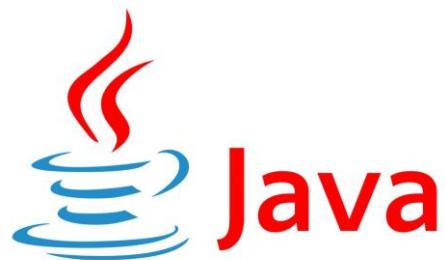




Assignment 3



歡迎各位同學來到作業三！這份作業將致力於訓練同學們掌握電腦科學中的重要概念。在這個過程中，你將學習如何建立基本的**類別**（class），深入了解**物件導向**程式設計的概念，並掌握如何運用**類別繼承**（class inheritance）的技巧。

如果作業卡關 **歡迎與助教討論**，stanCode 也非常鼓勵同學們互相討論作業的概念，但請不要把自己的 code 紿給任何人看，分享您的 code 會剝奪其他學生獨立思考的機會，同時會導致其他學生的程式碼與您的 code 極度相似，使得防抄襲軟體認定有抄襲之嫌疑。

Problem 1- Polymorphism

假設執行位於 **TestPeople** 中的程式碼，您會看到什麼結果？

請將答案詳細描述，如果發生錯誤，請寫 **Error**。

請在開始之前，觀看 **Problem 1** 說明影片：

<https://youtu.be/0eba2wV-q7A>

```
public class TestPeople{
    public static void main(String[] args){
        Person n = new Person("Neil", 12);
        Person a = new Grandma("Ada", 60);
        Grandma v = new Grandma("Vidya", 80);
        n.greet(a);
        n.greet(v);
        v.greet(a);
        v.greet((Grandma)a);
        a.greet(n);
        a.greet(v);
        ((Grandma) a).greet(v);
        ((Grandma) n).greet(v);
    }
}
```

```
public class Person{
    public String name;
    public int age;

    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }

    public void greet(Person other){System.out.println("Hello, "+other.name);}
}

public class Grandma extends Person{
    public Grandma(String name, int age){
        super(name, age);
    }
    @Override
    public void greet(Person other){System.out.println("Hello, young whippersnapper");}

    public void greet(Grandma other){System.out.println("How was bingo, "+other.name+"?");}
}
```

Problem 2 - Textbook.java

Book class 用於儲存有關書籍的資訊，以下是部分 **Book class** 的定義：

```
public class Book
{
    /** The title of the book */
    private String title;

    /** The price of the book */
    private double price;

    /** Creates a new Book with given title and price */
    public Book(String bookTitle, double bookPrice)
    { /* implementation not shown */ }

    /** Returns the title of the book */
    public String getTitle()
    { return title; }

    /** Returns a string containing the title and price of the Book */
    public String getBookInfo()
    {
        return title + " - " + price;
    }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

您將撰寫一個名為 **Textbook** 的 class，它是 **Book class** 的 subclass。

一個 **Textbook class** 具有一個版本編號，為一個正整數，用來識別書籍的不同版本。當在 **Textbook class** 呼叫 **getBookInfo method** 時，它會回傳一個包含**版本資訊的字串**，如下表所示。

有關書名和價格的資訊必須保持在 **Book class** 中，而有關版本編號的資訊必須保持在 **Textbook class** 中。

Textbook class 含有了一個額外的 **canSubstituteFor method**，當一 **Textbook** 能有效替代另一 **Textbook** 時即回傳 **true**；若否，則回傳 **false**。如果目前的 **Textbook** 和 **canSubstituteFor method** 引用的 **Textbook** 參數具有**相同的名稱**，並且目前的 **Textbook** 版本編號**大於等於** **Textbook** 參數的**版本編號**，則**目前的 Textbook 即可有效替代另一 Textbook**。

以下表格包含了範例程式碼的執行程序以及相應的結果，這些程式碼的執行程序出現在一個除了 **Book** 或 **Textbook** 之外的 class 中。

Statement	回傳值 (若無值為空白)	class 規範
<code>Textbook bio2015 = new Textbook("Biology" , 49.75, 2);</code>		<code>bio2015</code> 是一標題為 “Biology”，價格為 49.75 美元，版本編號為 2 的 Textbook 。
<code>Textbook bio2019 = new Textbook("Biology" , 39.75, 3);</code>		<code>bio2019</code> 是一標題為 “Biology”，價格為 39.75 美元，版本編號為 3 的 Textbook 。
<code>bio2019.getEdition();</code>	<code>3</code>	回傳版本編號。
<code>bio2019.getBookInfo();</code>	“Biology-39.75-3”	回傳包含 <code>bio2019</code> 標題、價格和版本編號的格式化字串。
<code>bio2019.</code> <code>canSubstituteFor(bio2015);</code>	<code>true</code>	<code>bio2019</code> 能有效替代 <code>bio2015</code> ，因為它們的標題相同，而且 <code>bio2019</code> 的版本編號大於或等於 <code>bio2015</code> 的版本編號。
<code>bio2015.</code> <code>canSubstituteFor(bio2019);</code>	<code>false</code>	<code>bio2015</code> 不能有效替代 <code>bio2019</code> ，因為 <code>bio2015</code> 的版本編號小於 <code>bio2019</code> 的版本編號。
<code>Textbook math = new Textbook("Calculus" , 45.25, 1);</code>		<code>math</code> 是一標題為 “Calculus”，價格為 45.25 美元，版本編號為 1 的 Textbook 。
<code>bio2015.</code> <code>canSubstituteFor(math);</code>	<code>false</code>	<code>bio2015</code> 不能有效替代 <code>math</code> ，因為 <code>bio2015</code> 的標題與 <code>math</code> 的標題不同。

請撰寫完整的 **Textbook** class，您的執行結果須符合所有的規範及表格的範例。

Problem 3 - HiddenWord.java

猜字遊戲中，玩家要試圖猜出一個隱藏的英文單詞，這個單詞全為大寫字母，而且玩家知道它的長度。玩家的**每次猜測也只能有大寫字母，並且要與隱藏單詞的長度相同**。

在進行猜測之後，玩家會得到一個提示，這個提示為隱藏的單詞和玩家的猜測之間的比較。提示中的每個位置都包含一個符號，該符號對應於猜測中相同位置的字母。以下規則決定提示中出現的符號：

若猜測的字母...	提示內對應的符號為
與隱藏單詞的位置相同	正確的字母
有在隱藏單詞內，但位置不同	'+'
沒有出現在隱藏單詞內	'*'

HiddenWord class 將用於表示遊戲中的隱藏單詞。隱藏單詞會被傳到該 constructor，其含有一個 **getHint method**，**可以接收猜測並生成提示**。

例如，假設一變數 **puzzle** 被如此宣告：

