整理:

Abstraction:

1.抽象類別和抽象方法 都必須使用abstract關鍵字來修飾才行

2.抽象類別可以包含非抽象方法

3.使用抽象定義抽象類別，不可以宣告為 final class

4.抽象類別不能夠用new關鍵字來產生物件

5.抽象方法只需要宣告不需要實作

6.專門被當作父類別的類別可以當作範本的作用

7.只能被繼承extends的關鍵字 抽象類別的子類別必須要實作父類別的抽象方法，不然子類別還是個抽象類別

8.抽象方法的封裝等級不能夠是private，而必須為public/protected/(default)，abstract 回傳值型別 抽象方法名稱(參數列);而沒有實體的方法必須以 ; 做為結束

9.抽象方法的類別必須要宣告是抽象類別，但是抽象類別不一定要有抽象方法

Good uses:

Class fruit {apple appleB; banana bananaB; watermelon watermelonB;}

Class apple {int name;}

Class banana {int name;}

Class watermelon{int name;}

此例子把物件分門別類且正確的命名

Bad uses:

Class fruit{ int apple; int banana; int watermelon;}

此例子沒把物件分門別類且命名隨便，沒有達到抽象化要求。

資料來源:

https://coffee0127.github.io/blog/2016/09/10/abstract-class-vs-interface/

http://jimmu-jimmu.blogspot.com/2012/10/abstract-classinterface.html

--------------------------------------------------------------------------------------------------------------------------------------------------

encapsulation

1. 屬性封印在類別中

2. 良好的方裝可以減少耦合

3. 類內部的結構可以自我修改

4. 可以對成員變數作更精確的控制

5. 隱藏信息也是一種防止外界呼叫隨意存取類別的屬性也就是說只讓類別的屬性給同個類別的方法存取，去存取物件內部實作細節的方法

Good uses:

Class fruit {apple appleB; banana banana B; watermelon watermelon B;}

Class apple {int name; float length; float area; }

Class banana {int name; float length; float area; }

Class watermelon{int name; float length; float area; }

此例子把物件分門別類且正確的命名，並正確的包裝再一起，並加以使用。

Bad uses:

Class fruit{ int appleB; float appleBLength;float appleBArea;

int bananaB; float bananaBLength;float bananaBArea;

int watermelonB; float watermelonBLength;float watermelonBArea;}

此例子把所有形狀資訊都塞在一起，沒有正確的包裝，導致資訊凌亂。

資料來源:

https://www.runoob.com/java/java-encapsulation.html

https://zh.wikipedia.org/wiki/%E5%B0%81%E8%A3%9D\_(%E7%89%A9%E4%BB%B6%E5%B0%8E%E5%90%91%E7%A8%8B%E5%BC%8F%E8%A8%AD%E8%A8%88)

http://kaiching.org/pydoing/java-guide/unit-09-encapsulationand-constructor.html

(貼圖) https://sites.google.com/a/mis.nsysu.edu.tw/cheng-shi-zi-xun-wang/java-jin-jie-pian/wu-jian-dao-xiang/3-shen-me-shi-feng-zhuang

----------------------------------------------------------------------------------------------------------------------------------------------------

Cohesion

1.執行某個功能所需要用到的程式與資料都塞在某一個模組（function, class, package, etc）之中，使得該模組可以被視為一個單獨的個體執行，那麼這個模組的內聚就愈高，就是把程式，資料這些有的沒的東西都『聚在一起』打包起來。

2.設計一個模組的基本精神，就是要『提高內聚力，降低耦合度』

3.提升內聚力好處就是要提升模組的『獨立性』，也就是說這個模組可以被單獨使用，也可以被單獨修改，可以被單獨使，就表示模組被『重複使用（reuse）』的機會變多了；可以被單獨修改，就表示開發人員可以放心大膽的修改模組而不用怕萬一改了模組之後會引發『漣波效應（ripple effect）』影響到其他原本功能正常的模組（就是可以避免不小心改一的地方錯十個地方）。

