Модуль 7: Использование ссылочных типов

Обзор

- Использование стандартных ссылочных типов
- Иерархия объектов с единым корнем
- Преобразование типов

Использование ссылочных переменных

- Сравнение размерных и ссылочных типов данных
- Объявление и удаление ссылочных переменных
- Неверное использование ссылок
- Сравнение значений и сравнение ссылок
- Множество ссылок на один и тот же объект
- Ссылки как параметры в методах

Сравнение размерных и ссылочных типов данных

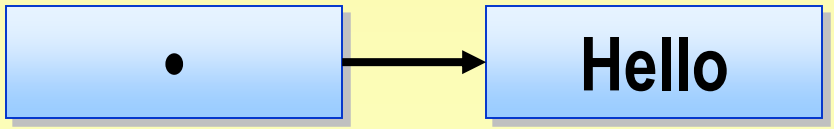
- **Размерные типы**
  - В переменной хранятся сами данные
  - Примеры: **char**, **int**

```
int mol;
mol = 42;
```

42

- **Ссылочные типы**
  - В переменной содержится ссылка на данные
  - Данные хранятся в отдельной области памяти

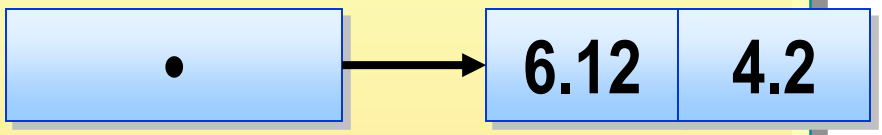
```
string mol;
mol = "Hello";
```



Объявление и удаление ссылочных переменных

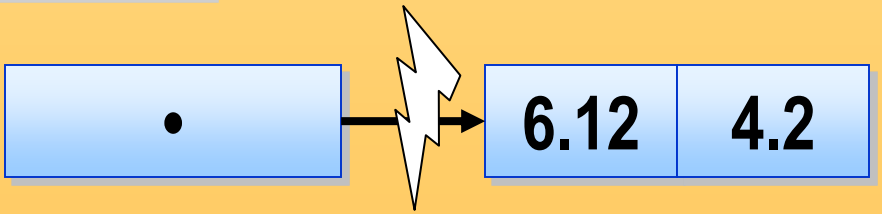
- **Объявление ссылочных переменных**

```
coordinate c1;
c1 = new coordinate();
c1.x = 6.12;
c1.y = 4.2;
```



- **Удаление ссылочных переменных**

```
c1 = null;
```

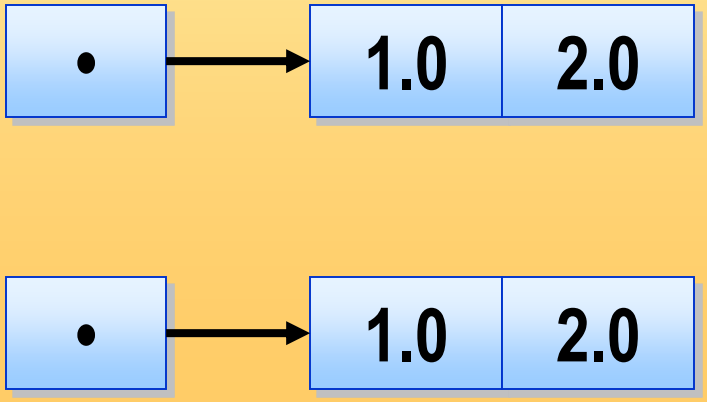


Неверное использование ссылок

- **Если вы используете неверную ссылку**
  - Вы не можете добраться до свойств и методов несуществующего объекта
- **Неправильные ссылки на этапе компиляции**
  - Компилятор отлавливает попытки использования не инициализированных переменных
- **Неправильные ссылки на этапе выполнения**
  - Система генерирует исключение

Сравнение значений и сравнение ссылок

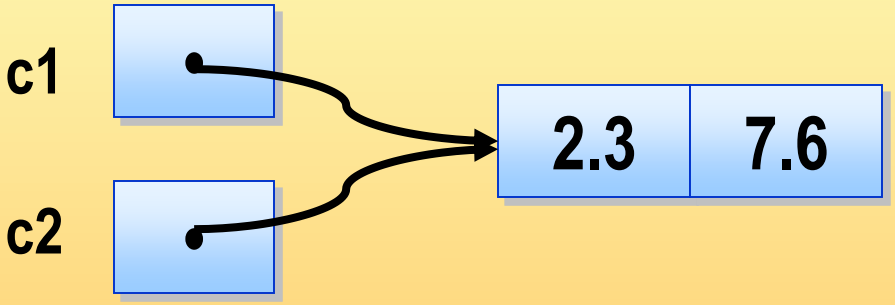
- **Сравнение размерных типов**
  - == и != сравнивают значения
- **Сравнение ссылочных типов**
  - == и != сравнивают ссылки, а не значения



