Классы
Абстрактные типы данных

Лекции по курсу

Языки программирования

семестр 2

Определение класса

```
public class Person
{
    private string Name;
    private int Age;
    public Person(string Name, int Age)
    {
        this.Name = Name;
        this.Age = Age;
    }
}
```

поля должны быть приватными

конструктор должен быть публичным

- Как осуществлять **доступ** к полям вне класса???

Свойства классов

Доступ к полям осуществляется с помощью **свойств**. Свойства – это "интеллектуальные поля", в которых разделяется доступ на чтение (getter) и доступ на запись (setter).

```
public class Person
{
    private string name;
    private int age;
    public Person(string Name, int Age)
    {
        this.Name = Name;
        this.Age = Age;
    }
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    public int Age
    {
        get { return age; }
        set {
            if (value >= 0)
               age = value;
            else age = 0;
        }
    }
}
...
var p = new Person("Иванов",18);
p.Age = -100; // p.Age == 0
```

можно делать дополнительные проверки

Автоматически реализуемые свойства

- Автосвойства используются если не требуется если не требуется сложной логики изменения свойств
- Для автосвойства генерируется приватное поле

```
public class Person
{
    public string Name { get; };
    public int Age { get; private set; };
    public Person(string Name, int Age)
    {
      this.Name = Name; // можно менять только в конструкторе
      this.Age = Age;
    }
    public void IncAge()
    {
      Age += 1; // можно менять во всех методах
    }
}
...
var p = new Person("Иванов",18);
p.Name = "Петров"; // нельзя!
p.Age = 100; // нельзя!
p.IncAge(); // только так
```

Классы и структуры

- Классы – ссылочный тип данных, структуры – размерный

```
public class Person
{
  ...
  public Person(string Name, int Age)
  {
    this.Name = Name;
    this.Age = Age;
  }
}

// В динамической памяти
var p = new Person("Иванов",18);

var p1 = p; // Присваивание ссылок

if (p1==p) ... // Сравнение ссылок

p = null; // сборка мусора

// передача ссылки по значению
void Print(Person p) {}
```

```
public struct Person
{
  ...
  public Person(string Name, int Age)
  {
    this.Name = Name;
    this.Age = Age;
  }
}

// На стеке (new обманчиво)
var p = new Person("Иванов",18);

var p1 = p; // Присваивание бруска

if (p1==p) ... // Ошибка!

// нет сборки мусора

// передача по ссылке
void Print(ref Person p) {}
```

Операции ?. и ??

- Решают проблему некорректного использования нулевой ссылки (частично)

```
Person p = null;
p.IncAge(); // тадам!

Необработанное исключение: System.NullReferenceException: Ссылка на объект не
указывает на экземпляр объекта.

p?.IncAge(); // OK – ничего не происходит

string s = p?.Name; // s == null

int? a = p?.Age; // int? – Nullable-тип; a == null
// int? типы могут принимать все значения
// базового размерного типа и null

int aa = 0;
if (a.HasValue)
  aa = a.Value; // или aa = (int)a;

int aaa = p?.Age ?? 0;
string ss = p?.Name ?? "Безымянный";
```

Класс как конкретный и абстрактный тип данных

Тип данных характеризуется:

- Хранение в памяти
- Множество значений
- Набор операций
- Реализация операций

Абстрактный тип данных (АТД) характеризуется:

- ~~Хранение в памяти~~
- Множество значений
- Набор операций
- ~~Реализация операций~~

Класс – можно рассматривать:

- и как конкретный тип данных (для того, кто реализует класс)
- и как абстрактный тип данных (АТД) – для тех, кто пользуется объектами класса (конструирует их, вызывает операции, методы)

Модификаторы защиты доступа позволяют скрыть доступ к внутреннему представлению, которое не хочет открывать разработчик класса

Помещение класса в откомпилированную библиотеку .dll позволяет скрыть реализацию (в некотором смысле)

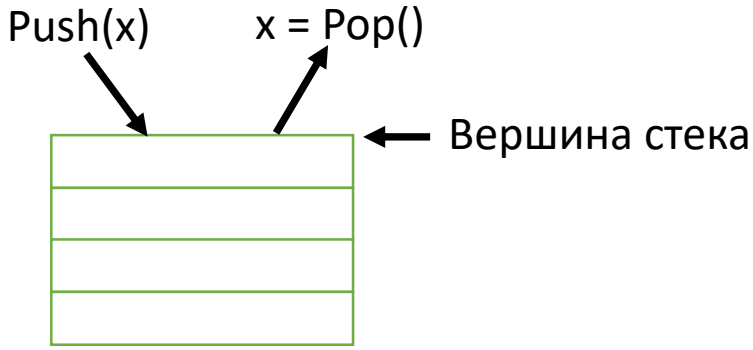
Принцип отделения интерфейса класса от реализации позволяет пользователю абстрагироваться от внутреннего устройства класса

