Examen práctico

1 Java

1.1 Escribe un programa que imprima una pirámide de asteriscos como el ejemplo siguiente:

```
    *
   ***
  *****
 *******
*********
```

```
public static void main(String[] args) {
        int altura = 5; // Define la altura de la pirámide
        // Bucle para recorrer cada fila de la pirámide
        for (int i = 1; i <= altura; i++) {
            // Imprimir espacios en blanco para centrar la pirámide
            for (int j = 1; j <= altura - i; j++) {
                System.out.print(" ");
            }
            // Imprimir asteriscos en la fila actual
            for (int k = 1; k <= 2 * i - 1; k++) {
                System.out.print("*");
            }
            // Imprimir un salto de línea al final de la fila
            System.out.println();
        }
}
```

1.2 Simulador de Red de Semáforos

Desarrolla un simulador de red de semáforos para una intersección compleja de una ciudad. Esta intersección involucra varias calles y direcciones, incluyendo giros a la derecha, izquierda y continuación recta. Tu tarea es gestionar los semáforos para asegurar un flujo eficiente del tráfico, minimizando las esperas y evitando colisiones.

Requisitos:

1. **Modelado de Intersección**: Crea una clase **Interseccion** que modele la intersección, incluyendo las diferentes calles y direcciones de tráfico.
2. **Gestión de Semáforos**: Implementa una serie de clases que representen los semáforos. Cada semáforo debe tener estados (rojo, amarillo, verde) y debe poder cambiar entre estos estados de manera controlada.

Examen práctico

3. **Lógica de Control de Tráfico:** Desarrolla un algoritmo que controle cuándo y cómo cambian los semáforos para maximizar la eficiencia del tráfico. Considera casos como el tráfico en horas pico y tráfico ligero.
4. **Detección de Colisiones:** Asegúrate de que tu sistema pueda prevenir colisiones, identificando situaciones potencialmente peligrosas y ajustando los semáforos en consecuencia.
5. **Simulación y Pruebas:** Crea un método de simulación que permita probar diferentes escenarios de tráfico. Incluye un sistema de registro para monitorear el comportamiento de los semáforos y el flujo del tráfico.
6. **Interfaz de Usuario (Opcional):** Para una visualización más interactiva, considera desarrollar una interfaz gráfica que muestre la intersección y el estado de los semáforos en tiempo real.

Desafío Adicional:

Integra un algoritmo de aprendizaje automático o heurísticas para optimizar la lógica de control de tráfico basándose en patrones de tráfico históricos.

```
public class Interseccion {
	private List<Calle> calles;
	private List<Semaforo> semaforos;

	public Interseccion() {
		this.calles = new ArrayList<>();
		this.semaforos = new ArrayList<>();
	}

	public void addCalle(Calle calle) {
		this.calles.add(calle);
	}

	public void addSemaforo(Semaforo semaforo, Direccion direccion) {
		this.semaforos.add(semaforo);
		// ... asociar el semáforo a la calle en la dirección indicada
	}

	public List<Calle> getCalles() {
		return calles;
	}

	public List<Semaforo> getSemaforos() {
		return semaforos;
	}

	// ... métodos adicionales para obtener información específica de la intersección
}

public class Calle {

	private String nombre;
	private int numeroCarriles;
	private Direccion direccion;

	public Calle(String nombre, int numeroCarriles, Direccion direccion) {
		this.nombre = nombre;
		this.numeroCarriles = numeroCarriles;
		this.direccion = direccion;
	}

	public String getNombre() {
		return nombre;
	}
```

```
	public int getNumeroCarriles() {
		return numeroCarriles;
	}

	public Direccion getDireccion() {
		return direccion;
	}
}

public class Semaforo {

	private Color estado;
	private int tiempoRestante;

	public Semaforo(Color estado) {
		this.estado = estado;
		this.tiempoRestante = 0;
	}

	public void cambiarEstado(Color nuevoEstado) {
		this.estado = nuevoEstado;
		this.tiempoRestante = getTiempoCiclo(nuevoEstado);
	}

	public Color getEstado() {
		return estado;
	}

	public int getTiempoRestante() {
		return tiempoRestante;
	}

	private int getTiempoCiclo(Color color) {
		return 0;
	}
}

public class ControladorTrafico {

	private Interseccion interseccion;
	private CicloCambio cicloCambio;

	public ControladorTrafico(Interseccion interseccion) {
		this.interseccion = interseccion;
		this.cicloCambio = new CicloCambio(interseccion);
	}

	public void actualizar() {
		cicloCambio.actualizar();
	}
}

public class CicloCambio {

	private Interseccion interseccion;
	private Fase actual;

	public CicloCambio(Interseccion interseccion) {
		this.interseccion = interseccion;
		this.actual = Fase.VERDE_NORTE_SUR;
	}

	public void actualizar() {
		for (Semaforo semaforo : interseccion.getSemaforos()) {
			semaforo.cambiarEstado(actual.getEstado(semaforo.getDireccion()));
		}
	}

}

public class Simulador {

	private Interseccion interseccion;
	private ControladorTrafico controladorTrafico;

	public Simulador(Interseccion interseccion) {
```

```
                this.interseccion = interseccion;
                this.controladorTrafico = new ControladorTrafico(interseccion);
        }

        public void run() {
                while (true) {
                        controladorTrafico.actualizar();
                        // ... simular el movimiento de vehículos
                        // ... registrar datos de tráfico
                        // ... visualizar la intersección (opcional)
                }
        }
}

public class Main {

        public static void main(String[] args) {
                Interseccion interseccion = new Interseccion();
                // ... crear calles y semaforos
                Simulador simulador = new Simulador(interseccion);
                simulador.run();
        }
}
```

