Промышленное программирование

Лекция 5

❶ Исключения
❷ Сборщик мусора
❸ Паттерны Dispose

1. **Исключения**
2. Сборщик мусора
3. Паттерны Dispose

try-catch, try-catch-finally, try-finally

```
StreamWriter? writer = null;

try
{
    writer = new StreamWriter("output.txt");
    writer.WriteLine("hello");
}
catch (IOException e)
{
    Console.WriteLine(e.Message);
}
finally
{
    writer?.Close();
}
```

IDisposable и using

```
StreamWriter? writer = null;
try
{
    writer = new StreamWriter("output.txt");
    writer.WriteLine("hello");
}
finally
{
    writer?.Dispose();
}
```

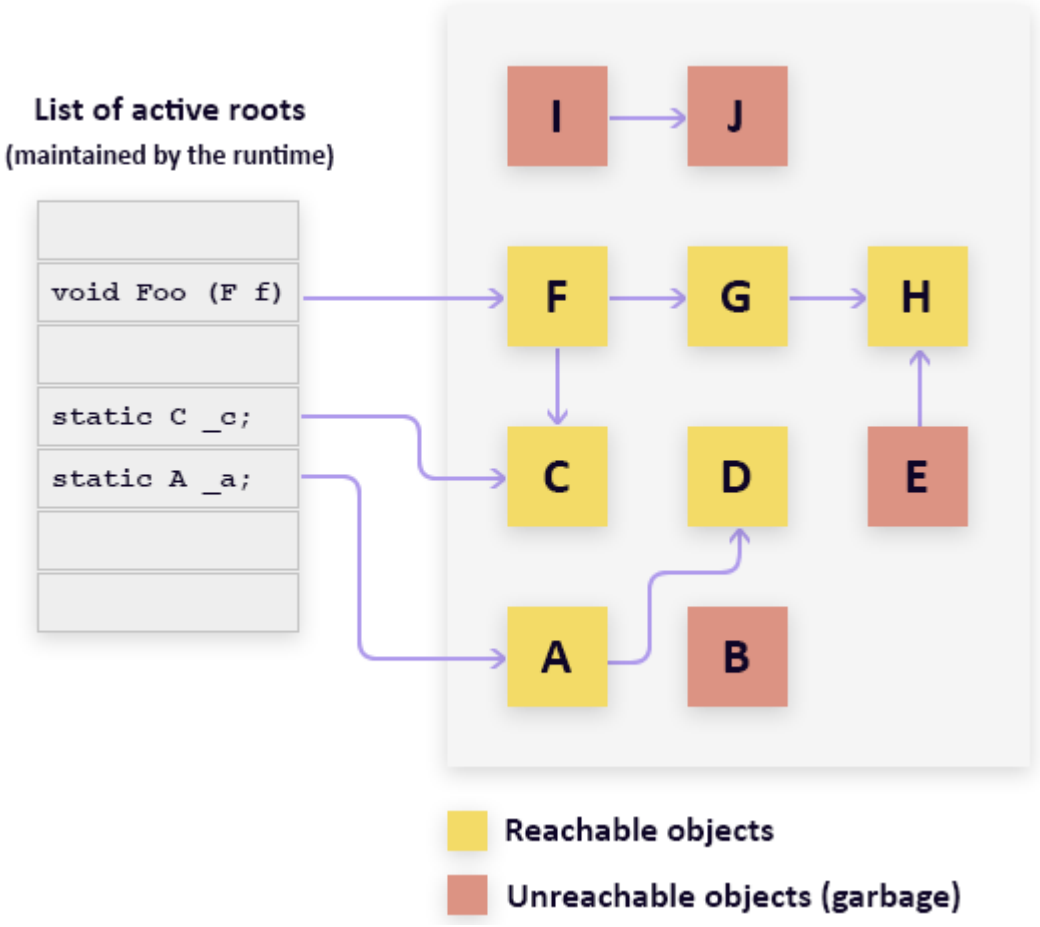
```
void Method()
{
    using (var writer = new StreamWriter("output.txt"))
    {
        writer.WriteLine("hello");
    }

    // writer здесь недоступен
}
```

```
namespace System
{
    public interface IDisposable
    {
        void Dispose();
    }
}
```

