Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

По дисциплине: “Рефакторинг программного обеспечения”

Лабораторная работа №2

“Рефакторинг программного кода. Перемещение функций между объектами”

Выполнил:

ст.гр. ИС/б-17-2

Долженко И.А.

Проверил:

Строганов В.А.

Севастополь

2020

1 ЦЕЛЬ РАБОТЫ

Исследовать эффективность перемещения функций между объектами при рефакторинге программного кода. Получить практические навыков применения приемов рефакторинга объектно-ориентированных программ.

2 ХОД РАБОТЫ

**1-3. Выделение класса, перемещение метода, перемещение поля.**

В качестве исходного кода были взяты классы игровых персонажей – Sword и Wizard из среды разработки игровых приложение Unity. Код данных классов представлен ниже:

Класс Wizard:

class Wizard : MonoBehaviour

{

    public float Speed;

    protected int damage;

    public int minDamage;

    public int maxDamage;

    protected Vector2 direction;

    private Animator animator;

    protected float timeBtwUseSpell;

    public float startTimeBtwUseSpell;

    protected bool isCooldown;

    private GameObject projectile;

    public float projectileForce;

    public int numberBullet;

    public GameObject teleportFX;

    void Update ()

    {

        TakeInput();

        Move();

        Attack();

        Dash();

    }

    void TakeInput()

    {

        direction = Vector2.zero;

*if* (Input.GetKey(KeyCode.W)) direction += Vector2.up \* 0.9f;

*if* (Input.GetKey(KeyCode.A)) direction += Vector2.left;

*if* (Input.GetKey(KeyCode.S)) direction += Vector2.down \* 0.9f;

*if* (Input.GetKey(KeyCode.D)) direction += Vector2.right;

    }

    void Move()

    {

*// Move*

    }

    void Attack()

    {

*if* (Input.GetMouseButtonDown(0) && Time.timeScale == 1)

        {

*// Attack*

        }

    }

    void Dash()

    {

*if* (timeBtwUseSpell <= 0)

        {

*if* (Input.GetKeyDown(KeyCode.F) && Time.timeScale == 1)

            {

*// Dash ability*

            }

        } *else* {

            timeBtwUseSpell -= Time.deltaTime;

        }

    }

}

Класс Sword:

class Sword : MonoBehaviour

{

    public float Speed;

    protected int damage;

    public int minDamage;

    public int maxDamage;

    protected Vector2 direction;

    private Animator animator;

    protected float timeBtwUseSpell;

    public float startTimeBtwUseSpell;

    protected bool isCooldown;

    private float timeBtwAttack;

    public float startTimeBtwAttack;

    public Transform attackPos;

    public LayerMask whatIsEnemies;

    public float attackRange;

    public GameObject armorFX;

    public float energy;

    void Update ()

    {

        Move();

        TakeInput();

        Attack();

        Energy();

        Armor();

    }

    void TakeInput()

    {

        direction = Vector2.zero;

*if* (Input.GetKey(KeyCode.W)) direction += Vector2.up \* 0.9f;

*if* (Input.GetKey(KeyCode.A)) direction += Vector2.left;

*if* (Input.GetKey(KeyCode.S)) direction += Vector2.down \* 0.9f;

*if* (Input.GetKey(KeyCode.D)) direction += Vector2.right;

    }

    void Move()

    {

*// Move*

    }

    void Attack()

    {

*if* (timeBtwAttack <= 0)

        {

*if* (Input.GetKeyDown(KeyCode.Mouse0))

            {

*// Attack*

            }

        } *else* {

            timeBtwAttack -= Time.deltaTime;

        }

    }

    void Energy()

    {

*if* (energy > 10 && Input.GetKey(KeyCode.Space)) {

            energy -= 0.5f;

            Speed = 3.7f;

        } *else* {

            energy += 0.5f;

            Speed = 2.5f;

        }

*if* (energy >= 100) energy = 100;

*if* ((energy == 10 && Input.GetKey(KeyCode.Space)) || energy <= 0) energy = 0;

    }

    void Armor()

    {

*if* (timeBtwUseSpell <= 0)

        {

*if* (Input.GetKeyDown(KeyCode.F) && Time.timeScale == 1)

            {

*// Armor ability*

            }

        } *else* {

            timeBtwUseSpell -= Time.deltaTime;

        }

    }

}

Можно заметить, что данные классы содержат множество повторяющихся полей и методов. Для решения данной проблемы создадим родительский класс PlayerBase, который будет наследован классами игровых персонажей и содержать общие поля и методы.

