昨日回顾.....2
设计模式.....3
单例模式.....3
实现MysqlDB类的单例模式及完整功能：.....4
资源和对象的辨析.....5
抽象类，抽象方法.....6
抽象类.....6
抽象方法：.....6
抽象类抽象方法细节关系描述.....7
PHP中的重载技术.....7
通常面向对象语言的重载技术.....7
属性重载.....8
方法重载.....9
接口interface.....10

昨日回顾

类的继承

基本概念：

继承：从上级类获取其特性信息（属性和方法）的过程。B<<==A，B继承自A

派生：A==>>B，从A派生出新的类B，B具有A的“几乎所有特性”，并可以具有自己的特性。

父类/子类，基类/派生类，上级类，下级类。

单继承：php中，类的继承模式只能是从一个父类继承。java，c#也都如此。

扩展：B从A继承了一些特性，并额外又加上了自己的一些特性，这种现象就可以称为扩展。

访问（权限）修饰符

在成员（属性/方法/常量）前面，可以加上访问修饰符：public, protected, private

所谓访问，就是类似这样的语法形式：

对象->属性/方法；

类::属性/方法；

访问的位置（范围）区别：

A一个类的内部（肯定是方法内）

B一个类的继承关系类的内部（也是方法内）

C一个类的外部。

public公有的：

protected 受保护的

private私有的

范围	A类的内部	B继承关系类中	C类的外部
public公有的	可以	可以	可以
protected受保护的	可以	可以	不
private私有的	可以	不	不

注意：

1，对于方法，如果省略访问修饰符，默认按public算

2，对于普通属性，不能省略，但var代表public

3，对于静态属性，可以省略，省略按public算

4，对于常量，默认都是public

parent代表父类：

可以在子类中使用它，以此代表父类去访问父类中的成员。

parent::属性，表示获取父类的该属性。

parent::方法()：表示调用父类的该方法。此时就可能会出现一个情形：

该父类方法中，可能会用到$this关键字，表示“当前对象”，但实际上此时表示的当前对象，并非是父类的对象，而是调用该方法的子类的对象。

此用法通常用于子类覆盖了父类的方法但又需要使用父类的同名方法的时候。

构造方法析构方法在继承中的表现：

如果没有定义，则他们都会自动调用父类的同名方法。

如果定义了，则就不会自动调用了，但可以在其中“手动调用”，类似这个模式：

```
parent::__constuct(...)
parent::__destruct()
```

