**Вопросы к лабораторной работы №11:**

1. Что такое рефлексия?

**Рефлексия** - процесс выявления типов во время выполнения приложения. Каждое приложение содержит набор используемых классов, интерфейсов, а также их методов и свойств. Рефлексия как раз и позволяет определить все эти составные элементы приложения.

1. Какие есть способы получения объекта типа Type. Приведите три

способа получения типа.

С помощью ключевого слова **typeof**, с помощью метода **GetType()** класса Object и применяя статический метод **Type.GetType().**

1) Type t = admin.GetType();  
2) Type t1 = Type.GetType(“System.Int32”);  
3) Type t2 = typeof (Point).

1. Охарактеризуйте классы из пространства имен System.Reflection.

Основной функционал рефлексии сосредоточен в пространстве имен System.Reflection. В нем мы можем выделить следующие основные классы:

* Assembly: класс, представляющий сборку и позволяющий манипулировать этой сборкой
* AssemblyName: класс, хранящий информацию о сборке
* MemberInfo: базовый абстрактный класс, определяющий общий функционал для классов EventInfo, FieldInfo, MethodInfo и PropertyInfo
* EventInfo: класс, хранящий информацию о событии
* FieldInfo: хранит информацию об определенном поле типа
* MethodInfo: хранит информацию об определенном методе
* PropertyInfo: хранит информацию о свойстве
* ConstructorInfo: класс, представляющий конструктор
* Module: класс, позволяющий получить доступ к определенному модулю внутри сборки
* ParameterInfo: класс, хранящий информацию о параметре метода

1. Как можно использовать класс System.Type? Перечислите его свойства

и методы.

Класс **System.Type** представляет изучаемый тип, инкапсулируя всю информацию о нем. С помощью его свойств и методов можно получить эту информацию. Некоторые из его свойств и методов:

* Метод **FindMembers()** возвращает массив объектов MemberInfo данного типа
* Метод **GetConstructors()** возвращает все конструкторы данного типа в виде набора объектов ConstructorInfo
* Метод **GetEvents()** возвращает все события данного типа в виде массива объектов EventInfo
* Метод **GetFields()** возвращает все поля данного типа в виде массива объектов FieldInfo
* Метод **GetInterfaces()** получает все реализуемые данным типом интерфейсы в виде массива объектов Type
* Метод **GetMembers()**возвращает все члены типа в виде массива объектов MemberInfo
* Метод **GetMethods()** получает все методы типа в виде массива объектов MethodInfo
* Метод **GetProperties()** получает все свойства в виде массива объектов PropertyInfo
* Свойство **IsAbstract** возвращает true, если тип является абстрактным
* Свойство **IsArray** возвращает true, если тип является массивом
* Свойство **IsClass** возвращает true, если тип представляет класс
* Свойство **IsEnum** возвращает true, если тип является перечислением
* Свойство **IsInterface** возвращает true, если тип представляет интерфейс.

1. Что такое позднее и раннее связывание?

**Раннее связывание** – связанное с формированием кода на этапе компиляции. При раннем связывании, программный код формируется на основе известной информации о типе (класс) ссылки. Как правило, это ссылка на базовый класс в иерархии классов.

**Позднее связывание** – связанное с формированием кода на этапе выполнения. Если в иерархии классов встречается цепочка виртуальных методов (с помощью слов virtual, override), то компилятор строит так называемое позднее связывание. При позднем связывании вызов метода происходит на основании типа объекта, а не типа ссылки на базовый класс. Позднее связывание используется, если нужно реализовать полиморфизм.

С помощью его статического метода *Activator.CreateInstance()* можно создавать экземпляры заданного типа.

1. Как динамически загрузить сборку в приложение?

Чтобы динамически загрузить сборку в приложение, надо использовать статические методы **Assembly.LoadFrom()** или **Assembly.Load().**

Метод **LoadFrom()** принимает в качестве параметра путь к сборке.

Метод **Load()** в качестве его параметра передается дружественное имя сборки, которое нередко совпадает с именем приложения.

1. Для чего предназначены BindingFlags? Какую комбинацию флагов

необходимо использовать, чтобы иметь возможность получать

приватные члены класса?

Флаги BindingFlags, связанные с получением.

