The Billion Dollar Mistake

Origin

*I call it my **billion-dollar mistake**. It was the invention of the **null reference** in 1965.*

At that time, I was designing the first comprehensive type system for references in an object oriented language. My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler.

*But I couldn't resist the temptation to put in a null reference, **simply because it was so easy to implement**. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused **a billion dollars of pain and damage** in the last forty years.*

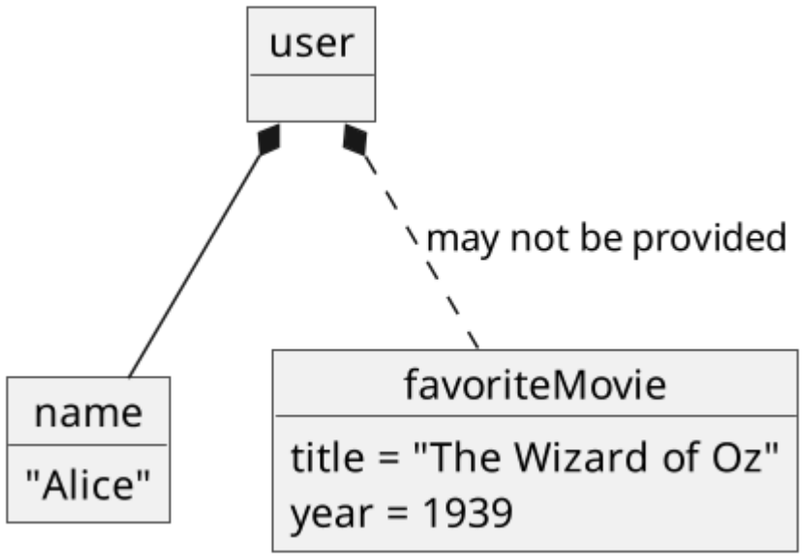
—Tony Hoare, 2009

”Billion Dollar”

- Could not find directly linked, catastrophic failure
- Toil of finding and fixing them

Null Reference

- a.k.a. Null Pointer
- Issue comes usually with dereferencing



Garden Variety (JS)

```
user = {};
console.log(user.favoriteMovie.title);
// => Uncaught TypeError: Cannot read properties of undefined (reading 'title')

user2 = { favoriteMovie: null };
console.log(user2.favoriteMovie.title);
// => Uncaught TypeError: Cannot read properties of null (reading 'title')
```

Real JS Issue

queue.isReady() throws error if client not initialized #1344

Open **gabegorelick** opened this issue on Jun 11, 2019 · 2 comments

gabegorelick commented on Jun 11, 2019 • edited ▾ Contributor •••

Description

If `queue._initializing` is not set, `queue.isReady()` will throw a `TypeError: Cannot read property 'then' of undefined` at

bull/lib/queue.js
Line 478 in `c0ee6be`

```
478     return this._initializing.then(() => {
```

Real JS Fix

```
  Queue.prototype.whenCurrentJobsFinished = function() {
    return new Promise((resolve, reject) => {
+     if (!this.bclientInitialized) {
+       // bclient not yet initialized, so no jobs to wait for
+       return resolve();
+     }
+
      //
      // Force reconnection of blocking connection to abort blocking redis
  call immediately.
      //
      const forcedReconnection = redisClientDisconnect(this.bclient).then(()
  => {
        return this.bclient.connect();
      });
```

Garden Variety (Java)

```java
class Movie {
    String title;
}

class User {
    String name;
    Movie favoriteMovie;
}

class GardenVariety {
    public static void main(String[] args) {
        var user = new User();
        System.out.println(user.favoriteMovie.title);
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException:
    Cannot read field "title" because "<local1>.favoriteMovie" is null
```

Real Java Issue

elastic / elasticsearch Public

Watch 2.7k

Code | Issues 3.6k | Pull requests 477 | Actions | Projects 1 | Security

7.16 _disk_usage API - NullPointerException #87761

Closed **ryanmorrisonlytics** opened this issue on Jun 16, 2022 · 2 comments · Fixed by #87826

Real Java Fix

```
@@ -385,8 +389,10 @@ void analyzePoints(SegmentReader reader, IndexDiskUsageStats stats) throws IOExc
385                 directory.resetBytesRead();
386                 if (field.getPointDimensionCount() > 0) {
387                     final PointValues values =
    pointsReader.getValues(field.name);
388 -                   values.intersect(new PointsVisitor());
389 -                   stats.addPoints(field.name, directory.getBytesRead());
390                 }
391             }
392         }
```

```
@@ -385,8 +389,10 @@ void analyzePoints(SegmentReader reader, IndexDiskUsageStats stats) throws IOExc
389                 directory.resetBytesRead();
390                 if (field.getPointDimensionCount() > 0) {
391                     final PointValues values =
    pointsReader.getValues(field.name);
392 +                   if (values != null) {
393 +                       values.intersect(new PointsVisitor());
394 +                       stats.addPoints(field.name, directory.getBytesRead());
395 +                   }
396                 }
397             }
398         }
```

