Sendable

“Your code runs fine… until it runs concurrently.”

Then it crashes. Or corrupts data.

What if Swift could stop you from writing unsafe code?

That’s what Sendable is here for. Let’s break it down

Ahmed Saeed Hendi

→

Wait, what's Sendable?

> Sendable is a protocol that marks a type as safe to pass across threads.

❌ Data races

✅ Value isolation

It's Swift's way of saying:

"I promise this type can safely move between concurrent tasks."

A thread-safe type whose values can be shared across arbitrary concurrent contexts without introducing a risk of data races.

https://developer.apple.com/documentation/Swift/Sendable

→

Why should I care?

Because in Swift Concurrency:

- Code can run in **parallel**
- Types shared between tasks might **mutate at the same time**
- That = undefined behavior

With Sendable, Swift checks your types at compile time, so you don’t have to face concurrency issues at runtime with unsafe code.

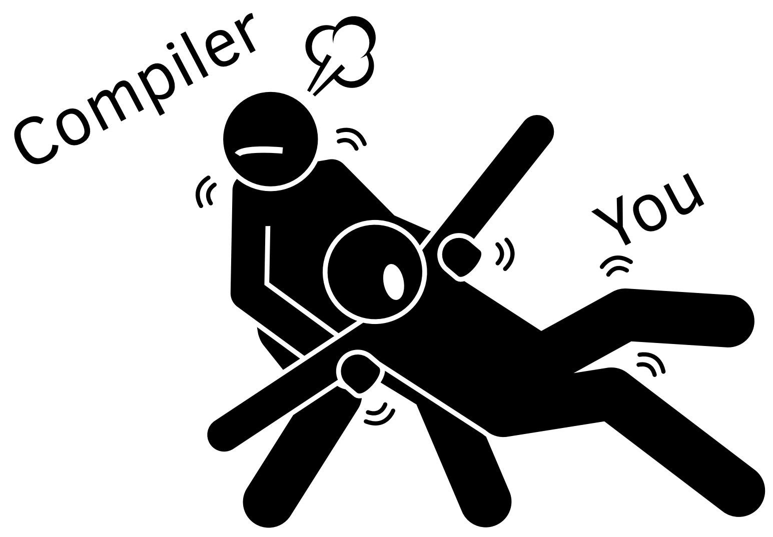
Ahmed Saeed Hendi

When does Swift check for Sendable?

When compiling code, whenever you use concurrency features like:

- Task { ... }
- DetachedTask
- actor
- @MainActor, @BackgroundActor
- @Sendable closures

Swift will enforce that any value you send into these is **Sendable**.

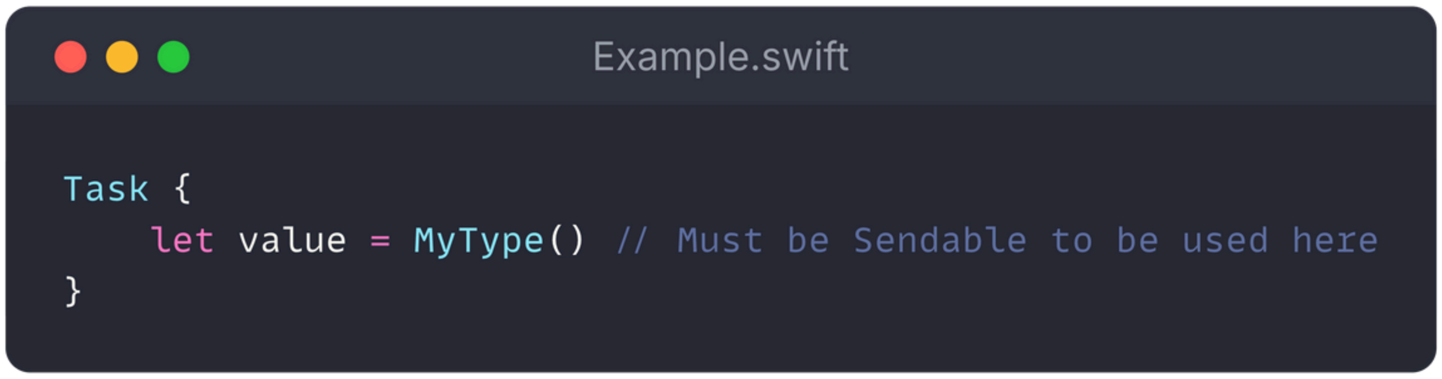


Will the compiler force me?