Создадим новый класс PlayerBase, воспользовавшись методами рефакторинга “Перемещение метода” и “Перемещение поля”:

**abstract class PlayerBase : MonoBehaviour**

**{**

**public float Speed;**

**protected int damage;**

**public int minDamage;**

**public int maxDamage;**

**protected Vector2 direction;**

**private Animator animator;**

**protected float timeBtwUseSpell;**

**public float startTimeBtwUseSpell;**

**protected bool isCooldown;**

**protected virtual void TakeInput()**

**{**

**direction = Vector2.zero;**

***if* (Input.GetKey (KeyCode.W)) direction += Vector2.up\*0.9f;**

***if* (Input.GetKey (KeyCode.A)) direction += Vector2.left;**

***if* (Input.GetKey (KeyCode.S)) direction += Vector2.down\*0.9f;**

***if* (Input.GetKey (KeyCode.D)) direction += Vector2.right;**

**}**

**protected void Move()**

**{**

***// Move***

**}**

**protected abstract void Attack();**

**}**

В результате создания базового класса из классов персонажей, были вынесены повторяющиеся поля и методы, а также создан абстрактный метод Attack, тело которого определяется в классах-наследниках.

Классы после рефакторинга представлены ниже:

class Wizard :**PlayerBase**

{

    private GameObject projectile;

    public float projectileForce;

    public int numberBullet;

    public GameObject teleportFX;

    void Update ()

    {

        TakeInput();

        Move();

        Attack();

        Dash();

    }

    void Attack()

    {

*if* (Input.GetMouseButtonDown(0) && Time.timeScale == 1)

        {

*// Attack*

        }

    }

    void Dash()

    {

*if* (timeBtwUseSpell <= 0)

        {

*if* (Input.GetKeyDown(KeyCode.F) && Time.timeScale == 1)

            {

*// Dash ability*

            }

        } *else* {

            timeBtwUseSpell -= Time.deltaTime;

        }

    }

}

class Sword : **PlayerBase**

{

    private float timeBtwAttack;

    public float startTimeBtwAttack;

    public Transform attackPos;

    public LayerMask whatIsEnemies;

    public float attackRange;

    public GameObject armorFX;

    public float energy;

    void Update ()

    {

        TakeInput();

        Move();

        Attack();

        Energy();

        Armor();

    }

    void Attack()

    {

*if* (timeBtwAttack <= 0)

        {

*if* (Input.GetKeyDown(KeyCode.Mouse0))

            {

*// Attack*

            }

        } *else* {

            timeBtwAttack -= Time.deltaTime;

        }

    }

    void Energy()

    {

*if* (energy > 10 && Input.GetKey(KeyCode.Space)) {

            energy -= 0.5f;

            Speed = 3.7f;

        } *else* {

            energy += 0.5f;

            Speed = 2.5f;

        }

*if* (energy >= 100) energy = 100;

*if* ((energy == 10 && Input.GetKey(KeyCode.Space)) || energy <= 0) energy = 0;

    }

    void Armor()

    {

*if* (timeBtwUseSpell <= 0)

        {

*if* (Input.GetKeyDown(KeyCode.F) && Time.timeScale == 1)

            {

*// Armor ability*

            }

        } *else* {

            timeBtwUseSpell -= Time.deltaTime;

        }

    }

}

**4. Встраивание класса.**

Данный метод противоположен “Выделению класса”. Им можно воспользоваться, когда от класса становится мало пользы. Часто это происходит в результате рефакторинга, оставившего в классе мало функций.

Фрагмент кода до рефакторинга:

class Person {

private String \_name;

private TelephoneNumber \_officeTelephone = new TelephoneNumber();

public String getName() {

return \_name;

}

public String getTelephoneNumber(){

return \_officeTelephone.getTelephoneNumber();

}

TelephoneNumber getOfficeTelephone() {

return \_officeTelephone;

}

}

class TelephoneNumber {

private String \_number;

private String \_areaCode;

public String getTelephoneNumber() {

return ("(" + \_areaCode + ") " + \_number);

}

String getAreaCode() {

return \_areaCode;

}

void setAreaCode(String arg) {

\_areaCode = arg;

}

String getNumber() {

return \_number;

}

void setNumber(String arg) {

\_number = arg;

}

}

В этом случае следует вставить класс TelephoneNumber в Person, так как класс TelephoneNumber несет слишком мало пользы, а класс Person чаще всего его использует.