重写override

子类定义跟父类同名的属性或方法，就是重写。

重写的基本要求：

1，访问权限修饰不能低于父类（必须同样级别或更开放）

2，对于方法，其形参必须跟父类一样。

最终类final class：不能（不允许）继承的类

```
final  class A{....}
```

最终方法final method：不能（不允许）覆盖的方法

```
finale  function  f1(){....}
```

设计模式

什么叫设计模式

设计模式就是应用中面对某种特定情形的问题而设计出来的某种常见的有效的解决办法，是经验的总结。

工厂模式：

就是设计来专门“生产”类的对象的一种结构模式。

设计模式

工厂模式：

单例模式

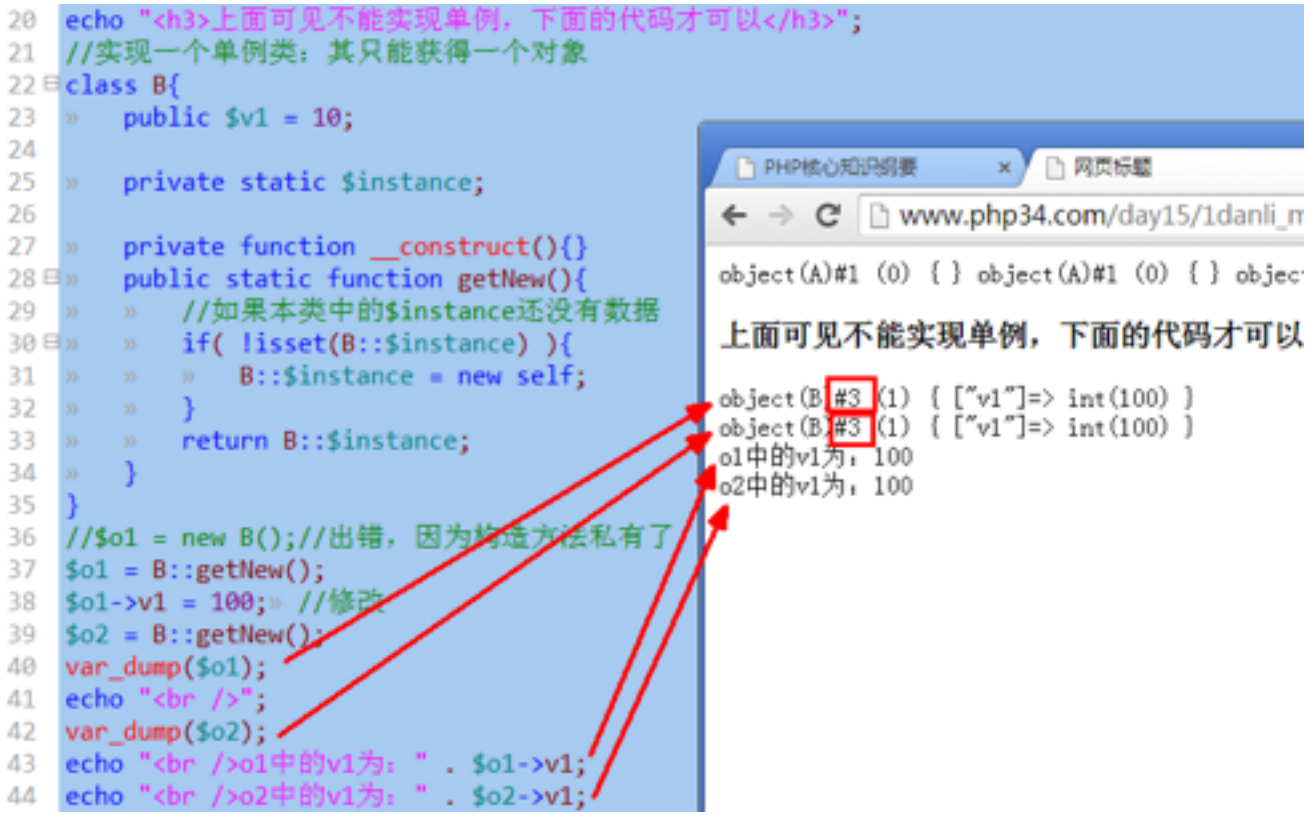
应用中的某种需求：

对于某些类，在使用它的时候，从头到尾（程序运行的开始到结束），都只需要一个对象，就可以完成所有任务。

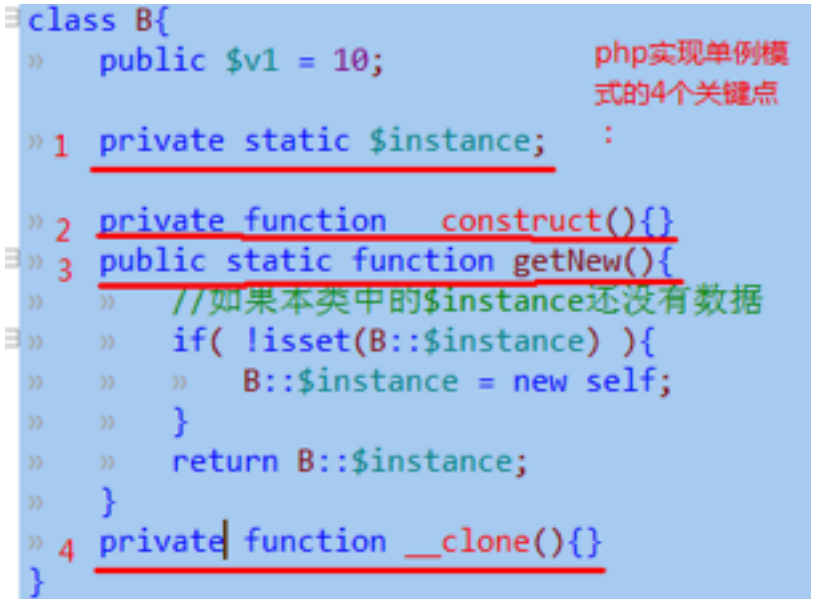
单例：

某个类，只允许其“创建”出一个对象，即使去进行多次创建，也只能得到一个对象。

```
$obj1 = new A();
$obj2 = $obj1;       //此时还是1个对象，$obj2只是最终指向了跟$obj1同样的对象
$obj3 = new A();    //这时候才有了第二个对象。
```



