

Parte 1: Principios de la POO(UF2404)

Desarrolla una aplicacion de gestion de biblioteca en Java que permita:

- Registrar libros(titulo,autor,ISBN,año)
- Registrar usuarios(nombre,DNI,correo)
- Prestar libros a usuarios (fecha de prestamo y devolucion)

Requisitos tecnicos:

- Requisitos tecnicos:
 - Uso de clases herencia, encapsulamiento y polimorfismo.
 - Interfaz por consola

```
BIBLIOTECA

MENÚ PRINCIPAL
1. Registrar Libros
2. Registrar Usuarios
3. Prestar Libros a usuarios
0. Salir

Opción: 1
REGISTRO DE NUEVO LIBRO

Título: EL VIENTO
Autor: PEPITO GONZALEZ
ISBN: 123654
Año: 1965

Hibernate: insert into libros (anio,autor,isbn,titulo) values (?,?,,?)

OPERACIÓN COMPLETADA CON ÉXITO

Libro "EL VIENTO" registrado correctamente con ID: 1
Presione Enter para continuar...
```

```
BIBLIOTECA

MENÚ PRINCIPAL
1. Registrar Libros
2. Registrar Usuarios
3. Prestar Libros a usuarios
0. Salir

Opción: 2
REGISTRO DE NUEVO USUARIO

Nombre: JUAN SANCHEZ
DNI: 45859525H
Correo: juansanchez@gmail.com

Hibernate: insert into usuarios (correo,dni,nombre,tipo_usuario) values (?,?,, 'REGULAR')

OPERACIÓN COMPLETADA CON ÉXITO

Usuario "JUAN SANCHEZ" registrado correctamente con ID: 2
Presione Enter para continuar...
```

```
1. Registrar Libros
2. Registrar Usuarios
3. Prestar Libros a usuarios
0. Salir

Opción: 3
REGISTRO DE NUEVO PRÉSTAMO

Hibernate: select u1_0.id,u1_0.tipo_usuario,u1_0.correo,u1_0.dni,u1_0.nombre from usuarios u1_0
Hibernate: select l1_0.id,l1_0.anio,l1_0.autor,l1_0.isbn,l1_0.titulo from libros l1_0

SELECCIONE USUARIO

IDX | NOMBRE | DNI | CORREO |
0 | pepito perez | 45454545L | pepitoperez@gmail.com |
1 | JUAN SANCHEZ | 45859525H | juansanchez@gmail.com |

Seleccione usuario (índice o nombre): pepito perez

SELECCIONE LIBRO

IDX | TÍTULO | AUTOR | ISBN | AÑO |
0 | EL VIENTO | PEPITO GONZALEZ | 123654 | 1965 |

Seleccione libro (índice o título): EL VIENTO

FECHAS DE PRÉSTAMO

Fecha de préstamo predeterminada: 2025-06-25 (fecha actual)
¿Desea usar esta fecha? (S/N): S

Fecha de devolución predeterminada: 2025-07-10 (15 días después)
¿Desea usar esta fecha? (S/N): s
```

1. Método para mostrar libros

```

- System.out.println(LIBROS_BANNER);
- List<Libro> libros = libroRepository.findAll();
-
- if (libros.isEmpty()) {
-     System.out.println("\nNo hay libros registrados en el sistema.\n");
-     return;
- }
-
- System.out.println("
+ _____
+ | ID | TÍTULO | AUTOR | ISBN
+ | AÑO |");
-
- System.out.println("
+ _____
+ |_____|_|_|_|");
-
- for (Libro libro : libros) {
-     System.out.printf("
+ | %-7d | %-26s | %-18s | %-12s | %-4d | \n",
+     libro.getId(),
+     truncateString(libro.getTitulo(), 26),
+     truncateString(libro.getAutor(), 18),
+     truncateString(libro.getIsbn(), 12),
+     libro.getAnio());
- }
-
- System.out.println("
+ _____
+ |_____|_|_|_|");
-
- System.out.println();
- esperarEnter();
- }

