



Risk

- Diego Barreiro Pérez (diego.barreiro.perez@rai.usc.es)
- Miguel Bugarín Carreira (miguel.bugarin@rai.usc.es)

```

fish /home/diego/Github/Risk x + v
[Hooker] $> rearmar Venezuela 3 AmeCentra
{
  numeroEjercitosInicialesOrigen: 8,
  numeroEjercitosInicialesDestino: 1,
  numeroEjercitosFinalesOrigen: 5,
  numeroEjercitosFinalesDestino: 4
}

[Hooker] $> asignar carta Infanteria&Perú
{
  código de error: 125,
  descripción: "El identificador no sigue el formato correcto"
}

[Hooker] $> rearmar ANorte 2 Egipto
{
  código de error: 99,
  descripción: "Comando no permitido en este momento"
}

[Hooker] $> atacar Egipto OMedio
{
  código de error: 99,
  descripción: "Comando no permitido en este momento"
}

[Hooker] $> ver mapa

```

Alaska 5	Tiroeste 1	Groenlan 6	Islandia 2	Escandinava 1	Siberia 1	Yakutsk 1	Kamchatka 1
Alberta 5	Ontario 4	Quebec 4		GBretaña 1	EurNorte 1	Rusia 6	Irkutsk 1
USA Oeste 1	USA Este 2			EurOcc 1	EurSur 1	Urales 1	Mongolia 1
	AmeCentra 4					Afgan 1	China 5
	Venezuela 5			ANorte 6	Egipto 5	OMedio 1	India 1
	Perú 3	Brasil 1		Congo 1	AOriental 1		Sisiático 1
	Argentina 1				Sudáfrica 1	Madagascar 1	Indonesia 10
							Guinea 7
							AusOccid 1
							AusOrient 1

NOTA: Para ejecutar el proyecto, se recomienda usar IntelliJ, o utilizar un terminal Linux. Otros entornos, como NetBeans permitirán ejecutarlo (aunque con un mapa más simple), pero otros como CMD y CygWin no soportan ciertos caracteres.

CorreccionRISK

Además, se ha hecho un comparador de salidas para el Risk, disponible en [barreeiro/CorreccionRISK](https://github.com/barreeiro/CorreccionRISK).

Juego

Para compilar el proyecto, se puede **ejecutar el script `build.sh`** que se encuentra en la raíz del proyecto, el cual generará todos bytecodes necesarios, y luego los empaquetará en un JAR disponible dentro de la carpeta `build/`.

Para ejecutarlo, basta con escribir `java -jar build/risk.jar`.

```
./build.sh && java -jar build/risk.jar
```

En cuanto a los archivos de comandos, estos se encuentran en el directorio `res/` , a modo de "resources" disponibles en todo momento.

Estructura del Proyecto

El proyecto se compone de los siguientes paquetes:

- **gal.sdc.usc.risk.comandos** : Clases con los comandos y su respectiva ejecución. Todos ellos implementan la interfaz `IComando` en el mismo paquete, y se agrupan según su estado en comandos genéricos, ya jugando (partida) y de preparación. Los comandos además tienen la anotación `Comando` , en la cual se especifica el estado de la partida en el que se puede ejecutar el comando del enum `Estado` , y su respectiva expresión regular que lo lanzará, especificado en el enum `Comandos` .
Además, en este mismo paquete se encuentra una clase `Ejecutor` . Este ejecutor es el encargado de ejecutar los comandos, tal como su nombre indica. Esta clase es una implementación de `Callable` , permitiendo ejecutar las clases como si fuesen métodos, y devolver un valor (un `Boolean` en este caso, indicando si hubo éxito o no). El `Ejecutor` , al llamar a procesar un comando, creará un nuevo thread y ejecutará el comando ahí. Todos los comandos se ejecutan llamando a su método `ejecutar()` .
- **gal.sdc.usc.risk.excepciones** : Clases con todas las posibles excepciones. Además, se encuentra el enum `Errores` , en el cual se especifican todos los posibles errores y su tipo, con el que se lanzará la respectiva excepción, siempre siendo una extensión de la clase abstracta `ExcepcionRISK` .
- **gal.sdc.usc.risk.jugar** : En este paquete se encuentran una serie de clases vitales para el desarrollo de la partida. Se encuentra la clase de `ComandosDisponibles` , la cual es la encargada de gestionar que comandos se pueden ejecutar. En la lista `lista` se encuentran las clases que se permiten ejecutar, y luego están unos métodos para gestionar que comandos se activan y desactivan. En la clase `Menu` , es donde tiene lugar la lectura de los comandos, y donde luego se dirige al `Ejecutor` para ser procesado. Y en la clase abstracta `Partida` se encuentran todos los datos relacionados con la partida, y los métodos para modificarse. Todos los datos almacenados son estáticos, para así guardar su estado a lo largo de las múltiples invocaciones, garantizando su unicidad.

