A COMPARATIVE STUDY OF GARBAGE COLLECTION IN C#, JAVASCRIPT, AND KOTLIN

Presentated by
Phanthira Kositjaroenkul (6630003)
Kaung Khant Lin (6540131)
Thit Lwin Win Thant (6540122)

Our Team

Thit Lwin Win Thant

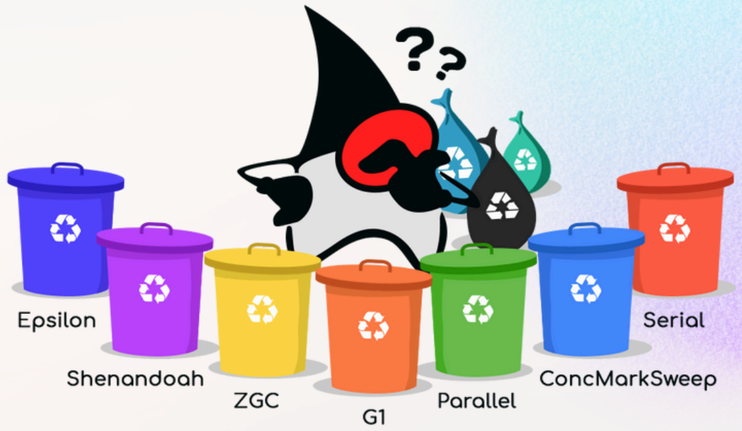
Phanthira Kositjaroenkul

Kaung Khant Lin

Introduction

What is it Garbage Collection?

- Garbage Collection (GC) = automatic memory management
- Frees memory from unused objects
- Prevents memory leaks & crashes
- Goal: Compare GC in C#, JavaScript, and Kotlin



Why These Languages?

JavaScript -

- Widely used, high-level language for web development
- Garbage collection handled by engines like V8
- Suited for event-driven, browser-based applications

Kotlin -

- Runs on the JVM and works seamlessly with Java
- Features safe and concise syntax
- Shows GC behavior in JVM-based apps

C# -

- Runs on .NET with generational garbage collection
- Statically typed and object-oriented
- Used in high-performance, enterprise applications

COMPARISON

Garbage Collection Strategy

JavaScript –

- Mark-and-Sweep algorithm with generational enhancements
- Incremental collection to keep UI responsive
- Handles circular references with ease

Kotlin –

- Uses JVM collectors (G1, ZGC, Shenandoah)
- Generational approach: short-lived vs long-lived objects
- Mark-and-Sweep / Mark-Compact under the hood

C# –

- Generational GC (Gen 0, 1, 2) with heap compaction
- Background & concurrent collection support
- Optimized for high-throughput enterprise systems

```javascript
function createCycle() {
    let a = {};
    let b = {};
    a.ref = b;
    b.ref = a;
    return null; // Both collectible once out of
}cope
```

```kotlin
class Node(var next: Node?)
fun main() {
    val a = Node(null)
    val b = Node(a)
    a.next = b
    // Both a and b are eligible for GC when unreachable
}
```

```csharp
class Node { public Node Next; }
void CreateCycle() {
    Node a = new Node();
    Node b = new Node();
    a.Next = b;
    b.Next = a;
    // Collected when no references remain
}
```

Manual vs Automatic Control

JavaScript –

- **Fully automatic; developer cannot trigger GC**
- **Browser engines manage memory behind the scenes**

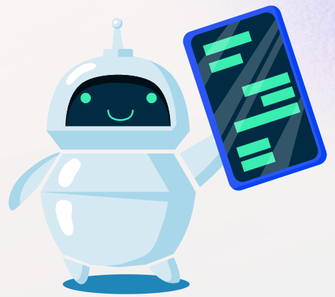
Kotlin –

- **Fully automatic; JVM decides when GC runs**
- **No API for manual GC calls**

C# –

- **Automatic by default**
- **Allows manual GC trigger with GC.Collect() (rarely recommended)**

```
// C# manual GC trigger
GC.Collect(); // Not recommended in most cases
```



