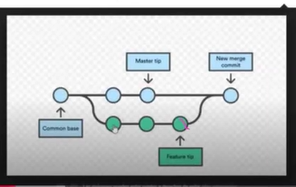
<https://github.com/pacho2712/Reto01.git>

[Evc2806/Java (github.com)](https://github.com/Evc2806/Java)



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 o 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 o 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 o 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 o 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 o 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. @devseniorcode Beneficios del Reto 1. Consolidación de Conceptos Básicos de Java o 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 o 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 o 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 o 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 o Presentar al usuario una lista de planetas del sistema solar para elegir el destino del viaje. o 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 o Calcular la distancia al planeta elegido y el tiempo estimado para completar el viaje, utilizando una velocidad fija (por ejemplo, 100,000 km/h). o Mostrar la información al usuario. 3. Gestión de Recursos o Calcular la cantidad de combustible y oxígeno necesarios para el viaje según la distancia. o Permitir al usuario revisar y ajustar los recursos antes de partir. 4. Simulación de Eventos Aleatorios o Durante el viaje, simular eventos aleatorios que afecten el progreso, como desvíos o fallos en el sistema. @devseniorcode o Solicitar al usuario tomar decisiones para resolver los problemas (por ejemplo, reparar la nave o cambiar de rumbo). 5. Monitoreo del Estado del Viaje o Mostrar en pantalla el progreso del viaje (porcentaje de la distancia total), el tiempo restante, y la cantidad de recursos disponibles. o Notificar al usuario si el viaje fue exitoso o si la nave se quedó sin recursos. 6. Interacción con el Usuario o 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)

EMPIEZA

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 o 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 o 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 o 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 o 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 o 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. @devseniorcode Beneficios del Reto 1. Consolidación de Conceptos Básicos de Java o 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 o 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 o 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 o 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 o Presentar al usuario una lista de planetas del sistema solar para elegir el destino del viaje. o 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 o Calcular la distancia al planeta elegido y el tiempo estimado para completar el viaje, utilizando una velocidad fija (por ejemplo, 100,000 km/h). o Mostrar la información al usuario. 3. Gestión de Recursos o Calcular la cantidad de combustible y oxígeno necesarios para el viaje según la distancia. o Permitir al usuario revisar y ajustar los recursos antes de partir. 4. Simulación de Eventos Aleatorios o Durante el viaje, simular eventos aleatorios que afecten el progreso, como desvíos o fallos en el sistema. @devseniorcode o Solicitar al usuario tomar decisiones para resolver los problemas (por ejemplo, reparar la nave o cambiar de rumbo). 5. Monitoreo del Estado del Viaje o Mostrar en pantalla el progreso del viaje (porcentaje de la distancia total), el tiempo restante, y la cantidad de recursos disponibles. o Notificar al usuario si el viaje fue exitoso o si la nave se quedó sin recursos. 6. Interacción con el Usuario o 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) en java

Selección de Destino Interplanetario

import java.util.Scanner;

public class SimuladorViajeInterplanetario {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String[] planetas = {"Mercurio", "Venus", "Marte", "Júpiter", "Saturno"};

double[] distancias = {91600000, 41400000, 78300000, 628730000, 1275000000}; // Distancias en km

System.out.println("Selecciona tu destino:");

for (int i = 0; i < planetas.length; i++) {

System.out.println((i + 1) + ". " + planetas[i]);

}

int eleccion = scanner.nextInt();

if (eleccion < 1 || eleccion > planetas.length) {

System.out.println("Selección inválida.");

return;

}

String planetaSeleccionado = planetas[eleccion - 1];

double distancia = distancias[eleccion - 1];

System.out.println("Has seleccionado: " + planetaSeleccionado);

System.out.println("Distancia desde la Tierra: " + distancia + " km");

// Calcular tiempo de viaje

double velocidad = 100000; // km/h

double tiempoViaje = distancia / velocidad;

System.out.println("Tiempo estimado de viaje: " + tiempoViaje + " horas");

