BANCO DE PREGUNTAS - PRIMER PARCIAL

Programación Visual

Instituto Tecnológico Lemas

Instrucciones

Este banco contiene 20 preguntas objetivas. Cada pregunta puede tener entre 2 y 4 respuestas correctas. Las respuestas correctas están marcadas con y en color verde para su revisión.

Pregunta 1: ¿Cuáles son características del proyecto Maven en JavaFX?

a) Gestiona las dependencias del proyecto automáticamente
b) No permite trabajar con bibliotecas externas
c) Utiliza el archivo pom.xml para configuración
d) Solo funciona con aplicaciones de consola

Pregunta 2: En el código del proyecto, ¿qué métodos HTTP son utilizados para consumir la PokeAPI?

a) GET para obtener información de pokémon
b) POST para crear nuevos pokémon
c) DELETE para eliminar pokémon
d) GET con diferentes endpoints para listar y detallar

Pregunta 3: ¿Qué elementos de JavaFX se utilizan en el archivo pokemon.fxml del proyecto?

a) VBox para organizar elementos verticalmente
b) TextField para el campo de búsqueda
c) Button para las acciones de búsqueda y carga
d) Canvas para dibujar gráficos complejos

Pregunta 4: ¿Cuál es la URL base de la PokeAPI según el código del proyecto?

a) https://pokemon.com/api/
b) https://pokeapi.co/api/v2/
c) https://api.pokemon.org/
d) Se define como constante BASE_URL en PokemonService

Pregunta 5: Qué propiedades se pueden aplicar a elementos JavaFX mediante CSS?

a) -fx-background-color para el color de fondo
b) -fx-text-fill para el color del texto
c) -fx-font-size para el tamaño de fuente
d) -fx-padding para el espaciado interno

Pregunta 6: En el proyecto, ¿qué clases de modelo se utilizan para mapear la respuesta de la API?

a) PokemonResponse para los datos completos del pokémon

b) PokemonSprites para las imágenes del pokémon
c) PokemonDatabase para almacenar en base de datos
d) PokemonTypes para los tipos de pokémon

Pregunta 7: ¿Qué biblioteca se utiliza en PokemonService para convertir JSON a objetos Java?

a) Jackson con ObjectMapper
b) GSON de Google
c) DeserializationFeature para configurar el mapeo
d) JSON.simple

Pregunta 8: ¿Qué operaciones asíncronas se implementan en el controlador?

a) CompletableFuture para llamadas no bloqueantes
b) Task de JavaFX para ejecutar en segundo plano
c) new Thread() para iniciar hilos separados
d) setOnSucceeded para manejar respuestas exitosas

Pregunta 9: ¿Qué información del pokémon se muestra en las tarjetas del proyecto?

a) Nombre formateado y número ID
b) Imagen del sprite frontal
c) Tipos con colores específicos
d) Evoluciones del pokémon

Pregunta 10: En el archivo CSS del proyecto, ¿qué estilos se aplican al campo de búsqueda?

a) Border-radius de 25 para bordes redondeados
b) Background-color negro
c) Cambio de border-color al enfocar el campo
d) Text-decoration underline

Pregunta 11: ¿Qué métodos del servicio PokemonService están implementados?

a) getPokemonList con límite y offset
b) getPokemonByURL para consultar por URL
c) getPokemonByName para buscar por nombre
d) makeHttpRequest como método privado auxiliar

Pregunta 12: ¿Qué características tiene el método showAlertDetailsPokemon?

a) Muestra altura, peso y experiencia base
b) Lista todas las habilidades del pokémon
c) Presenta las estadísticas con sus valores
d) Permite editar los datos del pokémon

Pregunta 13: ¿Qué anotaciones de JavaFX se utilizan en PokemonController?

a) @FXML para vincular elementos del archivo FXML
b) @Autowired para inyección de dependencias
c) @Override solo en métodos heredados
d) @FXML en el método initialize para inicialización

Pregunta 14: ¿Qué funcionalidades tiene el botón "Pokémon Aleatorio"?

a) Limpia el contenedor de pokémon
b) Genera un número aleatorio entre 1 y 1024
c) Muestra indicador de carga
d) Llama al método callPokemonByName con el número

Pregunta 15: En la clase PokemonService, ¿qué configuraciones se aplican a HttpURLConnection?

a) Timeout de conexión de 10000 ms
b) Método GET para las peticiones
c) Header "Accept: application/json"
d) Autenticación con Bearer token

Pregunta 16: ¿Qué propósito tienen las clases ItemStat, ItemsTypes y ItemsAbility?