但其实在php语言中，一个对象还可以使用clone运算符进行克隆，则就也会“生成”新对象。
因此，还需要在类中使用私有化措施来禁止克隆，最终，单例模式的实现如下：



实现MysqlDB类的单例模式及完整功能：

详细参看：《MySQLDB.class.php》

资源和对象的辨析

资源，迄今为止，我们只学过2个（种）资源：

$link = mysql_connect(“localhost”, “root”, “123”);　　//得到一个“连接到mysql数据库”的资源。

$result = mysql_query(“select .....”）；　　//这个是结果集资源

```php
<?php
$link = mysql_connect("localhost","root", "123");
mysql_query("set names utf8");
mysql_query("use php34;");
                    //得到一个“连接到mysql数据库”的资源。
                    //资源的理解：一个外部本来就有的“对象”（数据）
                //程序中，该资源变量只是一个“指向”该对象（数据）的标示符
var_dump($link);

$result = mysql_query("select * from money");//这是结果集资源
                    //对该结果集进行“fetch”操作才能获得php数据（数组）
echo "<br />";
var_dump($result);

class A{}
$o1 = new A();      //对象是我们通常代码完整创造出来。
echo "<br />";
var_dump($o1);

$arr = array(3, 'abc', true);
echo "<br />";
var_dump($arr);
?>
```

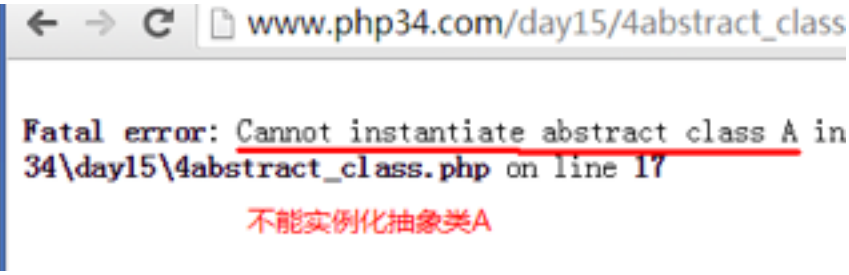
```
resource(3) of type (mysql link)
resource(4) of type (mysql result)
object(A)#1 (0) { }
array(3) { [0]=> int(3) [1]=> string(3) "abc" [2]=> bool(true) }
```

抽象类，抽象方法

抽象类

在正常定义类的前面，加上关键字：abstract，那就构成抽象类。
abstract class 类名{.....类的定义.....}

```php
<?php
abstract class A{
»
}
$o1 = new A();
var_dump($o1);
?>
```



可见，抽象类有什么用？
抽象类可以用来规范一些类的共同特性，但又不需要去对其进行实例化。
怎么规范：继承它。
也就是说，抽象类的使命是专门做“父类”：子类就继承了它的特性，这就是“规范作用”

抽象方法：

抽象方法是一个没有方法体（也不含大括号）的方法定义“头”而已。
前面需要加上abstract。
比如：abstract function f1($x1, $y, $m)；//注意，后面一定有分号。
抽象方法有什么用？
其实跟抽象类一样，配合抽象类，来实现对下级类的“行为规范”。
即相当于要求下级类去完成该功能（动作），但自己是不做的。


```php
//游戏中的“怪”类，抽象的！
abstract class guai{
    public $blood = 100;
    protected $distance = 30;  //开始发动攻击的距离
    protected abstract function attack();//攻击行为，抽象的
    //这个类不能实例化，也就是不能做出一个“怪”对象
    //但他规定了下级类（蛇怪，虎怪）的一些特性信息
    //其中有个特性行为，就是“攻击”
}
class snakeGuai extends guai{
    //
    function attack(){
        echo "<br />吐火攻击";
        $this->blood--;
    }
}
$o1 = new snakeGuai();
var_dump($o1);
echo "<br />";
$o1->attack(); //蛇攻击一次
echo "<br/>该蛇怪剩余的血为：" . $o1->blood;
```

