Universidad de Buenos Aires

Facultad de Ingeniería

75.10 Técnicas de Diseño

Trabajo práctico Nº 1

JGobstones

1er. Cuatrimestre de 2015

Alumnos:

Argüello, Osiris	***83062***
Alonso, Juan Manuel	***93419***
Carreras, Matías Nicolás	***93768***

Enunciado

Se deberá implementar una versión simplificada del lenguaje de programación Gobstones,
que permita escribir un programa y ejecutarlo, viendo la salida de dicha ejecución.

Objetivos

- Implementar una versión inicial de dicho lenguaje de programación.
- Aplicar las técnicas vistas en la teoría y en la práctica.

Herramientas a utilizar

- Java >= 1.8
- Maven >= 3
- JUnit >= 4.11
- Git

Restricciones

- Trabajo Práctico grupal implementado en java.
- Se deben utilizar las mismas herramientas que en el TP0 (git + maven + junit 4).
- Todas las clases del sistema deben estar justificadas.
- Todas las clases deben llevar un comentario con las responsabilidades de la misma.
- El uso de herencia debe estar justificado. Se debe explicar claramente el porqué de
  su conveniencia por sobre otras opciones.
- Se debe tener una cobertura completa del código por tests.

Responsabilidades de las clases

- Command: Representa un comando (simple o complejo) de la gramática de Gobstones.
- GobstonesEvaluator: Convertir el árbol generado por el parser en objetos de nuestro modelo.
- MoveCommand: Representa al comando Mover de Gobstones. Ejecuta la acción de Mover el cabezal en la dirección indicada en el argumento, en el tablero, si es posible.
- PlaceCommand: Representa al comando Poner de Gobstones. Ejecuta la acción de Poner una bolita del color indicado en el argumento, en el tablero, en la posición actual del cabezal.

- RemoveCommand: Representa al comando Sacar de Gobstones. Ejecuta la acción de Sacar una bolita del color indicado en el argumento, en el tablero, en la posición actual del cabezal, si es posible.
- RepeatCommand: Representa al comando repeat de Gobstones. Ejecuta un conjunto de comandos la cantidad de veces que indica el argumento.
- IllegalArgumentsNumberException: Representa una excepción que ocurre al pasarle a un Command una cantidad de Argument equivocada.
- NoSuchColorException: Representa una excepción que ocurre cuando se intenta sacar una bolita de un color que no existe en esa posición del tablero.
- OutOfBoundsException: Representa a una excepción que ocurre cuando se intenta mover el cabezal a una posición del tablero inexistente.
- ProgramTextEvaluationException: Representa una excepción que ocurre cuando el GobstonesEvaluator no puede parsear correctamente el texto del programa.
- ConsolePrinter: Implementación de la interfaz Printer que permite imprimir el tablero a través de la consola.
- FileReader: Implementación de la interfaz Reader que se encarga de leer un programa de Gobstones desde un archivo y convertirlo en un String para el GobstonesEvaluator.
- GobstonesGrammar: Parser de gramática del lenguaje Gobstones. Se encarga de convertir un string en un árbol parseado con los diferentes componentes de la gramática del lenguaje.
- ParserUtils: Clase utilitaria con métodos auxiliares para el GobstonesGrammar.
- Argument: Representa un argumento de un comando de Gobstones.
- Ball: Representa una pelota del modelo de Gobstones con su respectivo color.
- Board: Representa el tablero de Gobstones. Contiene la lista de las diferentes celdas del tablero, así como la posición del cabezal en el tablero.
- Cell: Representa una celda dentro de un tablero de Gobstones. Contiene las bolitas del tablero.
- Color: Representa el color de una bolita particular del tablero. Contiene los tipos de colores existentes en Gobstones.
- CommandBlock: Representa un bloque de código Gobstones. Al ejecutarlo se ejecutan todos los comandos que contiene.
- Expression: Representa una expresión de un comando complejo que indica la cantidad de veces que se ejecuta el comando.
- Gobstones: Invocar al Reader para obtener el texto que utilizará el GobstonesEvaluator para parsear el programa y una vez parseado invocar al Printer para imprimir el tablero.
- Orientation: Representa una dirección en la que se puede mover el cabezal.
- Position: Representa una posición particular del tablero.

- Program: Representa un programa de Gobstones que ejecuta comandos sobre un tablero.
- ReservedWords: Clase auxiliar con las palabras reservadas que se utilizan en Gobstones.
- ColorCreator: Convierte una palabra reservada de Gobstones a un Color de nuestro modelo.
- CommandCreator: Convierte una palabra reservada de Gobstones a un Command de nuestro modelo.
- OrientationCreator: Convierte una palabra reservada de Gobstones a un Orientation de nuestro modelo.
- PlaceCommandFactory: Implementación del patrón Factory para obtener un objeto concreto del tipo Command.
- RemoveCommandFactory: Implementación del patrón Factory para obtener un objeto concreto del tipo Command.
- RepeatCommandFactory: Implementación del patrón Factory para obtener un objeto concreto del tipo Command.
- MoveCommandFactory: Implementación del patrón Factory para obtener un objeto concreto del tipo Command.