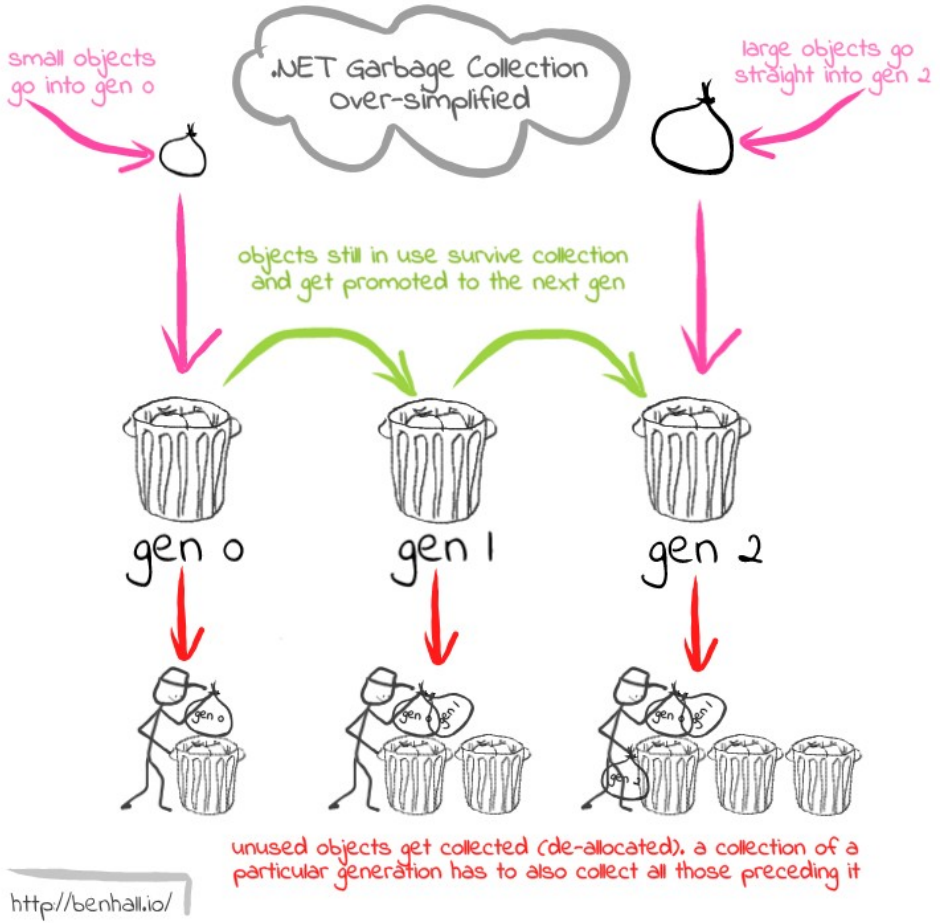
```
void Method()
{
    using var writer = new StreamWriter("output.txt");
    writer.WriteLine("hello");

    // writer доступен до конца секции
}
```

1. Исключения
2. **Сборщик мусора**
3. Паттерны Dispose

GC Roots


List of active roots
(maintained by the runtime)
void Foo (F f)
static C _c;
static A _a;
I
J
F
G
H
C
D
E
A
B
Reachable objects
Unreachable objects (garbage)

Логическая схема управляемой кучи



GC super-simplified animation

Пример

```
var small = new byte[50000];
var large = new int[50000];
Print(small, large);

GC.Collect();
Print(small, large);

GC.Collect();
Print(small, large);

GC.Collect();
Print(small, large);

static void Print(object a, object b)
{
    var ag = GC.GetGeneration(a);
    var bg = GC.GetGeneration(b);
    Console.WriteLine($"{ag} {bg}");
}
```