抽象类抽象方法细节关系描述

1，如果一个方法定义为抽象方法，则其所在的类必须定义为抽象类。
2，但，一个抽象类中，可以没有抽象方法——但通常意义不大。
3，子类继承自一个抽象类，则子类必须实现父类中的所有抽象方法，除非子类也继续作为抽象类
4，子类实现抽象父类的方法时，访问控制修饰符的范围不能降低，且方法的参数也须一致——其实这就是重写，所以要满足重写的要求。

PHP中的重载技术

通常面向对象语言的重载技术

其基本语法是这样的：

在一个类中，有多个同名的方法，每个方法的参数不同而已。这种现象就称为“重载”。

参数不同可以是：数量个数不同，或类型不同，或顺序不同。

比如：

```
class A{
```

```
        int function f1(int x){.....}
        int function f1(int x, int y){.....}
        int function f1(string s  int m){....}
    }
```

但，在php中，一个类中，根本就不可以定义多个同名方法——这直接是语法错误。

实际上，php中的重载，是另一个“概念”，其指的是：

属性重载：　如果使用一个不存在的属性，就会去自动调用类中预先定义好的某个方法以处理数据；

方法重载：　如果使用一个不存在的方法，就会去自动调用类中预先定义好的某个方法以处理该行为

属性重载

属性有哪些使用情形？其实跟变量一样，只有 4 种使用情形：

取值：$v1 = 对象->属性;

赋值：对象->属性 = XX值;

判断是否存在：isset(对象->属性;)

销毁：unset(对象->属性;)

所谓属性重载，就是在面对上述4种情形的属性使用场景中，该对象如果来“应对”的问题。

如果某属性不存在，但在语法中使用如下情形，则会发生：

取值：$v1 = 对象->属性;　　===>自动调用类中的__get()方法

赋值：对象->属性 = XX值;　　===>自动调用类中的__set()方法

判断是否存在：isset(对象->属性;)　　===>自动调用类中的__isset()方法

销毁：unset(对象->属性;)　　===>自动调用类中的__unset()方法

前提都是：类中要预先定义好这些方法。

通常，没有的属性，去使用，显然是报错：

```
14 class A{
15     public $p1 = 1;
16     /*
17     function __get( $){
18     
19     }
20     */
21 }
22 $o1 = new A();
23 echo "<br />o1的p1属性值为：" . $o1->p1;
24 echo "<br />o1的p2属性值为：" . $o1->p2;
25 ?>
```

www.php34.com/day15

o1的p1属性值为：1

Notice: Undefined property: A::$p2 in
34\day15\5shuxing_chongzai.php on 1

o1的p2属性值为：

先看属性的取值赋值（最常见情形）：

```php
class A{
    public $p1 = 1;
    public $propArr = array();//一个数组，用于存储“不存成的属性值”
    function __get( $prop_name ){
        if(isset($this->propArr[$prop_name])){
            return $this->propArr[$prop_name];
        }
        return "不存在属性$prop_name";
    }
    function __set( $prop_name, $value ){
        $this->propArr[$prop_name] = $value;
    }
}
$o1 = new A();
echo "<br />o1的p1属性值为：" . $o1->p1;//存在的属性
echo "<br />o1的p2属性值为：" . $o1->p2;//不存在的属性调用__get()
$o1->p2 = 2;    //给一个不存在的属性赋值，则会自动调用__set()
echo "<br />o1的p2属性值为：" . $o1->p2;//不存在的属性 调用__get()
$o1->p3 = 3; 不存在的属性，调用__set()
echo "<br />o1的p3属性值为：" . $o1->p3;//不存在的属性 调用__get()
```