Cumplimiento de principios SOLID

Respecto a los principios SOLID, se pueden ver reflejados en nuestro trabajo práctico los siguientes principios:

Single-Responsability: todas las clases de nuestro tp tienen una sola responsabilidad excepto la clase Gobstones que es nuestro main y esta encargada de llamar a los diferentes módulos del sistema.

Open-Close: Un ejemplo de este principio es la clase *Command* y sus clases herederas. La clase *Command* tiene el método execute que permite ejecutar un comando y cada clase heredera implementa su comportamiento.
Así, para ejecutar todos los comandos de un programa solo debemos recorrer la lista de comandos y ejecutar el método execute de cada uno. Al agregar un nuevo comando no deberemos hacer ningún cambio en el código, simplemente crear la nueva clase heredera de *Command* e implementar su execute.

Dependency Inversion: Esto podemos verlo en las interfases *Reader* y *Printer* utilizadas para leer un programa Gobstones y para mostrar el estado final del tablero, respectivamente. En este caso usamos dos interfases con los métodos *stringFromFile* y *print* que deben ser implementados por las diferentes clases *Reader y *Printer y que pueden ser intercambiados en cualquier momento. Así

invertimos la dependencia ya que la clase *Gobstones* utiliza las interfaces y no se preocupa por las diferentes implementaciones.

Decisiones de diseño

Al empezar a pensar el diseño de nuestro interpreter dividimos el problema en obstáculos más pequeños.
Las tareas que logramos identificar fueron:

- ***Lectura de datos de entrada:*** Consiste en la obtención del código de Gobstones. En el caso de esta primera entrega se obtiene de un archivo.
- ***Parseo del lenguaje Gobstones:*** Este problema fue resuelto por la cátedra utilizando java-petit-parser.
- ***Interpretación del lenguaje:*** Interpretar los datos que recibimos procesados por el parser y transformarlos en entidades de nuestro modelo, que pudiéramos utilizar.
- ***Modelización del lenguaje Gobstones:***
    - *Modelización de los comandos utilizados en Gobstones:* Consta de la creación de un modelo en Java que nos permita utilizar los mismos comandos que se utilizan en Gobstones. Para esta entrega sólo se consideraron los comandos Mover, Poner, Sacar y repeat.
    - *Modelización del modelo de Gobstones:* Modelar el tablero utilizado en Gobstones con sus respectivas interacciones.
- ***Impresión del código ejecutado:*** Representar el estado del modelo una vez finalizada la ejecucion del codigo que fue recibido como dato.

Primero resolvimos el modelo de Gobstones. Como la estructura de clases del modelo original es sencilla y clara, intentamos ser fieles a las clases y métodos utilizados en el modelo original, es decir, si Tablero implementa Mover(Direccion) y Poner(Color), nosotros implementamos los mismos métodos con los mismos parámetros en Java. Gracias a esta similitud entre el modelo de Gobstones y el nuestro, no nos representó un problema complejo implementar la lógica necesaria y suponemos que la implementación de comandos de Gob. que no son requeridos para la primera entrega tampoco lo será.

Los comandos utilizados en Gobstones los podemos separar en dos tipos, los que afectan al tablero directamente (Mover, Poner, etc) y los de control de flujo (for, while, etc). La lógica de los primeros ya se encuentra implementada en el modelo, por lo que es delegada a los componentes del mismo.

Los comandos conocen los argumentos que deben recibir, por lo que es responsabilidad de cada comando chequear que los argumentos recibidos sean del tipo y la cantidad adecuada.

Para el parseo utilizamos una clase GobstoneEvaluator que hereda de GobstonesGrammar y que sobreescribe los métodos que colocan en el árbol de parseo los comandos simples y complejos, los colores y las direcciones. Así al finalizar el parseo, en los nodos en los que debería haber texto para comandos, colores o direcciones ahora hay objetos. Luego se recorre el árbol de parseo y se extraen los comandos con sus argumentos respectivos, ignorando lo que no pertenezca a estas categorías. Esta lista de comandos extraída está en orden de ejecución y se coloca como atributo de una clase Program que luego podrá ejecutarlos secuencialmente.

Tanto en el caso de la entrada de datos como de la salida utilizamos interfaces, porque entendemos que pueden cambiar en futuras entregas.

El archivo de entrada se lee completo y se le indica al GobstonesEvaluator que, mediante uso del parser, lo transforme en objetos de nuestro modelo.

Para finalizar el printer recibe el tablero que debe imprimir.

Uso de herencia

En nuestro trabajo hemos utilizado herencia en solo dos casos: para los diferentes Command y para el GobstonesEvaluator.
En el primer caso utilizamos herencia para poder ejecutar la lista de comandos del programa en forma polimórfica, sin preocuparnos de qué tipo de Command estamos ejecutando. Además, como todos los comandos poseen el atributo “arguments” y no es un atributo constante no nos era posible utilizar una interfaz en este caso.
Respecto al GobstonesEvaluator, utilizamos herencia para modificar el comportamiento del GobstonesGrammar, cambiando texto por objetos al momento de parsear, pero aun asi conservar la estructura del arbol de parseo y saber en que secuencia se ejecutan los comandos.