Yes, when you have the following scenario

1- Concurrency Constructs:

Tasks: When you create a Task or DetachedTask, Swift ensures that any values passed into the task's closure or shared between tasks conform to Sendable. For example:

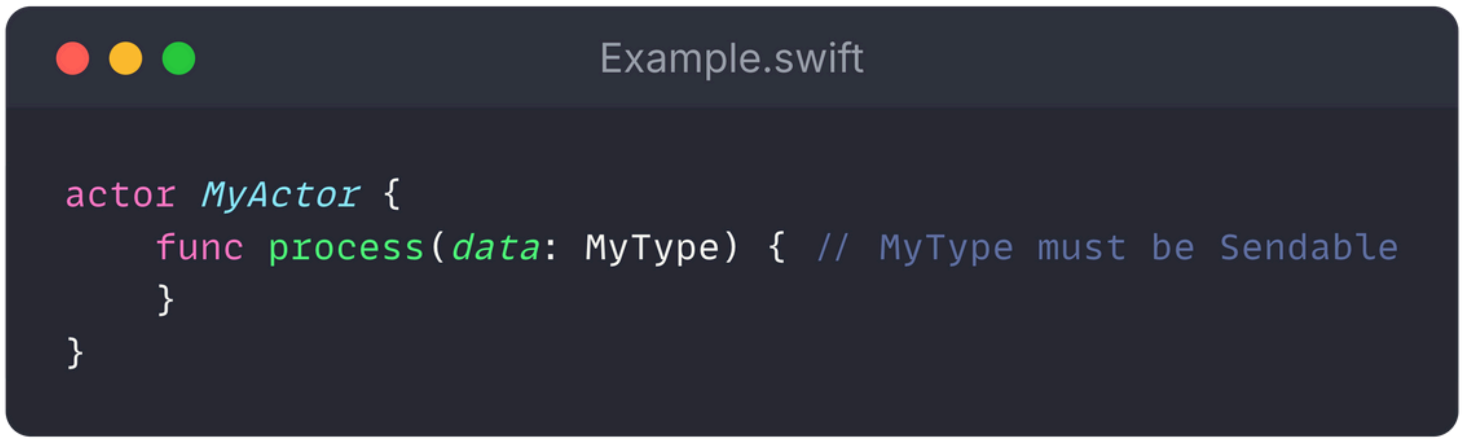


Example.swift

```swift
Task {
    let value = MyType() // Must be Sendable to be used here
}
```

MyType is your class or struct

Actors: When passing data to or from an actor, Swift checks that the data conforms to Sendable. For instance:



Example.swift

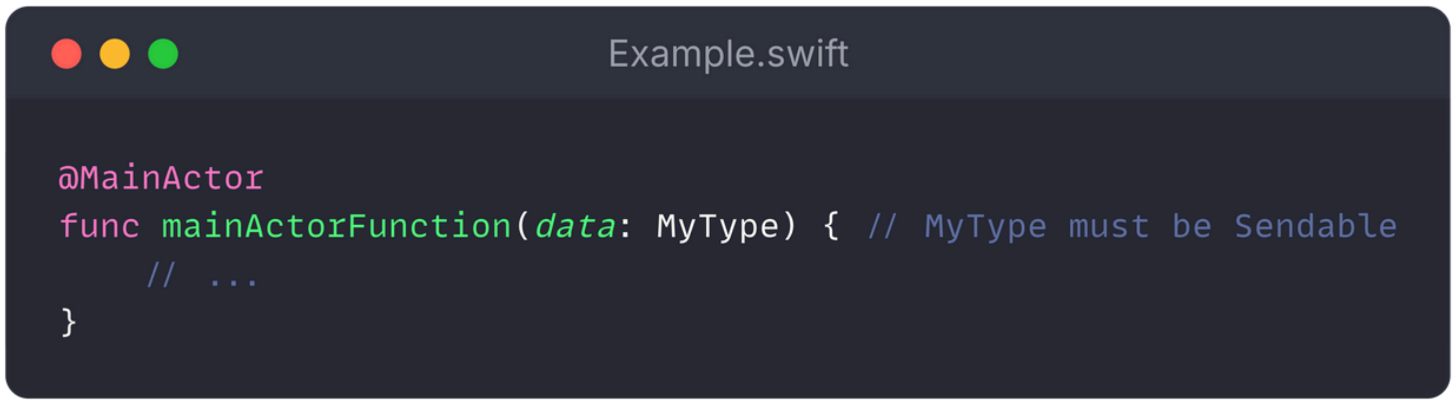
```swift
actor MyActor {
    func process(data: MyType) { // MyType must be Sendable
    }
}
```