Фрагмент кода после рефакторинга:

class Person {

private String \_name;

**private String \_officeAreaCode;**

**private String \_officeNumber;**

public String getName() {

return \_name;

}

**public String getTelephoneNumber() {**

**return ("(" + \_officeAreaCode + ") " + \_officeNumber);**

**}**

**String getOfficeAreaCode() {**

**return \_officeAreaCode;**

**}**

**void setOfficeAreaCode(String arg) {**

**\_officeAreaCode = arg;**

**}**

**String getOfficeNumber() {**

**return \_officeNumber;**

**}**

**void setOfficeNumber(String arg) {**

**\_officeNumber = arg;**

**}**

}

**5. Сокрытие делегирования.**

Существуют классы, представляющие работника и его отдел:

class Person {

private String \_name;

private Department \_department;

public String getName() {

return \_name;

}

public Department getDepartment() {

return \_department;

}

public void setDepartment(Department arg) {

\_department = arg;

}

}

class Department {

private String \_chargeCode;

private Person \_manager;

public Department (Person manager) {

\_manager = manager;

}

public Person getManager() {

return \_manager;

}

}

Если клиенту требуется узнать, кто является менеджером некоторого лица, он должен сначала узнать, в каком отделе это лицо работает:

manager = john.getDepartment().getManager();

Так клиенту открывается характер работы класса Department и то, что в нем хранятся данные о менеджере. Эту связь можно сократить, скрыв от клиента класс Department. Это осуществляется путем создания простого делегирующего метода в Person:

public Person getManager() {

return \_department.getManager();

}

Теперь для получения менеджера можно сразу обратиться к соответствующему методу:

manager = john.getManager();

**6. Введение внешнего метода**

Есть некий код, в котором нужно открыть новый период выставления счетов. Первоначально код выглядит так:

...

Date newStart = new Date (previousEnd.getYear(), previousEnd.getMonth(), previousEnd.getDate() + 1);

...

Код в правой части присваивания можно выделить в метод, который будет внешним для Date:

...

Date newStart = nextDay(previousEnd);

...

private static Date nextDay(Date arg) {

return new Date (arg.getYear(), arg.getMonth(), arg.getDate() + 1);

}

**7. Введение локального расширения**

В задаче требуется сгенерировать числа типа float, но стандартный класс Random не имеет такого метода. Создадим класс обертку, дополнив его необходимым методом.

Исходный код до рефакторинга:

static void Main(string[] args)

{

// В двухмерном массиве вещественных чисел поменять местами строки и столбцы с одинаковыми номерами.

Console.Write("Введите размерность матрицы: ");

int size = Int32.Parse(Console.ReadLine());

Console.WriteLine("Исходный массив:");

double[,] arr = new double[size, size];

Random rnd = new Random();

for (int i = 0; i < size; i++)

for (int j = 0; j < size; j++)

arr[i, j] = -10 + rnd.NextDouble() \* 20;

PrintArray(arr);

Console.WriteLine("Результат:");

for (int i = 0, length = arr.GetLength(0); i < length; i++)

for (int j = i + 1; j < length; j++)

{

double temp = arr[i, j];

arr[i, j] = arr[j, i];

arr[j, i] = temp;

}

PrintArray(arr);

Console.ReadLine();

}

После введения локального расширения:

class Program

{

static void Main(string[] args)

{

// В двухмерном массиве вещественных чисел поменять местами строки и столбцы с одинаковыми номерами.

Console.Write("Введите размерность матрицы: ");

int size = Int32.Parse(Console.ReadLine());

Console.WriteLine("Исходный массив:");

double[,] arr = new double[size, size];

RandomWrap rnd = new RandomWrap();

for (int i = 0; i < size; i++)

for (int j = 0; j < size; j++)

arr[i, j] = rnd.NextFloat(-10, 10);

PrintArray(arr);

Console.WriteLine("Результат:");

for (int i = 0, length = arr.GetLength(0); i < length; i++)

for (int j = i + 1; j < length; j++)

{

double temp = arr[i, j];

arr[i, j] = arr[j, i];

arr[j, i] = temp;

}

PrintArray(arr);

Console.ReadLine();

}

}

public class RandomWrap : Random

{

public double NextFloat(double minValue, double maxValue)

{

return minValue + base.NextDouble() \* (maxValue - minValue);

}

}

ВЫВОДЫ

В ходе выполнения лабораторной работы была исследована эффективность перемещения функций между объектами при рефакторинге программного кода. Получены практические навыки применения приемов рефакторинга объектно-ориентированных программ.