Разные

Множество ссылок на один и тот же объект

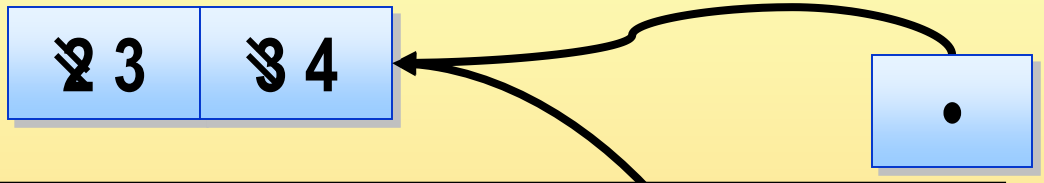
- **Две ссылки могут указывать на один объект**
  - Изменяя свойства объекта через одну ссылку, изменения будут доступны через другую




```
coordinate c1= new coordinate( );
coordinate c2;
c1.x = 2.3; c1.y = 7.6;
c2 = c1;
Console.WriteLine(c1.x + " , " + c1.y);
Console.WriteLine(c2.x + " , " + c2.y);
```

Ссылки как параметры в методах

- **Ссылки можно использовать для передачи параметров в методы**
  - При передаче по значению, данные могут измениться




```
static void PassCoordinateByValue(coordinate c)
{
  c.x++; c.y++;
}
```

```
loc.x = 2; loc.y = 3;
PassCoordinateByValue(loc);
Console.WriteLine(loc.x + " , " + loc.y);
```

Использование стандартных ссылочных типов

- Класс Exception
- Класс string
- Методы и свойства класса string
- Сравнение строк
- Операторы сравнения строк

Класс Exception

```
catch (InvalidTimeException caught)
{
        Console.WriteLine(caught);
}
```

- **Exception – это класс**
- **Объекты класса Exception используются для выбрасывания исключений**
  - Объект **Exception** создается с помощью ключевого слова **new**
  - Исключение выбрасывается с помощью ключевого слова **throw**
- **Все классы, описывающие исключения, являются производными от класса Exception**

```
if (minute < 1 || minute >= 60) {
  throw new InvalidTimeException(minute + " is not a valid minute");
```

Класс string

- Множество символов Unicode
- Псевдоним для класса System.String
- Неизменяемые

```
string s = "Hello";
char [] a={'l', 'i', 'm', 't', 'u'};
string v = new string (a); // v = limtu
string vv = new string(a, 0, 2); // vv = li
string xxx = new string('x', 3);
s[0] = 'c'; // Compile-time error
```

Методы и свойства класса string

- Метод Insert
- Свойство Length
- Метод Copy
- Метод Concat
- Метод Trim
- Методы ToUpper и ToLower
- Метод Join
- Метод Split

Сравнение строк

- **Метод Equals**
  - Сравнение значений
- **Метод Compare**
  - Лексическое сравнение строк
  - Параметр, определяющий учет регистра

Операции над строками

Над строками определены следующие операции:

- **присваивание (=):**
  - копирует одну строку в другую, т.е.строки ведут себя как значимые типы, хоть и реализованы как ссылочные типы
- **две операции проверки эквивалентности (==) и (!=);**
- **конкатенация или сцепление строк (+):**
  - сцепляет две строки, приписывая вторую строку к хвосту первой.
- **взятие индекса ([ ]).**
  - строку можно рассматривать как массив символов

Операторы сравнения строк

- **Операторы == и != перегружены для строк**
  - сравнивают значения строк, а не ссылки, т.е. эти операции выполняются как над значимыми типами.
- **Они равносильны методам String.Equals и !String.Equals**

```
string a = "Test";
string b = "Test";
if (a == b) ...   // Returns true
```

Класс StringBuilder

- **Класс *StringBuilder* используется для создания динамических (изменяемых) строк.**
- **Объекты этого класса всегда объявляются с явным вызовом конструктора класса**

```
//создание пустой строки, размер по умолчанию
StringBuilder a = new StringBuilder();
//инициализация строки и выделение необходимой памяти
StringBuilder b = new StringBuilder("abcd");
//создание пустой строки и выделение памяти
StringBuilder c = new StringBuilder(100);
//инициализация строки и выделение памяти
StringBuilder d = new StringBuilder("abcd", 100);
//инициализация подстрокой "bcd", и выделение памяти
StringBuilder d = new StringBuilder("abcd", 1, 3,100);
```

Операции над строками

Над строками класса *StringBuilder* определены практически те же операции с той же семантикой, что и над строками класса *String*:

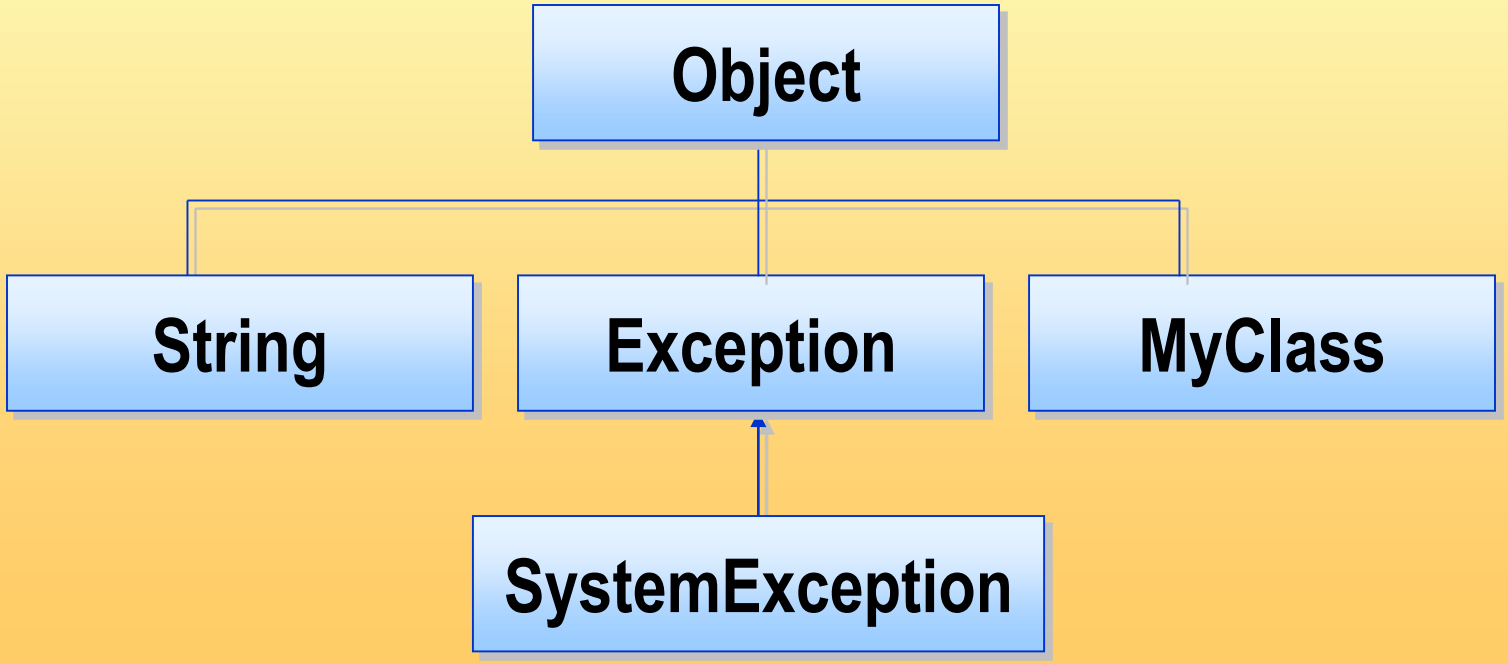
- **присваивание (=);**
- **две операции проверки эквивалентности (==) и (!=);**
- **взятие индекса ([ ])**
  - со строкой класса *StringBuilder* можно работать как с массивом: допускается не только чтение отдельного символа, но и его изменение
- **Операция конкатенации (+) не определена**
  - ее роль играет метод *Append*, дописывающий новую строку в хвост уже существующей

Иерархия объектов с единым корнем

- Тип object
- Общие методы
- Отражение

Тип object

- Синоним для System.Object
- Базовый класс для всех классов



Общие методы

- **Общие методы для всех ссылочных типов данных**
  - Метод **ToString**

    создает понятную для пользователя строку текста, в которой описывается экземпляр класса
  - Метод **Equals**

    поддерживает сравнение объектов
  - Метод **GetType**

    возвращает тип текущего экземпляра
  - Метод **Finalize**

    выполняет операции очистки перед автоматической утилизацией объекта

Отражение

- **Вы можете запросить информацию о типе объекта**
- **Пространство имен *System.Reflection***
- **Оператор *typeof* возвращает тип объекта**
  - Когда структура класса известна на этапе компиляции
- **Метод *GetType* для *System.Object***
  - Можно использовать в режиме реального времени для произвольных объектов

```
int i = 42;
System.Type type = i.GetType();
System.Console.WriteLine(type);
```

Лабораторная работа 7.1: Создание и использование ссылочных переменных

Дополнительно:

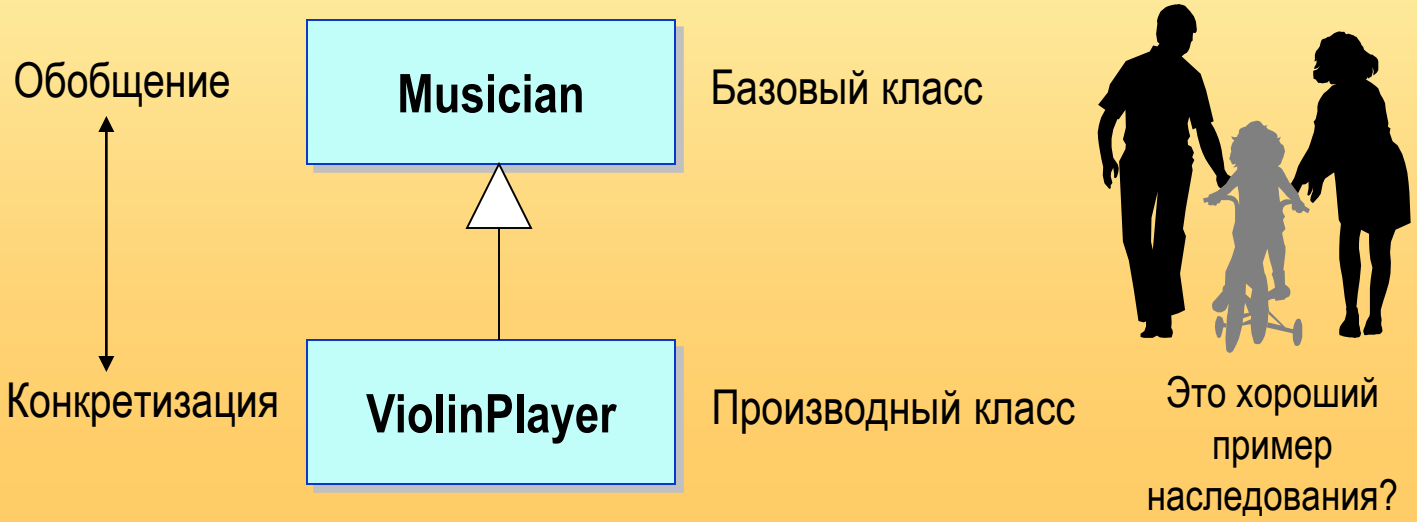
- **Разработать программу, которая для заданной строки s:**
  - вставляет символ a после каждого вхождения символа b;
  - определяет, какой из двух заданных символов встречается чаще в строке;
  - удаляет все символы e;

Разработка объектно-ориентированных систем

- Наследование
- Иерархия классов
- Единичное и множественное наследование
- Полиморфизм
- Абстрактные базовые классы
- Интерфейсы
- Раннее и позднее связывание

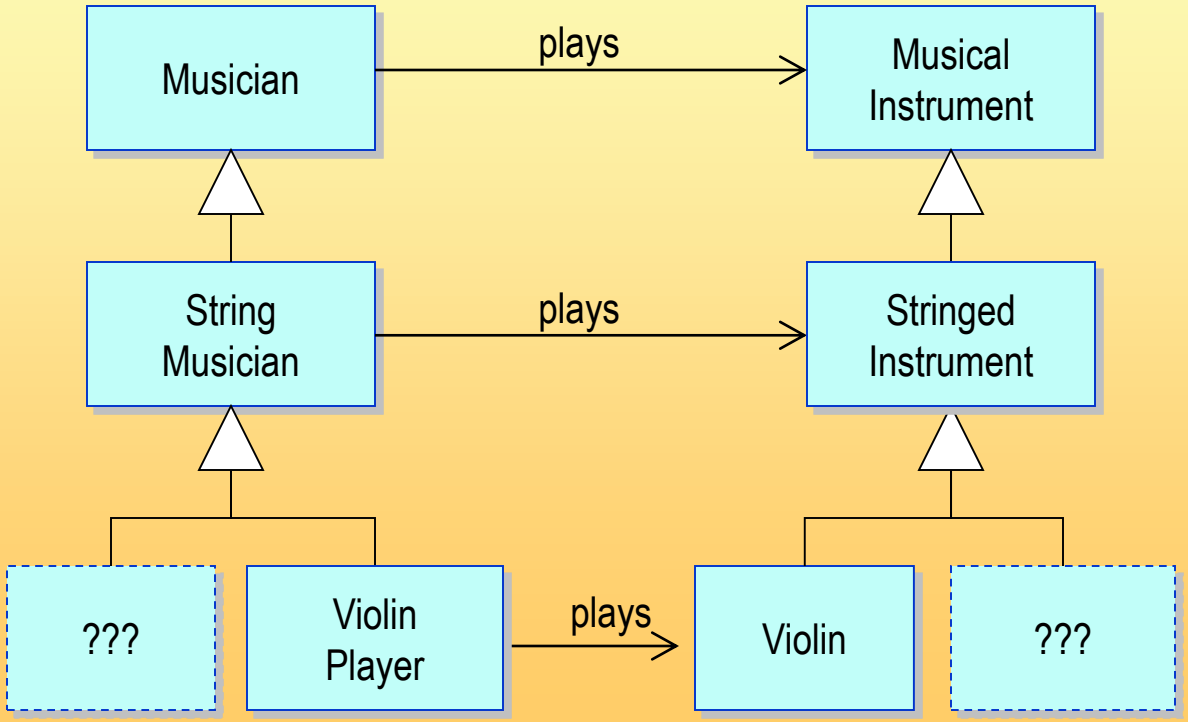
Наследование

- **Наследование определяет отношение “is a kind of"**
  - Наследование применяется к классам
  - Новые классы конкретизируют существующие



