Лекция 5. Переопределение Equals, GetHashCode, ToString

Методы класса Object

- Как мы помним, все типы данных в C# в конечном итоге наследуются от Object (но обычно принято использовать ключевое слово object), а значит, получают его методы

- Некоторые методы:

```
public bool Equals(object obj)        // сравнение
public int GetHashCode()              // хэш-код
public string ToString()              // преобразование в строку
```

Метод ToString

Метод ToString

- Метод **ToString**() предназначен для преобразования объекта в строку
- По умолчанию для наших классов этот метод работает так:
- MyClass o1 = new MyClass();
  Console.WriteLine(o1.ToString()); // MyProject.MyClass
- Т.е. получается строка с полным именем класса (с **namespace**'ом)

Метод ToString

- Чтобы метод **ToString** выводил что-то более осмысленное, его можно переопределить самим любым нужным нам образом
- Например, вывести поля объекта

- У многих стандартных классов **ToString** уже переопределен хорошим образом
- Но у массивов он не переопределен:
- ```
  int[] a = new int[10];
  Console.WriteLine(a.ToString());  // System.Int32[]
  ```
  ```

Метод ToString

- Метод **ToString**() можно определить как хочется
-
```
public class Vector {
    private int[] vector;

    public Vector(int[] vector) { this.vector = vector;  }

    public override string ToString() {
        return string.Join(“, ”, vector);
    }
}
```
-
```
int[] a = { 1, 2, 3, 4 };
Vector v = new Vector(a);
Console.WriteLine(v);   // 1, 2, 3, 4
```

string.Join – метод для преобразования массива в строку с использованием разделителя

Метод ToString

- Пусть у нас есть такой объект:
  - `Vector v = new Vector(new int[] { 1, 3, 5 });`
- Метод **ToString** можно вызвать самим явно:
  - `Console.WriteLine(v.ToString()); // 1, 3, 5`
- Также некоторые стандартные функции сами вызывают **ToString**:
  - `Console.WriteLine(v); // 1, 3, 5`
- Метод **ToString**() вызывается автоматически при конкатенации строки и объекта:
  - `Console.WriteLine("vector = " + v);`
    `// vector = 1, 3, 5`

Объект преобразовался в строку, строки конкатенируются

Сравнение ссылок. Метод Equals

Операторы сравнения для объектов

- Как мы помним, в C# операторы == и != для объектов сравнивают ссылки, а не проверяют равенство объектов, **если эти операторы не были переопределены**
- То есть в C#
  объект a == объект b
  тогда и только тогда, когда это один и тот же объект в памяти
- При этом не важен тип ссылки, важно что ссылки указывают на один и тот же объект:

```
A a = new A();
object b = a;
Console.WriteLine(a == b); // true, это тот же объект
```

- Для многих стандартных типов, например, string, операторы == и != определены правильно

Метод Equals

- Метод **Equals**(object o) предназначен для проверки на равенство содержимого объектов
- ```
  string a = “123”;
  string b =  Console.ReadLine();          // пусть вводят “123”
  Console.WriteLine(a.Equals(b));          // true
  ```
  ```

Метод Equals в классе object

- По умолчанию, в классе object, метод **Equals**(object o) просто проверяет равенство ссылок при помощи ==
- Многие стандартные классы, такие как string, переопределяют метод **Equals**, чтобы он сравнивал содержимое объектов
- Поэтому в своих классах, если мы хотим сравнивать их объекты, нужно переопределить метод **Equals**

Как переопределять метод equals?

```
public class Pair
{
    private int first;
    private int second;

    public Pair(int first, int second)
    {
        this.first = first;
        this.second = second;
    }

    public override bool Equals(object o)
    {
        // реализация
    }
}
```

Через равенство классов

- ```
  public class Pair {
    private int first;
    private int second;

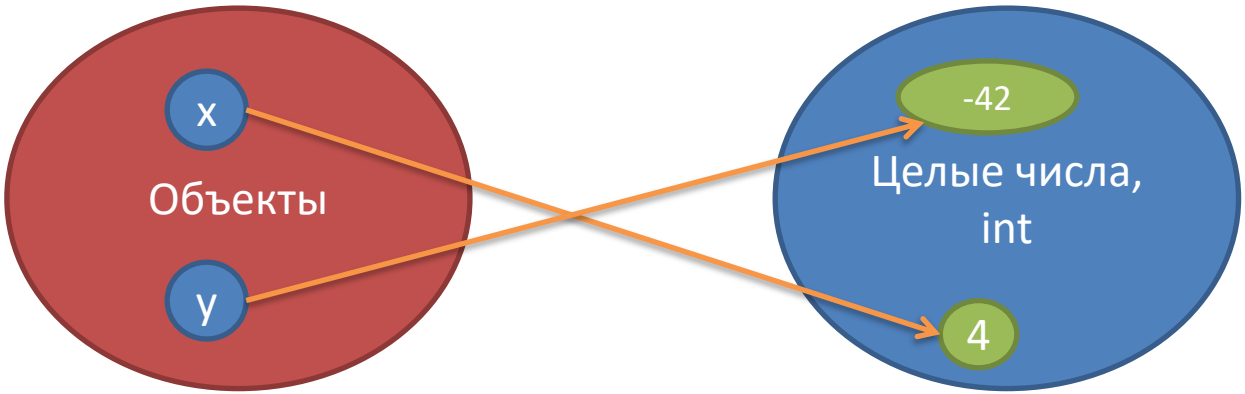
    public override bool Equals(object o) {
        // проверили что передали сам объект
        if (ReferenceEquals(o, this)) return true;
        // отсеяли null и объекты других классов
        if (ReferenceEquals(o, null) || o.GetType() != GetType())
  return false;
        // привели объект к Pair
        Pair p = (Pair) o;
        // проверили равенство ссылок и полей
        return first == p.first && second == p.second;
    }
  }
  ```