4.Higher cohesion(高凝聚性)想表達的其實就是事物凝聚的狀態或程度。

5一個類別擁有單一目標的程度

6.擁有高級內聚性的類別，越少被更動，越常被使用

7.下的情形會降低程式的內聚性：

(1)很多性能封裝在同一個型別內，可藉由方法供應外界使用，但性能彼此類似的地方不是很多。

(2)在方法中進行許多不同性能，使用的是相關性很低或不相關的資料。

低內聚性的缺點如下：

(1)提升理解模組的困難度。

(2)提升保護系統的困難度，因為一個邏輯修改會影響到很多模組，而一個模組的修改會把一些相關模組也修改。

(3)提升模組重複用的困難度，因為大部份的應用程式無法重複用一個由很多個不一定相關的性能組成的模組。

Good uses:

Class fruit {apple appleB; banana banana B; watermelon watermelon B;}

Class apple {int name; float length; float area; }

Class bananale {int name; float length; float area; }

Class watermelon{int name; float length; float area; }

此例子把物件分門別類且正確的命名，並正確的包裝再一起，並且可各別使用。

Bad uses:

Class fruit{ int appleB; float appleBLength;float appleBArea;

int bananaB; float bananaBLength;float bananaBArea;

int watermelonB; float watermelonBLength;float watermelonBArea;}

此例子把所有形狀資訊都塞在一起，沒有正確的包裝，導致資訊凌亂。

資料來源:

http://teddy-chen-tw.blogspot.com/2011/12/1.html

https://dotblogs.com.tw/jerryhuang0306/2016/03/08/113825

http://chuchu1313.blogspot.com/2017/11/day2-10-coupling-and-cohesion.html

https://zh.wikipedia.org/wiki/%E5%85%A7%E8%81%9A%E6%80%A7\_(%E8%A8%88%E7%AE%97%E6%A9%9F%E7%A7%91%E5%AD%B8)

-------------------------------------------------------------------------------------------------------------------------------------------------

Coupling

1.如果某一個模組跟另一個模組有關係性那麼這兩個模組就會耦合。

2.設計一個模組就是要提升內聚力，降低耦合度

3.低耦合度(低相依性)想表達的是程式單元之間牽連相依的程度，模組之間的相依性越寬鬆愈好。

4.類別A對類別B認識程度; 類別間如何互動

(1)若A只知道B暴露的interface，則為寬鬆耦合 (好)

(2)若A知道除了B暴露的interface以外的東西(包含有哪些實體變數)，則為緊耦合 (不好)

(3)若A依賴B class內的某個部份，則為緊耦合 (不好)

(4)若A知道B如何實作則為緊耦合 (不好)

(5)若A修改了和interface沒有關係的程式碼，卻影響到B，則為不好的設計(A可能根本不知道B用了A內部的東西)。如此的修改會造成B程式碼的破壞。

(6)若B封裝內的get, set method止依賴A內method，則為緊耦合(不好)

(7)寬鬆耦合將會有良好的封裝

(8)理想上，OO系統內object的互動都應該要透過型別的API，而一個良好的API應該要有很好的封裝才行。

5.性能設計是一種可以降低耦合性的方法，此方法以機能性的角度設法限制各模組需負責的事務。在類別A及B之間，若有以下任何一種情形，會提高二者的耦合性：

(1)A有一個屬性是參考類別B（此屬性的形態為類別B）

(2)A呼叫物件B提供的服務

(3)A有一個方法會參考類別B（此方式會傳回一形態為類別B的物性）

(4)A是類別B的子類別。

Good uses:

Class fruit {apple appleB; banana banana B; watermelon watermelon B;}

Class banana {int name; float length; float area; }

Class triangle {int name; float length; float area; }

Class watermelon{int name; float length; float area; }

此例子把物件分門別類且正確的命名，並正確的包裝再一起，並且可各別使用

Bad uses:

Class fruit{ circle circleB; triangle triangle B; watermelon watermelon B;}

Class apple{int name;float length;float area;}

Class banana{int name; float length;float area; }

Class watermelon{int name; float length;float area; }

Class appleRank{float rank;}

Class banana{ float rank;}

Class watermelon{ float rank;}

此例子有重複的物件類型，可以合而為一，才不會有Coupling狀況。

資料來源:

http://teddy-chen-tw.blogspot.com/2011/12/1.html

https://dotblogs.com.tw/jerryhuang0306/2016/03/08/113825

http://chuchu1313.blogspot.com/2017/11/day2-10-coupling-and-cohesion.html

https://zh.wikipedia.org/wiki/%E8%80%A6%E5%90%88%E6%80%A7\_(%E8%A8%88%E7%AE%97%E6%A9%9F%E7%A7%91%E5%AD%B8)