Implementación de Polimorfismo en Java

En este documento, se detalla la aplicación del **polimorfismo** en Java mediante la creación de un conjunto de clases y una interfaz destinados a representar personajes de un juego y sus respectivas acciones. El polimorfismo permite el tratamiento uniforme de objetos pertenecientes a diferentes tipos, proporcionando así flexibilidad y la capacidad de reutilizar el código.

La elección de implementar la interfaz `CharacterActions` y la clase abstracta `Character` se basa en la necesidad de modelar acciones comunes que todos los personajes de un videojuego pueden realizar, tales como saltar, moverse, protegerse, atacar y ejecutar habilidades especiales. Si bien algunos videojuegos presentan acciones genéricas compartidas por todos los personajes, también es común que algunas acciones tengan implementaciones específicas y distintivas para cada personaje. Por ejemplo, las habilidades especiales, como los ataques únicos de cada personaje, a menudo involucran animaciones y lógica de juego personalizadas.

En este contexto, la interfaz `CharacterActions` provee una abstracción que garantiza la presencia de estas acciones comunes, mientras que la clase abstracta `Character` ofrece una implementación base para evitar redundancias de código. Además, presenta un método abstracto que sirve como plantilla para la implementación específica de las acciones distintivas de cada personaje. De esta manera, se logra un equilibrio entre la uniformidad de las acciones compartidas y la flexibilidad para adaptarse a las particularidades de cada personaje.

CharacterActions

La interfaz `CharacterActions` define un conjunto de métodos que representan acciones comunes que un personaje de un juego puede llevar a cabo. Estas acciones incluyen saltar, moverse, protegerse, atacar y habilidad especial.

```
package com.game.characters;

/**
 * Interfaz CharacterActions que define un conjunto de métodos representando acciones
 * comunes que un personaje de un juego puede llevar a cabo.
 * Esta interfaz proporciona una abstracción para las acciones básicas de un personaje en un juego,
 * incluyendo saltar, moverse, protegerse, atacar y habilidades especiales.
 */
public interface CharacterActions {
    void jump();
    void move();
    void guard();
```

```
    void attack();
    void specialAbility();
}
```

Character

La clase abstracta `Character` representa un personaje en el juego e implementa la interfaz `CharacterActions`. Proporciona una implementación base para las acciones comunes de los personajes, como saltar, moverse, protegerse y atacar. Además, la habilidad especial está modelada como un método abstracto, reconociendo de esta manera la naturaleza única y personalizada de dicha habilidad para cada personaje del juego.

```
package com.game.characters;

/**
 * Clase abstracta que representa un personaje en un juego e implementa la interfaz CharacterActions.
 * La clase proporciona una implementación base para las acciones de un personaje,
 * como saltar, moverse, protegerse, atacar y habilidad especial.
 */
public abstract class Character implements CharacterActions {
    // Atributo para almacenar el nombre del personaje.
    String nombre;

    /**
     * Constructor de la clase Character.
     * Inicializa el nombre del personaje con el nombre de la clase.
     */
    public Character() {
        // Inicializa el nombre del personaje con el nombre de la clase.
        this.nombre = getClass().getSimpleName();
    }

    /**
     * Obtiene el nombre del personaje.
     * @return Cadena que contiene el nombre del personaje.
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * Implementación del método jump() de la interfaz CharacterActions.
     * Imprime un mensaje indicando que el personaje ha saltado.
     */
    @Override
```

```
    public void jump() {
        System.out.println(nombre + " salta");
    }

    /**
     * Implementación del método mover() de la interfaz CharacterActions.
     * Imprime un mensaje indicando que el personaje se ha movido.
     */
    @Override
    public void move() {
        System.out.println(nombre + " se mueve");
    }

    /**
     * Implementación del método guard() de la interfaz CharacterActions.
     * Imprime un mensaje indicando que el personaje se ha protegido.
     */
    @Override
    public void guard() {
        System.out.println(nombre + " se protege");
    }

    /**
     * Implementación del método attack() de la interfaz CharacterActions.
     * Imprime un mensaje indicando que el personaje ha atacado.
     */
    @Override
    public  void attack() {
        System.out.println(nombre + " ataca");
    }

    /**
     * Método abstracto specialAbility()
     * Debe ser implementado por las subclases para representar la habilidad
     * especial del personaje.
     */
    @Override
    public abstract void specialAbility();
}
```

Kirby, MetaKnight y KingDedede

Las clases `Kirby`, `MetaKnight` y `KingDedede` representan a personajes específicos en el juego y heredan de la clase base `Character`. Estas clases ofrecen una implementación específica para la habilidad especial de cada personaje.

```java
package com.game.characters;

/**
 * Clase que representa al personaje "Kirby" en el juego, hereda de la clase base Character.
 * La clase Kirby proporciona una implementación específica para la habilidad especial
 * del personaje, que consiste en movimientos individuales del personaje.
 */
public class Kirby extends Character {

    /**
     * Implementación del método abstracto specialAbility() de la clase base Character.
     * Realiza la habilidad especial única de Kirby: caída veloz.
     */
    @Override
    public void specialAbility() {
        // Imprime un mensaje indicando que la clase Kirby realiza su habilidad especial.
        System.out.println(getNombre() + " realiza caída veloz!");
    }
}
```

```java
package com.game.characters;

/**
 * Clase que representa al personaje "Meta Knight" en el juego, hereda de la clase base Character.
 * La clase MetaKnight proporciona una implementación específica para la habilidad especial
 * del personaje, que consisten en movimientos individuales del personaje.
 */
public class MetaKnight extends Character {

    /**
     * Implementación del método abstracto specialAbility() de la clase base Character.
     * Realiza la habilidad especial única de Meta Knight: supertornado.
     */
    @Override
    public void specialAbility() {
        // Imprime un mensaje indicando que la clase MetaKnight realiza su habilidad especial.
        System.out.println(getNombre() + " realiza supertornado!");
    }
}
```

```java
package com.game.characters;

/**
 * Clase que representa al personaje "King Dedede" en el juego, hereda de la clase base Character.
```

```
 * La clase MetaKnight proporciona una implementación específica para la habilidad especial
 * del personaje, que consisten en movimientos individuales del personaje.
 */
public class KingDedede extends Character {

    /**
     * Implementación del método abstracto specialAbility() de la clase base Character.
     * Realiza la habilidad especial única de King Dedede: martillo ígneo.
     */
    @Override
    public void specialAbility() {
        // Imprime un mensaje indicando que la clase KingDedede realiza su habilidad especial.
        System.out.println(getNombre() + " hace martillo ígneo!");
    }
}
```

