Auxiliar Revolucionaria 1

Patrones de diseño 2: Ejercicios resueltos y propuestos

Profesor: Alexandre Bergel
Auxiliares: Juan-Pablo Silva
Ignacio Slater
Semestre: Primavera 2019

1. Definiciones necesarias

Sea el juego de cartas *Yu-Gi-Oh!* definido como la tupla $Y = \ldots$
Sería interesante (?) dar una definición matemática para el juego, pero no lo voy a hacer.

Yu-Gi-Oh! es un *TCG* con varios tipos de cartas, pero para los siguientes ejercicios nos centraremos en 2 tipos en particular, *cartas de monstruos* y *cartas de hechizos*. La definición completa de las reglas del juego se pueden encontrar en https://img.yugioh-card.com/uk/rulebook/Rulebook_v9_en.pdf

2. Template method

Ejercicio 1 Zonas de juego

Si bien existen muchas diferencias entre estos dos tipos de cartas, por ahora consideraremos que solo se diferencian en la zona del campo en la que son jugadas.

Defina las cartas de monstruos y de hechizos para que puedan jugarse en su zona correspondiente siguiendo buenas metodologías de diseño.

Ejercicio 2 Niveles de monstruos

Además de los tipos de cartas del ejercicio 1 se tienen distintos niveles de monstruos que requerirán sacrificios para ser jugados. Los niveles son:

- **Niveles 1 a 4:** Se pueden jugar sin necesidad de hacer sacrificios.
- **Niveles 5 y 6:** Necesitan de un sacrificio para ser jugados.
- **Niveles 7 y superiores:** Necesitan de 2 sacrificios para jugarse.

Modifique el código para permitir esta mecánica.

Solución del ejercicio 1

Lo primero que necesitamos hacer es crear una interfaz y una clase abstracta que represente todos los tipos de cartas, así:

```java
// com.github.islaterm.yugi.card.ICard
public interface ICard {
  void playToMat(GameMat gameMat);
}

// com.github.islaterm.yugi.card.AbstractCard
public abstract class AbstractCard implements ICard {
  @Override
  public abstract void playToMat(GameMat gameMat);
}
```

Lo que estamos haciendo al definir el método como abstracto es relegar la labor de decidir cómo se jugarán las cartas a las implementaciones particulares de cada tipo de carta. Esto es lo que se conoce como *template method pattern*.

Ahora, las implementaciones particulares de cada método serán:

```java
// com.github.islaterm.yugi.card.monster.BasicMonsterCard
public class MonsterCard extends AbstractCard {

  @Override
  public void playToMat(@NotNull GameMat gameMat) {
    gameMat.addMonster(this);
  }
}

// com.github.islaterm.yugi.card.SpellCard
public class SpellCard extends AbstractCard {
  @Override
  public void playToMat(@NotNull GameMat gameMat) {
    gameMat.addSpell(this);
  }
}
```

Solución del ejercicio 2

Este problema también puede ser resuelto utilizando un *template*. Para esto necesitaremos extender el modelo de cartas de monstruos que llevamos hasta ahora.

```java
// com.github.islaterm.yugi.card.monster.IMonsterCard
public interface IMonsterCard extends ICard {
  void addTribute(IMonsterCard card);
}
// com.github.islaterm.yugi.card.monster.AbstractMonsterCard
public abstract class AbstractMonsterCard extends AbstractCard implements IMonsterCard {
  private Set<IMonsterCard> tributes = new HashSet<>();

  @Override
  public void addTribute(IMonsterCard card) {
    this.tributes.add(card);
  }
```

```
  @Override
  public void playToMat(GameMat gameMat) {
    if (enoughSacrifices()) {
      gameMat.addMonster(this);
      tributes.forEach(gameMat::removeMonster);
    }
  }

  protected abstract boolean enoughSacrifices();
}
```

Ahora, con esas clases definidas se pueden implementar los tres tipos de cartas de monstruos que necesitamos.

```
// com.github.islaterm.yugi.card.monster.BasicMonsterCard
public class BasicMonsterCard extends AbstractMonsterCard {
  @Override
  protected boolean enoughSacrifices() {
    return tributes.isEmpty();
  }
}
// com.github.islaterm.yugi.card.monster.OneTributeMonsterCard
public class OneTributeMonsterCard extends AbstractMonsterCard {
  @Override
  protected boolean enoughSacrifices() {
    return tributes.size() == 1;
  }
}
// com.github.islaterm.yugi.card.monster.TwoTributesMonsterCard
public class TwoTributesMonsterCard extends AbstractMonsterCard {
  @Override
  protected boolean enoughSacrifices() {
    return tributes.size() == 2;
  }
}
```