentaService service) {
16         this.service = service;
17     }
18
19     /**
20      * Registra una nueva venta
21      */
22     public int registrarVenta(int usuarioId, String usuarioNombre, double total) {
23         return service.registrarVenta(usuarioId, usuarioNombre, total);
24     }
25
26     /**
27      * Anula una venta
28      */
29     public void anularVenta(int id) {
30         service.anularVenta(id);
31     }
32
33     /**
34      * Lista todas las ventas
35      */
36     public List<Venta> listarTodas() {
37         return service.listarTodas();
38     }
39
40     /**
41      * Lista ventas del día
42      */
43     public List<Venta> listarDelDia() {
44         return service.listarDelDia();
45     }
46
47     /**
48      * Lista ventas por rango de fechas
49      */
50     public List<Venta> listarPorRango(LocalDateTime desde, LocalDateTime hasta) {
51         return service.listarPorRango(desde, hasta);
52     }
53
54     /**
55      * Lista ventas por usuario
56      */
57     public List<Venta> listarPorUsuario(int usuarioId) {
58         return service.listarPorUsuario(usuarioId);
59     }
60
61     /**
62      * Calcula el total de ventas del día
63      */
64     public double calcularTotalDelDia() {
65         return service.calcularTotalDelDia();
66     }
67
68     /**
69      * Calcula el total de ventas por rango
70      */
71     public double calcularTotalPorRango(LocalDateTime desde, LocalDateTime hasta) {
72         return service.calcularTotalPorRango(desde, hasta);
73     }
74 }

```

```

1 package cl.enmanuelchirinos.pnb.app;
2
3 import cl.enmanuelchirinos.pnb.controller.ProductoController;
4 import cl.enmanuelchirinos.pnb.controller.UsuarioController;
5 import cl.enmanuelchirinos.pnb.controller.VentaController;
6 import cl.enmanuelchirinos.pnb.repository.IProductoRepository;
7 import cl.enmanuelchirinos.pnb.repository.IUsuarioRepository;
8 import cl.enmanuelchirinos.pnb.repository.IVentaRepository;
9 import cl.enmanuelchirinos.pnb.repository.mock.ProductoRepositoryMock;
10 import cl.enmanuelchirinos.pnb.repository.mock.UsuarioRepositoryMock;
11 import cl.enmanuelchirinos.pnb.repository.mock.VentaRepositoryMock;
12 import cl.enmanuelchirinos.pnb.service.ProductoService;
13 import cl.enmanuelchirinos.pnb.service.UsuarioService;
14 import cl.enmanuelchirinos.pnb.service.VentaService;
15
16 /**
17  * Contenedor de Inversión de Control (IoC Container)
18  * Gestiona la creación e inyección de dependencias de toda la aplicación
19  *
20  * Patrón Singleton: Solo existe una instancia en toda la aplicación
21  */
22 public class ApplicationContext {
23
24     // Instancia única (Singleton)
25     private static ApplicationContext instance;
26
27     // ===== CAPA DE REPOSITORIOS =====
28     private IUsuarioRepository usuarioRepository;
29     private IProductoRepository productoRepository;
30     private IVentaRepository ventaRepository;
31
32     // ===== CAPA DE SERVICIOS =====
33     private UsuarioService usuarioService;
34     private ProductoService productoService;
35     private VentaService ventaService;
36
37     // ===== CAPA DE CONTROLADORES =====
38     private UsuarioController usuarioController;
39     private ProductoController productoController;
40     private VentaController ventaController;
41
42     /**
43      * Constructor privado (Singleton)
44      * Inicializa todas las capas de la aplicación
45      */
46     private ApplicationContext() {
47         inicializarRepositorios();
48         inicializarServicios();
49         inicializarControladores();
50     }
51
52     /**
53      * Inicializa la capa de repositorios
54      * Por ahora usamos implementaciones Mock
55      * En Clase 4 cambiaremos a implementaciones JDBC
56      */
57     private void inicializarRepositorios() {
58         System.out.println("[ApplicationContext] Inicializando repositorios Mock...");
59
60         // Implementaciones Mock (en memoria)
61         usuarioRepository = new UsuarioRepositoryMock();
62         productoRepository = new ProductoRepositoryMock();
63         ventaRepository = new VentaRepositoryMock();
64
65         System.out.println("[ApplicationContext] @ Repositorios inicializados");
66     }
67
68     /**
69      * Inicializa la capa de servicios
70      * Inyecta los repositorios en los servicios
71      */
72     private void inicializarServicios() {
73         System.out.println("[ApplicationContext] Inicializando servicios...");
74
75         // Inyección de dependencias por constructor
76         usuarioService = new UsuarioService(usuarioRepository);
77         productoService = new ProductoService(productoRepository);
78         ventaService = new VentaService(ventaRepository);
79
80         System.out.println("[ApplicationContext] @ Servicios inicializados");
81     }
82
83     /**
84      * Inicializa la capa de controladores
85      * Inyecta los servicios en los controladores
86      */
87     private void inicializarControladores() {
88         System.out.println("[ApplicationContext] Inicializando controladores...");
89
90         // Inyección de dependencias por constructor
91         usuarioController = new UsuarioController(usuarioService);
92         productoController = new ProductoController(productoService);
93         ventaController = new VentaController(ventaService);
94
95         System.out.println("[ApplicationContext] @ Controladores inicializados");
96     }
97
98     /**
99      * Obtiene la instancia única del ApplicationContext (Singleton)
100      */
101     public static ApplicationContext getInstance() {
102         if (instance == null) {
103             System.out.println("[ApplicationContext] Creando instancia única...");
104             instance = new ApplicationContext();
105             System.out.println("[ApplicationContext] @ Aplicación inicializada completamente\n");
106         }
107         return instance;
108     }
109
110     // ===== GETTERS PARA CONTROLADORES =====
111     // Las vistas solo deben usar controladores, no servicios ni repositorios
112     public UsuarioController getUsuarioController() {
113         return usuarioController;
114     }
115
116     public ProductoController getProductoController() {
117         return productoController;
118     }
119
120     public VentaController getVentaController() {
121         return ventaController;
122     }
123
124     // ===== MÉTODOS PARA TESTING (opcional) =====
125
126     /**
127      * Reinicia el contexto (útil para tests)
128      */
129     public static void reset() {
130         instance = null;
131     }
132 }
133

```