```

2. Método para mostrar usuarios

```
- public void mostrarUsuarios() {  
-     System.out.println(USUARIOS_BANNER);  
-     List<Usuario> usuarios = usuarioRepository.findAll();  
-  
-     if (usuarios.isEmpty()) {  
-         System.out.println("\nNo hay usuarios registrados en el sistema.\n");  
-         return;  
-     }  
-  
-  
-  
-     System.out.println("┌────────────────────────────────────────────────────────────────────────────────┐");  
-     System.out.println("│ ID │ NOMBRE │ DNI │");  
-     System.out.println("CORREO │");  
-  
-     System.out.println("┌────────────────────────────────────────────────────────────────────────────────┐");  
-     System.out.println("└────────────────────────────────────────────────────────────────────────────────┘");  
-  
-     for (Usuario usuario : usuarios) {  
-         System.out.printf("│ %-7d │ %-26s │ %-14s │ %-23s │\n",  
-             usuario.getId(),  
-             truncateString(usuario.getNombre(), 26),  
-             truncateString(usuario.getDni(), 14),  
-             truncateString(usuario.getCorreo(), 23));  
-     }  
-  
-  
-  
-     System.out.println("┌────────────────────────────────────────────────────────────────────────────────┐");  
-     System.out.println("└────────────────────────────────────────────────────────────────────────────────┘");  
-  
-     System.out.println();  
-     esperarEnter();  
- }
```

3. Método para mostrar préstamos

```
public void mostrarPrestamos() {
    System.out.println(PRESTAMOS_BANNER);
    List<Prestamo> prestamos = prestamoRepository.findAll();

    if (prestamos.isEmpty()) {
        System.out.println("\nNo hay préstamos registrados en el sistema.\n");
        return;
    }
}
```

```
System.out.println("ID | USUARIO | LIBRO |");
System.out.println("PRÉSTAMO | DEVOLUCIÓN |");
```

```
System.out.println(" |-----|-----|-----|
```

```
|-----|");
```

```
for (Prestamo prestamo : prestamos) {
    System.out.printf("| %-7d | %-26s | %-26s | %-12s | %-12s | \n",
        prestamo.getId(),
        truncateString(prestamo.getUsuario().getNombre(), 26),
        truncateString(prestamo.getLibro().getTitulo(), 26),
        prestamo.getFechaPrestamo(),
        prestamo.getFechaDevolucion());
}
```

```
System.out.println("
System.out.println();
esperarEnter();
}
```

Parte 2: Programación Web y BBDD (UF2405)

Conecta la aplicación a una base de datos relacional y desarrolla una API REST básica con Sprint Boot.

Requisitos Tecnicos:

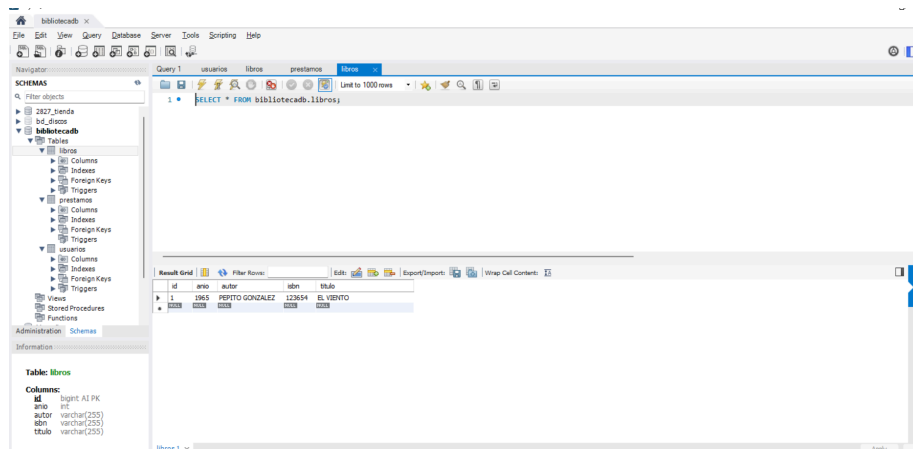
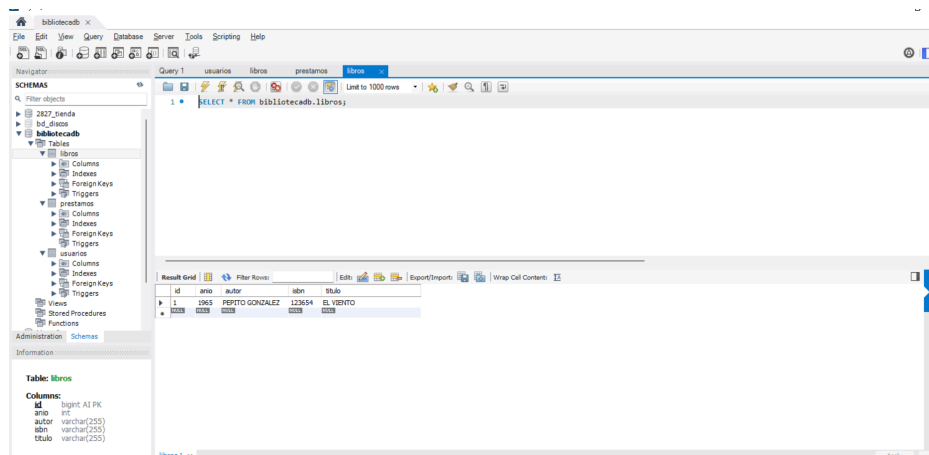
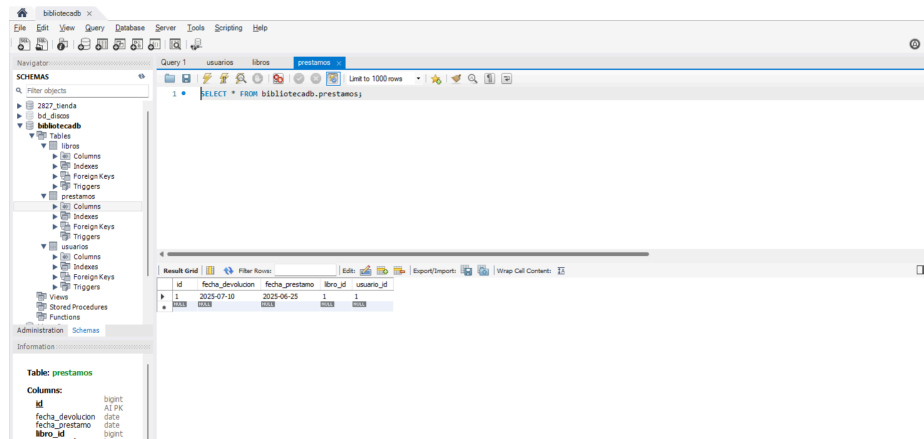
Base de Datos MySQL o SQLite con Tablas "Libros", "usuarios", "prestamos".

API REST con endpoints como:

GET/libros


POST/prestamos

GET/usuarios/{id}/prestamos



Sistema de Biblioteca

 Libros

 Usuarios

 Préstamos

Bienvenido al Sistema de Biblioteca

Seleccione una opción del menú para comenzar.



