Juegos de naipes
Casos de estudio para POO

TSSI Edwin Alexis Abot
Programación III y Laboratorio de Computación III

9 de abril de 2017

1. Introducción

Tomamos como caso de estudio algunos juegos de naipes que se pueden jugar con las barajas española o inglesa. Siguiendo el paradigma de la programación orientada a objetos vamos a plantear modelos para estos juegos. Además vamos a abordar el modelado de forma evolutiva siguiendo el desarrollo del temario de Programación III.

¿Qué significa que *vamos a modelar de forma evolutiva*?. Significa que nuestros modelos van a evolucionar en función a los nuevos conocimientos adquiridos de herencia, polimorfismo, principios SOLID y patrones GRASP. Esto quiere decir que en un principio vamos a crear modelos simples y subóptimos que deberán ser mejorados. Nuestros modelos incorporarán progresivamente más abstracciones y buenas prácticas que darán más flexilidad, mantenibilidad y correctitud a nuestro desarrollo.

Diseñar e implementar diferentes juegos de naipes será entonces una excusa para observar cómo la POO nos da herramientas para crear software reutilizable, mantenible, flexible y extensible. Que por otra parte respeta buenas prácticas y es performante.

1.1. La baraja española

Según Wikipedia:

> La baraja española consiste en un mazo de 48 'naipes' o 'cartas' que se barajan ..., aunque la versión más extendida solo tiene 40 cartas al eliminar los ochos y nueves.
>
> Tradicionalmente se divide en cuatro familias, también llamadas palos, que son: oros, copas, espadas y bastos... Cada palo tiene cartas numeradas del 1 al 9 (cartas numéricas)... y del 10 al 12 (figuras). Las figuras corresponden a los números 10 'sota', 11 'caballo' y 12 'rey', respectivamente...
>
> Actualmente ciertos mazos incluyen también 2 comodines. Por ello las barajas pueden ser de 40, 48 o de 50 naipes dependiendo del juego.

1.2. La baraja inglesa

Según Wikipedia:

> La baraja inglesa es un conjunto de naipes o cartas, formado por 52 unidades repartidas en cuatro palos. A menudo se incluyen en esta baraja dos cartas comodín, subiendo el número a 54 cartas.
>
> La baraja inglesa tiene su origen en la baraja francesa, pero está más difundida. Junto con la baraja española, es uno de los conjuntos de naipes más conocidos del mundo y se utiliza en juegos como el bridge, la canasta y el póquer.
>
> Cada palo está formado por 13 cartas, de las cuales 9 cartas son numerales y 4 literales. Se ordenan de menor a mayor rango de la siguiente forma: A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q y K. Las cartas con letras, las figuras, se llaman jack, queen, king y ace.

1.3. Objetivos

1. Construir diagramas de clases.
2. Incorporar nuevos conceptos de POO.
3. Adquirir nuevas buenas prácticas de desarrollo.
4. Refactorizar nuestros diseños aplicando los nuevos conceptos.
5. Crear algoritmos performantes

2. La carta más alta gana

Vamos a implementar nuestro primer juego: **La carta más alta gana**.

Las reglas Nuestras reglas son simples. Se tienen de dos a seis jugadores, se le reparte a cada jugador una carta (de la baraja española) cara arriba, es decir que todos los jugadores ven la carta de los demás jugadores. El jugador que tiene la carta más valiosa gana la mano. El valor de las cartas es incremental y va del 1 al 12. Los palos también tienen un valor incremental y son copas, bastos, oros y espadas. Entonces el 1 de copa es la menos valiosa y el 12 de espadas es la más valiosa. El jugador que tenga la carta más valiosa de la mano gana un punto. El partido termina cuando un jugador suma diez puntos y se lo declara ganador. Los comodines no se usan y las cartas que se reparten en cada mano no se devuelven al mazo. Una vez que no quedan más cartas para repartir se las vuelve a mezclar todas y se continúa con el juego.

Los requerimientos Queremos crear un programa orientado a objetos que simule un partido de este juego. El programa debe pedir el número de jugadores. Cuando se sepa la cantidad de jugadores el programa puede comenzar la simulación. El simulador debe simular el partido una mano por vez pidiendo la autorización del usuario para continuar entre manos. También debe permitir simular el partido completo sin interacción del usuario. El barajado de cartas debe ser automático, esto quiere decir que se deben mezclar las cartas y repartir

de forma automatizada. El simulador debe dar salida por consola de las cartas barajadas a cada jugador y el resultado de cada mano.

2.1. Ejercicios

Se debe resolver los siguientes ejercicios en el orden propuesto. En todos los puntos se debe realizar una puesta en común.

Ejercicio 1. Analizar las reglas del juego y comprender los requerimientos.

