GARBAGE COLLECTION

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

MANAGED VS UNMANAGED CODE

- Managed
- Code, der von der CLR des .NET Frameworks ausgeführt wird (IL Code)
- Bietet Typ-Sicherheit
- Speicherverwaltung
- …

- Unmanaged
- Code, der nicht von der CLR des .NET Frameworks ausgeführt wird
- Wird vom Betriebssystem direkt ausgeführt (Machine Code)

SPEICHERVERWALTUNG

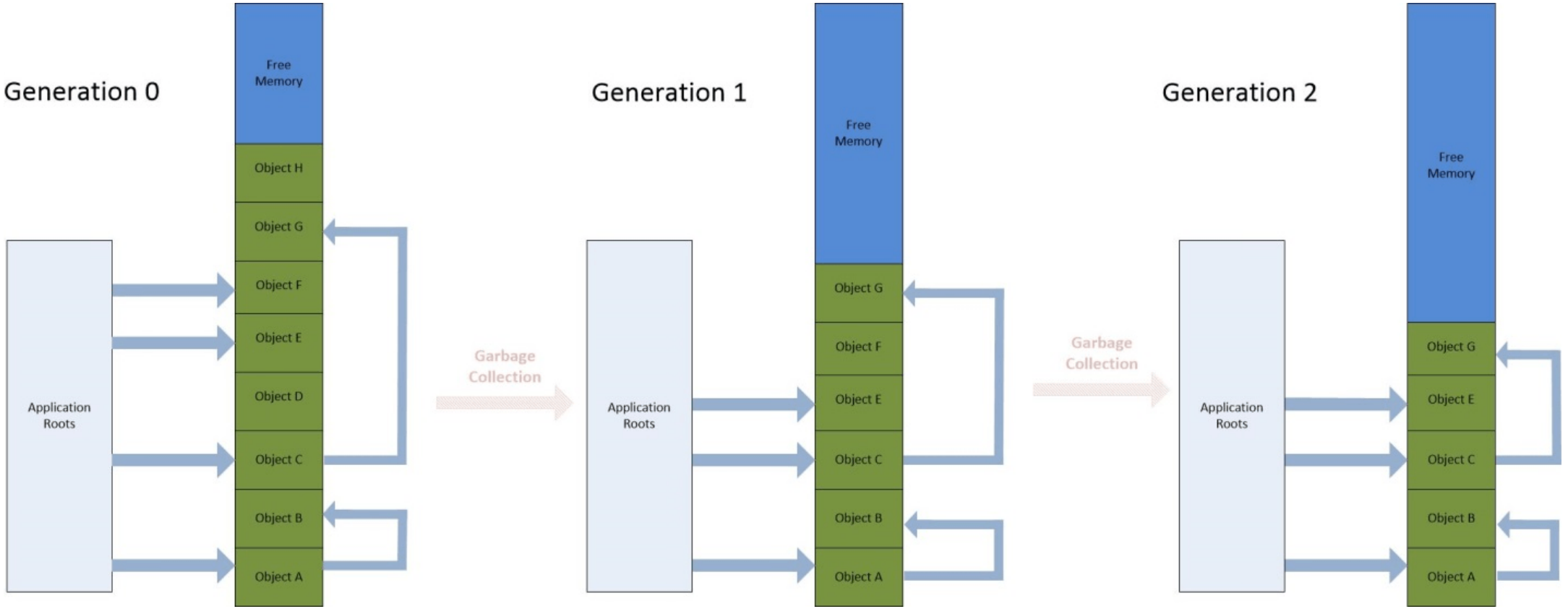
- Bei der Objekt Erzeugung wird der Konstruktor aufgerufen

- Der Garbage Collector (GC) untersucht den von der Anwendung verwendeten Arbeitsspeicher (Heap)
- Nicht mehr benötigter Speicher wird freigegeben und wenn vorhanden der Finalizer (Destruktor) aufgerufen
- Tritt ein wenn z.B. die Objektvariable nicht mehr im Gültigkeitsbereich liegt oder die Objektreferenz auf null gesetzt wird
- Die Aufrufe des GC sind nicht deterministisch und hängen von verschiedenen Faktoren ab
- Durch den Methodenaufruf GC.Collect() kann der GC manuell aufgerufen werden

SPEICHERVERWALTUNG

- Die Objekte im Heap werden in Generationen aufgeteilt
- Es gibt die Generationen 1, 2 und 3
- Die Einteilung in Generationen ermöglicht eine effizientere Speicherverwaltung, indem kurzlebige Objekte schneller freigegeben werden können
- Die einzelnen Generationen werden unterschiedlich oft untersucht und Objekte werden z.B. von der ersten in die zweite Generation verschoben

SPEICHERVERWALTUNG



FINALIZER

- Dient dem freigeben von nicht verwalteten Ressourcen (offene Datenbankverbindungen etc.)
- Wird automatisch vom GC aufgerufen
- Aufrufzeitpunkt ist nicht definiert
- Kann nicht manuell aufgerufen werden
- Ein Finalizer wird mit einem Tilde-Symbol (~) vor dem Klassennamen deklariert. Er kann keine Parameter haben und kann nicht überladen werden

```
public class Foo
{
    public Foo()
    {
        // Konstruktor
    }

    ~Foo()
    {
        // Finalizer
    }
}
```

FINALIZER

- Kann nur unmanaged Ressources freigeben, da auf eine managed Ressource zum Zeitpunkt der Ausführung bereits kein gültiger Verweis mehr existieren könnte

```
public class Foo
{
    public Foo()
    {
        // Konstruktor
    }

    ~Foo()
    {
        // Finalizer
    }
}
```

IDISPOSABLE

- IDisposable ist ein Interface, das die Dispose Methode definiert
- Ermöglicht die explizite Freigabe von Ressourcen, insbesondere für nicht verwaltete Ressourcen
- Kann im Vergleich zum Finalizer auch verwaltete Ressourcen freigeben
- Klassen, die nicht verwaltete Ressourcen halten, sollten IDisposable implementieren

- Durch die using Anweisung kann Dispose automatisch aufgerufen werden

```
using FileStream fileStream = new FileStream("test.txt", FileMode.OpenOrCreate);
using StreamWriter streamWriter = new StreamWriter(fileStream);
streamWriter.WriteLine("Hello World");
```

IDISPOSABLE IMPLEMENTIEREN

```
public class Foo : IDisposable
{
    private bool _disposed = false;

    ~Foo()
    {
        Dispose(false);
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }
```

IDISPOSABLE IMPLEMENTIEREN

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

        if (disposing)
        {
            // Freigabe verwalteter Ressourcen
        }

        // Freigabe nicht verwalteter Ressourcen
        _disposed = true;
    }
}
```

IDISPOSABLE IMPLEMENTIEREN

```
public class Program
{
    private static void Main()
    {
        using Foo foo = new Foo();
    }
}
```