Importaciones y Definiciones de Variables Globales

java

import java.util.Random;

public class App {

// Definir variables globales

static Scanner sc = new Scanner(System.in);

static Random r = new Random();

Importaciones:

java

import java.util.Random;

Importa la clase Random de la biblioteca estándar de Java, que se usa para generar números aleatorios.

Clase Principal:

java

public class App {

Define la clase principal App.

Variables Globales:

java

static Scanner sc = new Scanner(System.in);

static Random r = new Random();

Scanner sc: Se utiliza para leer la entrada del usuario desde la consola.

Random r: Se utiliza para generar números aleatorios, lo que será útil para simular eventos aleatorios durante el viaje.

Definiciones de Arrays de Planetas y Naves

java

static String[] planetas = { "Marte: Es un planeta frio y oscuro",

"Jupiter: Es el planeta mas grande del sistema solar",

"Neptuno: Es el planeta mas lejano del sistema solar",

"mercurio: Mercurio es el planeta más pequeño del sistema solar con un diámetro de 4,879

"venus: Se caracteriza por ser un planta rocoso, sin satélites ni anillos y con una atmó

"saturno: Saturno es el sexto planeta desde el Sol, y el último de los planetas conocido

"urano: Urano es un planeta gaseoso con una gran cantidad de metano en su superficie, l

Array de Planetas:

java

static String[] planetas = { "Marte: Es un planeta frio y oscuro",

"Jupiter: Es el planeta mas grande del sistema solar",

"Neptuno: Es el planeta mas lejano del sistema solar",

"mercurio: Mercurio es el planeta más pequeño del sistema solar con un diámetro de 4,879 km",

"venus: Se caracteriza por ser un planta rocoso, sin satélites ni anillos y con una atmósfera muy densa",

"saturno: Saturno es el sexto planeta desde el Sol, y el último de los planetas conocido por las civilizaciones antiguas",

"urano: Urano es un planeta gaseoso con una gran cantidad de metano en su superficie, lo que le confiere un tono azulado"

};

Define un array planetas que contiene descripciones de varios planetas del sistema solar. Cada elemento es una cadena de texto que incluye el nombre del planeta y una breve descripción.

Definiciones de Arrays de Distancias y Naves

java

static double[] distancias = { 225.0, 588.0, 4500.0, 77.0, 41.0, 1.2000, 2.871 }; // Distancia en millones de kilómetros

static String[] naves = {

"Nave Explorer: Capacidad para 5 personas, velocidad máxima 20,000 km/h",

"Nave Voyager: Capacidad para 8 personas, velocidad máxima 25,000 km/h",

"Nave Pioneer: Capacidad para 10 personas, velocidad máxima 22,000 km/h"

};

Array de Distancias:

java

static double[] distancias = { 225.0, 588.0, 4500.0, 77.0, 41.0, 1.2000, 2.871 };

Define un array distancias que contiene las distancias desde la Tierra hasta cada uno de los planetas listados en planetas. Las distancias se dan en millones de kilómetros.

Array de Naves:

java

static String[] naves = {

"Nave Explorer: Capacidad para 5 personas, velocidad máxima 20,000 km/h",

"Nave Voyager: Capacidad para 8 personas, velocidad máxima 25,000 km/h",

"Nave Pioneer: Capacidad para 10 personas, velocidad máxima 22,000 km/h"

};

Define un array naves que contiene descripciones de varias naves espaciales. Cada elemento es una cadena de texto que incluye el nombre de la nave, su capacidad y su velocidad máxima.

Resumen

Hasta aquí, hemos definido:

Las importaciones necesarias.

La clase principal App.

Variables globales para el lector de entrada Scanner y el generador de números aleatorios Random.

Arrays que contienen descripciones de planetas y naves espaciales, así como las distancias a esos planetas.

Definición de Constantes de Códigos de Color

java

// Definir constantes para códigos de color+

static final String GREEN = "\033[32m"; // Verde para barra de progreso

static final String YELLOW = "\033[33m"; // Amarillo para barra de progreso

static final String BLUE = "\033[34m"; // Azul para subtitulos

static final String ORANGE = "\033[38;5;208m"; // Naranja en las recomendaciones

static final String BOLD = "\033[1m"; // Negrita en subtitulos

static final String RESET = "\033[0m"; // Resetea el color

static final String BRIGHT\_BLUE = "\033[94m"; // Azul brillante Titulos

static final String BRIGHT\_GREEN = "\033[92m"; // Verde brillante Soluciones

static final String UNDERLINE = "\033[4m"; // Subrayado en titulos

static final String RED = "\033[32m"; // Rojo para barra de progreso

Aquí se definen varias constantes para códigos de colores ANSI, que se utilizarán para dar formato al texto en la consola.