// Llamar a otros métodos para continuar con la simulación

gestionarRecursos(distancia);

simularEventosAleatorios();

monitorearEstadoViaje(tiempoViaje, distancia);

}

// Métodos adicionales para gestionar recursos, simular eventos aleatorios y monitorear el estado del viaje

}

Gestión de Recursos

public static void gestionarRecursos(double distancia) {

// Supongamos que la nave consume 0.1 unidades de combustible por km y 0.05 unidades de oxígeno por km

double combustibleNecesario = distancia \* 0.1;

double oxigenoNecesario = distancia \* 0.05;

System.out.println("Combustible necesario: " + combustibleNecesario + " unidades");

System.out.println("Oxígeno necesario: " + oxigenoNecesario + " unidades");

// Permitir al usuario ajustar recursos (se podría implementar una lógica adicional aquí)

}

Simulación de Eventos Aleatorios

public static void simularEventosAleatorios() {

// Generar eventos aleatorios (por ejemplo, utilizando la clase Random)

System.out.println("Simulación de eventos aleatorios en progreso...");

// Lógica de eventos aleatorios aquí

}

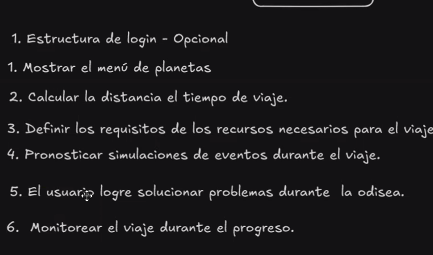
Monitoreo del Estado del Viaje

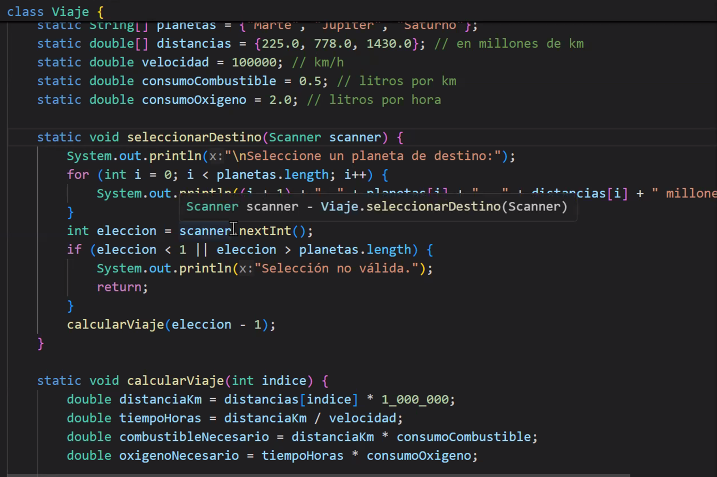
public static void monitorearEstadoViaje(double tiempoViaje, double distancia) {

// Monitorear el progreso del viaje

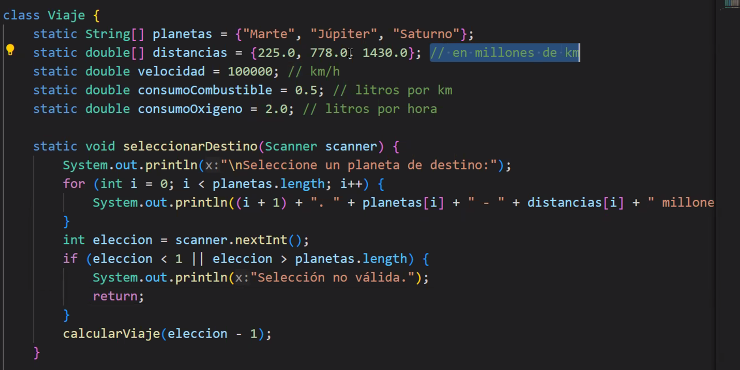
System.out.println("Monitoreando el estado del viaje...");

