dev Senior

Reto #1

JAVA

Simulador de Viaje Interplanetario

RETO 1 - Simulador de Viaje Interplanetario

Introducción

En el curso "Java de Cero a Senior: La Travesía Definitiva", los estudiantes aprenderán a desarrollar aplicaciones basadas en consola utilizando los conceptos fundamentales de Java, como estructuras de datos, control de flujo y métodos. En este segundo reto, titulado "Simulador de Viaje Interplanetario", los estudiantes crearán una simulación que les permita gestionar un viaje espacial a diferentes planetas del sistema solar. Este ejercicio es una oportunidad para practicar la lógica de programación y resolver problemas con creatividad.

El desafío consistirá en desarrollar un programa que simule la planificación y ejecución de un viaje interplanetario, donde el usuario podrá elegir destinos, calcular la distancia y el tiempo de viaje, y gestionar los recursos de la nave. El objetivo es aplicar conceptos básicos de programación de forma práctica y entretenida.

Objetivos

1. **Seleccionar destino interplanetario**
    - Permitir al usuario elegir un planeta destino entre una lista de planetas del sistema solar (Mercurio, Venus, Marte, Júpiter, Saturno, etc.).
2. **Calcular distancia y tiempo de viaje**
    - Calcular la distancia entre la Tierra y el planeta destino y el tiempo estimado del viaje, considerando una velocidad dada (por ejemplo, en kilómetros por hora).
3. **Gestionar recursos de la nave**
    - Calcular la cantidad de combustible necesario para el viaje y permitir al usuario administrar los recursos (combustible y oxígeno). La cantidad de recursos disponibles y necesarios variará según el destino.
4. **Simular eventos aleatorios durante el viaje**
    - Introducir eventos aleatorios que puedan afectar el viaje, como fallas en el sistema, asteroides, o desvíos, que requieran la intervención del usuario para resolverlos.
5. **Monitorear el estado del viaje**
    - Mostrar el progreso del viaje en cada paso (por ejemplo, porcentaje de distancia recorrida), la cantidad de recursos restantes y el tiempo faltante para llegar al destino.

Beneficios del Reto

1. **Consolidación de Conceptos Básicos de Java**
    - Este reto permite a los estudiantes practicar la manipulación de variables, operadores, estructuras de control de flujo y métodos en un contexto práctico.
2. **Desarrollo de Habilidades para Resolver Problemas**
    - Los estudiantes aprenderán a planificar soluciones dividiendo el problema en partes más pequeñas y utilizando funciones para organizar el código.
3. **Desafío de Lógica y Validación**
    - Se pondrán en práctica conceptos de control de flujo y validación de datos para manejar diferentes escenarios en la simulación.
4. **Introducción a la Manipulación de Arrays y Listas**
    - Los estudiantes trabajarán con arrays para manejar la lista de planetas y los recursos disponibles.

Requerimientos Funcionales para el Simulador de Viaje Interplanetario

1. **Selección de Destino**
    - Presentar al usuario una lista de planetas del sistema solar para elegir el destino del viaje.
    - Solicitar la confirmación del usuario y mostrar información básica del planeta (distancia en kilómetros desde la Tierra).
2. **Cálculo de Distancia y Tiempo de Viaje**
    - Calcular la distancia al planeta elegido y el tiempo estimado para completar el viaje, utilizando una velocidad fija (por ejemplo, 100,000 km/h).
    - Mostrar la información al usuario.
3. **Gestión de Recursos**
    - Calcular la cantidad de combustible y oxígeno necesarios para el viaje según la distancia.
    - Permitir al usuario revisar y ajustar los recursos antes de partir.
4. **Simulación de Eventos Aleatorios**
    - Durante el viaje, simular eventos aleatorios que afecten el progreso, como desvíos o fallos en el sistema.

- Solicitar al usuario tomar decisiones para resolver los problemas (por ejemplo, reparar la nave o cambiar de rumbo).