Ahmed Saeed Hendi

Will the compiler force me?

Yes, when you have the following scenario

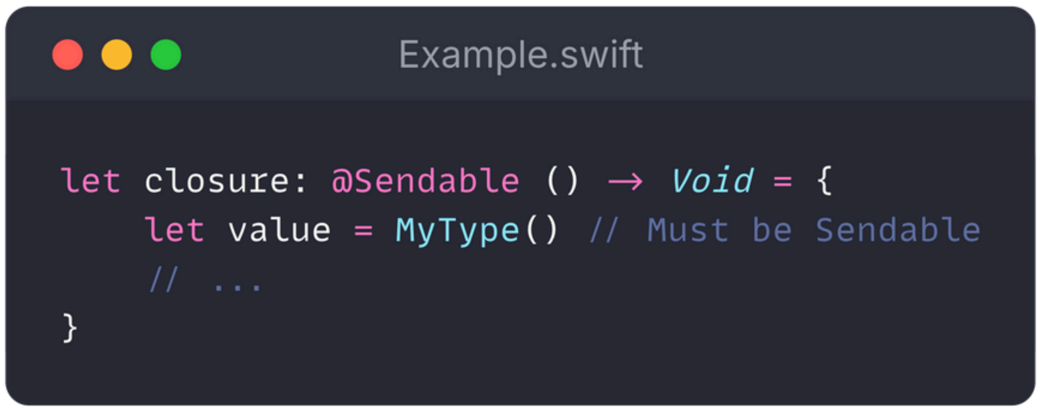
2- Actor Isolation Annotations:
When using attributes like @MainActor or custom actor-isolated contexts, Swift ensures that values passed across these isolation boundaries are Sendable. For example:


Example.swift

```swift
@MainActor
func mainActorFunction(data: MyType) { // MyType must be Sendable
    // ...
}
```

3- @Sendable Closures:

When a closure is marked with the @Sendable attribute, Swift enforces that any captured values in the closure conform to Sendable. For example:


Example.swift

```swift
let closure: @Sendable () → Void = {
    let value = MyType() // Must be Sendable
    // ...
}
```

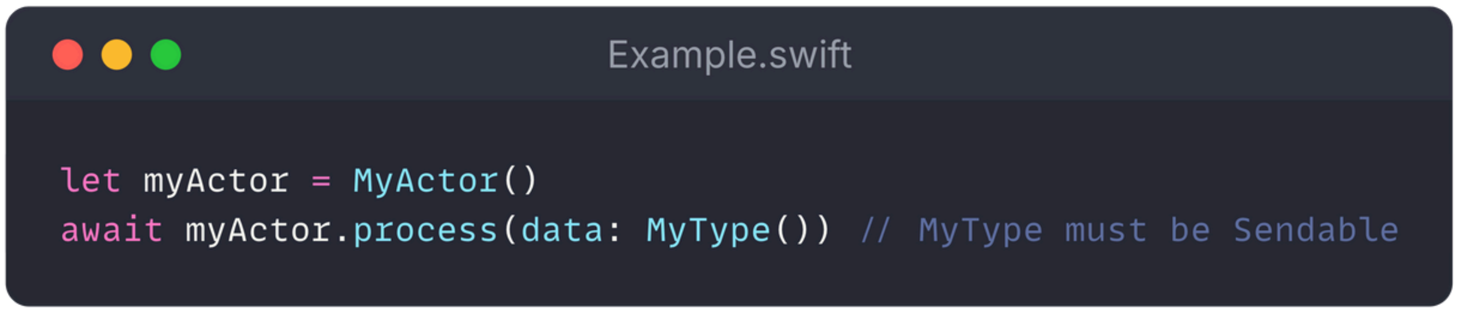
Ahmed Saeed Hendi

→

Will the compiler force me?