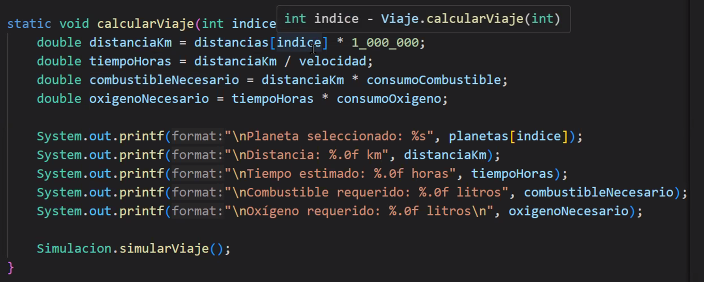
// Lógica de monitoreo aquí

}

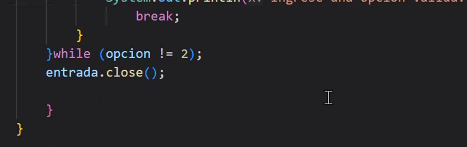


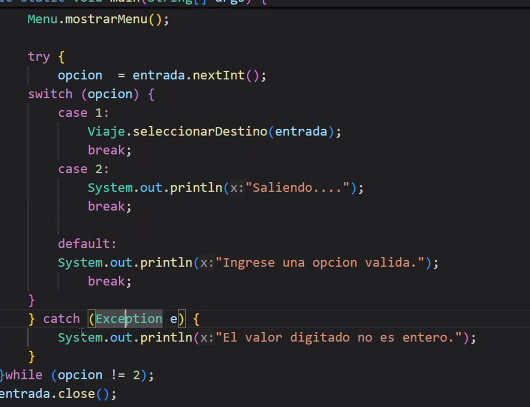


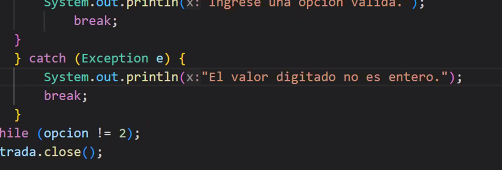












1. ava
2. import java.util.Scanner;

Esto importa la clase Scanner de la biblioteca java.util, que se utiliza para leer la entrada del usuario.

1. **Declaración de la clase SimuladorVuelo**:

java

public class SimuladorVuelo {

Aquí se define la clase principal del programa.

1. **Método principal (main)**:

java

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String[] planetas = {"Mercurio", "Venus", "Marte"};

double[] distancias = {91600000, 41400000, 78300000};

Se inicializa el objeto Scanner para leer la entrada del usuario. También se crean dos arreglos: planetas que contiene los nombres de los planetas y distancias que contiene las distancias desde la Tierra en kilómetros.

1. **Selección de destino**:

java

System.out.println("Selecciona tu destino:");

for (int i = 0; i < planetas.length; i++) {

System.out.println((i + 1) + ". " + planetas[i]);

}

Se imprime una lista de planetas para que el usuario elija su destino.

1. **Lectura de la elección del usuario**:

java

int eleccion = scanner.nextInt();

if (eleccion < 1 || eleccion > planetas.length) {

System.out.println("Selección inválida.");

return;

}

El programa lee la elección del usuario y verifica que sea válida. Si no lo es, imprime un mensaje de error y termina la ejecución.

1. **Mostrar selección y calcular distancia y tiempo de viaje**:

java

String planetaSeleccionado = planetas[eleccion - 1];

double distancia = distancias[eleccion - 1];

System.out.println("Has seleccionado: " + planetaSeleccionado);

System.out.println("Distancia desde la Tierra: " + distancia + " km");

double velocidad = 100000; // km/h

double tiempoViaje = distancia / velocidad;

System.out.println("Tiempo estimado de viaje: " + tiempoViaje + " horas");

El programa muestra el planeta seleccionado, la distancia desde la Tierra y el tiempo estimado de viaje calculado en función de una velocidad constante de 100,000 km/h.

1. **Gestión de recursos, eventos aleatorios y monitoreo del viaje**:

java

gestionarRecursos(distancia);

simularEventosAleatorios();

monitorearEstadoViaje(tiempoViaje, distancia);

Se llaman a tres métodos adicionales para gestionar recursos, simular eventos aleatorios y monitorear el estado del viaje.

