

# How Junior Engineers Can Utilize ChatGPT

國泰世華銀行 Samuel Luo

# Content

- Introduction to ChatGPT
- Examples of using ChatGPT
- Summarize

# Introduction to ChatGPT

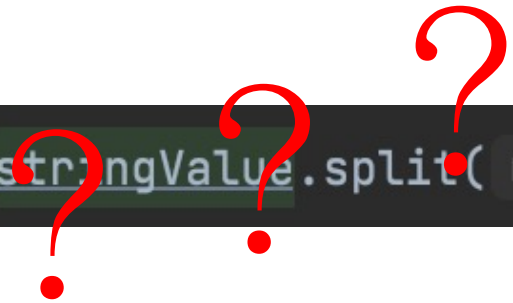


此相片 (作者: 未知的作者) 已透過 [CC BY-SA-NC](#) 授權

# Ex1. Unfamiliar with JAVA Features

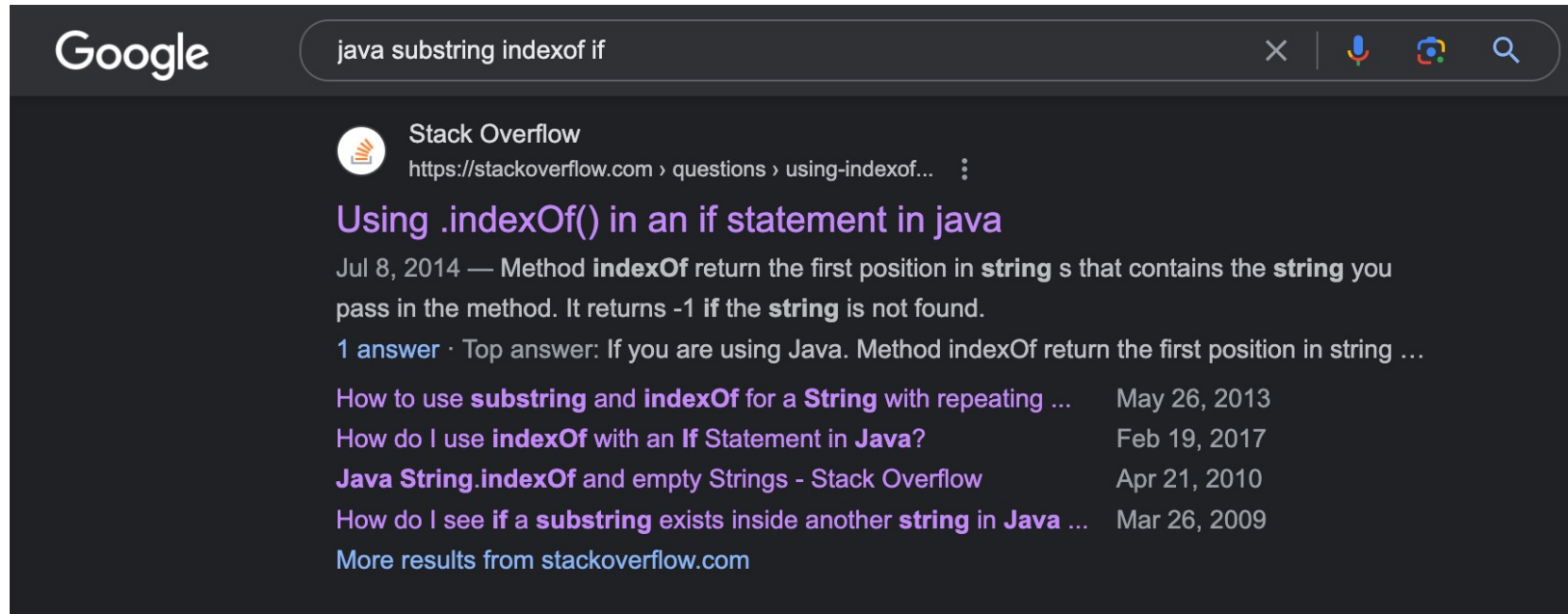
```
int index = stringValue.indexOf("/");  
if (index > -1) {  
    stringValue = stringValue.substring(0, index);  
}
```

```
stringValue = stringValue.split(regex: "/")[0];
```



# Ex1. Unfamiliar with JAVA Features

- If just using Google...