Patrones de diseño utilizados

El único patrón utilizado en nuestro tp es el patrón Factory Method para generar los diferentes Color, Orientation y Command, según lo parseado por el GobstonesEvaluator.

Utilización del programa

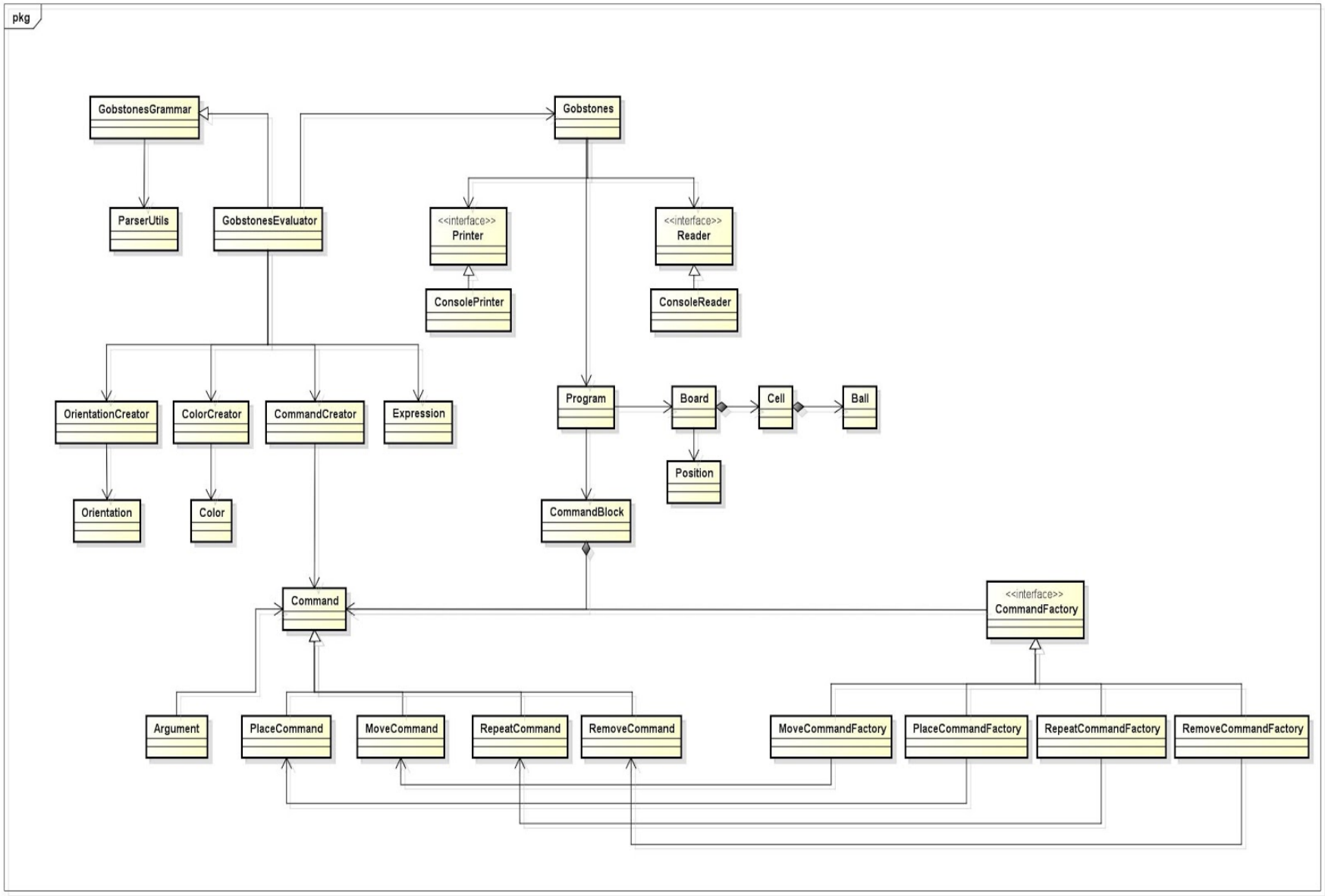
1) Realizar la instalación con Maven

   mvn clean install

2) Ejecutar el programa

   java -jar jgobstones-1.0-SNAPSHOT-shaded.jar -f path/to/file

Diagrama de clases


pkg
GobstonesGrammar
Gobstones
ParserUtils
GobstonesEvaluator
<<interface>>
Printer
<<interface>>
Reader
ConsolePrinter
ConsoleReader
OrientationCreator
ColorCreator
CommandCreator
Expression
Program
Board
Cell
Ball
Position
Orientation
Color
CommandBlock
Command
<<interface>>
CommandFactory
Argument
PlaceCommand
MoveCommand
RepeatCommand
RemoveCommand
MoveCommandFactory
PlaceCommandFactory
RepeatCommandFactory
RemoveCommandFactory
powered by Astah