- **gal.sdc.usc.risk.salida** : Este paquete es el encargado de gestionar las salidas de los resultados de las ejecuciones. La interfaz `Consola` y la clase `ConsolaNormal` son las encargadas de sacar los resultados por consola. A continuación, `Resultado`, contiene una serie de métodos estáticos que son llamados por los comandos, los cuales procesan la información y los imprimen llamando a `Consola`.
Las otras cuatro clases son objetos que son válidos para ser impresos. Están inspirados en el paquete `org.json` de Java con el que se puede estandarizar una salida. Existen `SalidaDupla` para representar una tupla, `SalidaLista` como array de datos, y `SalidaObjeto` como un diccionario de datos con sus claves. `SalidaUtils` es una clase con la que se obtiene la representación como string de estos objetos, en caso de haber más de un nivel hasta encontrar una cadena de texto.
- **gal.sdc.usc.risk.tablero** : Aquí se encuentran todos los objetos de la partida, del "tablero". Está una clase abstracta `Carta` (con sus subtipos en el paquete `gal.sdc.usc.risk.tablero.carta`) para las cartas de equipamiento, una clase `Celda` con los datos de una casilla del mapa, los objetos `Continente` y `Pais`, el cual utiliza la clase `Frontera`. De los jugadores, está la clase base `Jugador`, con sus respectivos datos, y la clase `Mision` indicando cuando un jugador puede obtener la victoria. También está la clase de `Ejercito`, que es usada tanto por `Pais` como por `Jugador` (en el caso de los países, se utiliza un constructor tipo *Builder* para crear un ejército sin color), y sus colores en su paquete `gal.sdc.usc.risk.tablero.ejercito`. Y finalmente, la clase `Mapa` viene siendo el tablero de juego, almacenando los países y continentes, principalmente. En cuanto al paquete `gal.sdc.usc.risk.tablero.valores`, aquí se almacena una serie de enums con constantes del juego, como los países y sus posiciones, continentes, misiones, cartas de equipamiento y fronteras marítimas disponibles.
- **gal.sdc.usc.risk.util** : Paquete con una serie de utilidades, como la gestión de `Colores` para imprimir por consola y del juego, una clase `Dado` para generar números aleatorios del 1 al 6, y una clase de `Recursos` para obtener los archivos de la carpeta `res/`.

Requisitos Parte 2

Jerarquía de cartas

La clase abstracta `Carta` se encuentra en el paquete `gal.sdc.usc.risk.tablero`, teniendo un constructor tipo *Builder* para facilitar su inicialización. Las cartas se pueden crear mediante `new Carta.Builder().withPais(pais).withSubEquipamiento(SubEquipamientos.X).builder()`. Esto creará una instancia de `Carta` con el país y subtipo especificado.

Esta clase es extendida por las subclases `Infanteria`, `Caballeria` y `Artilleria`, las cuales se encuentran en el paquete `gal.sdc.usc.risk.tablero.carta`, siendo también abstractas. Estas clases son extendidas por las respectivas subclases en el paquete `gal.sdc.usc.risk.tablero.carta.X`, siendo X el tipo en cuestión. Estos subtipos tienen constructores que reciben el país de la carta.

Jerarquía de ejércitos

La jerarquía especificada de ejércitos se encuentra en el paquete `gal.sdc.usc.risk.tablero`. Ahí dentro se encuentra la clase abstracta `Ejercito`, la cual es extendida por las otras subclases. Al igual que con la jerarquía de cartas, están en `gal.sdc.usc.risk.tablero.ejercitos`, estando estos en sus respectivos paquetes.

Además, aparte de tener `EjercitoBase` y `EjercitoCompuesto`, está también `EjercitoNuevo`, la cual no es abstracta, y permite crear un ejército sin color (ya que debido a la implementación de ejércitos, el traspaso siempre es mediante otro ejército), haciendo de "basura" de ejércitos o fábrica.

Excepciones

Las excepciones se encuentran en el paquete `gal.sdc.usc.risk.excepciones`. En ella, se encuentra la clase abstracta `ExcepcionRISK`, la cual es la superclase de todas las sub-excepciones. El tipo de excepción es `RuntimeException`, ya que así permite no tener que extender mediante `throws`. Las sub-excepciones todas extienden esta clase, recibiendo como argumento un elemento del enum `Errores`.

Existe el enum `Errores` con los valores constantes de los errores todos, el cual recibe el código de error, el mensaje del error y la sub-excepción que lanza. De esta forma, se consigue más versatilidad al estar definidas como constante todas las posibles.

La gestión de errores tiene lugar en la clase `Ejecutor`, dentro del paquete `gal.sdc.usc.risk.comandos`. Esta clase es una extensión del `Callable<Boolean>`, la cual permite "ejecutar" la clase y devolver un booleano (representando si la ejecución del comando tuvo éxito o no). La gestión de errores tiene lugar en el `public static void comando`, en el cual se intenta obtener el resultado de la ejecución. Los comandos lanzarán las excepciones durante la ejecución, y se gestionarán con la excepción `RuntimeException` (la cual extiende a `Exception`), la cual contendrá una serie de causas, siendo una de ellas una clase extendida de `ExcepcionRISK`.

Cuando esto se alcanza, se manda esta excepción al gestor de errores para escribirlo en el archivo y lanzarlo por consola.

Interfaz *Consola*

La interfaz `Consola` está en el paquete `gal.sdc.usc.risk.jugar`, la cual, además de los requisitos, tiene un método con el que imprimir una línea en blanco. La clase `ConsolaNormal` que extiende `Consola` está en el mismo paquete.

La clase `Partida` (que contiene todos los datos del juego) tiene una instancia estática de `Consola` con la que se puede acceder desde cualquier punto del programa.