a) Representar objetos anidados en el JSON de respuesta
b) Almacenar datos en base de datos local
c) Encapsular propiedades específicas de stats, tipos y habilidades
d) Validar datos de entrada del usuario

Pregunta 17: ¿Qué elementos visuales se aplican a las tarjetas de pokémon mediante estilos?

a) Sombra con dropshadow effect
b) Background-color blanco
c) Border-radius de 10 para esquinas redondeadas
d) Padding de 15 para espaciado interno

Pregunta 18: ¿Qué manejo de errores se implementa en el proyecto?

a) setOnFailed para capturar excepciones en Task
b) Try-catch en los métodos del servicio
c) Mensajes de error impresos en consola
d) Retry automático en caso de fallo

Pregunta 19: El método loadMorePokemon utiliza:

a) El tamaño actual de lstPokemon como offset
b) Reinicia la lista desde cero
c) La constante POKEMON_LIMIT como cantidad a cargar
d) Carga todos los pokémon disponibles

Pregunta 20: ¿Qué ventajas ofrece usar CompletableFuture en el servicio?

a) Permite ejecución asíncrona sin bloquear la UI
b) Facilita el encadenamiento de operaciones
c) Retorna inmediatamente permitiendo otras operaciones
d) Garantiza que nunca habrá errores en las peticiones

EXAMEN DE PROGRAMACIÓN VISUAL - Los Simpsons

Descripción del Proyecto

La aplicación **Los Simpsons - Explorador de Personajes** es una aplicación JavaFX que consume la API de Los Simpsons para mostrar información sobre diferentes personajes de la serie. La aplicación permite buscar personajes, obtener personajes aleatorios, cargar listas paginadas y ver detalles completos de cada personaje.

Estado Actual del Proyecto

El proyecto está parcialmente implementado. La funcionalidad de carga de imágenes está completa y **NO DEBE SER MODIFICADA**. Sin embargo, faltan varias funcionalidades críticas que debes implementar:

1. **Inicialización del servicio y configuración inicial**
2. **Búsqueda de personajes por nombre**
3. **Obtención de personajes aleatorios**
4. **Carga de más personajes con paginación**
5. **Creación de componentes de la tarjeta de personaje**
6. **Mostrar detalles completos del personaje**

Tareas a Realizar

TAREA 1: Completar el método initialize()

Ubicación: src/main/java/com/simpsons/PrimaryController.java
Objetivo: Inicializar el servicio de Simpsons y la lista de personajes, configurar el layout del FlowPane y cargar los personajes iniciales.

Pasos a seguir:

1. Inicializar simpsonService con una nueva instancia de SimpsonService
2. Inicializar currentCharactersList como una nueva ArrayList<>()
3. Llamar al método setupFlowPaneLayout() para configurar el layout
4. Llamar al método loadInitialCharacters() para cargar los personajes iniciales

Código a completar:

```
public void initialize() {
```

```
        // TODO: Inicializar simpsonService con new SimpsonService()
        // TODO: Inicializar currentCharactersList como new ArrayList<>()
        // TODO: Llamar a setupFlowPaneLayout()
        // TODO: Llamar a loadInitialCharacters()
}
```

TAREA 2: Completar el método searchCharacter()
Ubicación: src/main/java/com/simpsons/PrimaryController.java
Objetivo: Implementar la búsqueda de personajes por nombre en la lista actual.

Pasos a seguir:

1. Obtener el texto del campo de búsqueda usando `searchField.getText().trim()`
2. Validar que el texto no esté vacío. Si está vacío, mostrar una alerta con título "Error" y mensaje "Por favor ingresa el nombre de un personaje", luego retornar
3. Limpiar el contenedor de personajes llamando a `clearCharactersContainer()`
4. Mostrar el indicador de carga llamando a `showLoadingIndicator()`
5. Buscar el personaje en `currentCharactersList` usando un stream que filtre por nombre (comparación sin distinguir mayúsculas/minúsculas) y obtener el primero con `findFirst().orElse(null)`
6. Si se encuentra el personaje:
    - Ocultar el indicador de carga con `hideLoadingIndicator()`
    - Limpiar el contenedor nuevamente
    - Mostrar el personaje con `displayCharacter(foundCharacter)`
7. Si no se encuentra:
    - Ocultar el indicador de carga
    - Mostrar una alerta con título "Información" y mensaje "Personaje no encontrado en la lista actual. Intenta cargar más personajes o busca por ID."

