abstract：抽象類

1. 可以用來修飾的結構：類、方法
2. 具體的：

abstract修飾類：抽像類

\* > 此類不能實例化

\* > 抽像類中一定有構造器，便於子類實例化時調用（涉及：子類對象實例化的全過程）

\* > 開發中，都會提供抽像類的子類，讓子類對象實例化，完成相關的操作　->抽象的使用前提:繼承性。

abstract修飾方法：抽象方法

\* > 抽象方法只有方法的聲明，沒有方法體

\* > 包含抽象方法的類，一定是一個抽像類。反之，抽像類中可以沒有抽象方法的。

\* > 若子類重寫了父類中的所有的抽象方法後，此子類方可實例化，若子類沒有重寫父類中的  
　　　 所有的抽象方法，則此子類也是一個抽像類，需要使用abstract修飾

3. 注意點：

1.abstract不能用來修飾：屬性、構造器等結構

2.abstract不能用來修飾私有方法、靜態方法、final的方法(因為不能被重寫)、final的類

4.abstrack的應用舉例:

舉例一:



舉例二:

(加上abstrcak是為了不讓他造對象,讓其子類去重寫抽象方法,然後再去實現方法)

abstack class GeometricObject{

public abstrack double findArea();

}

public class Circle extends GeometricObject {

private double radius;

public double findArea(){

return 3.14 \* radius \* radius;

}

}

舉例三:IO流中設計到的抽象類:InputStream / OutputStream / Reader / Writer。在其內部定義了抽象的read()、write()方法。

模板方法的設計模式

1.解決的問題  


2. 舉例

**abstract** **class** Template{

//計算某段代碼執行所需要花費的時間

**public** **void** spendTime(){

**long** start = System.*currentTimeMillis*();

**this**.code();//不確定的部分、易變的部分

**long** end = System.*currentTimeMillis*();

System.***out***.println("花費的時間為：" + (end - start));

}

**public** **abstract** **void** code();

}

**class** SubTemplate **extends** Template{

@Override

**public** **void** code() {

**for**(**int** i = 2;i <= 1000;i++){

**boolean** isFlag = **true**;

**for**(**int** j = 2;j <= Math.*sqrt*(i);j++){

**if**(i % j == 0){

isFlag = **false**;

**break**;

}

}

**if**(isFlag){

System.***out***.println(i);

}

}

}

1. 應用場景