对应，写出__iset(),和__unset()：

```php
    function __isset($prop_name){
        if(isset($this->propArr[$prop_name])){
            return true;
        }
        return false;
    }
    function __unset($prop_name){
        unset($this->propArr[$prop_name]);
    }
```

方法重载

当使用一个对象调用一个不存在的普通方法的时候，会自动去调用预先定义好的"__call"方法。其中，该方法必须带 2 个参数，见下图：

```php
function f1(){
    echo "<br />f1函数被执行（任务完成）";
}
function f2($x, $y){
    echo "<br />f1函数被执行（任务完成）";
    return $x+$y;
}
class A{
    public $p1 = 1;
    //当使用一个对象调用一个不存在的方法时，就会自动调用本方法
    //其中$name就是本来要调用的方法名
    //$array就是本来调用该方法时使用的实参数据，都装入该数组
    function __call($name, $array){
        $count = count($array);
        if($count==0){
            f1();
        }
        else if($count == 2){
            return f2($array[0], $array[1]);
        }
    }
}
$o1 = new A();
$o1->f1();    //不存在的方法，不带实参
$v1 = $o1->f1(1,2);//不存在的方法，并带2个实参
echo "<br />o1里面的p1=" . $o1->p1;
echo "<br />结果v1=" . $v1;
```

这个例子实现了其他面向对象语言中的常见的“重载技术”

当使用一个对象（类）调用一个不存在的静态方法的时候，会自动去调用预先定义好的 " _ _ callStatic " 方法。
其中，该方法必须带 2 个参数。其实跟前面一样！

上面所学的几个方法都被称为“魔术方法”:
__get(), __set(), __isset(), __unset(), __call(), __callstatic();

接口interface

类：有属性，有方法；有类常量；
抽象类：有属性，有方法；有类常量； 还可以有抽象方法（只有方法头）
“更”抽象类(就是接口)：有接口常量；还可以有抽象方法（只有方法头）

可见，接口中，只有两类最简单特性信息：

接口常量：　其实就是常量。
抽象方法：　只有方法头——下级自然就应该去“实现”它。

接口有什么用?
　　接口可以看作是抽象类的更高层面的“抽象规范”，不考虑接口常量的话，那么就相当于之规定了“下级类”要做什么——至于怎么就，没有规定。

接口也可以说，在一个抽象类中只有类常量和抽象方法的一种特例情形。
　　但是！！！
　　接口可以实现“多继承”（多实现），而类不行。

鸟：有翅膀，能飞;
鸵鸟是鸟吗?
一个类“继承”接口中的特性信息，被称为实现“implements），其实本质跟继承一样，即下级类就可以使用上级类的数据（方法）了。
但因为接口中的方法都是抽象的，则下级类就必须:
　　1，要么实现该抽象方法（写出其方法体）
　　2，要么也声明为抽象方法——等待更下一级的类去实现。