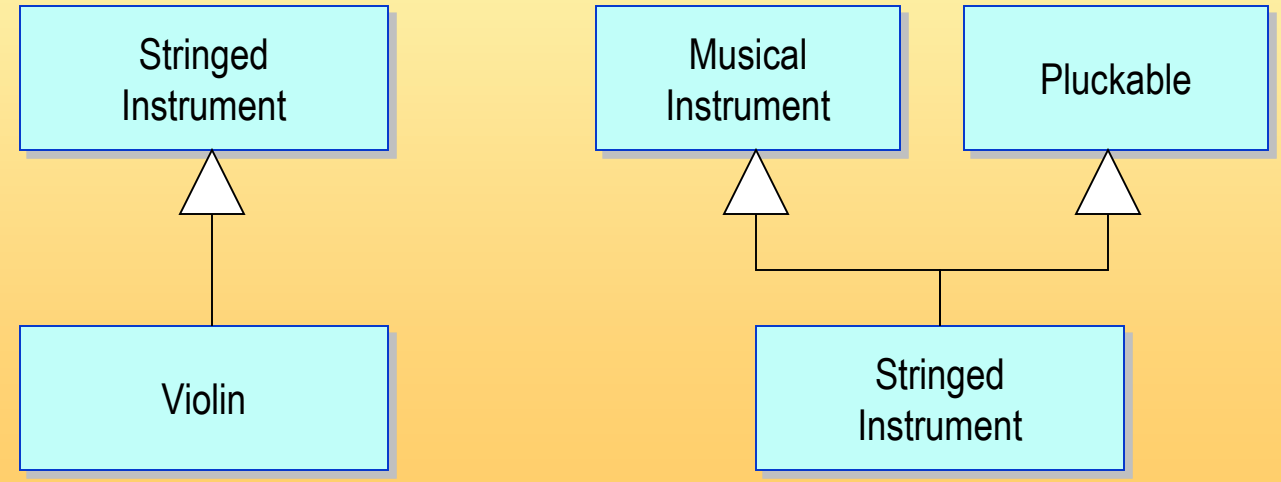
Иерархии классов

- **Классы, полученные через наследование, образуют классовые иерархии**



Единичное и множественное наследование

- **Единичное наследование: наследование от одного базового класса**
- **Множественное наследование: наследование от двух или более базовых классов**