Runtime Environments

JavaScript –

- Runs in browser engines (V8, SpiderMonkey)
- Also powers Node.js for server-side apps

Kotlin –

- Runs on JVM (Java Virtual Machine)
- Benefits from decades of JVM GC research

C# –

- Runs on .NET CLR (Common Language Runtime)
- Supports workstation and server GC modes

Performance Goals & Trade-offs

JavaScript –

- **Prioritizes smooth UI and low pause times**
- **May face memory pressure with long-lived apps**

Kotlin –

- **JVM collectors can be tuned (pause vs throughput)**
- **Aimed at Android/mobile performance balance**

C# –

- **Offers tuning for desktop (low latency) or server (high throughput)**
- **Background sweeping reduces pause impact**

Special Features & Tools

JavaScript –

- WeakMap, WeakSet for weak references
- WeakRef & FinalizationRegistry for advanced memory control

Kotlin –

- WeakReference & SoftReference (via JVM)
- JVM flags for choosing GC type (e.g., -XX:+UseG1GC)

C# –

- WeakReference<T>, GCSettings for fine tuning
- IDisposable & using for unmanaged resource cleanup

```javascript
let cache = new WeakMap();
(function() {
    let obj = { temp: true };
    cache.set(obj, "data");
    // 'obj' becomes collectible when out of scope
})();
```

```kotlin
val data = MyData("important")
val weak = WeakReference(data)
// 'data' is collectible when no strong refs remain
```

```javascript
var weak = new WeakReference<MyClass>(new MyClass());
// Object collectible when no strong reference exists
```

Memory Leak Risks

JavaScript –

- Closures holding unused variables
- DOM event listeners not removed
- Accidental globals in non-strict mode

Kotlin –

- Static or singleton references holding Activities
- Long-lived coroutines not canceled
- Observers or listeners not unregistered

C# –

- Event handler subscriptions left active
- Static fields holding objects indefinitely
- Unmanaged resources not disposed (IDisposable not used)

INSIGHTS

Implications for Learning and Productivity

- GC makes programming more accessible and speeds up prototyping
- Can lead to a false sense of memory safety and unnoticed leaks
- Highlights the need for better developer education and tooling

The Future: Smarter GC and Static Analysis

- Growing complexity demands smarter, adaptive GC algorithms
- Deeper static analysis should help identify leaks earlier
- Connecting analysis to memory patterns will improve application robustness

Trade-off Between Memory and Performance

- GC improves productivity but can impact performance
- High-performance apps combine GC with manual memory strategies
- Developers balance convenience and efficiency based on requirements

References

Java Script

- Understand JS Garbage Collector in 4 mins
- https://www.geeksforgeeks.org/javascript/garbage-collection-in-javascript/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Memory_management#garbage_collection

Kotlin

- https://newrelic.com/blog/best-practices/java-garbage-collection
- https://www.netdata.cloud/academy/java-garbage-collection/
- https://www.geeksforgeeks.org/java/garbage-collection-in-java/
- https://kotlinlang.org/docs/native-memory-manager.html
- https://kotlinlang.org/api/core/kotlin-stdlib/kotlin.native.runtime/-g-c/
- https://moldstud.com/articles/p-a-comprehensive-comparison-of-kotlin-and-java-performance-in-real-world-native-applications
- https://appmaster.io/blog/kotlin-memory-management-and-garbage-collection
- https://dev.to/arsenikavalchuk/manual-memory-management-and-garbage-collection-in-kotlin-multiplatform-native-shared-libraries-1ll3
- https://dev.to/arsenikavalchuk/memory-management-and-garbage-collection-in-kotlin-multiplatform-xcframework-15pa

C#

- https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview
- https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/
- https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals
- https://www.geeksforgeeks.org/c-sharp/garbage-collection-in-c-sharp-dot-net-framework/
- https://learn.microsoft.com/en-us/dotnet/api/system.weakreference?view=net-9.0

THANK YOU