Reflector

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Runtime.Remoting.Contexts;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using static System.Net.Mime.MediaTypeNames;

namespace OOP11
{

    static class Reflector
    {
        delegate void ResearchedClass<T>();
        public static void ResearchClass<T>(T obj)
        {
            var type = obj.GetType();
            var props = type.GetProperties();

            using (StreamWriter writer = new StreamWriter(type.Name + ".txt", false))
            {
                writer.WriteLineAsync(props[0].DeclaringType.Assembly.ToString());

                writer.WriteLineAsync("-------------Конструкторы-------------");
                foreach (var constructor in type.GetConstructors())
                {
                    writer.WriteLineAsync(constructor.ToString());
                }

                writer.WriteLineAsync("------------------Методы----------------");
                foreach (var method in type.GetMethods())
                {
                    writer.WriteLineAsync(method.ToString());
                }

                writer.WriteLineAsync("-----------------Поля----------------");
                foreach (var field in type.GetFields())
                {
                    writer.WriteLineAsync(field.ToString());
                }
                writer.WriteLineAsync("-----------------Свойства----------------");
                foreach (var field in type.GetProperties())
                {
                    writer.WriteLineAsync(field.ToString());
                }
                writer.WriteLineAsync("-----------------Интерфейсы----------------");
                foreach (var interface_ in type.GetInterfaces())
                {
                    writer.WriteLineAsync(interface_.ToString());
                }
                writer.WriteLineAsync("------------------Методы 2----------------");
                var methods = type.GetMethods();
                foreach (var method in methods)
                {
                    foreach (var param in method.GetParameters())
                    {
                        if (param.Name.GetType() == typeof(System.String))
                            writer.WriteLineAsync(method.ToString());
                    }
                }
            }
        }

        public static void Invoke(object obj, string className, string methodName, object[] params_)
        {
            Type t = Type.GetType(className);
            var handler = t.GetMethod(methodName);
            handler.Invoke(obj, new object[] { });
        }
        public static T Create<T>() where T : new()
        {
            T ob = new T();
            return ob;
        }
    }
    public class Person
    {
        public string Name { get; }
        public int age;
        public Person(string name) => Name = name;
        public Person(int name){}
        public Person()
        {
            Name = "Ivan";
            age = 18;
        }

        public void Walk()
        {
            Console.WriteLine("Я сплю");
        }
        public void Eat(string food) {}
        public void Sleep(int hour) {}

    }

    internal class Program
    {
        public static void Main(string[] args)
        {
            Person person = new Person("Ivan");
            Reflector.ResearchClass(person);
            Reflector.Create<Person>();
            string text;
            using (StreamReader reader = new StreamReader("param.txt"))
            {
                text = reader.ReadToEnd();
                Console.WriteLine(text);
            }
            string[] words = text.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
            Reflector.Invoke(person, words[0], words[1], params_: null);
            Console.ReadLine();
        }
    }
}
```