Main

La clase principal `Main` ejemplifica la creación de una lista de personajes utilizando tanto una variable de referencia de tipo `CharacterActions` como de tipo `Character`. Además, demuestra la ejecución de acciones para los personajes del juego mediante la interfaz `CharacterActions`.

```
package com.main;

import com.game.characters.Character;
import com.game.characters.*;

import java.util.*;

/**
 * Clase principal que demuestra la creación y ejecución de acciones
 * para los personajes del juego.
 */
public class Main {

    /**
     * Método principal que crea instancias de personajes y realiza acciones sobre ellos.
     * Se demuestra tanto el uso de la clase abstracta Character como la interfaz CharacterActions.
     */
    public static void main(String[] args) {
        // Crear y mostrar acciones para personajes utilizando la clase abstracta Character.
        List<Character> characterList = createList(Character.class);
        System.out.println("---------------------------------");
        System.out.println("--------- Tipo Character --------");
        System.out.println("---------------------------------");
```

```
        performActions(characterList);

        // Crear y mostrar acciones para personajes utilizando la interfaz CharacterActions.
        List<CharacterActions> characterActionsList = createList(CharacterActions.class);
        System.out.println("---------------------------------");
        System.out.println("-----  Tipo CharacterActions ----");
        System.out.println("---------------------------------");
        performActions(characterActionsList);
    }

    /**
     * Crea una lista de personajes utilizando reflexion y cast().
     *
     * @param objectType Clase que define el tipo de objeto a crear y agregar a la lista characters.
     * @return           La lista de personajes creada de acuerdo al tipo de objeto recibido.
     *
     */
    private static <T> List<T> createList(Class<T> objectType) {
        List<T> characters = new ArrayList<>();
        characters.add(objectType.cast(new Kirby()));
        characters.add(objectType.cast(new MetaKnight()));
        characters.add(objectType.cast(new KingDedede()));
        return characters;
    }

    /**
     * Realiza las acciones para cada personaje en la lista, como saltar, moverse, protegerse, atacar
     * y realizar su habilidad especial.
     *
     * @param characterList Lista de personajes sobre los que se realizaran las acciones.
     *                      Debe de ser una lista que contenga objetos de clases que
     *                      implementen la interfaz CharacterActions.
     */
    private static void performActions(List<? extends CharacterActions> characterList) {
        for (CharacterActions character : characterList) {
            character.jump();
            character.move();
            character.guard();
            character.attack();
            character.specialAbility();
            System.out.println();
        }
    }
}
```

Durante la ejecución, se imprime una serie de mensajes que describen las acciones llevadas a cabo por cada personaje. La prueba valida la capacidad del polimorfismo al tratar tanto a la interfaz como a la clase abstracta

de manera uniforme, permitiendo así que la lista almacene objetos de diferentes tipos pero relacionados por su capacidad de llevar a cabo las acciones definidas en la interfaz.

Finalmente, la salida en la consola confirma la ejecución exitosa de las acciones, proporcionando un rastro detallado de las operaciones realizadas por cada personaje en el juego.

```
---------------------------------
--------- Tipo Character --------
---------------------------------
Kirby salta
Kirby se mueve
Kirby se protege
Kirby ataca
Kirby realiza caída veloz!

MetaKnight salta
MetaKnight se mueve
MetaKnight se protege
MetaKnight ataca
MetaKnight realiza supertornado!

KingDedede salta
KingDedede se mueve
KingDedede se protege
KingDedede ataca
KingDedede hace martillo ígneo!

---------------------------------
-----  Tipo CharacterActions ----
---------------------------------
Kirby salta
Kirby se mueve
Kirby se protege
Kirby ataca
Kirby realiza caída veloz!

MetaKnight salta
MetaKnight se mueve
MetaKnight se protege
MetaKnight ataca
MetaKnight realiza supertornado!

KingDedede salta
KingDedede se mueve
KingDedede se protege
```

```
KingDedede ataca
KingDedede hace martillo ígneo!
```

En esta salida, se confirma que la ejecución se realiza de manera exitosa, confirmando la aplicación efectiva del polimorfismo en la manipulación de la lista de personajes. La diferenciación entre el tipo de referencia de las variables, ya sea como Character o CharacterActions, no afecta la capacidad de ejecutar acciones específicas para cada personaje.

En la sección correspondiente a **Tipo Character**, los personajes son tratados como instancias de la clase abstracta `Character`, lo que permite acceder a los métodos definidos en dicha clase. Por otro lado, en Tipo CharacterActions, la lista es tratada como una colección de objetos que implementan la interfaz `CharacterActions`.