2 Spring Boot

2.1 Sistema de Reservas para un Hotel

Desarrolla un sistema de reservas en línea para un hotel utilizando Spring Framework. Este sistema deberá permitir a los usuarios reservar habitaciones, gestionar reservas existentes y consultar información sobre habitaciones y precios.

Requisitos:

1. **Modelado de Datos**: Diseña entidades para representar habitaciones, reservas, clientes y cualquier otro elemento relevante para un hotel. Utiliza JPA para la persistencia de datos.
2. **API REST**: Crea una API REST para gestionar las reservas. Esto incluirá endpoints para crear, modificar, cancelar y consultar reservas, así como obtener información sobre las habitaciones.
3. **Seguridad**: Implementa seguridad en tu API. Utiliza Spring Security para manejar la autenticación y autorización. Asegúrate de que solo los usuarios registrados puedan hacer reservas.
4. **Gestión de Errores**: Maneja adecuadamente los errores y excepciones, proporcionando respuestas claras y útiles a los usuarios de la API.
5. **Pruebas Unitarias y de Integración**: Escribe pruebas unitarias y de integración utilizando JUnit para asegurarte de que todos los componentes de tu aplicación funcionen correctamente.
6. **Interfaz de Usuario (Opcional)**: Si lo deseas, desarrolla una interfaz de usuario sencilla utilizando alguna tecnología frontend para interactuar con tu API.
7. **Documentación de la API**: Documenta tu API REST utilizando herramientas como Spring REST Docs.

```
@Entity
@Table(name = "habitaciones")
public class Habitacion {
```

```
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	private String numero;
	private TipoHabitacion tipo;
	private int capacidad;
	private double precio;
	private boolean disponible;

	// Getters and setters...
}

@Entity
@Table(name = "tipos_habitaciones")
public class TipoHabitacion {

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

	// Getters and setters...
}

@Entity
@Table(name = "clientes")
public class Cliente {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	private String nombre;
	private String apellido;
	private String email;
	private String telefono;

	// Getters and setters...
}

@Entity
@Table(name = "reservas")
public class Reserva {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	private Cliente cliente;
	private Habitacion habitacion;
	private LocalDate fechaEntrada;
	private LocalDate fechaSalida;
	private int numeroPersonas;
	private double precioTotal;

	// Getters and setters...
}

@RestController
@RequestMapping("/api/reservas")
public class ReservaController {

	@Autowired
	private ReservaService reservaService;

	@PostMapping
	public ResponseEntity<?> crearReserva(@RequestBody ReservaDTO reservaDTO) {
		return ResponseEntity.ok(reservaService.crearReserva(reservaDTO));
	}

	@GetMapping
	public ResponseEntity<?> obtenerReservas() {
		return ResponseEntity.ok(reservaService.obtenerReservas());
	}

	@GetMapping("/{id}")
	public ResponseEntity<?> obtenerReservaPorId(@PathVariable Long id) {
```

```
                return ResponseEntity.ok(reservaService.obtenerReservaPorId(id));
        }

        @PutMapping("/{id}")
        public ResponseEntity<?> modificarReserva(@PathVariable Long id, @RequestBody ReservaDTO reservaDTO) {
                return ResponseEntity.ok(reservaService.modificarReserva(id, reservaDTO));
        }

        @DeleteMapping("/{id}")
        public ResponseEntity<?> cancelarReserva(@PathVariable Long id) {
                reservaService.cancelarReserva(id);
                return ResponseEntity.noContent().build();
        }
}

@Service
public class ReservaServiceImpl implements ReservaService {

        @Autowired
        private ReservaRepository reservaRepository;

        @Autowired
        private HabitacionRepository habitacionRepository;

        @Override
        public Reserva crearReserva(ReservaDTO reservaDTO) {
                Habitacion habitacion = habitacionRepository.findById(reservaDTO.getHabitacionId()).orElseThrow();
                Cliente cliente = new Cliente(); // Se debería obtener el cliente autenticado
                Reserva reserva = new Reserva(cliente, habitacion, reservaDTO.getFechaEntrada(),
                reservaDTO.getFechaSalida(), reservaDTO.getNumeroPersonas(),
                habitacion.getPrecio() * reservaDTO.getNumeroPersonas());
                return reservaRepository.save(reserva);
        }

        // ... Implementación de los demás métodos del servicio ...
}

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http.csrf().disable()
                .authorizeRequests()
                .antMatchers("/api/reservas/**").authenticated()
                .and()
                .formLogin();
        }

        @Autowired
        public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
                auth.inMemoryAuthentication()
                .withUser("usuario").password("password").roles("USER");
        }
}


@ControllerAdvice
public class GlobalExceptionHandler {

        @ExceptionHandler(Exception.class)
        public ResponseEntity<?> handleException(Exception ex) {
                // Log the exception for debugging purposes
                log.error("An error occurred:", ex);

                // Create a generic error response with a meaningful message
                ErrorResponse errorResponse = new ErrorResponse("Internal server error");

                // Return a 500 Internal Server Error response
                return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(errorResponse);
        }

        // Handler for specific exceptions, providing more detailed error messages
        @ExceptionHandler(ResourceNotFoundException.class)
        public ResponseEntity<?> handleResourceNotFoundException(ResourceNotFoundException ex) {
                ErrorResponse errorResponse = new ErrorResponse(ex.getMessage());
```

```
                return ResponseEntity.status(HttpStatus.NOT_FOUND).body(errorResponse);
        }

}
```