АТД Стек и его интерфейс

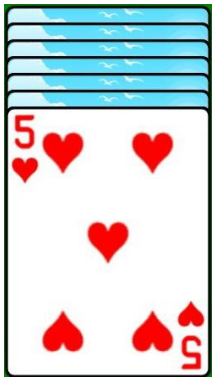
Стек (магазин) – набор данных, устроенный по принципу **LIFO:**
Last In – First Out (последним пришёл – первым ушёл)

Интерфейс класса Стек:

```
class Stack<T>
{
  public int Count { get; }
  public void Push(T item);
  public T Pop();
  public T Peek();
}
```



Примеры стеков:



Выражение в ПОЛИЗ

- ПОЛИЗ – польская бесскобочная инверсная запись выражения

Выражение в обычной записи

```
2 + 3 * 4
```

Выражение в ПОЛИЗ

```
2 3 4 * +
3 4 * 2 +
```

Выражение в обычной записи

```
(2 + 3) * 4
```

Выражение в ПОЛИЗ

```
2 3 + 4 *
```

Вычисление выражения в ПОЛИЗ с помощью стека

Выражение в обычной записи

```
(1 + 2) * 4 + 3
```

Выражение в ПОЛИЗ

```
1 2 + 4 * 3 +
```

Алгоритм

Пока не конец ввода
- Считываем лексему
- Если это число, то кладём на стек
- Если это – знак операции, то
  - снимаем со стека два последних операнда
  - применяем к ним указанную операцию
  - кладём результат на стек

Снимаем со стека результат
Assert(стек пуст)

Пример работы

Ввод	Операция	Стек
1	поместить в стек	1
2	поместить в стек	1, 2
+	сложение	3
4	поместить в стек	3, 4
*	умножение	12
3	поместить в стек	12, 3
+	сложение	15

Реализация алгоритма

```
var expr = "1 2 + 4 * 3 +";
var ss = expr.Split();
var st = new Stack<int>();

foreach (var x in ss)
    if (char.IsDigit(x[0]))
        st.Push(int.Parse(x));
    else if (x == "+")
        st.Push(st.Pop() + st.Pop());
    else if (x == "*")
        st.Push(st.Pop() * st.Pop());

WriteLine(st.Pop());

System.Diagnostics.Debug.Assert(st.Count == 0);
// срабатывает только в режиме Debug
```

Риторический вопрос

Мешало ли нам при написании алгоритма вычисления выражения в ПОЛИЗ то, что мы не знали внутреннее устройство стека?

Риторический вопрос

Мешало ли нам при написании алгоритма вычисления выражения в ПОЛИЗ то, что мы не знали внутреннее устройство стека?

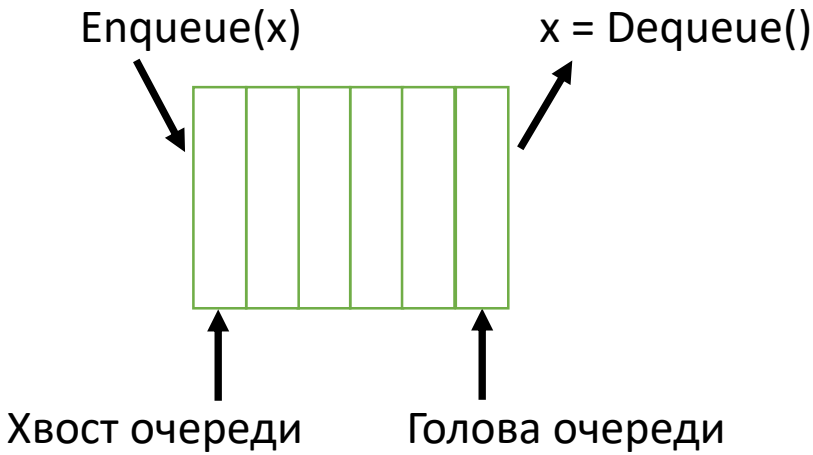
Правильный ответ: **помогало**

АТД Очередь и его интерфейс

Очередь – набор данных, устроенный по принципу **FIFO:**
First In – First Out (первым пришёл – первым ушёл)