Cada constante tiene un código de escape ANSI que se utiliza para cambiar el color o el estilo del texto.

Verificación de Selección de Nave y Planeta

java

// Verificación de selección de nave y planeta

static boolean isPlanetSelected = false; // Verifica si se ha seleccionado un planeta

static boolean isShipSelected = false; // Verifica si se ha seleccionado una nave

static int selectedShipIndex = -1; // Indice de nave seleccionada

static int selectedPlanetIndex = -1; // Indice de planeta seleccionado

Estas variables se utilizan para rastrear si un planeta o una nave han sido seleccionados:

isPlanetSelected: Indica si se ha seleccionado un planeta.

isShipSelected: Indica si se ha seleccionado una nave.

selectedShipIndex: Almacena el índice de la nave seleccionada.

selectedPlanetIndex: Almacena el índice del planeta seleccionado.

Método main

java

public static void main(String[] args) {

int opcion;

do {

showMenu();

opcion = sc.nextInt();

switch (opcion) {

case 1:

seleccionarPlaneta();

break;

case 2:

seleccionarNaveEspacial();

break;

case 3:

calcularRecursos();

break;

case 4:

iniciarViaje();

break;

case 5:

monitorearEstadoViaje();

break;

case 6:

simularEventosinconvenientes();

break;

case 7:

System.out.println("Saliendo del programa.......");

break;

default:

System.out.println("Opción no validad !!!!!");

break;

}

} while (opcion != 7);

sc.close();

}

Desglose

Declaración de la Variable opcion:

java

int opcion;

Bucle do-while:

java

do {

showMenu();

opcion = sc.nextInt();

switch (opcion) {

// Opciones del menú

}

} while (opcion != 7);

sc.close();

El bucle do-while se ejecuta al menos una vez y continúa mientras opcion no sea igual a 7.

showMenu(): Muestra el menú de opciones.

opcion = sc.nextInt(): Lee la entrada del usuario.

Sentencia switch:

java

switch (opcion) {

case 1:

seleccionarPlaneta();

break;

case 2:

seleccionarNaveEspacial();

break;

case 3:

calcularRecursos();

break;

case 4:

iniciarViaje();

break;

case 5:

monitorearEstadoViaje();

break;

case 6:

simularEventosinconvenientes();

break;

case 7:

System.out.println("Saliendo del programa.......");

break;

default:

System.out.println("Opción no validad !!!!!");

break;

}

La sentencia switch maneja las diferentes opciones del menú:

case 1: Llama al método seleccionarPlaneta().

case 2: Llama al método seleccionarNaveEspacial().

case 3: Llama al método calcularRecursos().

case 4: Llama al método iniciarViaje().

case 5: Llama al método monitorearEstadoViaje().

case 6: Llama al método simularEventosinconvenientes().

case 7: Muestra un mensaje de salida y termina el bucle.

default: Maneja cualquier opción no válida ingresada por el usuario.

Cerrar el Escáner:

java

sc.close();

Cierra el objeto Scanner para liberar los recursos asociados con él.

Resumen

Este fragmento de código define constantes para formatear el texto en la consola y declara variables para rastrear las selecciones del usuario (planeta y nave). Luego, en el método main, se muestra un menú de opciones en un bucle do-while, y según la opción seleccionada por el usuario, se llama a diferentes métodos para realizar diversas acciones. El programa continúa ejecutándose hasta que el usuario selecciona la opción 7 para salir.