Gestión de Libros

Registrar, consultar y administrar libros



Gestión de Usuarios

Registrar, consultar y administrar usuarios



Gestión de Préstamos

Registrar, consultar y administrar préstamos

Tabla endpoints

Método	URL	Descripción
GET	/libros	Obtiene todos los libros
GET	/libros/{id}	Obtiene un libro por su ID
POST	/libros	Crea un nuevo libro
GET	/usuarios	Obtiene todos los usuarios
GET	/usuarios/{id}	Obtiene un usuario por su ID
POST	/usuarios	Crea un nuevo usuario
GET	/prestamos	Obtiene todos los préstamos
GET	/prestamos/{id}	Obtiene un préstamo por su ID
GET	/prestamos/usuario/{usuarioid}	Obtiene todos los préstamos de un usuario
POST	/prestamos	Crea un nuevo préstamo

application.properties

```
# Configuración de la base de datos MySQL
spring.datasource.url=jdbc:mysql://localhost:3306/bibliotecadb?createDatabaseIf
NotExist=true&useSSL=false&serverTimezone=UTC
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=1234
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
# Configuración JPA
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Entidad Libro

```
@Entity
@Table(name = "usuarios")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "tipo_usuario")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    protected String nombre;
    protected String dni;
    protected String correo;
    // ...
}
```

Entidad Usuario

```
@Entity
@Table(name = "prestamos")
public class Prestamo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "usuario_id")
    private Usuario usuario;

    @ManyToOne
    @JoinColumn(name = "libro_id")
    private Libro libro;

    private LocalDate fechaPrestamo;
    private LocalDate fechaDevolucion;
    // ...
}
```

Entidad préstamo

```
@Entity
@Table(name = "prestamos")
public class Prestamo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "usuario_id")
    private Usuario usuario;

    @ManyToOne
    @JoinColumn(name = "libro_id")
    private Libro libro;

    private LocalDate fechaPrestamo;
    private LocalDate fechaDevolucion;
    // ...
}
```


Repositorios JPA

```
@Repository
public interface LibroRepository extends JpaRepository<Libro, Long> {
    // Métodos de consulta personalizados si son necesarios
}

@Repository
public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
    // Métodos de consulta personalizados si son necesarios
}

@Repository
public interface PrestamoRepository extends JpaRepository<Prestamo, Long> {
    // Método para encontrar préstamos por usuario
    List<Prestamo> findByUsuario(Usuario usuario);

    // Método para encontrar préstamos por ID de usuario
    List<Prestamo> findByUsuarioId(Long usuarioId);
}
```

Controladores REST

LibroController

```
@RestController
@RequestMapping("/libros")
public class LibroController {
    private final BibliotecaService bibliotecaService;

    @Autowired
    public LibroController(BibliotecaService bibliotecaService) {
        this.bibliotecaService = bibliotecaService;
    }

    @GetMapping
    public ResponseEntity<List<Libro>> getAllLibros() {
        return ResponseEntity.ok(bibliotecaService.getAllLibros());
    }

    @GetMapping("/{id}")
    public ResponseEntity<Libro> getLibroById(@PathVariable Long id) {
        Libro libro = bibliotecaService.getLibroById(id);
        if (libro != null) {
            return ResponseEntity.ok(libro);
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    @PostMapping
    public ResponseEntity<Libro> createLibro(@RequestBody Libro libro) {
        return new ResponseEntity<>(bibliotecaService.saveLibro(libro),
        HttpStatus.CREATED);
    }
}
```

UsuarioController

```
@RestController
@RequestMapping("/usuarios")
public class UsuarioController {

    private final BibliotecaService bibliotecaService;

    @Autowired
    public UsuarioController(BibliotecaService bibliotecaService) {
        this.bibliotecaService = bibliotecaService;
    }

    @GetMapping
    public ResponseEntity<List<Usuario>> getAllUsuarios() {
        return ResponseEntity.ok(bibliotecaService.getAllUsuarios());
    }

    @GetMapping("/{id}")
    public ResponseEntity<Usuario> getUsuarioById(@PathVariable Long id) {
        Usuario usuario = bibliotecaService.getUsuarioById(id);
        if (usuario != null) {
            return ResponseEntity.ok(usuario);
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    @PostMapping
    public ResponseEntity<Usuario> createUsuario(@RequestBody UsuarioRegular
usuario) {
        return new ResponseEntity<>(bibliotecaService.saveUsuario(usuario),
HttpStatus.CREATED);
    }
}
```

PrestamoController