Ejercicio 2. Identificar objetos y sus responsabilidades. Especificar los atributos y comportamientos.

Ejercicio 3. Crear un diagrama de clases, con UML, que represente los objetos del programa y sus relaciones.

Ejercicio 4. Crear pseudocódigo que describa la lógica para cada métodos de los objetos.

Ejercicio 5. Crear una aplicación Java con NetBeans y crear todas las clases como se indican en el diagrama de clases. Implementar todos los métodos como se indican en el pseudocódigo.

2.2. Proposiciones

Se debe analizar las siguientes proposiciones y determinar su valor de verdad. En todos los casos debe justificar porqué la proposición es verdadera o falsa.

Se debe buscar evidencia concreta en la resolución de los ejercicios anteriores y/o en la bibliografía.

Proposición A. Los constructores se utilizan para inicializar los atributos de un objeto y no deben resolver reglas de negocio.

Proposición B. Los métodos de clase pueden hacer uso de atributos de instancia.

Proposición C. Los métodos de instancia pueden ser utilizados aunque no se haya instanciado la clase.w

Proposición D. Los miembros públicos son accesibles por todo el mundo. Los miembros privados de una clase pueden ser accedidos solo por los métodos de esa clase.

Proposición E. Los métodos de acceso, también conocidos como getters y setters, nunca deben incluír validaciones para aquellos valores que se asignarán los atributos.

3. La carta más alta II

Luego de haber implementado nuestra primer versión de "La carta más alta" y como se mencionó en la introducción de esta guía vamos a hacer evolucionar nuestro software.

Ya que nos sentimos un poco más cómodos con POO vamos a mejorar nuestra implementación. Vamos a aprovechar los primeros conceptos de Herencia y Polimorfismo.

Debemos comenzar a preparar el terreno para poder implementar nuevos juegos. Entonces ahora clases que teníamos como concretas serán abstractas. Las nuevas abstracciones nos permitirán reutilizar mucho de lo ya implementado. Un ejemplo claro de esto es el método "mezclar()".

Con todo esto comenzaremos a hablar de un partido de "La carta más alta", de un partido de Poker, de un partido de 7 y 1/2, etc. De barajas españolas o inglesas y ¿porqué no de barajas de Uno u otros juegos?.

Los nuevos requerimientos Necesitamos hacer que nuestras clases Carta y Baraja sean reutilizables y que representen un ente abstracto. Esto nos va a permitir modelar juegos de cartas con otros tipos de barajas. Queremos que nuestro programa considere a cada palo como una entidad separada.

Necesitamos mejorar la asignación de responsabilidades. Normalmente cuando nos juntamos a jugar cartas, en cada vuelta de cualquier partido, un jugador diferente se hace cargo de repartir las cartas. Entonces debemos buscar un mecanismo para asignar esa responsabilidad a un jugador e implementar esta rotación.

3.1. Ejercicios

Se debe resolver los siguientes ejercicios en el orden propuesto. En todos los puntos se debe realizar una puesta en común.

Ejercicio 1. Analizar y comprender los requerimientos.

Ejercicio 2. Identificar objetos y sus responsabilidades. Identificar los entes abstractos y concretos. Especificar los atributos y comportamientos.

Ejercicio 3. Crear un diagrama de clases, con UML, que represente los objetos del programa y sus relaciones.

Ejercicio 4. Crear pseudocódigo que describa la lógica para cada nuevo método.

Ejercicio 5. Refactorizar la aplicación Java creada anteriormente con NetBeans y crear todas las clases como se indican en el diagrama de clases. Implementar todos los métodos como se indican en el pseudocódigo.

3.2. Proposiciones

Se debe analizar las siguientes proposiciones y determinar su valor de verdad. En todos los casos debe justificar porqué la proposición es verdadera o falsa.

Se debe buscar evidencia concreta en la resolución de los ejercicios anteriores y/o en la bibliografía.

Proposición A. El orden de ejecución de los constructores en una cadena de herencia es de las subclases a las superclases.

Proposición B. El órden de resolución de métodos, durante la instanciación de una clase, en una cadena de herencia indica que: la implementación más cercana en la jerarquía de clases es la que estará asociada a la instancia.

Proposición C. La sobrecarga de métodos, consecuencia del polimorfismo, establece que es posible crear métodos con distintos conjuntos de parámetros de entrada pero iguales tipos de devolución.

Proposición D. Los objetos heredan de sus padres su interfaz. En el caso de Java toda clase B que herede de una clase A heredará todos los métodos públicos o protegidos.

Proposición E. La sobreescritura de métodos puede ser total o parcial.

Proposición F. El principio de sustitución de Liskov, aplicado a la POO, nos indica que: una variable de tipo A puede contener una referencia a un objeto de tipo B si, y solo si, B respeta el contrato del tipo de dato A.