1. **Método para gestionar recursos**:

java

public static void gestionarRecursos(double distancia) {

double combustibleNecesario = distancia \* 0.1;

double oxigenoNecesario = distancia \* 0.05;

System.out.println("Combustible necesario: " + combustibleNecesario + " unidades");

System.out.println("Oxígeno necesario: " + oxigenoNecesario + " unidades");

}

Este método calcula y muestra la cantidad de combustible y oxígeno necesarios para el viaje.

1. **Método para simular eventos aleatorios**:

java

public static void simularEventosAleatorios() {

System.out.println("Simulación de eventos aleatorios...");

// Lógica de eventos aleatorios aquí

}

Este método simula eventos aleatorios durante el viaje. La lógica específica de los eventos se debe implementar.

1. **Método para monitorear el estado del viaje**:

java

public static void monitorearEstadoViaje(double tiempoViaje, double distancia) {

System.out.println("Monitoreando el estado del viaje...");

// Lógica de monitoreo aquí

}

Este método monitorea el progreso del viaje. La lógica específica de monitoreo se debe implementar.

Espero que esta explicación te ayude a entender mejor el algoritmo. Si tienes alguna pregunta o necesitas más detalles, ¡déjame saber!

public class AvionMonitor {

private int numeroPasajeros;

private int cantidadAlimentacion;

private double combustibleEstable;

private double proximidadAterrizaje;

// Constructor

public AvionMonitor(int numeroPasajeros, int cantidadAlimentacion, double combustibleEstable, double proximidadAterrizaje) {

this.numeroPasajeros = numeroPasajeros;

this.cantidadAlimentacion = cantidadAlimentacion;

this.combustibleEstable = combustibleEstable;

this.proximidadAterrizaje = proximidadAterrizaje;

}

// Métodos para monitorear el avión

public void mostrarEstadoAvion() {

System.out.println("Estado del Avión:");

System.out.println("Número de Pasajeros: " + numeroPasajeros);

System.out.println("Cantidad de Alimentación: " + cantidadAlimentacion);

System.out.println("Combustible Estable: " + combustibleEstable + " litros");

System.out.println("Proximidad al Aterrizaje: " + proximidadAterrizaje + " km");

}

public static void main(String[] args) {

// Ejemplo de uso

AvionMonitor avion = new AvionMonitor(150, 300, 5000.0, 50.0);

avion.mostrarEstadoAvion();

}

}

public class AvionMonitor {

private int numeroPasajeros;

private int cantidadAlimentacion;

private double combustibleEstable;

private double proximidadAterrizaje;

private double nivelOxigeno;

private double velocidadKmH;

// Constructor

public AvionMonitor(int numeroPasajeros, int cantidadAlimentacion, double combustibleEstable,

double proximidadAterrizaje, double nivelOxigeno, double velocidadKmH) {

this.numeroPasajeros = numeroPasajeros;

this.cantidadAlimentacion = cantidadAlimentacion;

this.combustibleEstable = combustibleEstable;

this.proximidadAterrizaje = proximidadAterrizaje;

this.nivelOxigeno = nivelOxigeno;

this.velocidadKmH = velocidadKmH;

}

// Métodos para monitorear el avión

public void mostrarEstadoAvion() {

System.out.println("Estado del Avión:");

System.out.println("Número de Pasajeros: " + numeroPasajeros);

System.out.println("Cantidad de Alimentación: " + cantidadAlimentacion);

System.out.println("Combustible Estable: " + combustibleEstable + " litros");

System.out.println("Proximidad al Aterrizaje: " + proximidadAterrizaje + " km");

System.out.println("Nivel de Oxígeno: " + nivelOxigeno + " %");

System.out.println("Velocidad: " + velocidadKmH + " km/h");

}

public static void main(String[] args) {

// Ejemplo de uso

AvionMonitor avion = new AvionMonitor(150, 300, 5000.0, 50.0, 21.0, 900.0);

avion.mostrarEstadoAvion();

}

}