Herencia- Pokemon

Leonardo Hernández Chacón - B43262

21 de enero de 2017

Índice

---

1. Enunciado

Implemente un programa en C++ utilizando herencia que modele las tres aves legendarias de Pokemon: Articuno, Moltres y Zapdos:

- Implemente una clase abstracta Pokemon que contenga:
  - Atributos: name, species, HP, ATK, DEF, sATK, sDEF, SPD, EXP, call.
  - Métodos:
    - virtual void atk1(Pokemon other) = 0
    - virtual void atk2(Pokemon other) = 0
    - virtual void atk3(Pokemon other) = 0
    - virtual void atk4(Pokemon other) = 0
    - string call()
    - void printInfo()
- Implemente cuatro clases llamadas Electric, Fire, Water y Flying que modelan los tipos de aves míticas. Estas clases heredan de la clase Pokemon.
  - Métodos
    - static string type()
    - static string strongVs()
    - static string weakVs()
- Implemente tres clases concretas llamadas Zapdos, Articuno y Moltres.
  - Atributos
    - Type, strongVs, weakVs
  - Metodos
    - void print()
- Las clases Zapdos, Articuno y Moltres deben implementar los métodos virtuales puros y un método que imprima el estado del pokemon con todos sus atributos. Haga un programa de prueba en donde se creen dos objetos e interactuen entre ellos(se ataquen un par de veces).

2. Desglose

Se implementó las clases en programas como sigue:

- Clase Pokemon
  - Pokemon.cpp

        - Pokemon.hpp
    - Clases Tipo
        - Electric
            - Electric.cpp
            - Electric.hpp
        - Water
            - Water.cpp
            - Water.hpp
        - Fire
            - Fire.cpp
            - Fire.hpp
        - Flying
            - Flying.cpp
            - Flying.hpp
    - Clases Pokemones Legendarios
        - Zapdos
            - Zapdos.cpp
            - Zapdos.hpp
        - Moltres
            - Moltres.cpp
            - Moltres.hpp
        - Articuno
            - Articuno.cpp
            - Articuno.hpp

3. Pokemon

3.1. Pokemon.hpp

```
#ifndef POKEMON_HPP
#define POKEMON_HPP

#include <iostream>

using namespace std;

class Pokemon {
public:
    Pokemon();
    virtual ~Pokemon();
```

```
    //Metodos
    void printInfo();
    string callPokemon();

    //Metodos virtuales puros, que estan implementados en una clase inferior
    virtual void atk1(Pokemon &other)=0;
    virtual void atk2(Pokemon &other)=0;
    virtual void atk3(Pokemon &other)=0;
    virtual void atk4(Pokemon &other)=0;

    //Atributos
    string name;
    string species;
    int life;
    int HP;
    int ATK;
    int DEF;
    int sATK;
    int sDEF;
    int SPD;
    int EXP;
    string call;

};

#endif /* POKEMON_HPP */
```

3.2. Pokemon.cpp

```
#ifndef WATER_HPP
#define WATER_HPP

#include "Pokemon.hpp"
#include <iostream>

using namespace std;

class Water : virtual public Pokemon {
public:

    Water();
    virtual ~Water();
//Metodos  static string
    static string type();
    static string strongVs();
    static string weakVs();
};
```

```
#endif /* WATER_HPP */
```

4. Clase Electric

4.1. Electric.cpp

```
#include "Electric.hpp"
//Constructor de Electric
Electric::Electric() {
}
//Destructor de Electric
Electric::~Electric() {
}
//Metodos
string Electric::type() {
    return "Electric";
}

string Electric::strongVs() {
    return "Ground";
}

string Electric::weakVs() {
    return "Water, Flying";
}
```

4.2. Electric.hpp

```
#ifndef ELECTRIC_HPP
#define ELECTRIC_HPP

#include "Pokemon.hpp"
#include <iostream>

using namespace std;

class Electric : virtual public Pokemon {
public:

    Electric();
    virtual ~Electric();

//Metodos estaticos string
    static string type();
    static string strongVs();
    static string weakVs();
};

#endif /* ELECTRIC_HPP */
```

