



Functional Interfaces

- Functional Interface : 只有一個 method 需實作的 interface 。
- Lambda 表示式 必須搭配 functional interfaces 。
- 因為Functional Interface只有一個方法，可以預期使用方式，因此Java 8 在套件 `java.util.function` 內建了許多的 functional interfaces，可以直接使用。

Functional Interface Demo 1/2

```
@FunctionalInterface
interface StringAnalyzer {
    public boolean analyze(String target, String searchStr);
}

class ContainsAnalyzer implements StringAnalyzer {
    public boolean analyze(String target, String searchStr) {
        return target.contains(searchStr);
    }
}

class StartWithAnalyzer implements StringAnalyzer {
    public boolean analyze(String target, String searchStr) {
        return target.startsWith(searchStr);
    }
}
```

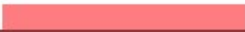


The java.util.function Package

- Java 8 內建的 functional interfaces 依其必須實作的方法分四大類：
 - Predicate：沒有參數，且回傳 boolean
 - Consumer：可以傳入參數，且沒有回傳 (void)
 - Function：將傳入的參考由 T 型別轉換成 U 型別
 - Supplier：如同工廠方法，提供T型別的實例/物件
- OCA考試必須了解Predicate的種類，使用Predicate在於需「作出論斷」，因此唯一的方法回傳true/false。



Predicate



```
package java.util.function;

public interface Predicate<T> {
    public boolean test(T t);
}
```

使用Predicate<Animal>取代IChecker

```
public class LambdaPredicateSearch {
    public static void main(String[] args) {
        List<Animal> animals = new ArrayList<Animal>();
        animals.add(new Animal("fish", false, true));
        animals.add(new Animal("monkey", true, false));
        animals.add(new Animal("rabbit", true, false));
        animals.add(new Animal("human", true, true));
        print(animals, a -> a.canJump());
    }
    private static void check(List<Animal> animals, Predicate<Animal> checker) {
        for (Animal animal : animals) {
            if (checker.test(animal))
                System.out.print(animal + " ");
        }
    }
}
```



-
- 這樣，就連IChecker介面也不需要建立！






END ~~

Thank you!!





Lambda表示式推導過程-2

- Lambda 表示式只需要強調方法的內容：
 - 傳入參數名稱 a
 - 回傳 (return) a.canSwim() 的執行結果
- 所以使用「a -> a.canJump()」來表達，這樣就不需要建立CheckCanJump類別。
- 也可以視需要改用「a -> a.canSwim()」
- 所以只需要一個Functional Interface來定義方法規格。實做類別不再需要。

Lambda 表示式的使用方式

- Lambda 表示式必須含 3 部分：
 - 1) Argument List : 方法參數
 - 2) Arrow Token : ->
 - 3) Body : 方法內容
- 範例：
 - `(int x, int y) -> x + y`
 - `(x, y) -> x + y`
 - `(x, y) -> { system.out.println(x + y); }` //body超過1行時
 - `(String s) -> s.contains("word")`
 - `s -> s.contains("word")`

範例	結果	說明
<code>(int x, int y) -> x + y</code>	OK	
<code>(x, y) -> x + y</code>	OK	Java自動推斷參數，可以省略
<code>(x, y) -> { system.out.println(x + y);}</code>	OK	body超過1行時
<code>(String s) -> s.contains("word")</code>	OK	
<code>s -> s.contains("word")</code>	OK	只有1個參數時，可以去除()
<code>() -> true</code>	OK	方法不帶參數
<code>a, b -> a.startsWith("test")</code>	NG	2個以上參數需有()
<code>a -> a.startsWith("test");</code>	OK	不論方法是void或回傳boolean，都OK
<code>a -> { return a.startsWith("test");}</code>	NG	方法內容加上{}後，每行程式碼都要有分號(;)區隔
<code>(a, b) -> { int a = 0; return 5;}</code>	NG	方法內容裡的區域變數名稱不能和參數相同
<code>(a, b) -> { int c = 0; return 5;}</code>	OK	



3/3

使用內建的 **Functional Interfaces**