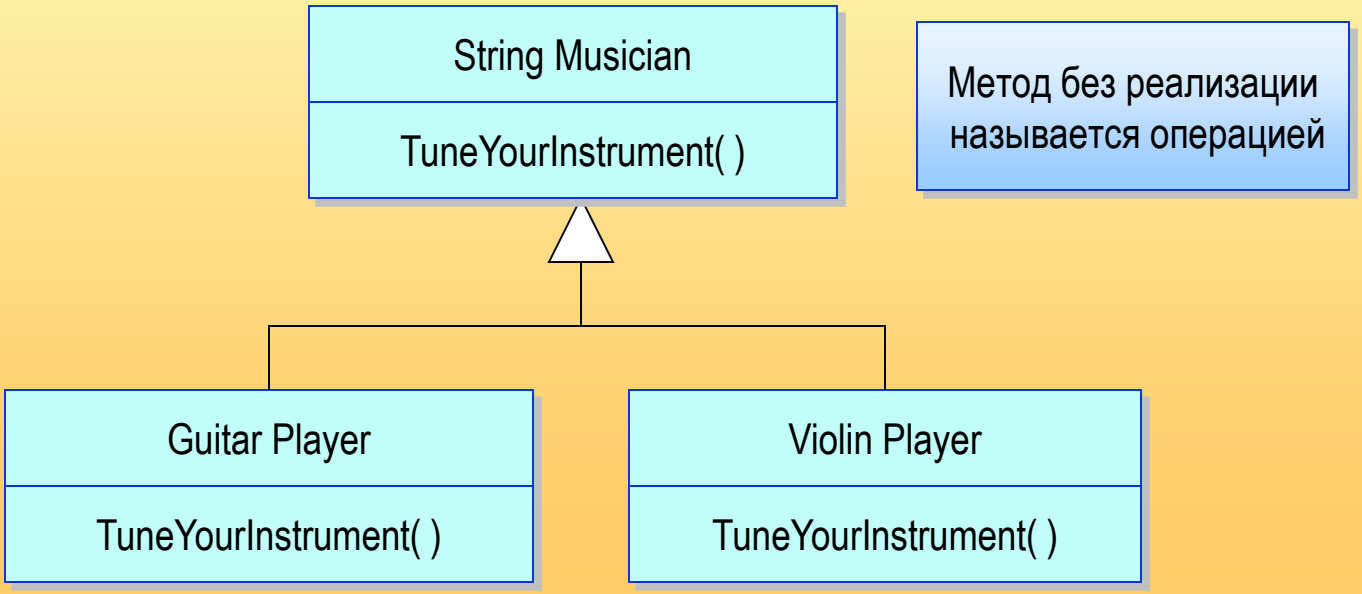


У класса **Violin** один базовый класс

У класса **StringedInstrument** два базовых класса

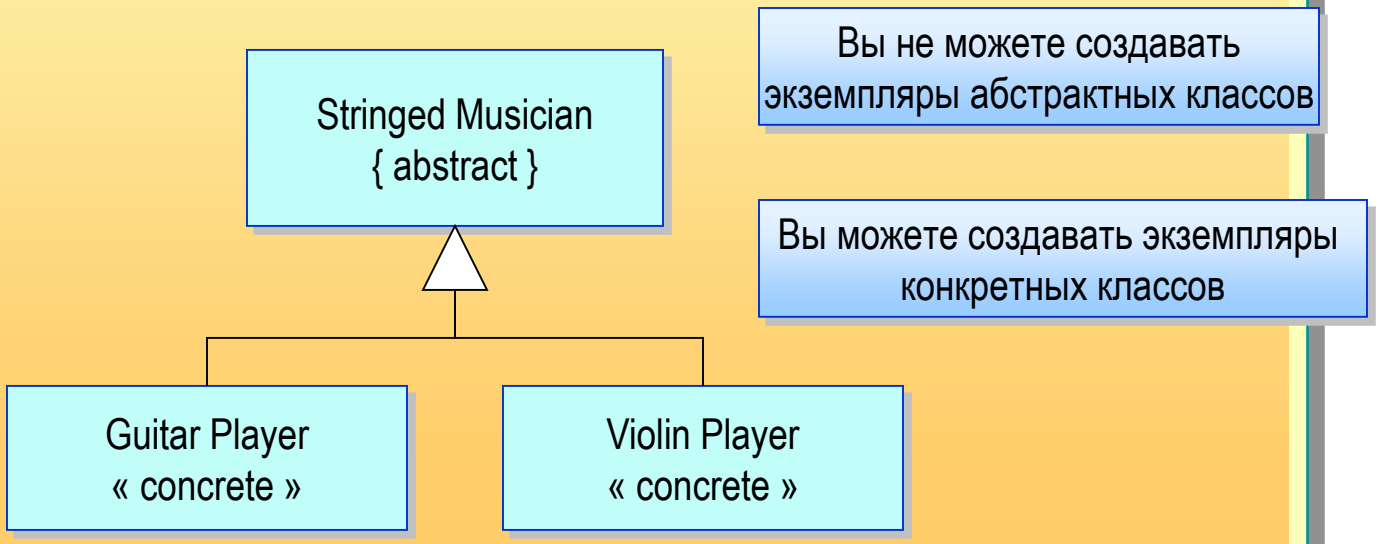
Полиморфизм

- **Имя метода определяется в базовом классе**
- **Реализация метода определяется в производном классе**



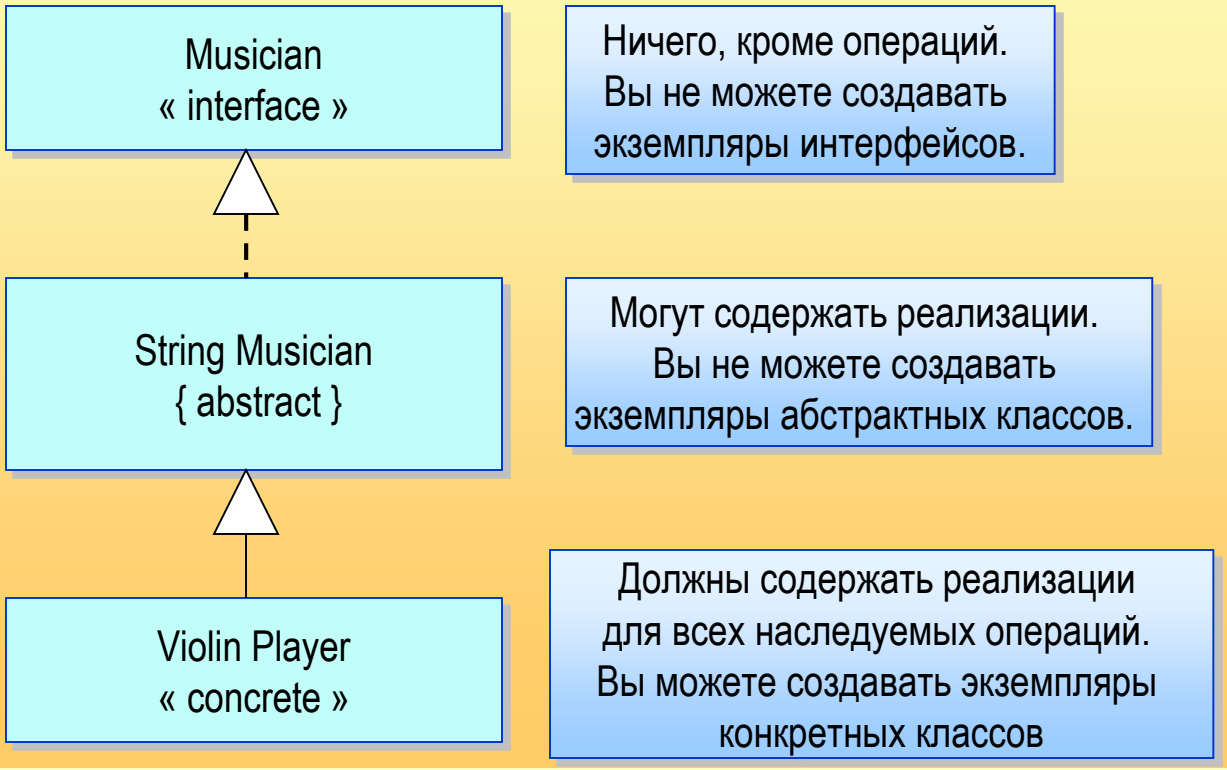
Абстрактные базовые классы

- **Некоторые классы нужны лишь для того, чтобы от них наследовать**
  - Нет смысла создавать экземпляры таких классов
  - Эти классы *абстрактны*