```
HiddenWord puzzle = new HiddenWord( "HARPS" );
```

以下表格顯示了幾個猜測及其產生的提示：

呼叫 getHint	回傳字串
puzzle.getHint("AAAAA")	"+A+++"
puzzle.getHint("HELLO")	"H****"
puzzle.getHint("HEART")	"H*++*"
puzzle.getHint("HARMS")	"HAR*S"
puzzle.getHint("HARPS")	"HARPS"

請撰寫完整的 **HiddenWord class**，包含任何必要的 instance variables、其 constructor 及上述描述的 **getHint method**。您可以假設猜測的長度與隱藏單詞的長度相同。

Problem 4 - FrogSimulation.java

這個問題為一隻青蛙在一條直線上跳躍的模擬：青蛙試圖在指定的跳躍次數內跳到目標位置。該模擬被涵蓋在以下的 **FrogSimulation** class 中。您需要撰寫此 class 中的兩個 method。

```
public class FrogSimulation
{
    /** Distance, in inches, from the starting position to the goal. */
    private int goalDistance;

    /** Maximum number of hops allowed to reach the goal. */
    private int maxHops;

    /** Constructs a FrogSimulation where dist is the distance, in inches, from the starting
     *  position to the goal, and numHops is the maximum number of hops allowed to reach the goal.
     *  Precondition: dist > 0; numHops > 0
     */
    public FrogSimulation(int dist, int numHops)
    {
        goalDistance = dist;
        maxHops = numHops;
    }

    /** Returns an integer representing the distance, in inches, to be moved when the frog hops.
     */
    private int hopDistance()
    { /* implementation not shown */ }

    /** Simulates a frog attempting to reach the goal as described in part (a).
     *  Returns true if the frog successfully reached or passed the goal during the simulation;
     *  false otherwise.
     */
    public boolean simulate()
    { /* to be implemented in part (a) */ }

    /** Runs num simulations and returns the proportion of simulations in which the frog
     *  successfully reached or passed the goal.
     *  Precondition: num > 0
     */
    public double runSimulations(int num)
    { /* to be implemented in part (b) */ }
}
```

(a) 請撰寫 **simulate** method，模擬青蛙在一條直線上從起始位置 **0** 跳到目標位置的過程，並限制最大跳躍次數。如果青蛙在最大跳躍次數內成功到達目標位置，回傳 **true**；若否，則回傳 **false**。

FrogSimulation class 提供了一個名為 **hopDistance** 的 method，此 method 會回傳一個整數，表示青蛙在跳躍時要移動的距離（可以是正數或負數）。正數的距離表示向目標移動，負數的距離表示遠離目標。回傳的距離在每次呼叫時可能會有所不同。每次青蛙跳躍時，它的位置都會根據 **hopDistance** method 的回傳值進行調整。

青蛙會跳躍直到發生任一以下條件：

- 青蛙已經到達或超過了目標位置
- 青蛙已經到達了一個小於 **0** 的位置
- 青蛙已經進行了最大次數的跳躍，但沒有到達目標位置

下面範例展示了一個 **FrogSimulation** object 的宣告，其中目標距離為 **24** 英寸，最大跳躍次數為 **5**。表格顯示了呼叫 **simulate** method 的一些可能結果：

FrogSimulation sim = new FrogSimulation(24, 5);

	hopDistance() 回傳值	青蛙最終位置	sim.simulate() 回傳值
Example 1	5, 7, -2, 8, 6	24	true
Example 2	6, 7, 6, 6	25	true
Example 3	6, -6, 31	31	true
Example 4	4, 2, -8	-2	false
Example 5	5, 4, 2, 4, 3	18	false

(b) 請撰寫 **runSimulations** method，執行一定數量的模擬，並回傳青蛙成功到達或超過目標的比例。例如，如果傳遞給 **runSimulations** 的參數是 **400**，且呼叫 **400** 次 **simulate** method 中，有 **100** 次回傳 **true**，則 **runSimulations** method 應該回傳 **0.25**。

請在下方完成 **runSimulations** method。假設 **simulate** method 按說明執行，並先不論您在(a)部分中寫的內容，您必須適當地使用 **simulate** method 以獲得全部的分數。

Problem 5 - WordMatch.java

這個問題有關 **WordMatch** class，該 class 儲存一個秘密字串，並提供了比較其他字串與秘密字串的 method。您需要在 **WordMatch** class 中撰寫兩個 method。