Возможно, не все поля должны участвовать в Equals, например, если некоторые поля являются вспомогательными
  ```

Хэш-функция. hashCode

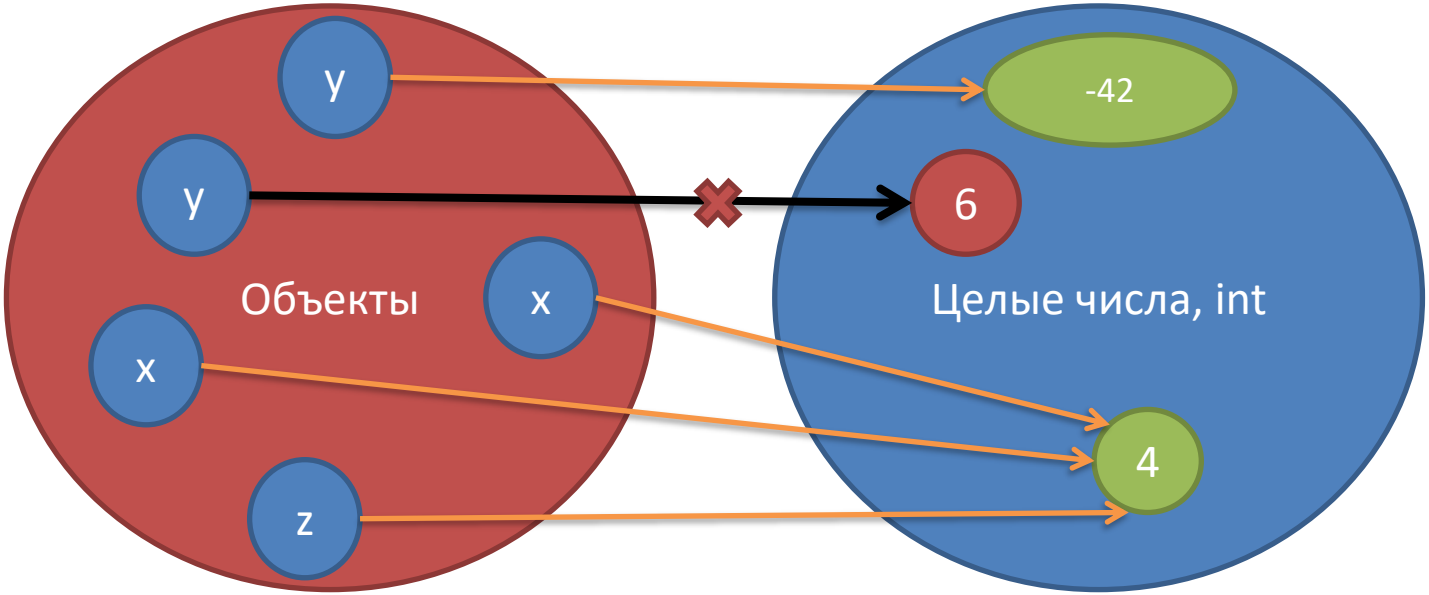
Хэш-функция

- **Хэш-функция** – это функция, которая принимает объект и, используя данные объекта, вычисляет целое число, и обладает некоторыми свойствами
(см. через несколько слайдов)
- Результат применения хэш-функции к объекту называется **хэш-код** или просто **хэш**
- В C# за вычисление хэш-функции отвечает метод **GetHashCode**()



Свойства хэш-функции

- Хэш функция должна быть связана со сравнением объектов следующим образом:
  - если объекты равны, то их хэши должны быть равны
  - если объекты не равны, то их хэши желательно должны быть разными, но это не обязательно



Хэш-функция

- Пример хэш-функции
-
```
public class A {
    private int a;

    public override int GetHashCode() {
        return a;    // все формальности соблюдены
    }                // это int, для равных объектов он совпадает