Método showMenu

java

public static void showMenu() {

System.out.println(BRIGHT\_BLUE + UNDERLINE + BOLD+"\n\t=== SIMULADOR DE VAIJES INTERPLANETARIO ===\n" + RESET);

System.out.println(BLUE+BOLD + "1)." + RESET + " Planetas destino");

System.out.println(BLUE+BOLD + "2)." + RESET + " Naves disponible");

System.out.println(BLUE+BOLD + "3)." + RESET + " Calcular recursos");

System.out.println(BLUE+BOLD + "4)." + RESET + " Iniciar Viaje");

System.out.println(BLUE+BOLD + "5)." + RESET + " Monitorear el estado del Viaje");

System.out.println(BLUE+BOLD + "6)." + RESET + " Simular eventos o inconvenientes");

System.out.println(BLUE+BOLD + "7)." + RESET + " Salir");

System.out.print(BOLD + "Selecciones una opción: " + RESET);

}

Desglose

Mostrar Menú: Este método imprime el menú principal del programa en la consola. Utiliza códigos de color y formato para hacer el menú más atractivo visualmente.

System.out.println(...): Muestra cada opción del menú, utilizando los códigos de color y formato definidos anteriormente.

System.out.print(...): Muestra un mensaje pidiendo al usuario que seleccione una opción.

Método seleccionarPlaneta

java

public static void seleccionarPlaneta() {

System.out.println(BOLD+BRIGHT\_BLUE+UNDERLINE+"\n\tMenú Planetas y sus distancias desde la Tierra "+RESET+"\n");

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

System.out.println(BOLD+BRIGHT\_BLUE+(i + 1) + "). " +RESET+ planetas[i] +BLUE+BOLD+ " - Distancia: " + distancias[i] + " millones de km."+RESET);

}

System.out.print(BOLD+"Elige el número de tu planeta destino: "+RESET);

var seleccion = sc.nextInt();

if (seleccion >= 1 && seleccion <= planetas.length) {

selectedPlanetIndex = seleccion - 1;

System.out.println(BRIGHT\_GREEN+"\nHas seleccionado " + planetas[selectedPlanetIndex] + "."+RESET);

isPlanetSelected = true;

} else {

System.err.println(ORANGE+"Selección no válida. Por favor, intenta de nuevo."+RESET);

}

}

Desglose

Mostrar Planetas: Este método imprime una lista de planetas y sus distancias desde la Tierra.

System.out.println(...): Muestra el título del menú.

for loop: Recorre el array planetas y muestra cada planeta junto con su distancia.

System.out.print(...): Pide al usuario que seleccione un planeta.

var seleccion = sc.nextInt(): Lee la selección del usuario.

Verificación de Selección:

Si la selección es válida, actualiza selectedPlanetIndex y isPlanetSelected y muestra el planeta seleccionado.

Si la selección no es válida, muestra un mensaje de error.

Método seleccionarNaveEspacial

java

public static void seleccionarNaveEspacial() {

if (!isPlanetSelected) {

System.err.println(ORANGE + "Primero debes seleccionar un planeta destino." + RESET);

return;

}

System.out.println(BOLD + BRIGHT\_BLUE + UNDERLINE + "\n\tMenú de Naves Espaciales Disponibles" + RESET + "\n");

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

System.out.println(BOLD + BRIGHT\_BLUE + (i + 1) + "). " + RESET + naves[i]);

}

System.out.print(BOLD + "Elige el número de tu nave espacial: " + RESET);

var seleccion = sc.nextInt();

if (seleccion >= 1 && seleccion <= naves.length) {

selectedShipIndex = seleccion - 1;

System.out.println(BRIGHT\_GREEN + "\nHas seleccionado " + naves[selectedShipIndex] + "." + RESET);

isShipSelected = true;

} else {

System.err.println(ORANGE + "Selección no válida. Por favor, intenta de nuevo." + RESET);

}

}

Desglose

Verificación de Selección de Planeta:

Verifica si se ha seleccionado un planeta antes de continuar. Si no se ha seleccionado un planeta, muestra un mensaje de error y termina el método con return.

Mostrar Naves: Este método imprime una lista de naves espaciales disponibles.

System.out.println(...): Muestra el título del menú.

for loop: Recorre el array naves y muestra cada nave.

System.out.print(...): Pide al usuario que seleccione una nave.

var seleccion = sc.nextInt(): Lee la selección del usuario.

Verificación de Selección:

Si la selección es válida, actualiza selectedShipIndex y isShipSelected y muestra la nave seleccionada.

Si la selección no es válida, muestra un mensaje de error.