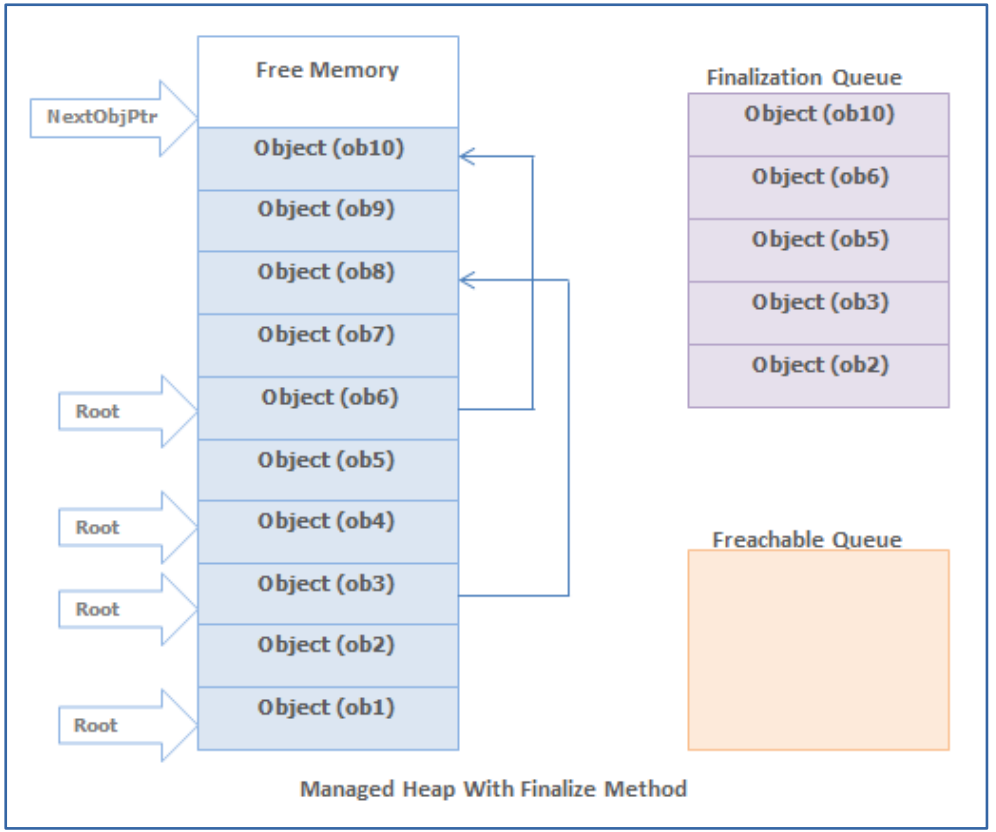
Слабые ссылки

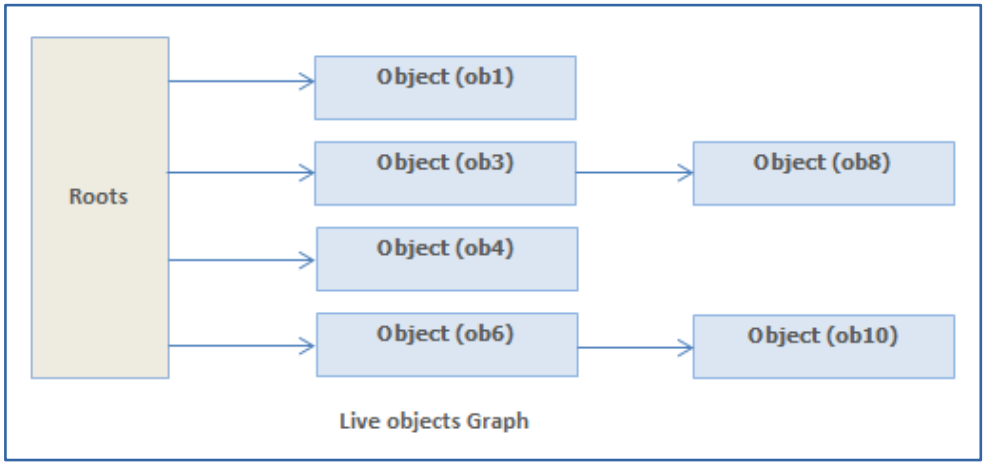
```
WeakReference<string> CreateWeak() {
    var strong = new string("hello");
    return new WeakReference<string>(strong);
}

void Test(WeakReference<string> weak) {
    if (weak.TryGetTarget(out var strong))
        Console.WriteLine(strong);
    else
        Console.WriteLine("collected");
}

void Main() {
    var weak = CreateWeak();
    Test(weak);
    GC.Collect();
    Test(weak);
}
```