5. Clase Water

5.1. Water.cpp

```
#include "Water.hpp"
//Constructor de Water
Water::Water() {

}
//Destructor de Water
Water::~Water() {

}
//Metodos
string Water::type() {
    return "Water";
}

string Water::strongVs() {
    return "Electric, Grass";
}

string Water::weakVs() {
    return "Fire, Ground, Rock";
}
```

5.2. Water.hpp

```
#ifndef WATER_HPP
#define WATER_HPP

#include "Pokemon.hpp"
#include <iostream>

using namespace std;

class Water : virtual public Pokemon {
public:

    Water();
    virtual ~Water();
//Metodos  static string
    static string type();
    static string strongVs();
    static string weakVs();
};

#endif /* WATER_HPP */
```

6. Clase Fire

6.1. Fire.cpp

```
#include "Fire.hpp"
//Constructor de Fire
Fire::Fire() {

}
//Destructor de Fire
Fire::~Fire() {

}

//Metodos
string Fire::type() {
    return "Fire";
}

string Fire::strongVs() {
    return "Water, Ground, Rock";
}

string Fire::weakVs() {
    return "Grass, Ice, Bug, Steel";
}
```

6.2. Fire.hpp

```
#ifndef FIRE_HPP
#define FIRE_HPP

#include "Pokemon.hpp"
#include <iostream>

using namespace std;

class Fire : virtual public Pokemon {
public:

    Fire();
    virtual ~Fire();
//Metodos static string
    static string type();
    static string strongVs();
    static string weakVs();
};

#endif /* FIRE_HPP */
```

7. Clase Flying

7.1. Flying.cpp

```
#include "Flying.hpp"
//Constructor de Flying
Flying::Flying() {

}
//Destructor de Flying
Flying::~Flying() {
}
//Metodos
string Flying::type(){
    return "Flying";
}

string Flying::strongVs() {
    return "Electric, Ice, Rock";
}

string Flying::weakVs() {
    return "Grass, Fight, Bug";
}
```

7.2. Flying.hpp

```
#ifndef FLYING_HPP
#define FLYING_HPP

#include "Pokemon.hpp"
#include <iostream>

using namespace std;

class Flying : virtual public Pokemon {
public:
    Flying();
    virtual ~Flying();
//Metodos static string
    static string type();
    static string strongVs();
    static string weakVs();
};


#endif /* FLYING_HPP */
```

8. Clase Zapdos

8.1. Zapdos.hpp

```
#ifndef ZAPDOS_HPP
#define ZAPDOS_HPP

#include "Pokemon.hpp"
#include <iostream>
#include "Electric.hpp"
#include "Flying.hpp"

using namespace std;
//Zapdos es un pokemon tipo electric y tipo flying
//as  que hereda de ambas clases.
class Zapdos : public Electric, public Flying {
public:

    Zapdos();
    virtual ~Zapdos();
//Atributos
    string Type;
    string strongVs;
    string weakVs;

//Implementaci n de los m todos virtuales puros.
   void atk1(Pokemon &other);
   void atk2(Pokemon &other);
   void atk3(Pokemon &other);
   void atk4(Pokemon &other);

   void printPokemon();

};

#endif /* ZAPDOS_HPP */
```