Método valcularrRecursos

java

public static void valcularrRecursos(double distancia) {

// Supongamos que la nave consume 1.55 unidades de combustible por km y 0.15 unidades de oxígeno por km

double combustibleNecesario = distancia \* 1.55;

double oxigenoNecesario = distancia \* 0.15;

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)

}

Desglose

Calcular Recursos: Este método calcula la cantidad de combustible y oxígeno necesarios para el viaje.

Cálculo de Combustible: double combustibleNecesario = distancia \* 1.55;

Supone que la nave consume 1.55 unidades de combustible por kilómetro.

Cálculo de Oxígeno: double oxigenoNecesario = distancia \* 0.15;

Supone que la nave consume 0.15 unidades de oxígeno por kilómetro.

Mostrar Recursos: System.out.println(...)

Muestra la cantidad de combustible y oxígeno necesarios.

Ajuste de Recursos: // Permitir al usuario ajustar recursos...

Comentario indicando que se podría implementar lógica adicional para permitir que el usuario ajuste los recursos.

Resumen

Este fragmento de código incluye métodos para mostrar el menú principal, seleccionar un planeta, seleccionar una nave espacial y calcular los recursos necesarios para el viaje. Cada método utiliza bucles y verificaciones para asegurar que las selecciones del usuario sean válidas y proporcionar feedback adecuado.

Método simularEventosinconvenientes

java

public static void simularEventosinconvenientes() {

if (!isPlanetSelected) {

System.err.println(ORANGE + "Primero debes seleccionar un planeta destino." + RESET);

return;

}

if (!isShipSelected) {

System.err.println(ORANGE + "Primero debes seleccionar una nave espacial." + RESET);

return;

}

System.out.println(BRIGHT\_GREEN + "\nSimulación de eventos aleatorios en progreso..." + RESET);

String[] eventos = {

"Tormenta de meteoritos: La nave ha sufrido daños menores.",

"Agujero negro cercano: La trayectoria ha sido ligeramente desviada.",

"Se detecta una anomalía gravitacional: Velocidad reducida temporalmente.",

"Falla en los sistemas de soporte vital: Los sistemas se están reiniciando.",

"Encuentro con una nave alienígena: Intercambio de saludos amistosos."

};

// Generar un evento aleatorio

int eventoIndex = r.nextInt(eventos.length);

// Mostrar el evento aleatorio generado

System.out.println(YELLOW + eventos[eventoIndex] + RESET);

// Lógica adicional según el evento (puedes ajustar según necesites)

switch (eventoIndex) {

case 0: // Tormenta de meteoritos

System.out.println(RED + "Reparando daños menores..." + RESET);

break;

case 1: // Agujero negro cercano

System.out.println(BRIGHT\_BLUE + "Recalculando trayectoria..." + RESET);

break;

case 2: // Anomalía gravitacional

System.out.println(BRIGHT\_BLUE + "Velocidad reducida temporalmente." + RESET);

break;

case 3: // Falla en los sistemas de soporte vital

System.out.println(RED + "Reiniciando sistemas de soporte vital..." + RESET);

break;

case 4: // Encuentro con una nave alienígena

System.out.println(BRIGHT\_GREEN + "Intercambio de saludos completado." + RESET);

break;

}

}

Desglose

Verificación de Selección de Planeta y Nave:

Verifica si se ha seleccionado un planeta y una nave antes de continuar.

Si no se ha seleccionado un planeta, muestra un mensaje de error y termina el método.

Si no se ha seleccionado una nave, muestra un mensaje de error y termina el método.

Mensaje de Progreso:

java

System.out.println(BRIGHT\_GREEN + "\nSimulación de eventos aleatorios en progreso..." + RESET);

Muestra un mensaje indicando que la simulación de eventos aleatorios está en progreso.

Definición de Eventos:

java

String[] eventos = {

"Tormenta de meteoritos: La nave ha sufrido daños menores.",

"Agujero negro cercano: La trayectoria ha sido ligeramente desviada.",

"Se detecta una anomalía gravitacional: Velocidad reducida temporalmente.",

"Falla en los sistemas de soporte vital: Los sistemas se están reiniciando.",

"Encuentro con una nave alienígena: Intercambio de saludos amistosos."

};

Define un array eventos que contiene diferentes eventos posibles que pueden ocurrir durante el viaje.