```
@RestController
@RequestMapping("/prestamos")
public class PrestamoController {

    private final BibliotecaService bibliotecaService;

    @Autowired
    public PrestamoController(BibliotecaService bibliotecaService) {
        this.bibliotecaService = bibliotecaService;
    }
}
```

```
@GetMapping
public ResponseEntity<List<Prestamo>> getAllPrestamos() {
    return ResponseEntity.ok(bibliotecaService.getAllPrestamos());
}

@GetMapping("/{id}")
public ResponseEntity<Prestamo> getPrestamoById(@PathVariable Long id) {
    Prestamo prestamo = bibliotecaService.getPrestamoById(id);
    if (prestamo != null) {
        return ResponseEntity.ok(prestamo);
    } else {
        return ResponseEntity.notFound().build();
    }
}

@GetMapping("/usuario/{usuarioId}")
public ResponseEntity<List<Prestamo>> getPrestamosByUsuarioId(@PathVariable
Long usuarioId) {
    return
ResponseEntity.ok(bibliotecaService.getPrestamosByUsuarioId(usuarioId));
}

@PostMapping
public ResponseEntity<Prestamo> createPrestamo(@RequestBody Prestamo
prestamo) {
    return new ResponseEntity<>(bibliotecaService.savePrestamo(prestamo),
HttpStatus.CREATED);
}
}
```

Parte 3: Ciclo de vida del Desarrollo(UF2406)

Tareas:

- Documento Técnico que incluya:
- Análisis de requisitos.
- Diagrama UML
- Diseño Base de datos
- Planificación del desarrollo
- Pruebas realizadas

Análisis de Requisitos del Sistema de Biblioteca

El análisis de requisitos del sistema de biblioteca se divide en dos categorías principales: requisitos funcionales y no funcionales.

Requisitos Funcionales

1. Gestión de Libros

- Registro de libros : El sistema permite registrar nuevos libros con información como título, autor, ISBN y año de publicación.
- Consulta de libros : Posibilidad de consultar los libros existentes en el sistema.

2. Gestión de Usuarios

- Registro de usuarios : El sistema permite registrar nuevos usuarios con datos como nombre, DNI y correo electrónico.
- Consulta de usuarios : Capacidad para consultar los usuarios registrados en el sistema.

3. Gestión de Préstamos

- Registro de préstamos : Funcionalidad para registrar préstamos de libros a usuarios.