8.2. Zapdos.cpp

```
#include "Zapdos.hpp"
//Constructor de Zapdos
Zapdos::Zapdos() {
    //Se concatena el llamado a los metodos string type() de Electric y Flying
    //que retornan el tipo.
    Type= Electric::type() + "/" + Flying::type();
    //Se concatena el llamado a los metodos string strongVs() de Electric y Flying
    //que retornan el tipo contra contra quien es     fuerte el pokemon.
    strongVs= Electric::strongVs() + ", " + Flying::strongVs();
    //Se concatena el llamado a los metodos string weakVs() de Electric y Flying
```

```
    //que retornan el tipo contra contra quien es debil el pokemon.
    weakVs= Electric::weakVs() + ", " + Flying::weakVs();
}
//Destructor de Zapdos
Zapdos::~Zapdos() {
}

//Metodo para imprimir estado
void Zapdos::printPokemon(){
 cout << "Tipo: "<< Type << endl;
 cout << "Es fuerte contra los tipo: "<< strongVs << endl;
 cout << "Es debil contra los tipo: "<< weakVs << endl;

}
//Ataque 1
void Zapdos::atk1(Pokemon &other){

 if (ATK >= other.DEF)
   {
     other.life = other.life - 2*(ATK - other.DEF);
     EXP=EXP +2;
   }

   else
   {
     life = life - 2*(other.DEF - ATK);
     other.EXP= other.EXP +2;
   }
}

//Ataque 2
void Zapdos::atk2(Pokemon &other){
 if (sATK >= other.sDEF)
   {
     other.life = other.life - 3*(sATK - other.sDEF);
     EXP=EXP +3;
   }

   else
   {
     life = life - 3*(other.sDEF - sATK);
     other.EXP= other.EXP +3;
   }

}
//Ataque 3
void Zapdos::atk3(Pokemon &other){
if (SPD >= other.SPD)
```

```
    {
      other.life = other.life - (SPD - other.SPD);
      EXP=EXP +1;
    }

    else
    {
      life = life - (other.SPD - SPD);
      other.EXP= other.EXP +1;
    }

}
//Ataque 4
void Zapdos::atk4(Pokemon &other){
if (life < 50)
    {
      life = life +9;
    }

    else
    {
     life=life;
    }
}
```

9. Clase Moltres

9.1. Moltres.hpp

```
#ifndef MOLTRES_HPP
#define MOLTRES_HPP

#include "Pokemon.hpp"
#include <iostream>
#include "Fire.hpp"
#include "Flying.hpp"

using namespace std;
//Moltres es un pokemon tipo fire y tipo flying, as  que hereda de ambas clases.
class Moltres : public Fire , public Flying {
public:

    Moltres();
    virtual ~Moltres();
//Atributos
    string Type;
    string strongVs;
    string weakVs;
```

```
//Implementaci n de los m todos virtuales puros.
    void atk1(Pokemon &other);
    void atk2(Pokemon &other);
    void atk3(Pokemon &other);
    void atk4(Pokemon &other);

    void printPokemon();
};

#endif /* MOLTRES_HPP */
```

9.2. Moltres.cpp

```
#include "Moltres.hpp"
//Constructor de Moltres
Moltres::Moltres() {
     //Se concatena el llamado a los metodos string type() de Fire y Flying
     //que retornan el tipo.
     Type= Fire::type() + "/" + Flying::type();
    //Se concatena el llamado a los metodos string strongVs() de Fire y Flying
    //que retornan el tipo contra contra quien es     fuerte el pokemon.
     strongVs= Fire::strongVs() + ", " + Flying::strongVs();
 //Se concatena el llamado a los metodos string weakVs() de Fire y Flying
 //que retornan el tipo contra contra quien es debil el pokemon.
     weakVs= Fire::weakVs() + ", " + Flying::weakVs();
}
//Destructor de Moltres
Moltres::~Moltres() {
}
//Metodo para imprimir estado
void Moltres::printPokemon(){
 cout << "Tipo: "<< Type << endl;
 cout << "Es fuerte contra los tipo: "<< strongVs << endl;
 cout << "Es debil contra los tipo: "<< weakVs << endl;

}

//Primer ataque
void Moltres::atk1(Pokemon &other){
if (ATK >= other.DEF)
    {
      other.life = other.life - (ATK - other.DEF);
      EXP=EXP +1;
    }

    else
```

```
    {
      life = life - (other.DEF - ATK);
      other.EXP= other.EXP +1;
    }

}
//Segundo ataque
void Moltres::atk2(Pokemon &other){
if (sATK >= other.sDEF)
    {
      other.life = other.life - 3*(sATK - other.sDEF);
      EXP=EXP +3;
    }

    else
    {
      life = life - 3*(other.sDEF - sATK);
      other.EXP= other.EXP +3;
    }
}
//Tercer ataque
void Moltres::atk3(Pokemon &other){
if (SPD >= other.SPD)
    {
      other.life = other.life - 2*(SPD - other.SPD);
      EXP=EXP +2;
    }

    else
    {
      life = life - 2*(other.SPD - SPD);
      other.EXP= other.EXP +2;
    }
}
//Cuarto ataque
void Moltres::atk4(Pokemon &other){
if (life < 70)
    {
      life = life +6;
    }

    else
    {
     life=life;
    }
}
```