Generación de Evento Aleatorio:

java

int eventoIndex = r.nextInt(eventos.length);

Genera un índice aleatorio que corresponde a un evento en el array eventos.

Mostrar Evento Generado:

java

System.out.println(YELLOW + eventos[eventoIndex] + RESET);

Muestra el evento aleatorio seleccionado en color amarillo.

Lógica Adicional Según el Evento:

java

switch (eventoIndex) {

case 0: // Tormenta de meteoritos

System.out.println(RED + "Reparando daños menores..." + RESET);

break;

case 1: // Agujero negro cercano

System.out.println(BRIGHT\_BLUE + "Recalculando trayectoria..." + RESET);

break;

case 2: // Anomalía gravitacional

System.out.println(BRIGHT\_BLUE + "Velocidad reducida temporalmente." + RESET);

break;

case 3: // Falla en los sistemas de soporte vital

System.out.println(RED + "Reiniciando sistemas de soporte vital..." + RESET);

break;

case 4: // Encuentro con una nave alienígena

System.out.println(BRIGHT\_GREEN + "Intercambio de saludos completado." + RESET);

break;

}

Dependiendo del índice del evento seleccionado, se ejecuta un caso específico en el switch que muestra un mensaje adicional relacionado con el evento.

Método calcularRecursos

java

public static void calcularRecursos() {

if (!isPlanetSelected) {

System.err.println(ORANGE + "Primero debes seleccionar un planeta destino." + RESET);

return;

}

if (!isShipSelected) {

System.err.println(ORANGE + "Primero debes seleccionar una nave espacial." + RESET);

return;

}

double distancia = distancias[selectedPlanetIndex];

double combustibleNecesario = distancia \* 1.55; // Suponemos que la nave consume 1.55 unidades de combustible por km

double oxigenoNecesario = distancia \* 0.15; // Suponemos que la nave consume 0.15 unidades de oxígeno por km

System.out.println(BRIGHT\_BLUE + "\nRecursos necesarios para el viaje a " + planetas[selectedPlanetIndex] + ":" + RESET);

System.out.println(BRIGHT\_GREEN + "Combustible necesario: " + combustibleNecesario + " unidades" + RESET);

System.out.println(BRIGHT\_GREEN + "Oxígeno necesario: " + oxigenoNecesario + " unidades" + RESET);

// Permitir al usuario ajustar recursos (puedes implementar lógica adicional aquí si es necesario)

}

Desglose

Verificación de Selección de Planeta y Nave:

Verifica si se ha seleccionado un planeta y una nave antes de continuar.

Si no se ha seleccionado un planeta, muestra un mensaje de error y termina el método.

Si no se ha seleccionado una nave, muestra un mensaje de error y termina el método.

Cálculo de Recursos:

Cálculo de Combustible: double combustibleNecesario = distancia \* 1.55;

Calcula la cantidad de combustible necesario basado en la distancia al planeta seleccionado.

Cálculo de Oxígeno: double oxigenoNecesario = distancia \* 0.15;

Calcula la cantidad de oxígeno necesario basado en la distancia al planeta seleccionado.

Mostrar Recursos Necesarios:

java

System.out.println(BRIGHT\_BLUE + "\nRecursos necesarios para el viaje a " + planetas[selectedPlanetIndex] + ":" + RESET);

System.out.println(BRIGHT\_GREEN + "Combustible necesario: " + combustibleNecesario + " unidades" + RESET);

System.out.println(BRIGHT\_GREEN + "Oxígeno necesario: " + oxigenoNecesario + " unidades" + RESET);

Muestra la cantidad de combustible y oxígeno necesarios para el viaje en la consola.

Ajuste de Recursos:

Comentario indicando que se podría implementar lógica adicional para permitir que el usuario ajuste los recursos.

Método monitorearEstadoViaje

java

public static void monitorearEstadoViaje() {

if (!isPlanetSelected) {

System.err.println(ORANGE + "Primero debes seleccionar un planeta destino." + RESET);

return;

}

if (!isShipSelected) {

System.err.println(ORANGE + "Primero debes seleccionar una nave espacial." + RESET);

return;

}

System.out.println(BRIGHT\_GREEN + "Monitoreando el estado del viaje hacia " + planetas[selectedPlanetIndex] + "..." + RESET);

Desglose

Verificación de Selección de Planeta y Nave:

Verifica si se ha seleccionado un planeta y una nave antes de continuar.