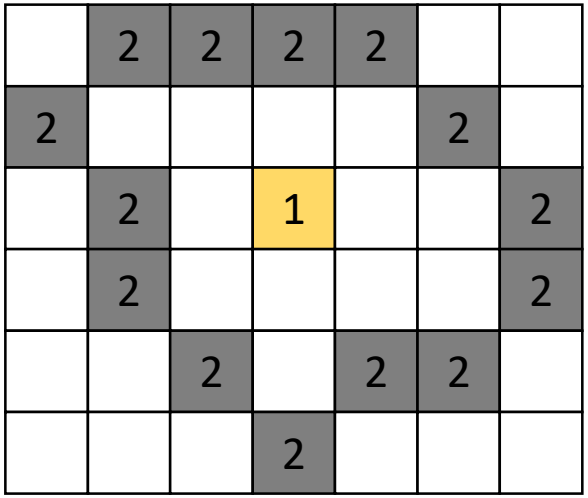
Интерфейс класса Очередь:

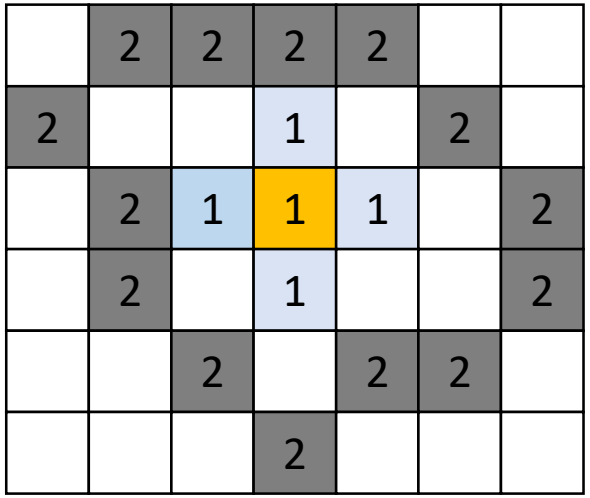
```
class Queue<T>
{
  public int Count { get; }
  public void Enqueue(T item);
  public T Dequeue();
  public T Peek();
}
```

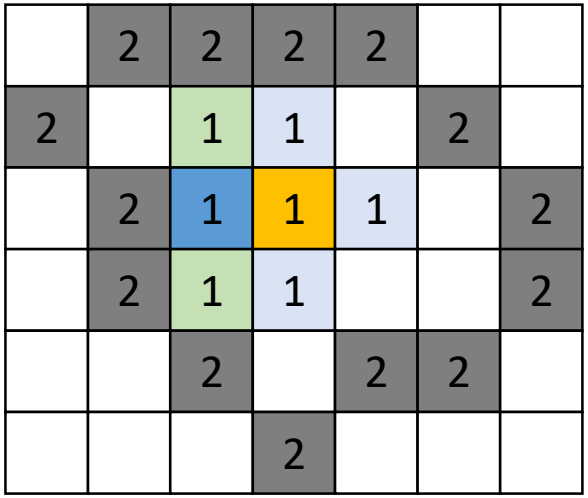


Примеры очередей:

Алгоритм FloodFill заливки области








```
a[y,x] = 0 – не залито
a[y,x] = 1 – залито
a[y,x] = 2 – цвет границы
```

 – начальная точка

Реализация алгоритма FloodFill

```
static void AddPixel(int[,] a, int x, int y, Queue<(int, int)> q)
{
  if (a[y,x] == 0)
  {
    a[y, x] = 1;
    q.Enqueue((x,y));
  }
}
static void FloodFill(int[,] a, int x, int y)
{
  var q = new Queue<(int,int)>();
  AddPixel(a, x, y, q);
  while (q.Count > 0)
  {
    (var xx, var yy) = q.Dequeue();
    AddPixel(a, xx-1, yy, q);
    AddPixel(a, xx+1, yy, q);
    AddPixel(a, xx, yy-1, q);
    AddPixel(a, xx, yy+1, q);
  }
}
```

Риторический вопрос

Мешало ли нам при написании алгоритма FloodFill то, что мы не знали внутреннее устройство очереди?

Стандартные классы коллекций .NET

Стандартные классы коллекций находятся в пространстве имён System.Collections.Generic

Нам известны следующие:

- Stack<T>
- Queue<T>
- List<T> - динамический массив с возможностью расширения
- HashSet<T> - несортированное множество (операции Add, Remove, Contains выполняются за время Θ(1))
- SortedSet<T> - сортированное множество (операции Add, Remove, Contains выполняются за время Θ(log(n)))

Пример создания и использования:

```
var hs = new HashSet<int> {1, 2, 5, 8, 3, 15 };
foreach (var x in hs)
  Write($"{x} ");
```

Q & A