Pistas:

- Usa `stream()`, `filter()`, `findFirst()`, `orElse(null)` para la búsqueda
- La comparación debe ser case-insensitive usando `toLowerCase()`
- Usa `contains()` para buscar el texto dentro del nombre

Código a completar:

```
@FXML
```

```
private void searchCharacter() {
    // TODO: Obtener texto del campo de búsqueda y validar que no esté vacío
    // TODO: Limpiar contenedor y mostrar indicador de carga
    // TODO: Buscar personaje en currentCharactersList usando stream
    // TODO: Si se encuentra, ocultar indicador, limpiar y mostrar personaje
    // TODO: Si no se encuentra, ocultar indicador y mostrar alerta informativa
}
```

TAREA 3: Completar el método getRandomCharacter()
Ubicación: src/main/java/com/simpsons/PrimaryController.java
Objetivo: Implementar la obtención de un personaje aleatorio de la API.

Pasos a seguir:

1. Limpiar el contenedor de personajes
2. Mostrar el indicador de carga
3. Crear un Task<Personajes> que:
   - En el método call(): genere un ID aleatorio entre 1 y 1182 (usando Random), luego llame a simpsonService.getCharacterById(randomId).get() y retorne el resultado
4. Configurar el setOnSucceeded del Task para:
   - Ocultar el indicador de carga
   - Obtener el personaje con randomTask.getValue()
   - Si el personaje no es null: limpiar currentCharactersList, agregar el personaje a la lista y mostrarlo con displayCharacter(character)
5. Configurar el setOnFailed del Task para:
   - Ocultar el indicador de carga
   - Mostrar una alerta con título "Error" y mensaje "No se pudo obtener un personaje aleatorio"
6. Iniciar el Task en un nuevo hilo con new Thread(randomTask).start()

Pistas:

- Usa Random random = new Random() y random.nextInt(1182) + 1 para el ID aleatorio
- El Task debe extender Task<Personajes>
- Usa Platform.runLater() si es necesario para actualizar la UI

Código a completar:

```
@FXML
private void getRandomCharacter() {
    // TODO: Limpiar contenedor y mostrar indicador de carga
```

```
    // TODO: Crear Task<Personajes> que genere ID aleatorio y obtenga personaje
    // TODO: Configurar setOnSucceeded para mostrar personaje
    // TODO: Configurar setOnFailed para mostrar error
    // TODO: Iniciar Task en nuevo hilo
}
```

TAREA 4: Completar el método loadMoreCharacters()
Ubicación: src/main/java/com/simpsons/PrimaryController.java
Objetivo: Implementar la carga de más personajes con paginación.

Pasos a seguir:

1. Validar que no se esté cargando (isLoading) y que haya más páginas (hasMorePages). Si alguna condición no se cumple, retornar
2. Establecer isLoading = true
3. Mostrar el indicador de carga

4. Crear un Task<SimpsonResponse> que:
    - En el método call(): llame a simpsonService.getCharacters(currentPage).get() y retorne el resultado
5. Configurar el setOnSucceeded del Task para:
    - Ocultar el indicador de carga
    - Obtener la respuesta con loadTask.getValue()
    - Si la respuesta no es null y response.getResults() no es null:
        - Obtener la lista de nuevos personajes: List<Personajes> newCharacters = response.getResults()
        - Si currentPage == 1, limpiar currentCharactersList
        - Agregar todos los nuevos personajes a currentCharactersList con addAll()
        - Iterar sobre newCharacters y mostrar cada personaje con displayCharacter()
        - Verificar si hay más páginas: hasMorePages = response.getNext() != null && !response.getNext().isEmpty()
        - Incrementar currentPage++
        - Si no hay más páginas: deshabilitar loadMoreButton con setDisable(true) y cambiar el texto a "No hay más personajes"
    - Establecer isLoading = false
6. Configurar el setOnFailed del Task para:
    - Ocultar el indicador de carga
    - Mostrar una alerta con título "Error" y mensaje "No se pudieron cargar más personajes"
    - Establecer isLoading = false

7. Iniciar el Task en un nuevo hilo

Código a completar:

```
@FXML
private void loadMoreCharacters() {
    // TODO: Validar isLoading y hasMorePages
    // TODO: Establecer isLoading = true y mostrar indicador
    // TODO: Crear Task<SimpsonResponse> para obtener personajes
    // TODO: Configurar setOnSucceeded para procesar respuesta y actualizar UI
    // TODO: Configurar setOnFailed para manejar errores
    // TODO: Iniciar Task en nuevo hilo
}
```

TAREA 5: Completar la creación de componentes en createCharacterCard()

Ubicación: src/main/java/com/simpsons/PrimaryController.java
Objetivo: Crear los componentes de la tarjeta del personaje (sin modificar la parte de imagen que ya está completa).