Si no se ha seleccionado un planeta, muestra un mensaje de error y termina el método.

Si no se ha seleccionado una nave, muestra un mensaje de error y termina el método.

Mensaje de Monitoreo:

java

Resumen

Este fragmento de código incluye métodos para simular eventos inconvenientes, calcular los recursos necesarios para el viaje y monitorear el estado del viaje. Cada método utiliza verificaciones para asegurar que se haya seleccionado un planeta y una nave antes de proceder, y proporciona feedback adecuado en la consola.

Método monitorearEstadoViaje

java

System.out.println(BRIGHT\_GREEN + "Monitoreando el estado del viaje hacia " + planetas[selectedPlanetIndex] + "..." + RESET);

double distanciaTotal = distancias[selectedPlanetIndex];

double distanciaRecorrida = 0;

double velocidad = 100; // Suponiendo una velocidad constante de 100 millones de km por ciclo

double combustible = distanciaTotal \* 1.55;

double oxigeno = distanciaTotal \* 0.15;

Desglose

Mensaje de Monitoreo:

java

System.out.println(BRIGHT\_GREEN + "Monitoreando el estado del viaje hacia " + planetas[selectedPlanetIndex] + "..." + RESET);

Muestra un mensaje indicando que el estado del viaje hacia el planeta seleccionado está siendo monitoreado.

Inicialización de Variables:

double distanciaTotal = distancias[selectedPlanetIndex];

Calcula la distancia total al planeta seleccionado.

double distanciaRecorrida = 0;

Inicializa la distancia recorrida a 0.

double velocidad = 100;

Supone una velocidad constante de 100 millones de km por ciclo.

double combustible = distanciaTotal \* 1.55;

Calcula la cantidad total de combustible necesario.

double oxigeno = distanciaTotal \* 0.15;

Calcula la cantidad total de oxígeno necesario.

Bucle while para Monitorear el Progreso del Viaje

java

while (distanciaRecorrida < distanciaTotal) {

distanciaRecorrida += velocidad;

if (distanciaRecorrida > distanciaTotal) {

distanciaRecorrida = distanciaTotal;

}

combustible -= velocidad \* 1.55;

oxigeno -= velocidad \* 0.15;

System.out.println(BRIGHT\_BLUE + "Progreso del viaje: " + RESET + (distanciaRecorrida / distanciaTotal \* 100) + "%");

System.out.println(BRIGHT\_BLUE + "Combustible restante: " + RESET + combustible + " unidades");

System.out.println(BRIGHT\_BLUE + "Oxígeno restante: " + RESET + oxigeno + " unidades");

try {

Thread.sleep(1000); // Pausa de 1 segundo para simular el tiempo de viaje

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println(BRIGHT\_GREEN + "\n¡El viaje ha sido completado! Has llegado a " + planetas[selectedPlanetIndex] + "." + RESET);

Desglose

Bucle while:

java

while (distanciaRecorrida < distanciaTotal) {

El bucle continúa ejecutándose mientras la distancia recorrida sea menor que la distancia total.

Actualizar Distancia Recorrida:

java

distanciaRecorrida += velocidad;

if (distanciaRecorrida > distanciaTotal) {

distanciaRecorrida = distanciaTotal;

}

Incrementa la distancia recorrida por la velocidad.

Si la distancia recorrida supera la distancia total, ajusta la distancia recorrida a la distancia total.

Actualizar Combustible y Oxígeno:

java

combustible -= velocidad \* 1.55;

oxigeno -= velocidad \* 0.15;

Disminuye el combustible y el oxígeno según la distancia recorrida.

Mostrar Progreso:

java

System.out.println(BRIGHT\_BLUE + "Progreso del viaje: " + RESET + (distanciaRecorrida / distanciaTotal \* 100) + "%");

System.out.println(BRIGHT\_BLUE + "Combustible restante: " + RESET + combustible + " unidades");

System.out.println(BRIGHT\_BLUE + "Oxígeno restante: " + RESET + oxigeno + " unidades");

Muestra el progreso del viaje, el combustible restante y el oxígeno restante en la consola.