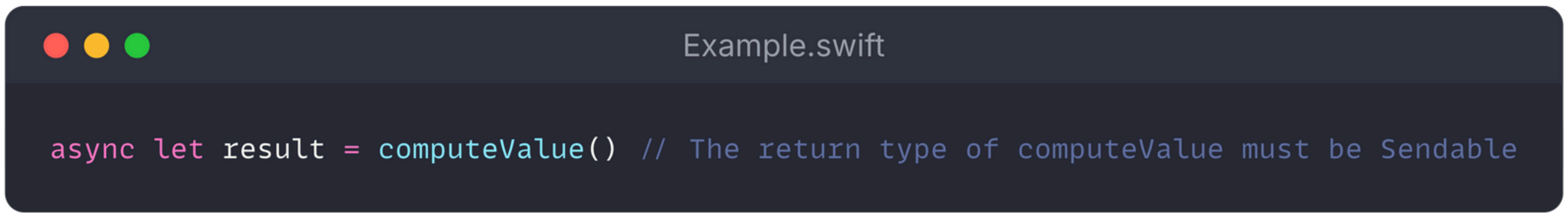
Yes, when you have the following scenario

4- Cross-Actor Messaging:
When you call a method on an actor or pass data between actors, Swift checks that the arguments and return types conform to Sendable. For example:



```swift
let myActor = MyActor()
await myActor.process(data: MyType()) // MyType must be Sendable
```

5- Structured Concurrency:
When a closure is marked with the @Sendable attribute, Swift enforces that any captured values in the closure conform to Sendable. For example:



```swift
async let result = computeValue() // The return type of computeValue must be Sendable
```

→

Overview

Values of the type may have no shared mutable state, or they may protect that state with a lock or by forcing it to only be accessed from a specific actor.
You can safely pass values of a sendable type from one concurrency domain to another — for example, you can pass a sendable value as the argument when calling an actor's methods. All of the following can be marked as sendable:

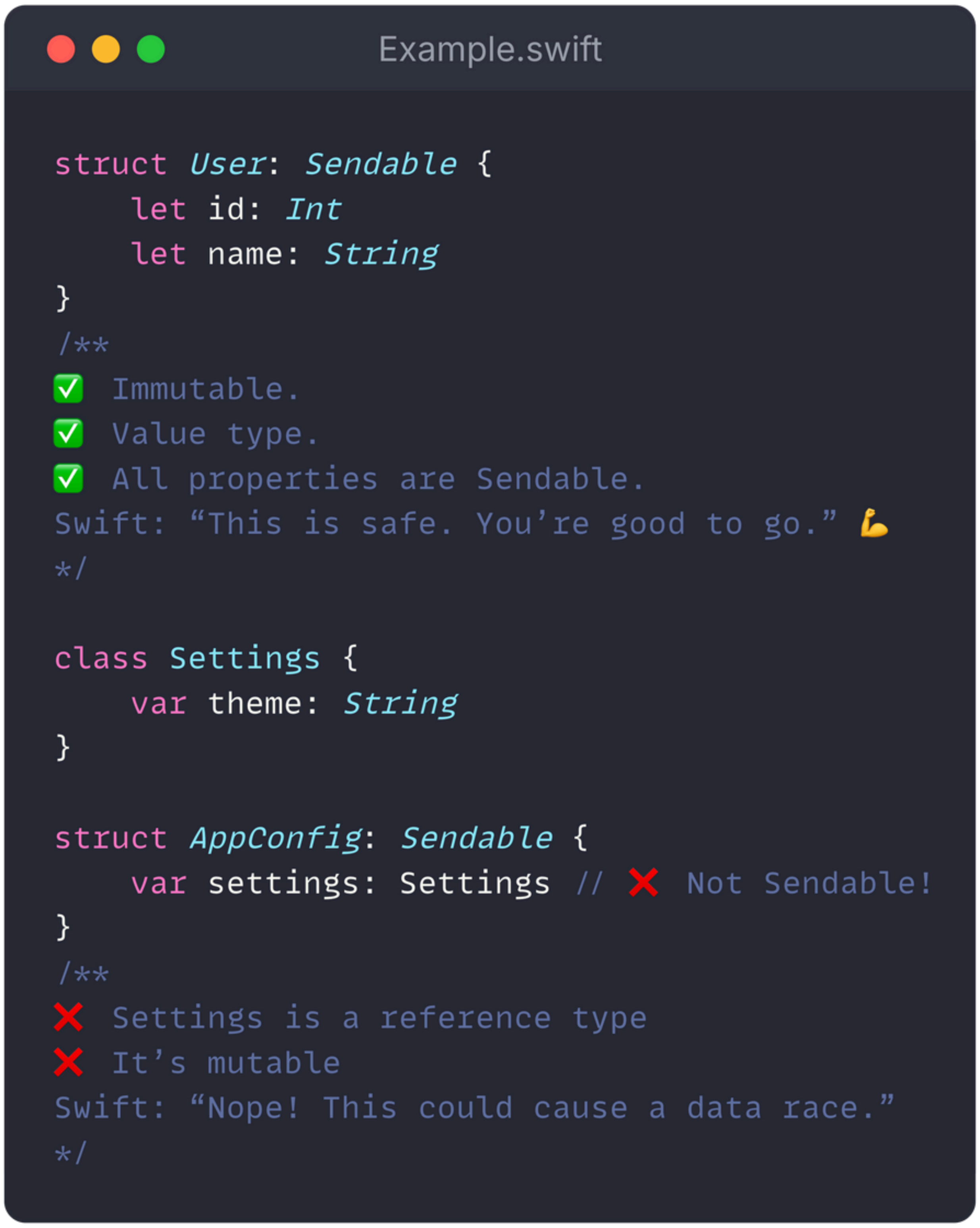
- Value types
- Reference types with no mutable storage
- Reference types that internally manage access to their state
- Functions and closures (by marking them with @Sendable)

Although this protocol doesn't have any required methods or properties, it does have semantic requirements that are enforced at compile time. These requirements are listed in the sections below. Conformance to Sendable must be declared in the same file as the type's declaration.
To declare conformance to Sendable without any compiler enforcement, write @unchecked Sendable. You are responsible for the correctness of unchecked sendable types, for example, by protecting all access to its state with a lock or a queue. Unchecked conformance to Sendable also disables enforcement of the rule that conformance must be in the same file.

https://developer.apple.com/documentation/Swift/Sendable

Structs are friends

Implicit Sendable Conformance: Some types, like structs and enums with Sendable members (String, Int, etc conform to Sendable by default).


Example.swift

```swift
struct User: Sendable {
    let id: Int
    let name: String
}
/**
✅ Immutable.
✅ Value type.
✅ All properties are Sendable.
Swift: “This is safe. You’re good to go.” 💪
*/

class Settings {
    var theme: String
}

struct AppConfig: Sendable {
    var settings: Settings // ❌ Not Sendable!
}
/**
❌ Settings is a reference type
❌ It’s mutable
Swift: “Nope! This could cause a data race.”
*/
```