Финализаторы

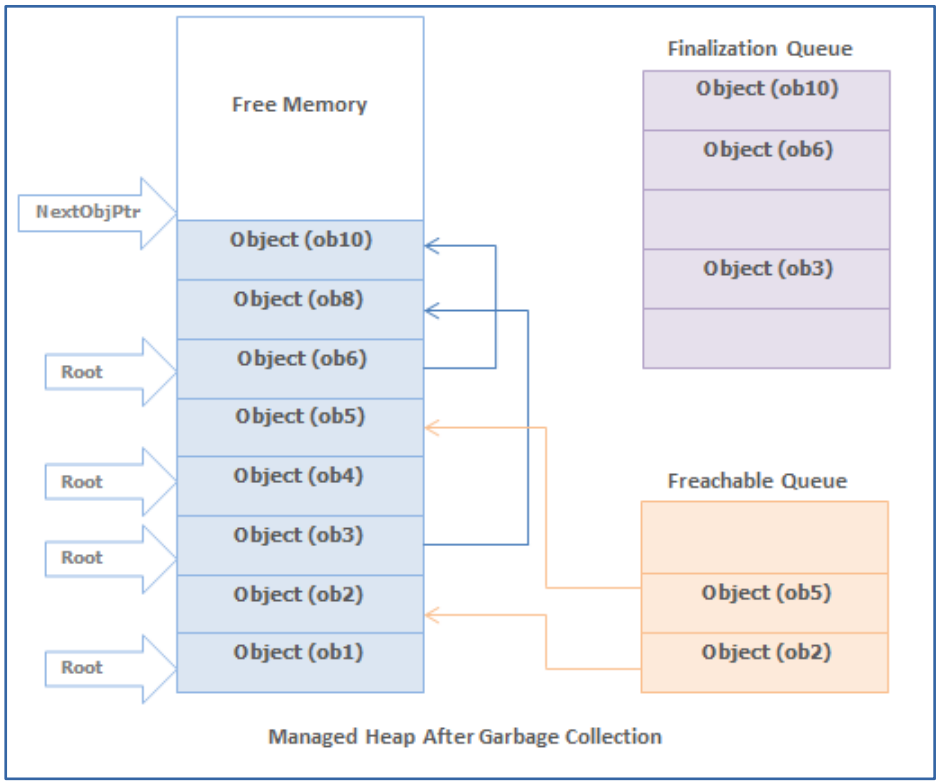
```
class Demo {
    ~Demo() => Console.WriteLine("~Demo()");

    // protected override void Finalize() {
    //     try {
    //         // Body
    //     }
    //     finally {
    //         base.Finalize();
    //     }
    // }
}

public static void Test() => new Demo();

public static void Main() {
    Test();
    Console.WriteLine("#1");
    GC.Collect();
    Console.WriteLine("#2");
}
```

