Tarea 1 - Pokémon TCG: Elementos básicos

Profesor: Alexandre Bergel
Auxiliar: Juan-Pablo Silva, Ignacio Slater

El objetivo de esta tarea (y las siguientes (!)) será implementar un clon del juego de cartas Pokémon. Su aplicación debe estar debidamente testeada y documentada, y debe utilizar buenas prácticas de diseño enseñadas en el curso. Los detalles de las funcionalidades del juego que debe implementar serán especificadas en el enunciado de la tarea correspondiente, por lo que las reglas del juego se irán explicando por partes.

1. Requisitos

En este juego los participantes asumen el rol de entrenadores Pokémon y usan sus Pokémon para pelear con sus oponentes. Los entrenadores juegan sus cartas al campo de juego con el objetivo principal de derrotar a los Pokémon de su oponente.

Para esta entrega se considerarán dos tipos de entidades principales, la entidad **Entrenador** y la entidad **Carta**.

1.1. Entrenador

En el juego, un entrenador será lo que representará a un jugador. Un entrenador debe tener **un solo** *Pokémon activo* en el campo de batalla y hasta *5 Pokémon en la banca* (sumando un máximo de 6 Pokémon en juego por entrenador), además tendrá una mano con un número ilimitado de cartas.

Un entrenador debe ser capaz de ver sus cartas, tanto las que tiene en la mano como las que están en el campo, debe poder jugar una carta de su mano y seleccionar alguna de las habilidades de su Pokémon activo para usarla.

1.2. Carta

Las cartas son la unidad fundamental del juego, y entre ellas se darán la mayoría de las interacciones. Todas las cartas tienen un nombre (no necesariamente único) y deben poder ser jugadas por el entrenador que la tenga, de acuerdo a ciertas condiciones.

Para esta tarea, se implementarán **dos tipos de cartas**, los *Pokémon* y las *Energías*.

1.2.1. Energía

Una energía es una carta que sirve como *moneda* para realizar ciertas acciones. Considere que existen 6 tipos de energías: *planta*, *agua*, *fuego*, *lucha*, *psíquica* y *relámpago*.

Al jugar una carta de energía, se le asocia una energía del tipo indicado en la carta a un Pokémon en juego. Para simplificar la implemetación, asuma que al jugar una carta de energía, su efecto se aplica siempre sobre el Pokémon activo del entrenador.

1.2.2. Pokémon

Los Pokémon siempre son jugados a la banca del entrenador, y si el entrenador no tiene ningún Pokémon activo, entonces un Pokémon de la banca pasa a ser el activo (por simplicidad puede asumir que el Pokémon de la banca que pasará a activo será siempre el mismo e.g. el que esté primero en la banca).

Un Pokémon debe tener algún número que lo identifique[1], una cantidad de puntos de salud y debe llevar la cuenta de cuantas energías tiene asociadas junto con el tipo de éstas. Si los puntos de salud de una carta se reducen a 0, ésta no puede seguir peleando y debe ser reemplazada por otra carta de la banca.

Además, un Pokémon puede tener hasta 4 habilidades, y un entrenador debe ser capaz de usar cualquiera de las habilidades de su Pokémon activo. Por ahora, se considerará que todas las habilidades son ataques. Un *ataque* tiene un nombre, un daño base, un texto descriptivo y un coste que indica cuántas energías de cada tipo debe tener asociadas el Pokémon para poder usarlo. Cuando un Pokémon ataca a otro, se reducen los puntos de vida del Pokémon atacado en la cantidad de daño base del ataque, a menos que éste presente alguna debilidad o resistencia. Si un Pokémon es resistente a otro, entonces el daño recibido por un ataque se reduce en 30 puntos, mientras que si es débil a otro el daño recibido es duplicado.

Cada Pokémon tiene un tipo, y de acuerdo a éste se definen las debilidades y resistencias, los tipos y sus relaciones se muestran en el cuadro 1.

Tipo	Debilidad	Resistencia
Planta	Fuego	Agua
Fuego	Agua	
Agua	Planta, Rayo	Lucha
Rayo	Lucha	
Lucha	Psíquico, Planta	
Psíquico	Psíquico	Lucha

Cuadro 1: Resistencias y debilidades según tipo

Un Pokémon puede ser jugado directamente al campo de batalla siempre y cuando la banca del entrenador no esté llena.

[1] Considere que pueden existir distintas versiones de un mismo Pokémon, por lo que es posible que hayan múltiples cartas con el mismo id. Puede usar el número de la Pokédex como identificador.