Pasos a seguir:

En el método createCharacterCard(), después de la creación del imageContainer (que ya está completo), debes crear:

1. **Contenedor de contenido (contentContainer):**
   - Crear un VBox con espaciado de 12
   - Agregar la clase de estilo "character-content"

2. **Label del nombre:**
   - Crear un Label con el nombre del personaje: character.getName()
   - Agregar la clase de estilo "character-name"

3. **Label de ocupación:**
   - Crear un Label con el texto " " + ocupación
   - La ocupación debe ser character.getOccupation() si no es null ni vacío, sino "Sin ocupación"
   - Agregar la clase de estilo "character-occupation"

4. **Fila de información (infoRow):**
   - Crear un HBox con espaciado de 10
   - Agregar la clase de estilo "character-info-row"
   - Establecer alineación a Pos.CENTER_LEFT
   - **Edad:** Si character.getAge() no es null y es mayor que 0:

- Crear un Label con texto " " + edad + " años"
- Agregar clase de estilo "character-age"
- Agregar el label al infoRow

- **Estado:** Crear un Label con el estado (si es null, usar "Desconocido")
  - Si el estado es "Alive" (ignorando mayúsculas): agregar clase "character-status-alive"
  - Si el estado es "Deceased" (ignorando mayúsculas): agregar clase "character-status-deceased"
  - Si no: agregar clase "character-age"
  - Agregar el label al infoRow

5. **Label de frase:**

- Crear un Label
- Si character.getPhrases() no es null y tiene al menos un elemento: establecer texto como """ + primera frase + """
- Si no: establecer texto como "Sin frase famosa"
- Agregar clase de estilo "character-phrase"

6. **Botón de detalles:**

- Crear un Button con texto "Ver Detalles"
- Agregar clase de estilo "character-details-button"
- Configurar el evento setOnAction para llamar a showCharacterDetails(character)

7. **Agregar componentes al contenedor:**

- Agregar todos los componentes creados al contentContainer usando getChildren().addAll()
- Agregar imageContainer y contentContainer a la tarjeta (card)

Import necesario:

```
import javafx.geometry.Pos;
```

Código a completar:

```
// Después de crear imageContainer (que ya está completo):
// TODO: Crear contentContainer (VBox con espaciado 12 y clase "character-content")
// TODO: Crear label del nombre con clase "character-name"
// TODO: Crear label de ocupación con emoji      y clase "character-occupation"
// TODO: Crear infoRow (HBox) con edad y estado
// TODO: Crear label de frase con clase "character-phrase"
// TODO: Crear botón de detalles con evento que llame a showCharacterDetails
// TODO: Agregar todos los componentes al contentContainer
```

```
// TODO: Agregar imageContainer y contentContainer a la tarjeta
```

TAREA 6: Completar el método showCharacterDetails()
Ubicación: src/main/java/com/simpsons/PrimaryController.java
Objetivo: Mostrar los detalles completos del personaje en un Alert.

Pasos a seguir:

1. Crear un Alert de tipo Alert.AlertType.INFORMATION
2. Establecer el título como "Detalles de " + nombre del personaje
3. Establecer el header como null
4. Crear un StringBuilder llamado details
5. Agregar al StringBuilder:
    - "ID: " + character.getId() + "\n"
    - "Nombre: " + character.getName() + "\n"
    - Si character.getAge() no es null: "Edad: " + edad + " años\n"
    - Si character.getBirthdate() no es null ni vacío: "Fecha de Nacimiento: " + fecha + "\n"
    - Si character.getGender() no es null ni vacío: "Género: " + género + "\n"
    - Si character.getOccupation() no es null ni vacío: "Ocupación: " + ocupación + "\n"
    - "Estado: " + (estado si no es null, sino "Desconocido") + "\n"
    - Si character.getPhrases() no es null y tiene elementos:
        - Agregar "\nFrases Famosas:\n"
        - Iterar sobre el array de frases y agregar cada una como "• "" + frase + ""\n"
6. Establecer el contenido del Alert con alert.setContentText(details.toString())
7. Hacer el Alert redimensionable con alert.setResizable(true)
8. Establecer el ancho preferido del DialogPane a 500: alert.getDialogPane().setPrefWidth(500)
9. Mostrar el Alert con alert.showAndWait()