# Ex1. Unfamiliar with JAVA Features



有沒有tricky的寫法 或是 import 方法



當然，有一些 "tricky" 的寫法可以達到同樣的效果，但這些方法可能會犧牲一些可讀性。以下 是一個使用三元運算符的示例：

java

Copy code

```
String value = "your-string-here";  
int index = value.indexOf("/");  
value = index != -1 ? value.substring(0, index) : value;
```

或者，如果你真的想要一個簡單的一行解決方案，可以這樣：

java

Copy code

```
value = value.split("/")[0];
```

## Ex2. Unfamiliar with JUnit Feature

```
public class TennisTest {  
  
    private Tennis tennis;  
  
    @BeforeEach  
    void setUp() { tennis = new Tennis(); }
```

# Ex2. Unfamiliar with JUnit Feature



官方檔案哪裡有說 junit 為每個測試方法創建新的類實例



我為你找了JUnit 5的官方用戶指南。在其中，對於這個特性的描述可以在"Test Instance Lifecycle"部分找到。



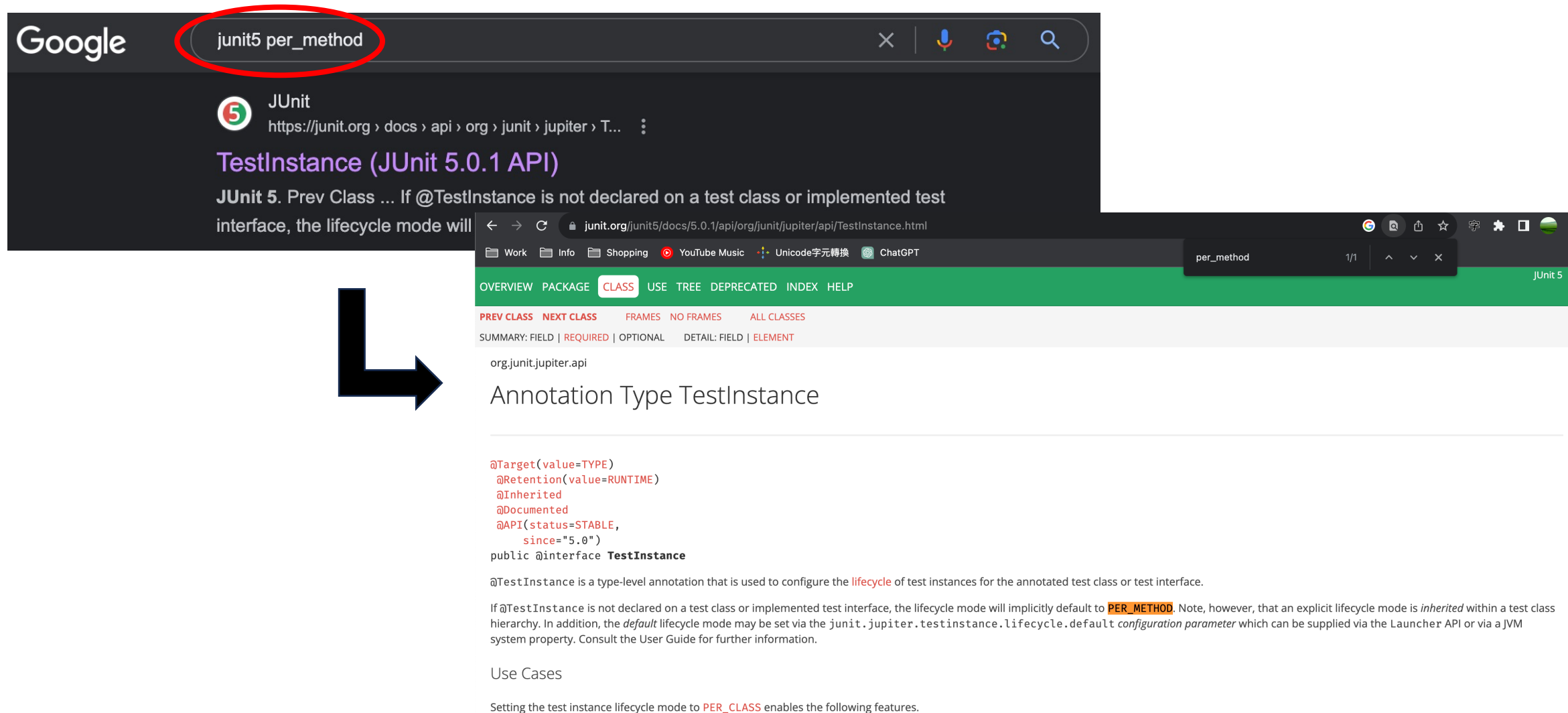
在JUnit 5中，可以選擇使用兩種不同的測試實例生命週期：

1. **PER\_METHOD (默認)**：默認情況下，JUnit會為每個@Test、@RepeatedTest、@ParameterizedTest、@TestFactory和@TestTemplate方法創建新的測試實例。這與JUnit 4的行為相符，可以確保測試之間的完全隔離。
2. **PER\_CLASS**：當測試類使用@TestInstance(Lifecycle.PER\_CLASS)註解時，JUnit會為整個測試類創建一個測試實例。這允許你在多個測試方法之間共享測試實例的狀態。此選項可用於某些特定的情境，例如當使用`@BeforeAll`或`@AfterAll`方法時不想使用靜態方法。

在JUnit 5的官方用戶指南中，對於這個特性有詳細的說明。你可以查看該部分以獲得更多關於測試實例生命週期的資訊。



# Ex2. Unfamiliar with JUnit Feature



Google search results for "junit5 per\_method". The top result is from JUnit.org, showing the JUnit 5.0.1 API page for `TestInstance`. The page title is "JUnit 5.0.1 API" and the subtitle is "JUnit 5. Prev Class ... If @TestInstance is not declared on a test class or implemented test interface, the lifecycle mode will".