5. **Monitoreo del Estado del Viaje**
   - Mostrar en pantalla el progreso del viaje (porcentaje de la distancia total), el tiempo restante, y la cantidad de recursos disponibles.
   - Notificar al usuario si el viaje fue exitoso o si la nave se quedó sin recursos.
6. **Interacción con el Usuario**
   - El sistema debe proporcionar un menú interactivo para que el usuario elija las opciones disponibles (selección de destino, ajuste de recursos, iniciar el viaje, etc.).
   - Validar la entrada del usuario para asegurarse de que sea correcta (por ejemplo, seleccionar un número correspondiente a un planeta de la lista).

Metodología para el Reto 1

Para llevar a cabo el Reto 1 del curso "Java de Cero a Senior: La Travesía Definitiva", se seguirá una metodología colaborativa en la que los estudiantes trabajarán en parejas. Esto les permitirá no solo mejorar sus habilidades técnicas, sino también practicar la colaboración en equipo, la comunicación y el uso de herramientas profesionales como Git y GitHub. A continuación, se detalla la metodología a seguir:

1. **Trabajo en Parejas**
   - **Formación de Parejas:** Los estudiantes deberán organizarse en parejas para realizar el reto. La colaboración entre los integrantes será fundamental para compartir ideas, distribuir tareas y resolver problemas conjuntamente.
   - **División de Tareas:** Se recomienda que las parejas distribuyan las tareas de manera equitativa, según las fortalezas de cada integrante. Por ejemplo, uno de los estudiantes puede encargarse de la lógica principal del programa, mientras que el otro puede centrarse en la validación de datos y la interacción con el usuario. Se debe fomentar la rotación de roles para que ambos estudiantes desarrollen habilidades tanto en la parte lógica como en la interfaz de usuario.
2. **Uso de Git y GitHub**
   - **Creación del Repositorio en GitHub:** Cada pareja deberá crear un repositorio en GitHub donde subirán el código fuente de la aplicación. El

repositorio debe ser público para que pueda ser revisado por los docentes del curso.

- **Uso de Ramas y Control de Versiones:**
  - Se sugiere que cada estudiante trabaje en una rama separada para desarrollar funcionalidades específicas. Esto evitará conflictos de código y permitirá que ambos puedan trabajar en paralelo. Una vez
  - que ambas ramas estén listas, se puede realizar un merge en la rama principal (generalmente main o master). Es importante que antes de hacer el merge, se realicen revisiones de código mutuas para verificar la calidad y funcionalidad del mismo.

- **Commits Regulares:** Los estudiantes deben realizar commits frecuentes a lo largo del desarrollo del proyecto, con mensajes claros que describan los cambios realizados en el código. Esto garantizará un historial claro del progreso del reto. Se recomienda usar convenciones para los mensajes de commit, como "fix:" para correcciones de errores, "feat:" para nuevas funcionalidades, y "docs:" para actualizaciones en la documentación.

3. **Video de Presentación**

- **Grabación del Video:** Una vez que el programa esté completo y funcional, los estudiantes deberán grabar un video en el que ambos aparezcan explicando la funcionalidad de la aplicación. Se debe asegurar que el audio y la calidad del video sean adecuados para que los docentes puedan seguir la explicación sin dificultades.

- **Contenido del Video:**
  - **Introducción:** Los estudiantes deben presentarse y dar una breve descripción del reto y de los objetivos del sistema de gestión de productos. Se espera que mencionen los principales desafíos encontrados y cómo los abordaron.
  - **Demostración del Programa:** Durante el video, se debe mostrar una ejecución del programa, explicando las características implementadas, como la gestión de productos, la compra, la búsqueda y el cálculo del valor del stock. Además, deben demostrar cómo se manejan las validaciones de entrada y los mensajes de error.
  - **Reparto Equitativo:** Ambos estudiantes deben participar activamente en la presentación del video, alternando la explicación

de las distintas funcionalidades. La exposición debe ser balanceada para reflejar el trabajo colaborativo.

- **Duración:** El video no debe exceder los 10 minutos y debe ser lo suficientemente claro para que los docentes puedan evaluar la comprensión del reto y la correcta implementación de los requerimientos funcionales. Se debe animar a los estudiantes a ensayar antes de grabar el video para asegurarse de que cubren todos los puntos en el tiempo indicado.