2. Requerimientos adicionales

- **Cobertura**: Cree los tests unitarios, usando JUnit 4, que sean necesarios para tener al menos un coverage del 90 % de las líneas por paquete. Todos los tests de su proyecto deben estar en el paquete `test`.
- **Javadoc**: Cada interfaz, clase pública y método público debe estar debidamente documentado siguiendo las convenciones de Javadoc[2]. En particular necesita `@author` y una pequeña descripción para su clase e interfaz, y `@param`, `@return` (si aplica) y una descripción para los métodos.
- **Resumen**: Debe entregar un archivo **pdf**[3]que contenga su nombre, rut, usuario de Github, un link al repositorio de su tarea y un diagrama UML que muestre las clases, interfaces y métodos que usted definió en su proyecto. **No debe incluir los tests** en su diagrama.
- **Git**: Debe hacer uso de git para el versionamiento de su proyecto. Luego, esta historia de versionamiento debe ser subida a Github.
- **Readme**: En la raíz de su proyecto debe crear un archivo *markdown* llamado `README.md`. Deberá dar una descripción de lo que implementó, cómo lo hizo, a grandes rasgos qué patrones de diseño usó, cómo correr su programa, y otras anotaciones que usted encuentre pertinentes. Aquí puede encontrar un *cheatsheet*[4] y un template[5]. El template incluye mucha más información que la solicitada, pero así es como debe presentarse un proyectoprofesionalmente.

3. Evaluación

- **Código fuente (4.0 puntos)**: Este ítem comprende dos categorías.
  - **Funcionalidad**: Se analizará que su código provea la funcionalidad pedida.
  - **Diseño**: Se analizará que su código provea la funcionalidad implementada utilizando un buen diseño.
- **Coverage (1.0 puntos)**: Sus casos de prueba deben crear diversos escenarios y contar con un coverage de las líneas de al menos 90 % por paquete. No está de más decir que sus tests deben testear algo (es decir, no ser tests vacios o sin asserts).
- **Javadoc (0.5 puntos)**: Cada clase, interfaz y método público debe ser debidamente documentado. Se descontará por cada falta.

[2] `http://www.oracle.com/technetwork/articles/java/index-137868.html`
[3] Realizar la entrega en otro formato es motivo de descuento.
[4] `https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet`
[5] `https://gist.github.com/PurpleBooth/109311bb0361f32d87a2`

- **Resumen y readme (0.5 puntos)**: El resumen y readme mencionados en la sección anterior. Se evaluará que haya incluido el diagrama UML de su proyecto. **En caso de no enviarse el resumen con el link a su repositorio su tarea no será corregida**[6].

4. Entrega

Suba el código de su tarea a *Github*[7]. Su repositorio de *Github* debe ser **privado**, y debe llamarse **cc3002-tarea1**, lo cual significa que si su usuario de *Github* es *juanpablos*, el link a su repositorio será `https://github.com/juanpablos/cc3002-tarea1`. Además, deberá invitar a la cuenta del equipo docente, *CC3002EquipoDocente*, como colaboradores de su repositorio para que podamos acceder a él. Para hacer esto entre a su repositorio, luego a *Settings*, *Collaborators*, ingrese su contraseña y finalmente escriba *CC3002EquipoDocente* en el campo para buscar por usuario, seleccione *Add collaborator* y nos habrán invitado exitosamente. Las invitaciones se aceptan al momento de corregir su tarea, por lo que no se asuste si aparece como solicitud pendiente.

Para la entrega de las tareas del curso usted dispondrá de 72 horas de retraso repartidas entre todas las tareas para usar como estime conveniente. Más detalles al respecto serán publicados en el foro.

Usted debe subir a U-Cursos **solamente el resumen**, con su nombre, rut, usuario de *Github*, link a su repositorio y diagrama UML. El código de su tarea será bajado de *Github* directamente. El plazo de entrega es hasta el **27 de abril** a las 23:59 hrs. Se verificará que efectivamente el último *commit* se haya hecho antes de la fecha límite. **No se aceptarán peticiones de extensión de plazo**.

5. Recomendaciones

No estamos interesados en un programa que solamente funcione. Este curso contempla el diseño de su solución y la aplicación de buenas prácticas de programación que ha aprendido en el curso. No se conforme con el primer diseño que se le venga a la mente, intente ver si puede mejorarlo y hacerlo más extensible.

Las tareas serán incrementales, por lo que tendrá que agregar funcionalidades a su programa en las siguientes entregas; tenga esto en cuenta al momento de escribir su código.

No comience su tarea a último momento[8]. Esto es lo que se dice en todos los cursos, pero es particularmente importante/cierto en éste. Si usted hace la tarea a último minuto lo más seguro es que no tenga tiempo para reflexionar sobre su diseño, y termine entregando un diseño deficiente o sin usar lo enseñado en el curso.

[6] Porque no tenemos su código.
[7] `https://github.com/`
[8] Por favor[9]
[9] De verdad

Tenga en cuenta que típicamente la documentación de los proyectos de software suele estar escrita en inglés, no es requisito para la entrega, pero puede aprovechar esta oportunidad para practicar y acostumbrarse a documentar en ese idioma.

Les pedimos encarecidamente que las consultas referentes a la tarea las hagan por el **foro de U-Cursos**, en la categoría "Consultas Tarea: Tarea 1". En caso de no obtener respuesta en un tiempo razonable, pueden hacernos llegar un correo al auxiliar o ayudantes.

¡Exito!