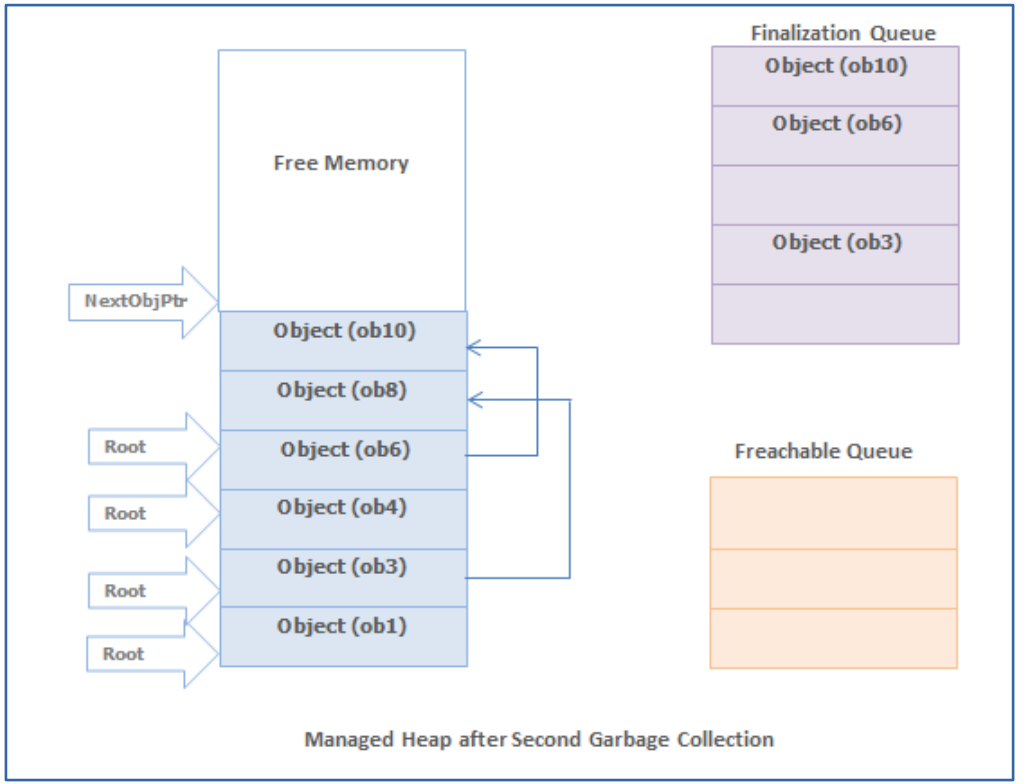
Сборщик мусора и финализируемые объекты





Сборщик мусора и финализируемые объекты





SafeHandle

```
public abstract class SafeHandle : IDisposable
{
    public abstract bool IsInvalid { get; }

    ~SafeHandle();

    public void Close();

    public void Dispose();

    protected abstract bool ReleaseHandle();

    protected IntPtr handle;

    ...
}
```

Пример использования SafeHandle

```
public class FileHandle : SafeHandleZeroOrMinusOneIsInvalid {
    private const int O_WRONLY = 1;  private const int O_CREAT = 64;
    private const int S_IRUSR = 256; private const int S_IWUSR = 128;

    [DllImport("libc")] private static extern int open(string path, int flags, int mode);
    [DllImport("libc")] private static extern int write(int fd, byte[] data, uint count);
    [DllImport("libc")] private static extern int close(int fd);

    public FileHandle(string path) : base(true) {
        var fd = open(path, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
        SetHandle(new IntPtr(fd));
    }

    public void Write(string line) {
        if (IsInvalid)
            throw new InvalidOperationException();
        var bytes = Encoding.UTF8.GetBytes(line);
        write(handle.ToInt32(), bytes, (uint) bytes.Length);
    }

    protected override bool ReleaseHandle() => close(handle.ToInt32()) == 0;
}
```

1. Исключения
2. Сборщик мусора
3. **Паттерны Dispose**

Управляемые ресурсы, sealed-класс

```
sealed class WrapperClass : IDisposable {
    public WrapperClass(Stream stream) {
        _stream = stream;
    }

    public void Dispose() {
        if (_disposed)
           return;

        // TODO: Dispose managed objects
        _stream.Dispose();

        _disposed = true;
    }

    private readonly Stream _stream;
    private bool _disposed;
}
```

Базовый класс

```
class BaseClass : IDisposable {
  public void Dispose() {
    Dispose(true);
    GC.SuppressFinalize(this);
  }

  ~BaseClass() => Dispose(false);

  protected virtual void Dispose(bool disposing) {
    if (_disposed)
      return;

    if (disposing)
      // TODO: Dispose managed objects

    // TODO: Free unmanaged resources

    _disposed = true;
  }

  private bool _disposed;
}
```

⚠ **Все неуправляемые ресурсы лучше выносить в отдельные управляемые обёртки SafeHandle**

Если есть уверенность, что в наследниках неуправляемых ресурсов быть не может:

```
class BaseClass : IDisposable {
  public void Dispose() =>
    DisposeManagedResources();

  protected virtual void DisposeManagedResources() {
    if (_disposed)
      return;

    // TODO: Dispose managed objects

    _disposed = true;
  }

  private bool _disposed;
}
```

Заключение

Любой объект может владеть ресурсами двух типов:
1) неуправляемые ресурсы (например, IntPtr);
2) ссылки на управляемые (IDisposable) ресурсы.

В обоих случаях объект, содержащий один из этих ресурсов, сам становится управляемым ресурсом.

Заключение

1. Нужно писать код без финализаторов

2. Если необходима работа с неуправляемыми ресурсами, то обернуть каждый из них в наследник SafeHandle

3. Во всех других случаях использовать паттерны Dispose для управляемых ресурсов в зависимости от типа класса (sealed или base)

Заключение

1. Гарантий вызова финализатора нет

2. Гарантий вызова Dispose нет

3. За вызов Dispose отвечает программист
   1. Конструкция using гарантирует вызов Dispose
   2. Если using использовать невозможно
      (ресурс хранится в поле или свойстве класса-обёртки),
      то класс-обёртка должен реализовывать паттерн Dispose.
      ⚠ К классу-обёртке вновь применяется пункт 3.