4. **Revisión y Feedback**

   - **Entrega del Reto:** Una vez finalizado el reto, los estudiantes deberán compartir el enlace al repositorio de GitHub y el enlace del video de presentación en YouTube. Deben asegurarse de que el repositorio esté bien documentado, con un archivo README que explique cómo ejecutar el proyecto.

   - **Feedback:** Los docentes revisarán tanto el código en el repositorio como el video explicativo para proporcionar retroalimentación constructiva. Se evaluará la correcta implementación de los requerimientos funcionales, el uso adecuado de Git, y la claridad en la presentación. Los estudiantes deben estar preparados para recibir sugerencias de mejora y podrán solicitar una reunión para discutir el feedback recibido si lo consideran necesario.

Requerimientos Funcionales para la Aplicación Simulador de Viaje Interplanetario

1. **Selección de Planeta de Destino**

   - **Elegir un planeta para el viaje:** El sistema debe permitir al usuario seleccionar un planeta de destino para su viaje interplanetario. Los planetas disponibles serán Marte, Júpiter, y Saturno, cada uno con diferentes distancias desde la Tierra.

   - **Mostrar detalles del planeta:** Una vez seleccionado el planeta, el sistema debe mostrar información básica, como la distancia en millones de kilómetros y una breve descripción.

2. **Gestión de la Nave Espacial**

- **Seleccionar una nave para el viaje:** El sistema ofrecerá al usuario una lista de naves espaciales, cada una con características básicas, como la velocidad máxima y la capacidad de pasajeros.
- **Ingresar la cantidad de pasajeros:** El usuario deberá especificar la cantidad de pasajeros que viajarán. El sistema solo verificará que se ingrese un valor positivo, pero no limitará la cantidad máxima.
- **Cálculo de la duración del viaje:** Basado en la velocidad de la nave y la distancia al planeta seleccionado, el sistema calculará la duración estimada del viaje en días.

3. **Simulación del Viaje**

- **Simulación básica del progreso del viaje:** El sistema mostrará el avance del viaje mediante una serie de mensajes en la consola, indicando el porcentaje completado del trayecto.
- **Uso de estructuras de control para simular el viaje:** El progreso se calculará utilizando bucles básicos (por ejemplo, un bucle for para incrementar el progreso en intervalos regulares), y se presentarán mensajes condicionales (usando if y switch) para indicar etapas clave del viaje, como "Inicio del viaje", "Mitad del camino", y "Llegada al destino".

4. **Interacción con el Usuario**

- **Menú de opciones:** El sistema debe proporcionar un menú interactivo en consola que permita al usuario elegir entre las siguientes opciones:
  - Seleccionar un planeta de destino.
  - Seleccionar una nave espacial.
  - Iniciar la simulación del viaje.
  - Salir del programa.
- **Validación de opciones:** El sistema debe verificar que el usuario elija una opción válida en el menú. Si el usuario ingresa una opción no válida, se mostrará un mensaje y se volverá a presentar el menú.

5. **Manipulación de Datos**

- **Uso de variables y tipos de datos:** Los datos como la distancia al planeta, la velocidad de la nave, y la cantidad de pasajeros se manejarán utilizando tipos de datos básicos como int, double, y String.

- **Almacenamiento de planetas y naves en arrays:** Los planetas y las naves espaciales se almacenarán en arrays para facilitar su selección e información. Por ejemplo, un array de strings para los nombres de los planetas y otro array para las distancias correspondientes.

6. **Modularidad y Métodos**

   - **Descomposición del programa en métodos:** El sistema debe utilizar métodos para separar la lógica de selección del planeta, cálculo de la duración del viaje, y simulación del trayecto. Cada funcionalidad principal se implementará en un método separado.

   - **Uso de parámetros y retorno de valores:** Los métodos aceptarán parámetros para recibir datos del usuario (por ejemplo, la distancia y velocidad) y retornarán resultados como la duración del viaje.