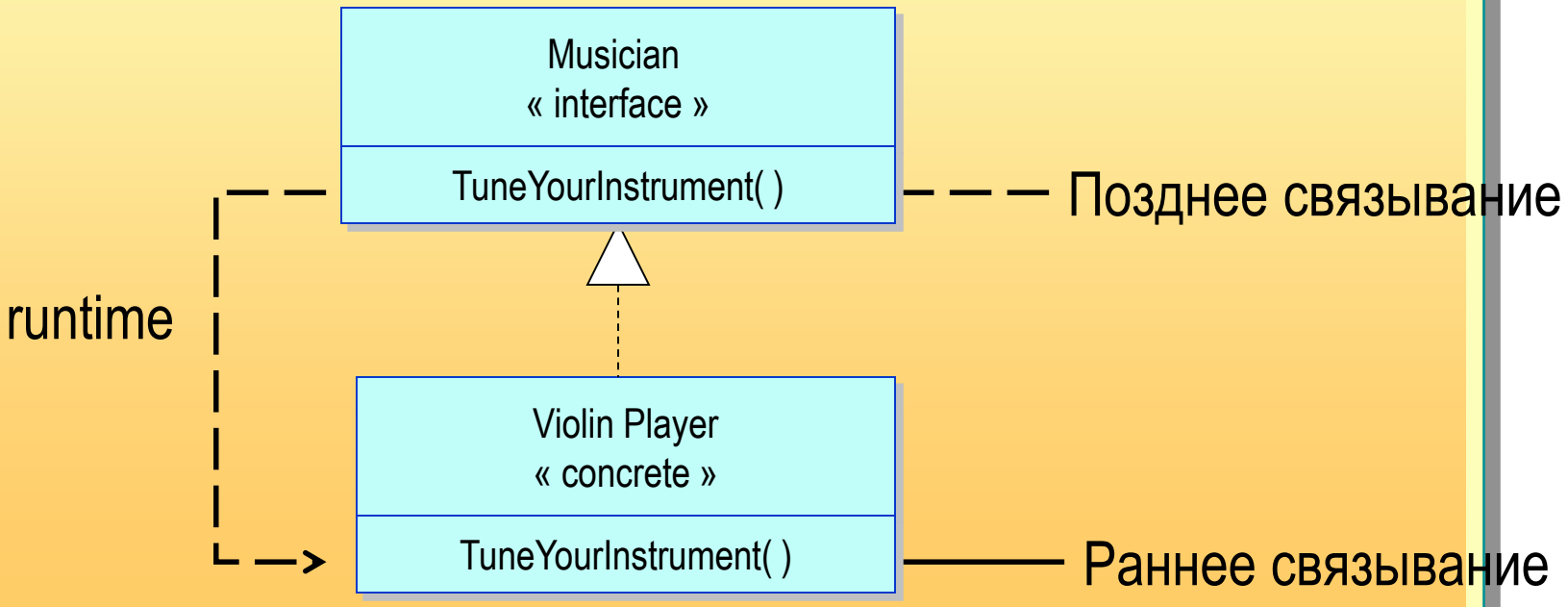
Интерфейсы

- **Интерфейсы содержат лишь операции, без реализаций**



Раннее и позднее связывания

- **Вызовы обычных методов обрабатываются на этапе компиляции**
- **Вызовы полиморфных методов обрабатываются на этапе выполнения**



Приведение типов

- Приведение размерных типов
- Преобразование к базовому/производному классу
- Оператор is
- Оператор as
- Преобразования и тип object
- Преобразования и интерфейсы
- Упаковка и распаковка

Приведение размерных типов

- **Неявное приведение**
- **Явное приведение**
  - Оператор приведения типов

Преобразование к базовому/производному классу

- **Преобразование к базовому классу**
  - Неявное или явное
  - Работает всегда
  - Ссылке на тип object можно присвоить любой объект
- **Преобразование к производному классу**
  - Необходимо явно указать тип, к которому необходимо преобразовывать
  - Происходит проверка правильности явного преобразования
  - В случае неправильного преобразования будет выброшено исключение **InvalidCastException**

Преобразование к базовому/производному классу

```
class Child:Parent
 {...}
 Parent p1 = new Parent(), p2 = new Parent();
Child ch1 = new Child(), ch2 = new Child();
```