10. Clase Articuno

10.1. Articuno.hpp

```
#ifndef ARTICUNO_HPP
#define ARTICUNO_HPP

#include "Pokemon.hpp"
#include <iostream>
#include "Water.hpp"
#include "Flying.hpp"

using namespace std;
//Articuno es un pokemon tipo water y tipo flying
//asi que hereda de ambas clases.
class Articuno : public Water, public Flying {
public:

    Articuno();
    virtual ~Articuno();

//Atributos
    string Type;
    string strongVs;
    string weakVs;

//Implementaci n de los m todos virtuales puros.
   void atk1(Pokemon &other);
   void atk2(Pokemon &other);
   void atk3(Pokemon &other);
   void atk4(Pokemon &other);

   void printPokemon();


};

#endif /* ARTICUNO_HPP */
```

10.2. Articuno.cpp

```
#include "Articuno.hpp"

//Constructor de Articuno
Articuno::Articuno() {
    //Se concatena el llamado a los metodos string type() de Water y Flying
    //que retornan el tipo.
    Type= Water::type() + "/" + Flying::type();
    //Se concatena el llamado a los metodos string strongVs() de Water y Flying
```

```
    //que retornan el tipo contra contra quien es fuerte el pokemon.
    strongVs= Water::strongVs() + ", " + Flying::strongVs();
    //Se concatena el llamado a los metodos string weakVs() de Water y Flying
    //que retornan el tipo contra contra quien es debil el pokemon.
    weakVs= Water::weakVs() + ", " + Flying::weakVs();
}
//Destructor de Articuno
Articuno::~Articuno() {

}
//Metodo para imprimir estado
void Articuno::printPokemon(){
 cout << "Tipo: "<< Type << endl;
 cout << "Es fuerte contra los tipo: "<< strongVs << endl;
 cout << "Es debil contra los tipo: "<< weakVs << endl;

}
//Primer ataque
void Articuno::atk1(Pokemon &other){
if (ATK >= other.DEF)
   {
     other.life = other.life - 3*(ATK - other.DEF);
     EXP=EXP +3;
   }

   else
   {
     life = life - 3*(other.DEF - ATK);
     other.EXP= other.EXP +3;
   }
}
//Segundo ataque
void Articuno::atk2(Pokemon &other){
if (sATK >= other.sDEF)
   {
     other.life = other.life - 2*(sATK - other.sDEF);
     EXP=EXP +2;
   }

   else
   {
     life = life - 2*(other.sDEF - sATK);
     other.EXP= other.EXP +2;
   }
}
//Tercer ataque
void Articuno::atk3(Pokemon &other){
if (SPD >= other.SPD)
```

```
    {
      other.life = other.life - (SPD - other.SPD);
      EXP=EXP +1;
    }

    else
    {
      life = life - (other.SPD - SPD);
      other.EXP= other.EXP +1;
    }
}
//Cuarto ataque
void Articuno::atk4(Pokemon &other){
if (life < 90)
    {
      life = life +4;
    }

    else
    {
     life=life;
    }
}
```