Descripción General del Flujo del Sistema

1. **Inicio del Programa:**

   - El programa se inicia y muestra al usuario un menú interactivo. El usuario puede elegir la acción que desea realizar (seleccionar un planeta, elegir una nave, iniciar el viaje, o salir del programa).

2. **Selección del Planeta y la Nave:**

   - El usuario elige un planeta de destino de una lista proporcionada por el sistema. El sistema muestra la distancia y una breve descripción del planeta.

   - A continuación, el usuario selecciona una nave para el viaje y especifica la cantidad de pasajeros.

3. **Simulación del Viaje:**

   - Al iniciar la simulación, el sistema muestra el progreso del viaje en intervalos, indicando etapas clave y calculando la duración total del trayecto en función de la velocidad de la nave y la distancia al planeta.

4. **Finalización del Viaje:**

   - Una vez que el viaje alcanza el 100% de progreso, el sistema muestra un mensaje indicando la llegada exitosa al planeta de destino.

5. **Salir del Programa:**

   - Si el usuario selecciona la opción para salir, el sistema cierra el programa de manera correcta.

Rúbrica de Evaluación para el Reto 1: "Simulador de Viaje Interplanetario"

Esta rúbrica detalla los criterios de evaluación para el Reto 1 del curso "Java de Cero a Senior: La Travesía Definitiva". Está diseñada para garantizar que se consideren todos los aspectos del proyecto, desde la implementación técnica hasta la presentación del trabajo. La calificación total será de 100 puntos.

Criterio	Descripción	Puntos
1. Implementación Técnica		**50**
Selección de Planeta y Nave	El sistema permite seleccionar un planeta y una nave, mostrando la información correcta del destino y la nave.	10
Cálculo de la Duración del Viaje	El sistema calcula correctamente la duración del viaje en función de la distancia y la velocidad de la nave.	10
Simulación del Progreso del Viaje	El programa muestra el avance del viaje de manera clara, utilizando bucles y condicionales para simular etapas.	10
Interacción con el Usuario	El menú del programa funciona correctamente, permitiendo la selección de opciones y validando entradas.	10
Manipulación de Datos y Modularidad	Uso adecuado de variables, arrays y métodos para gestionar la lógica del programa de manera modular.	10
2. Uso de Git y GitHub		**20**
Uso de ramas y commits regulares	Se realizaron commits frecuentes y se utilizaron ramas para separar funcionalidades y evitar conflictos.	10
Mensajes de commits descriptivos	Los mensajes de los commits son claros y explican los cambios realizados en cada etapa del desarrollo.	10
3. Presentación del Video		**20**
Introducción y descripción del reto	Se presenta claramente el reto, describiendo los objetivos y la funcionalidad del simulador.	5

Demostración de la funcionalidad	El video muestra el funcionamiento completo del programa, incluyendo selección de planeta, nave y simulación.	10
Claridad y equidad en la explicación	Ambos estudiantes participan activamente en la presentación, explicando diferentes partes del proyecto de manera clara y comprensible.	5
4. Documentación		**10**
Comentarios en el código	El código está bien comentado, explicando las funciones y bloques clave.	5
README en GitHub	El repositorio contiene un archivo README bien redactado, que describe el proyecto, cómo ejecutarlo y sus características.	5

Desglose de Puntos:

- **Implementación Técnica:** 50 puntos
- **Uso de Git y GitHub:** 20 puntos
- **Presentación del Video:** 20 puntos
- **Documentación:** 10 puntos

Total Máximo: 100 puntos

Puntuación Final:

- **Excelente (90-100 puntos):** El proyecto está completamente funcional, el uso de Git/GitHub es adecuado, y la presentación en video es clara y detallada.
- **Bueno (80-89 puntos):** El proyecto es funcional con algunos detalles mejorables. La interacción es adecuada, pero podría haber mayor claridad en la documentación o el video.
- **Satisfactorio (70-79 puntos):** El proyecto cumple con la mayoría de los requerimientos, pero presenta errores en la funcionalidad o en el uso de Git/GitHub. La presentación necesita mejoras.
- **Insuficiente (menos de 70 puntos):** El proyecto tiene deficiencias graves en la implementación, uso de Git/GitHub o la presentación en video.