- **Преобразование к базовому классу**

допустимые:

```
p1 = p2; p2= p1; ch1=ch2; ch2 = ch1; p1 = ch1; p1 = ch2;
```

- **Преобразование к производному классу**

недопустимые:

```
ch1 = p1; ch2 = p1; ch2 = p2; ch1 = p2;
```

допустимые:

```
p1 = ch1; ch1 = (Child)p1;
```

Оператор is

- Возвращает значение true, если преобразование возможно

```
expression is type
```

```
Bird b;
if (a is Bird)
    b = (Bird) a; // Safe
else
    Console.WriteLine("Not a Bird");
```

- С помощью оператора is можно проверить, совместим ли рассматриваемый объект с определенным типом

Оператор as

- **Проводит преобразование**

```
expression as type
```

  - равносильно:

```
expression is type ? (type)expression : (type)null
```

- **При возникновении ошибки**
  - Возвращает null
  - Не выбрасывает исключение

```
Bird b = a as Bird; // Convert

if (b == null)
      Console.WriteLine("Not a bird");
```

Преобразования и тип object

- **Тип object является базовым для всех типов**
- **Ссылке на тип object можно присвоить любой объект**
- **Любому объекту можно присвоить ссылку на тип object**
  - С явным указанием типа, к которому необходимо преобразовывать, и выполнением необходимых проверок
- **Тип object и оператор is**

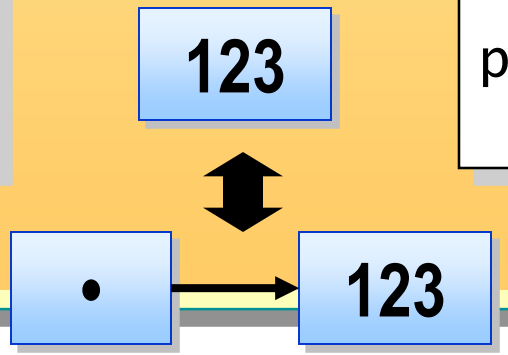
```
object ox;
ox = a;
ox = (object) a;
ox = a as object;
```

```
b = (Bird) ox;
b = ox as Bird;
```

Упаковка и распаковка

- Унифицированная система типов
- Упаковка
- Распаковка
- Вызов методов объекта для размерных типов

```
int p = 123;
object box;
box = p;
```




```
p = (int)box;
```

Обобщенные типы (Generic Types)

- **Обобщения – часть системы типов .NET Framework, которая позволяет определять тип.**
- **Преимущества:**
  - производительность;

    вместо использования объектов можно использовать класс List<T> из пространства имен System.Collection.Generic, который позволяет определить тип элемента при создании коллекции.

```
List<int> list = new List<int>();
list.Add(44);//нет упаковки – элементы
                    //сохраняются в List<int>
int il=list[0];// распаковка не нужна
foreach (int i2 in list)
{     Console.WriteLine(i2);
      }
```

- **Преимущества:**
  - безопасность типов:

    например, когда в классе ArrayList сохраняются объекты, то в коллекцию могут быть вставлены объекты различных типов.

```
ArrayList list = new ArrayList();
list.Add(44); // вставка целого
list.Add("mystring"); // вставка строки
list.Add(new MyClass ()) ; // вставка объекта
```

```
foreach (int i in list) {
Console.WriteLine(i);
}// возникнет ошибка во время выполнения!!!
```

```
List<int> list = new List<int>();
list.Add(44) ;
list.Add("mystring");   // ошибка компиляции
list.Add(new MyClass());   // ошибка компиляции
```

Создание обобщений

```
class Gen<T, U>
{
public T t;
public U u;
public Gen(T _t, U _u)
      {
            t = _t;
            u = _u;
      }
}
```

- **Класс Gen имеет два члена типа *T* и *U*.**
  - Код, использующий этот класс, определит типы для *T* и *U*. В зависимости от того, как класс Gen используется в коде, могут иметь тип string, int, пользовательский тип или другую их комбинацию.

Применение обобщений

- **Для использования обобщений нужно указать его тип.**

```
// Add two strings using the Gen class

Gen<string, string> ga = new Gen<string, string>("Hello, ", "World!");
Console.WriteLine(ga.t + ga.u);

// Add a double and an int using the Gen class

Gen<double, int> gb = new Gen<double, int>(10.125, 2005);
Console.WriteLine(gb.t + gb.u);
```

Использование ограничений

- **Ограничения — позволяют определить требования к типам, которыми разрешено заменять обобщения в коде.**
- **Обобщения поддерживают четыре типа ограничений:**
  - По интерфейсу.
  - По базовому классу.
  - По конструктору.
  - По ссылочному или значимому типу.
- **Для применения ограничений к обобщению используется секция `where`.**

```
// класс обобщения может использоваться только типами,
// реализующими интерфейс IComparable
class CompGen<T>
where T : IComparable
{
public T t1;
public T t2;
public CompGen(T _t1, T _t2)
      {
            t1 = _t1;
            t2 = _t2;
      }
public T Max()
            {
      if (t2.CompareTo(t1) < 0)
            return t1;
      else
            return t2;
            }
}
```

Лабораторная работа 7.2: Приведение типов