Simulación de Tiempo de Viaje:

java

try {

Thread.sleep(1000); // Pausa de 1 segundo para simular el tiempo de viaje

} catch (InterruptedException e) {

e.printStackTrace();

}

Pausa la ejecución durante 1 segundo para simular el tiempo de viaje.

Mensaje de Llegada:

java

System.out.println(BRIGHT\_GREEN + "\n¡El viaje ha sido completado! Has llegado a " + planetas[selectedPlanetIndex] + "." + RESET);

Muestra un mensaje indicando que el viaje ha sido completado y se ha llegado al planeta seleccionado.

Método iniciarViaje

java

public static void iniciarViaje() {

if (!isPlanetSelected) {

System.err.println(ORANGE + "Primero debes seleccionar un planeta destino." + RESET);

return;

}

if (!isShipSelected) {

System.err.println(ORANGE + "Primero debes seleccionar una nave espacial." + RESET);

return;

}

System.out.println(BRIGHT\_GREEN + "Iniciando el viaje hacia " + planetas[selectedPlanetIndex] + "..." + RESET);

// Simular el progreso del viaje

for (int i = 0; i <= 100; i += 10) {

try {

Thread.sleep(500); // Pausa de medio segundo para simular progreso

} catch (InterruptedException e) {

e.printStackTrace();

}

if (i < 50) {

System.out.print(GREEN + "=" + RESET);

} else if (i < 80) {

System.out.print(YELLOW + "=" + RESET);

} else {

System.out.print(RED + "=" + RESET);

}

}

System.out.println(BRIGHT\_GREEN + "\n¡Has llegado a " + planetas[selectedPlanetIndex] + "!" + RESET);

}

Desglose

Verificación de Selección de Planeta y Nave:

java

if (!isPlanetSelected) {

System.err.println(ORANGE + "Primero debes seleccionar un planeta destino." + RESET);

return;

}

if (!isShipSelected) {

System.err.println(ORANGE + "Primero debes seleccionar una nave espacial." + RESET);

return;

}

Verifica si se ha seleccionado un planeta y una nave antes de continuar.

Si no se ha seleccionado un planeta, muestra un mensaje de error y termina el método.

Si no se ha seleccionado una nave, muestra un mensaje de error y termina el método.

Mensaje de Inicio de Viaje:

java

System.out.println(BRIGHT\_GREEN + "Iniciando el viaje hacia " + planetas[selectedPlanetIndex] + "..." + RESET);

Muestra un mensaje indicando que el viaje hacia el planeta seleccionado está comenzando.

Simulación del Progreso del Viaje:

java

for (int i = 0; i <= 100; i += 10) {

try {

Thread.sleep(500); // Pausa de medio segundo para simular progreso

} catch (InterruptedException e) {

e.printStackTrace();

}

if (i < 50) {

System.out.print(GREEN + "=" + RESET);

} else if (i < 80) {

System.out.print(YELLOW + "=" + RESET);

} else {

System.out.print(RED + "=" + RESET);

}

}

System.out.println(BRIGHT\_GREEN + "\n¡Has llegado a " + planetas[selectedPlanetIndex] + "!" + RESET);

Desglose

Bucle for:

java

for (int i = 0; i <= 100; i += 10) {

El bucle se ejecuta desde i = 0 hasta i = 100 en incrementos de 10, simulando el progreso del viaje en pasos del 10%.

Simulación de Tiempo de Viaje:

java

try {

Thread.sleep(500); // Pausa de medio segundo para simular progreso

} catch (InterruptedException e) {

e.printStackTrace();

}

Pausa la ejecución durante medio segundo para simular el tiempo de viaje.

Mostrar Progreso del Viaje:

java

if (i < 50) {

System.out.print(GREEN + "=" + RESET);

} else if (i < 80) {

System.out.print(YELLOW + "=" + RESET);

} else {

System.out.print(RED + "=" + RESET);

}

Muestra una barra de progreso en diferentes colores según el porcentaje completado:

Verde (GREEN) para el 0-50%.

Amarillo (YELLOW) para el 50-80%.

Rojo (RED) para el 80-100%.

Mensaje de Llegada:

java

System.out.println(BRIGHT\_GREEN + "\n¡Has llegado a " + planetas[selectedPlanetIndex] + "!" + RESET);

Espero que esta explicación te haya ayudado a comprender mejor estas partes del código.