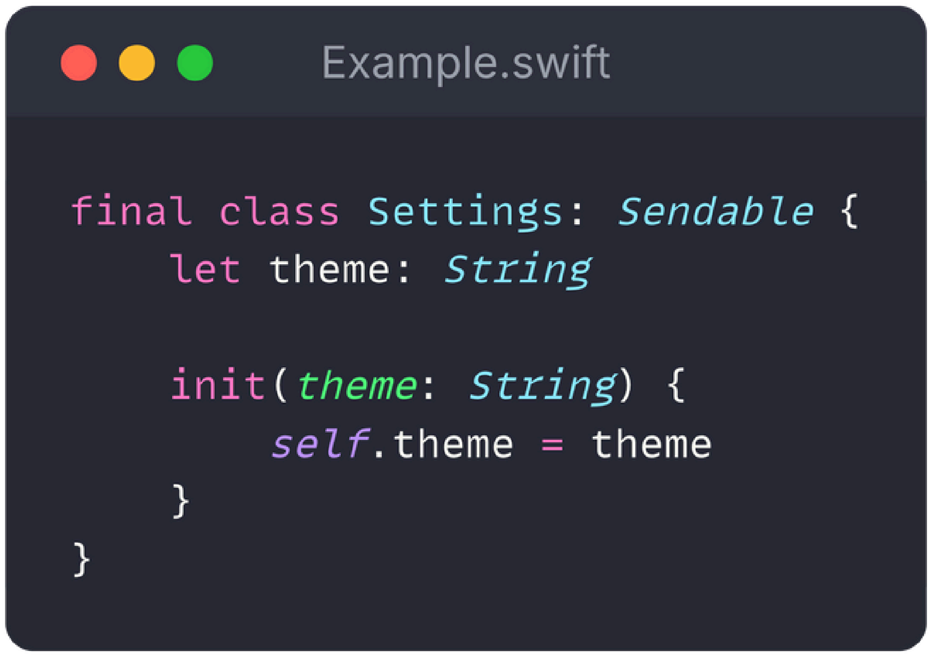
codesnap.dev

Ahmed Saeed Hendi

→

Fix it with immutability

let’s mark it with sendable protocol and make theme property constant instead of variable.



Example.swift

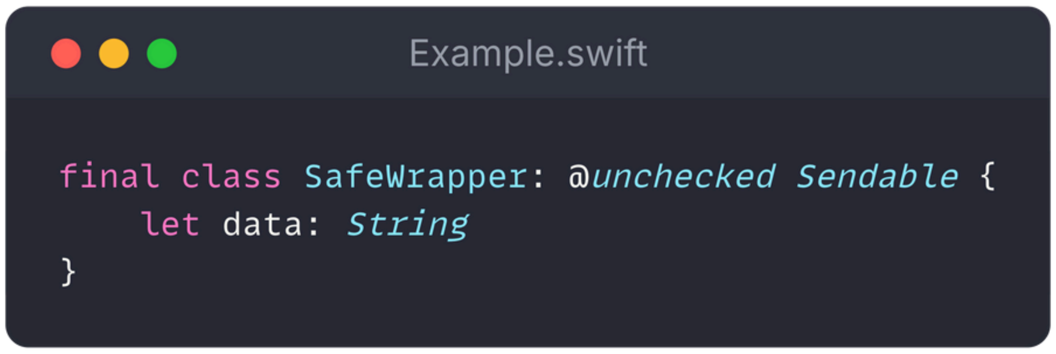
```swift
final class Settings: Sendable {
    let theme: String

    init(theme: String) {
        self.theme = theme
    }
}
```

✅ All properties are immutable
✅ Final class means no subclassing

Ahmed Saeed Hendi

→

Use @unchecked Sendable carefully



Example.swift

```swift
final class SafeWrapper: @unchecked Sendable {
    let data: String
}
```

You’re telling Swift: “I promise this is thread-safe.”

Use this ONLY if:
All properties are immutable
You know what you’re doing

Recap

✅ **Sendable** = compile-time thread safety
✅ **Protects** from data races
✅ Works with Swift Concurrency
✅ **Easy** with structs, **tricky** with classes
✅ Use @Sendable closures
⚠️ Use @unchecked Sendable only **if you really understand the risks**

Note: In Swift 6, Copyable is related but different.

- Copyable = can be copied safely
- Sendable = can be shared across threads safely
- Not interchangeable but both focus on safety.

I have posted about copyable before and I will post a more detailed post soon.

Thank you

Ahmed Saeed Hendi

qabilah

Ahmed Saeed

@ihendi

Senior IOS Developer

امسح الرمز لفتح صفحتي المهنية
وسؤالي عن أي شيء

اسألني عن أي شيء

أنا هنا للإجابة على أسئلتك!

https://qabilah.com/profile/ihendi/

https://www.linkedin.com/in/ihendi/