Standing Guard

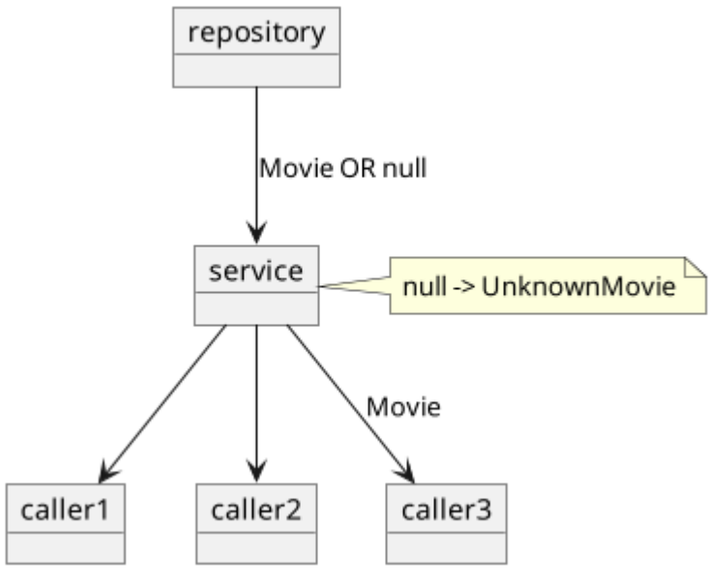
- Identify and handle unwanted nulls early
- Usually at the entryway to your system (HTTP, parsing files)
- Guard clauses or, better yet, a validation library

```
function processUserRequest(req) {
    if (!req.user) {
        throw new Error('No user provided');
    }
    if (!req.user.favoriteMovie) {
        throw new Error ('No favorite movie provided');
    }
    if (!req.user.favoriteMovie.title) {
        throw new Error('No title provided');
    }

    // Happy-path code goes here.
}
```

Null Object Pattern

- a.k.a. Active Nothing
- Turn your "elses" into encapsulated behavior
- **Not like sentinel values**



Null Object Example

```
const UNKNOWN_MOVIE = {
    title = '';
    year = '';
}

async function loadUserForTemplate(userId) {
    const user = await userRepo.find(userId, { relations: ['favoriteMovie'] });
    if (!user.favoriteMovie) {
        user.favoriteMovie = UNKNOWN_MOVIE;
    }
    return user;
}
```

Real Null Objects

- "The show must go on."
- Can be inconspicuous or… **obvious**.

Direct Dealing

- Downstream things must handle nulls

```
// If Statement
let title = '';
if (user.favoriteMovie) {
  title = user.favoriteMovie.title;
}

// Ternary
user.favoriteMovie ? user.favoriteMovie.title : '';

// Optional Chaining (?.) + Nullish Coalescing (??)
user.favoriteMovie?.title ?? '';
```

Optional Chaining

- First **bad** access short-circuits the whole thing
- Quick and easy, but can be a bit dirty
- Many languages have this feature

```
user = {};

user.favoriteMovie
// => undefined

user.favoriteMovie.title
// => Uncaught TypeError: Cannot read properties of undefined (reading 'title')

user.favoriteMovie?.title;
// => undefined

user.favoriteMovie?.title.toUpperCase();
// => undefined
```

Taking It Too Far

- Don't **?.?.?.ing** do this

```
user?.bestFriend?.address?.zipCode?.toUpperCase?.();
```

Static Type Systems

- In some compiled langs, ref types are nullable
- Let's avoid this uncertainty altogether
- Optionals can prevent this issue from happening at runtime

TypeScript's "? :"

```
class User {
  favoriteMovie?: Movie; // Equivalent to `Movie | undefined`.
}

const user = new User();
console.log(user.favoriteMovie.title);
```

During compilation:

```
'user.address' is possibly 'undefined'.
```

Optionals

- Requires handling of null case

```
class User {
    Optional<Movie> favoriteMovie;
}
// ...
var unknownMovie = new Movie("");
user.favoriteMovie.orElse(unknownMovie).title;
// Or, if no way to handle it
user.favoriteMovie.orElseThrow().title;
```

Summary

- Know your type system
- Validate at the edges
- Some "nulls" are latent "null objects"
- Use optional chaining operator sparingly
- Use explicit optionals instead of implicit nulls