    public override bool Equals(object obj) {
        if (ReferenceEquals(obj, this)) return true;
        if (ReferenceEquals(obj, null) || GetType() != obj.GetType())
            return false;
        return (a == ((A)obj).a);
    }
}
```

Свойства хэш-функции

- Хэш-функция должна быть связана со сравнением объектов следующим образом:
  - если объекты равны, то их хэши должны быть равны
  - если объекты не равны, то их хэши желательно должны быть разными, но это не обязательно
  - если объект не менялся, то хэш не должен меняться
- Получается следующее:
  - если хэши разные, то объекты точно разные
  - если хэши равны, то не факт, что равны объекты, надо проверять дальше при помощи сравнения
- Так как хэш-функция связана со сравнением, то методы **GetHashCode** и **Equals** всегда надо переопределять вместе

Хэш-функция

Методы GetHashCode и Equals оба переопределены. Свойства выполняются

```
public class A {
    private int a;

    public override int GetHashCode() {
        return a;    // все формальности соблюдены
    }                // это int, для равных объектов он совпадает

    public override bool Equals(object obj) {
        if (ReferenceEquals(obj, this)) return true;
        if (ReferenceEquals(obj, null) || GetType() != obj.GetType())
            return false;
        return (a == ((A)obj).a);
    }
}
```

Как написать hashCode?

```
public class A {
    private int a;
    private double b;
    private B c;

    public override int GetHashCode() {
        int prime = 37;
        int hash = 1;
        hash = prime * hash + a;
        hash = prime * hash + b.GetHashCode();
        hash = prime * hash + (c != null ? c.GetHashCode() : 0);
        return hash;
    }
}
```

В качестве начального значения берут число != 0

Потом поле за полем, делают
hash = prime * hash + (хэш от этого поля)

prime – некоторое нечетное простое число, например, 37

Зачем нужна хэш-функция?

- Позволяет быстро определить, что объекты не равны.
  Если хэши разные, то объекты – разные.
  - Но если хэши совпали, то надо проверять, что объекты равны
  - Хэш-функция обычно вычисляется быстро, и это дешевле, чем выполнить полное сравнение
- На хэш-функции основана структура данных **хэш-таблица**, которая позволяет осуществлять быстрый поиск
- Хэш-таблица будет рассмотрена в следующих лекциях

Методы Equals, GetHashCode для структур

- У структур автоматически правильно определены методы **Equals** и **GetHashCode**
- Но при этом они будут работать медленно (т.к. стандартная реализация через **Reflection**), поэтому если эти методы нужны, их следует переопределить самим

Records

Records

- Начиная с C# 9 появился новый вид типов данных – **records**
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record
- **Record** позволяет кратко создать тип (класс или структуру), в котором сразу будут реализованы конструктор и свойства (при необходимости), а также **Equals**, **GetHashCode**, **ToString**, операторы **==** и **!=, Equals**, который принимает не object, а тип этого **record**'а, и некоторые другие вещи
- В основном **records** используются для передачи данных (т.е. чтобы сделать тип с полями без дополнительной логики)

Records классы и структуры

- В C# 9 изначально можно было создавать только record-классы (т.е. они были ссылочными типами):
- public record Person(string FirstName, string LastName);

- Начиная с C# 10 появилась возможность создавать record-структуры:
- public record struct Person(string FirstName, string LastName);

- А для классов теперь можно указать так:
- public record class Person(string FirstName, string LastName);

Изменяемость свойств

- public record Person(string FirstName, string LastName);
- В таком кратком варианте (он называется **позиционный синтаксис**) сразу создается конструктор, свойства и др.
- При этом если это record-класс, то свойства будут неизменяемые, а если record-структура, то изменяемые
- Чтобы сделать свойства record-структур неизменяемыми, нужно указать модификатор readonly:
- public readonly struct record Person(string FirstName, string LastName);

Records

- Для records можно не указывать круглые скобки, а объявить свойства самим
- public record Person
{
    public string FirstName { get; init; } = default!;
    public string LastName { get; init; } = default!;
}

default! используется, чтобы не было warning'а

- Здесь для сеттеров используется модификатор init, чтобы свойства были неизменяемыми:
- https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/proposals/csharp-9.0/init
- Но можно использовать set;
  - Тогда свойство будет изменяемым

Оператор with

- Оператор with позволяет создать копию record'а, но с измененными значениями нужных свойств
- Это удобно, т.к. record'ы часто неизменяемые, и иначе пришлось бы копировать все свойства и только менять нужные
- Person p1 = new Person("Ivan", "Ivanov");
  Person p2 = p1 with { FirstName = "Petr" };
- Также можно создать копию:
- Person p3 = p1 with {};
- Делается неглубокое копирование свойств

Наследование

- Record-классы (не структуры) могут наследоваться друг от друга
- В этом случае есть довольно много специфичного синтаксиса и деталей
- Если понадобится, см. в документации:
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record#inheritance