```
public class WordMatch
{
    /** The secret string. */
    private String secret;

    /** Constructs a WordMatch object with the given secret string of lowercase letters. */
    public WordMatch(String word)
    {
        /* implementation not shown */
    }

    /** Returns a score for guess, as described in part (a).
     *  Precondition: 0 < guess.length() <= secret.length()
     */
    public int scoreGuess(String guess)
    { /* to be implemented in part (a) */ }

    /** Returns the better of two guesses, as determined by scoreGuess and the rules for a
     *  tie-breaker that are described in part (b).
     *  Precondition: guess1 and guess2 contain all lowercase letters.
     *      guess1 is not the same as guess2.
     */
    public String findBetterGuess(String guess1, String guess2)
    { /* to be implemented in part (b) */ }
}
```

(a) 請撰寫 **WordMatch** class 的 **scoreGuess** method。為了確定要回傳的分數，**scoreGuess** method 會找出 **guess** 在 **secret** 中作為子字串出現的次數，然後將該次數乘以 **guess** 長度的平方。**guess** 可以在 **secret** 中重複出現。

請假設 **guess** 的長度小於或等於 **secret** 的長度，且 **guess** 不是空字串。

以下是 **WordMatch** object 的宣告範例。表格顯示了對 **scoreGuess** method 進行一些可能的呼叫後的結果。

WordMatch game = new WordMatch("mississippi");

guess 值	子字串 出現次數	分數計算： (子字串出現次數) x (guess 長度的 平方)	game.scoreGuess(guess) 回傳值
'i'	4	$4 * 1 * 1 = 4$	4
"iss"	2	$2 * 3 * 3 = 18$	18
"issipp"	1	$1 * 6 * 6 = 36$	36
"mississippi"	1	$1 * 11 * 11 = 121$	121

WordMatch game = new WordMatch("aaaabb");

guess 值	子字串 出現次數	分數計算： (子字串出現次數) x (guess 長度的平方)	game.scoreGuess(guess) 回傳值
'a'	4	$4 * 1 * 1 = 4$	4
"aa"	3	$3 * 2 * 2 = 12$	12
"aaa"	2	$2 * 3 * 3 = 18$	18
"aabb"	1	$1 * 4 * 4 = 16$	16
'c'	0	$0 * 1 * 1 = 0$	0

(b) 請撰寫 **WordMatch** class 的 **findBetterGuess** method，該 method 根據兩個 **String** 參數 **guess1** 和 **guess2** 回傳更好的猜測。如果 **scoreGuess** method 對 **guess1** 和 **guess2** 回傳**不同的值**，則回傳具有較高分數的猜測；如果 **scoreGuess** method 對 **guess1** 和 **guess2** 回傳**相同的值**，則回傳字母順序較大的猜測。

以下是 **WordMatch** object 的宣告範例，以及對 **scoreGuess** 和 **findBetterGuess** method 進行一些可能的呼叫後的結果。

```
WordMatch game = new WordMatch( "concatenation" );
```

呼叫 method	回傳值	說明
game.scoreGuess("ten");	9	1 * 3 * 3
game.scoreGuess("nation");	36	1 * 6 * 6
game.findBetterGuess("ten" , "nation");	"nation"	由於 scoreGuess method 對於 "nation" 回傳 36，而對於 "ten" 回傳 9，因此具有較高分數的猜測 "nation" 將被回傳。
game.scoreGuess("con");	9	1 * 3 * 3
game.scoreGuess("cat");	9	1 * 3 * 3
game.findBetterGuess("con" , "cat");	"con"	由於 scoreGuess method 對於 "con" 和 "cat" 都回傳 9，因此字母順序較大的猜測 "con" 將被回傳。

C/C++ 作業講解影片

以下提供 C/C++ 版本的作業講解影片，透過觀看，同學們可以輕鬆地轉換至不同的語言環境，同時深入瞭解不同語言作業所需的程式概念和語法。相信將為同學提供更廣泛的學習視野。

影片連結：<https://youtu.be/N9Om4c1RPaw>

希望這份資源對同學們的學習旅程有所助益，並取得更好的學習成果！

stanCode

Should you have any idea or questions, please feel free to contact.