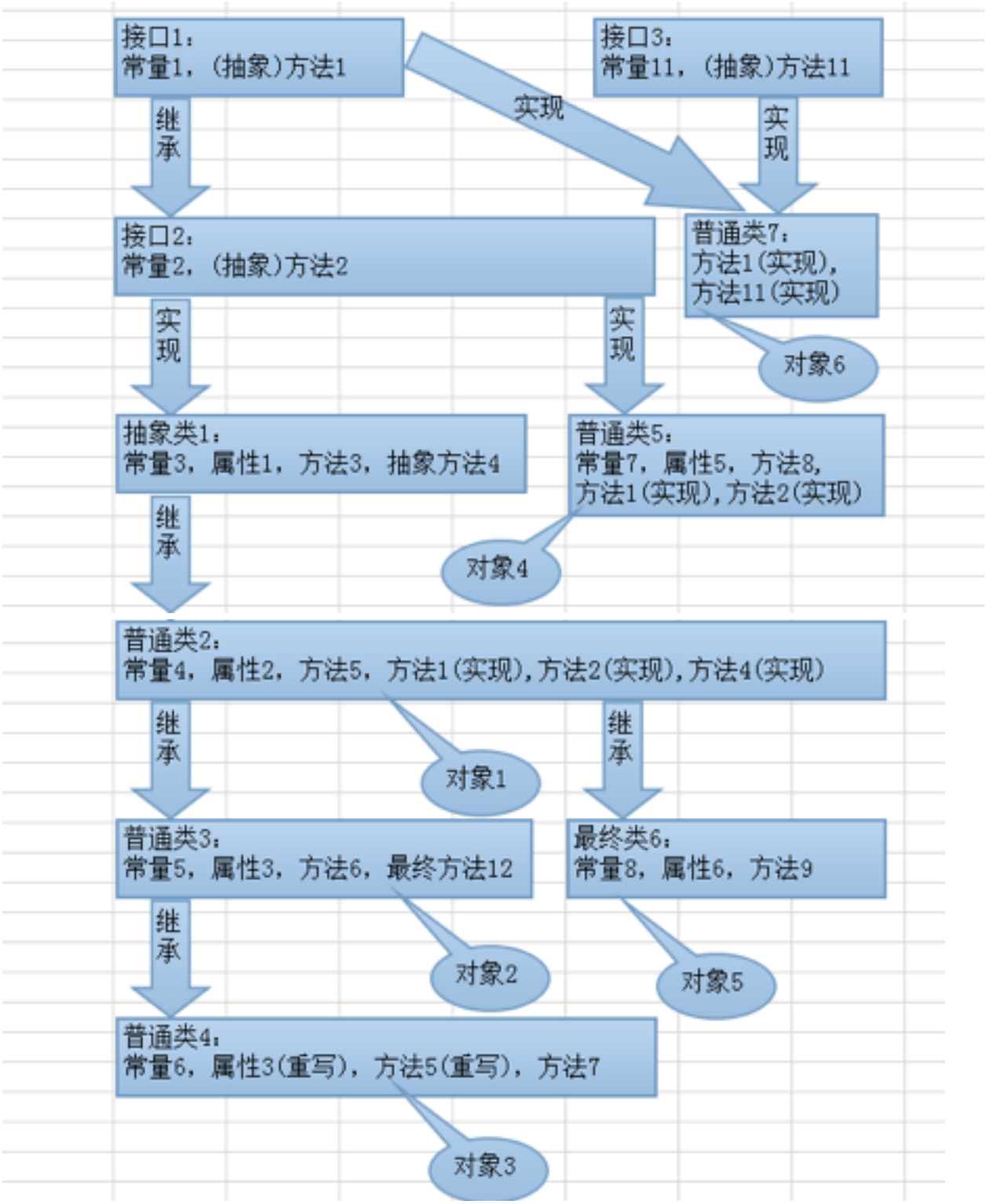
```php
interface flyAnimal{
    function fly();//接口中的方法都是抽象方法，无需声明抽象
}
class bird{
    public $wings = "2";//2个翅膀
}
//“继承”接口的特性，被称为：实现（implements）
class maque extends bird implements flyAnimal{

}
$m1 = new maque();
?>
</body>
</html>
```

一个类只能继承一个父类，但可以实现多个“上级接口”，语法形式:
class 类A extends 类B implements 接口1，接口2，.... { 。。。类定义语句。。。}

一个接口同样可以继承另一个接口（也是单继承）:
interface 接口1 extends 接口2 {接口定义}

最终的接口，普通类，抽象类，普通方法，抽象方法，各种常量的总结（图）:


接口1：
常量1，(抽象)方法1
接口3：
常量11，(抽象)方法11
实现
继承
实现
接口2：
常量2，(抽象)方法2
普通类7：
方法1(实现)，
方法11(实现)
实现
实现
对象6
抽象类1：
常量3，属性1，方法3，抽象方法4
普通类5：
常量7，属性5，方法8，
方法1(实现)，方法2(实现)
继承
对象4
普通类2：
常量4，属性2，方法5，方法1(实现)，方法2(实现)，方法4(实现)
继承
继承
对象1
普通类3：
常量5，属性3，方法6，最终方法12
最终类6：
常量8，属性6，方法9
继承
对象2
对象5
普通类4：
常量6，属性3(重写)，方法5(重写)，方法7
对象3