3.- Crea un programa en Java que consuma el API PokeAPI. El programa debe poder interactuar con los recursos del api para realizar una búsqueda (el parametro de busqueda es a tu elección, para esto revisa la documentación del API sobre que recursos puedes consumir)

Ejemplo busqueda por nombre se consume la URL de la siguiente manera:

https://pokeapi.co/api/v2/pokemon/pikachu

```
private static String getPokemonData(String nombrePokemon) throws IOException {
        URL url = new URL("https://pokeapi.co/api/v2/pokemon/" + nombrePokemon);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Accept", "application/json");

        int responseCode = connection.getResponseCode();
        if (responseCode != 200) {
                throw new IOException("Error al obtener datos del Pokemon: " + responseCode);
        }

        BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));
        String inputLine;
        StringBuilder response = new StringBuilder();

        while ((inputLine = reader.readLine()) != null) {
                response.append(inputLine);
        }

        reader.close();
        return response.toString();
}

private static void mostrarInformacionPokemon(String nombrePokemon, String jsonData) throws Exception {
        JSONParser parser = new JSONParser();
        JSONObject jsonObject = (JSONObject) parser.parse(jsonData);

        String nombre = (String) jsonObject.get("name");
        int id = (int) jsonObject.get("id");
        int altura = (int) jsonObject.get("height");
        int peso = (int) jsonObject.get("weight");

        System.out.println("Nombre: " + nombre);
        System.out.println("ID: " + id);
        System.out.println("Altura: " + altura + " cm");
        System.out.println("Peso: " + peso + " kg");
}

public static void main(String[] args) throws Exception {

        String nombrePokemon = "pikachu"; // Cambiar por el nombre del Pokemon que se desea buscar



        String jsonData = getPokemonData(nombrePokemon);

        mostrarInformacionPokemon(nombrePokemon, jsonData);

}
```