Código a completar:

```
private void showCharacterDetails(Personajes character) {
    // TODO: Crear Alert de tipo INFORMATION
    // TODO: Establecer título y header
    // TODO: Crear StringBuilder y agregar todos los detalles
    // TODO: Agregar frases famosas si existen
    // TODO: Configurar Alert y mostrarlo
}
```

Criterios de Evaluación

TAREA 1 (1 punto)

- Se inicializa correctamente simpsonService
- Se inicializa correctamente currentCharactersList
- Se llama a setupFlowPaneLayout()
- Se llama a loadInitialCharacters()

TAREA 2 (1.5 puntos)

- Se obtiene y valida el texto del campo de búsqueda
- Se muestra alerta si el campo está vacío
- Se limpia el contenedor y se muestra indicador de carga
- Se busca correctamente usando stream con filtro case-insensitive
- Se maneja correctamente cuando se encuentra el personaje
- Se maneja correctamente cuando NO se encuentra el personaje

TAREA 3 (1.5 puntos)

- Se limpia el contenedor y se muestra indicador de carga
- Se crea correctamente el Task con generación de ID aleatorio
- Se configura correctamente setOnSucceeded para mostrar el personaje
- Se configura correctamente setOnFailed para manejar errores
- Se inicia el Task en un nuevo hilo

TAREA 4 (2 puntos)

- Se valida correctamente isLoading y hasMorePages
- Se crea correctamente el Task para obtener personajes
- Se procesa correctamente la respuesta en setOnSucceeded
- Se actualiza correctamente la lista y se muestran los personajes
- Se verifica correctamente si hay más páginas

- Se deshabilita el botón cuando no hay más páginas

- Se maneja correctamente el error en `setOnFailed`
- Se actualiza correctamente el estado `isLoading`

TAREA 5 (2 puntos)

- Se crea correctamente el `contentContainer` con estilos
- Se crea correctamente el label del nombre

- Se crea correctamente el label de ocupación con validación

- Se crea correctamente el `infoRow` con edad y estado
- Se valida correctamente la edad antes de mostrarla

- Se aplican correctamente las clases de estilo según el estado

- Se crea correctamente el label de frase con validación

- Se crea correctamente el botón de detalles con evento

- Se agregan correctamente todos los componentes

TAREA 6 (1 punto)

- Se crea correctamente el Alert

- Se construye correctamente el StringBuilder con todos los datos

- Se validan correctamente los campos opcionales (edad, fecha, género, ocupación)

- Se muestran correctamente las frases famosas si existen

- Se configura correctamente el Alert (redimensionable, ancho)

- Se muestra el Alert correctamente

Total: 9 puntos

Notas Importantes

1. **NO MODIFICAR código de carga de imágenes:** Los métodos `createCharacterImage()`, `downloadAndLoadImage()` y `downloadImageBytes()` están completos y **NO DEBEN SER MODIFICADOS**.

2. **Manejo de null:** Siempre valida que los objetos no sean null antes de usarlos para evitar `NullPointerException`.
3. **Threading:** Recuerda que las actualizaciones de la UI deben hacerse en el hilo de JavaFX. El código de `displayCharacter()` ya usa `Platform.runLater()`, así que no necesitas agregarlo ahí.

4. **Streams y Lambdas:** Usa streams de Java 8 para búsquedas y filtros. Recuerda que las comparaciones de strings deben ser case-insensitive.

5. **Testing:** Prueba la aplicación para verificar que:

- Los personajes se cargan correctamente al iniciar
- La búsqueda funciona correctamente
- El personaje aleatorio se obtiene y muestra
- La paginación funciona correctamente
- Los detalles se muestran correctamente
- Los casos especiales (campos null, listas vacías) funcionan bien

---

Estructura del Proyecto

```
src/main/java/com/simpsons/
├── App.java                    (  Completo - No modificar)
├── PrimaryController.java      (⚠ Completar - TODAS LAS TAREAS)
├── model/
│   ├── Personajes.java         (  Completo - No modificar)
│   └── SimpsonResponse.java    (  Completo - No modificar)
└── service/
    └── SimpsonService.java     (  Completo - No modificar)
```

Tiempo Estimado

- TAREA 1: 5 minutos
- TAREA 2: 10 minutos
- TAREA 3: 15 minutos
- TAREA 4: 20 minutos
- TAREA 5: 25 minutos

- TAREA 6: 10 minutos
- **Total: 85 minutos**

GITHUB
https://github.com/wgaibor/ExamenProgramacionVisualB25/tree/master/simpsons