The page content shows the JUnit 5.0.1 API documentation for `TestInstance`. The page title is "Annotation Type TestInstance". The page content includes the following code snippet:

```
@Target(value=TYPE)
@Retention(value=RUNTIME)
@Inherited
@Documented
@API(status=STABLE,
      since="5.0")
public @interface TestInstance
```

The page also includes a description of `@TestInstance` and its usage:

`@TestInstance` is a type-level annotation that is used to configure the lifecycle of test instances for the annotated test class or test interface.

If `@TestInstance` is not declared on a test class or implemented test interface, the lifecycle mode will implicitly default to `PER_METHOD`. Note, however, that an explicit lifecycle mode is *inherited* within a test class hierarchy. In addition, the *default* lifecycle mode may be set via the `junit.jupiter.testinstance.lifecycle.default` configuration parameter which can be supplied via the Launcher API or via a JVM system property. Consult the User Guide for further information.

Use Cases

Setting the test instance lifecycle mode to `PER_CLASS` enables the following features.

## Ex3. Improvement in Coding Style

```
BankBO bankBO = localCacheBankMap.get(response.getBankId());  
if (bankBO == null) {  
    bankName = response.getBankName();  
} else {  
    bankName = bankBO.getBankName();  
}
```

# Ex3. Improvement in Coding Style



```
BankBO bankBO = localCacheBankMap.get(response.getBankId());  
    if (bankBO == null) {  
        bankName = response.getBankName();  
    } else {  
        bankName = bankBO.getBankName();  
    }  
use Optional?
```



Certainly! You can use Java 8's `Optional` to make the code more concise and readable.  
Here's how you can rewrite the provided code using `Optional`:



java

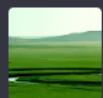
Copy code

```
BankBO bankBO = localCacheBankMap.get(response.getBankId());  
bankName = Optional.ofNullable(bankBO)  
    .map(BankBO::getBankName)  
    .orElse(response.getBankName());
```

## Ex3. Improvement in Coding Style

```
// BankBO bankBO = localCacheBankMap.get(response.getBankId());  
bankName = Optional.ofNullable(localCacheBankMap) Optional<Map<String, TennisTest.BankBO>>  
    .map(bankMap -> localCacheBankMap.get(response.getBankId())) Optional<TennisTest.BankBO>  
    .map(BankBO::getBankName) Optional<String>  
    .orElse(response.getBankName());
```

## Ex4. Regex



我要寫一個正則：如果包含/handshakewb 則回傳 handshakewb，如果包含 /M\*\*\*\*%%%/，則回傳M\*\*\*\*%%%，而且\*代表任意英文字母，%代表任意數字



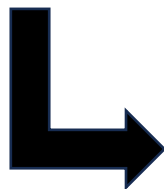
根據您的需求，以下是一個正則表達式來匹配指定的模式：



regex

Copy code

```
(/handshakewb)|(/M[a-zA-Z]*\d*\d/)
```



```
(/handshakewb)|(/M[A-Z]{4}\\d{3}/)
```

# Ex4. Regex



regex不能實現捕獲 跟 group輸出不一樣嗎



正則表達式的捕獲和分組是相關的概念，但兩者有所不同。在正則表達式中：



1. **捕獲 (Capturing)**：當你使用括號 ``(`` 來定義正則表達式的一部分時，你正在進行捕獲。被括號包圍的部分將被捕獲並可以在之後被參照或提取。每個捕獲組都有一個與之相關的編號，從1開始，依次遞增。
2. **分組 (Grouping)**：括號 ``(`` 也可以用來進行分組，而不一定是捕獲。分組允許你將多個字元或子表達式組合成一個單一的單位，從而可以對這個單元應用量詞或其他操作。

在許多程式語言和工具中（如Java、Python、JavaScript等），當你執行一個正則表達式匹配時，你可以獲取到所有的捕獲組。主匹配（通常稱為 ``group(0)`` 或整體匹配）是整個正則表達式的匹配結果，而 ``group(1)``、``group(2)`` 等是各個捕獲組的結果。

例如，對於正則表達式 ``/M([a-zA-Z]*\d*)\//``，``group(0)`` 將返回整體匹配（例如 ``/Mabc123/``），而 ``group(1)`` 將只返回捕獲組的匹配，也就是 ``abc123``。

希望這樣能幫助您更好地理解正則表達式中的捕獲和分組。

```
/handshakewb|/M[A-Z]{4}\\d{3}/
```

```
replaceAll("/", "");
```

# Summarize

- Understanding my communication and converting it into draft code
- Analyzing the document and giving me some keyword.
- A new approach to learning.