- Fechas de préstamo : Establecimiento de fechas de préstamo y devolución.
- Consulta de préstamos : Posibilidad de consultar los préstamos existentes.

4. Interfaz de Usuario

- Interfaz por consola : Interfaz de línea de comandos para operaciones básicas del sistema.
- API REST : Interfaz de programación para integración con otros sistemas.

Requisitos No Funcionales

1. Persistencia

- Base de datos relacional : Almacenamiento de datos en una base de datos relacional (MySQL).
- JPA/Hibernate : Uso de JPA/Hibernate para el mapeo objeto-relacional.

2. Arquitectura

- Spring Boot : Aplicación basada en el framework Spring Boot.
- Arquitectura en capas : Implementación siguiendo una arquitectura en capas (controlador, servicio, repositorio, modelo).

3. Usabilidad

- Interfaz de consola : Mantenimiento de la interfaz de consola existente para operaciones básicas.
- API REST : Provisión de una API REST para acceso programático a las funcionalidades.

Diagrama UML.

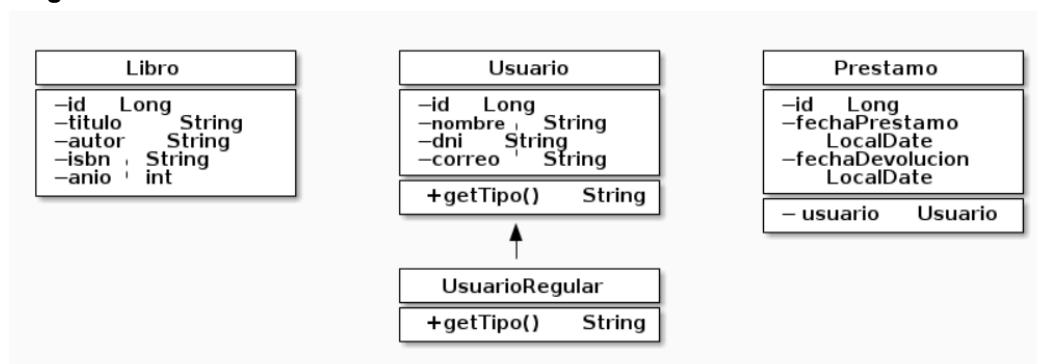
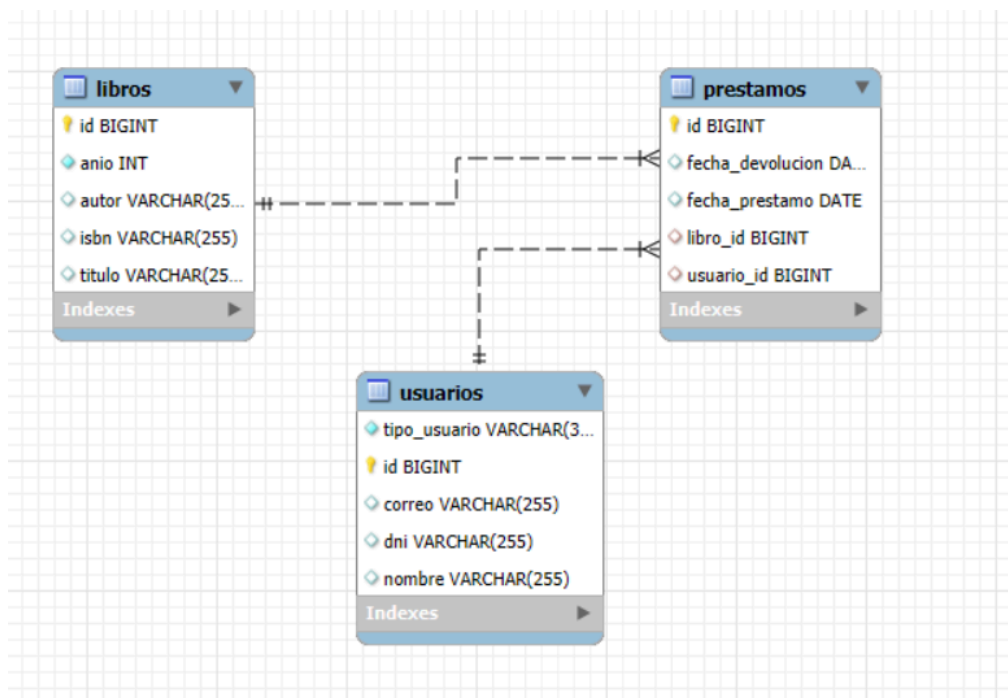


Diagrama de Casos de uso



Diseño Base de datos



- Pruebas realizadas con Mockito.

```
[INFO] --- compiler:3.11.0:compile (default-compile) @ biblioteca-app ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- resources:3.3.1:testResources (default-testResources) @ biblioteca-app ---
[INFO] Copying 1 resource from src\test\resources to target\test-classes
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ biblioteca-app ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- surefire:3.0.0:test (default-test) @ biblioteca-app ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running com.biblioteca.app.ejemplos.EjemploMockitoTest
WARNING: A Java agent has been loaded dynamically (C:\Users\Desarrollo\.m2\repository\net\bytebuddy\byte-buddy-agent\1.14.9\byte-buddy-agent-1.14.9.jar)
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.573 s - in com.biblioteca.app.ejemplos.EjemploMockitoTest
[INFO] Results:
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.147 s
[INFO] Finished at: 2025-06-25T13:15:24+02:00
[INFO] -----
```