11. Main.cpp

```
#include <iostream>
#include "Pokemon.hpp"

#include "Electric.hpp"
#include "Flying.hpp"
#include "Fire.hpp"
#include "Water.hpp"

#include "Zapdos.hpp"
#include "Moltres.hpp"
#include "Articuno.hpp"


using namespace std;

int main(int argc, char** argv) {

cout << "Valores Iniciales:   " << endl;
cout <<  endl;
//Creamos objeto tipo Zapdos:
     Zapdos pZ0;
//Asiganamos estos valores los atributos
```

```
//los demas mantienen su valor por defecto.
     pZ0.name= "Juancito";
     pZ0.species= "Zapdos";
     pZ0.HP = 5;
     pZ0.ATK = 6;
     pZ0.DEF = 4;
     pZ0.sATK = 8;
     pZ0.sDEF = 6;
     pZ0.SPD = 9;
     pZ0.EXP = 0;
     pZ0.call = "Grrrr";



//Creamos Objeto tipo Articuno:
     Articuno pZ1;
//Asiganamos estos valores los atributos
//los demas mantienen su valor por defecto.
     pZ1.name= "Pedrito";
     pZ1.species= "Articuno";
     pZ1.HP = 3;
     pZ1.ATK = 8;
     pZ1.DEF = 7;
     pZ1.sATK = 7;
     pZ1.sDEF = 5;
     pZ1.SPD = 11;
     pZ1.EXP = 0;
     pZ1.call = "Brrrr";


//Vamos a simular una batalla pokemon.
// Estado inicial
     pZ0.printInfo();
  cout << endl << endl;
     pZ1.printInfo();
  cout << endl << endl;

//PRIMER ATAQUE
    pZ1.atk1(pZ0);
    pZ0.atk1(pZ1);

     cout << "PRIMERA RONDA ATAQUE: " << endl << endl;

     pZ0.printInfo();
  cout << endl << endl;
     pZ1.printInfo();
  cout << endl << endl;
```

```
//SEGUNDO ATAQUE
    pZ1.atk2(pZ0);
    pZ0.atk2(pZ1);

     cout << "SEGUNDA RONDA ATAQUE: " << endl << endl;

     pZ0.printInfo();
  cout << endl << endl;
     pZ1.printInfo();
  cout << endl << endl;

//TERCER ATAQUE
   pZ1.atk3(pZ0);
   pZ0.atk3(pZ1);

  cout << "TERCERA RONDA ATAQUE: " << endl << endl;

    pZ0.printInfo();
  cout << endl << endl;
    pZ1.printInfo();
  cout << endl << endl;

//Se puede continuar con los ataques, y definir una regla para que un pokemon gane
//Esto era una demostracion de los pokemon interactuan correctamente.

    return 0;
};
```

12. Conclusiones

Para mostrar los resultados de una batalla pokemon, se imprime en consola los valores iniciales de los atributos de cada pokemon con el método printInfo(), y posteriormente los valores de los atributos luego de cada ronda de ataques, la vida(atributo life) y la experiencia(atributo EXP) son los atributos a los cuales se debe prestar atención pues de acuerdo a como definimos los ataques esos son los atributos que van a modificarse. Veamos un ejemplo de como se va a imprimir todo esto en pantalla:

Y así sucesivamente se desarrolla la batalla pokemon:

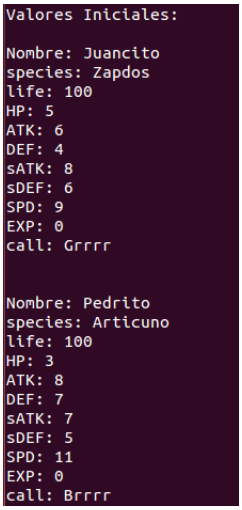

Figura 1: Valores Iniciales

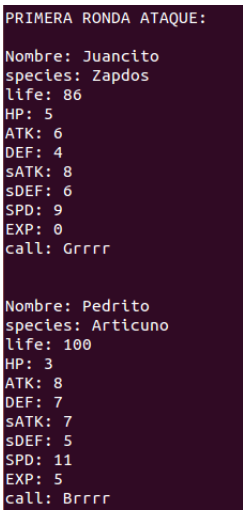

Figura 2: Primer Ataque

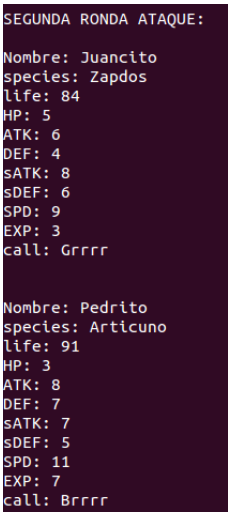

Figura 3: Segundo Ataque