<http://blog.csdn.net/chjttony/article/details/6648534> good文章笔记系列

## 1.1

## 1.2每一个对象都有一个接口

1. 类：类描述了具有相同特性（数据元素）和行为（功能）的对象集合。
2. 挑战之一：将问题空间的元素和解空间的对象之间建立一对一的映射。
3. 接口（类中定义的方法）确定了对某一类对象所能发出的请求。
4. 向某个对象“发送消息或者请求”：有某一个对象的引用，调用这个对象的某一个方法，那么相对应的代码就会被执行。

（向某一个对象发送消息，这个对象就知道此消息的目的，就会执行相应的代码。）

## 1.3每一个对象都提供服务

1. 一个个对象就是一个个服务提供者。
2. 将问题分解成一个个服务提供者（对象：已有的使用，没有创建）。
3. 软件设计原则：高内聚，低耦合。（有助于设计出高内聚的对象）。
4. 每一个类型不需要身兼数职，完成一个任务即可。单一职责原则。

## 1.4被隐藏的具体实现(权限修饰符)

——权限修饰符（access specifier）

（1）让客户端程序员无法触及他们不应该触及的部分。

（2）允许库设计者改变类内部工作方式二不用担心会影响到客户端程序员。

|  |  |
| --- | --- |
| 权限修饰符（从大到小） | 作用域 |
| public | 最大访问控制权限，对所有的类都可见 |
| protected | 同一个包以及其他包中子类使用 |
| Default（默认不写） | 包访问权限（同一个包下的类可以访问） |
| private | 本类内部（方法）使用, 对外一切类都不可以访问(反射机制可以访问) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 作用域 | 本类 | 同package | 子孙类 | 其他package |
| public | √ | √ | √ | √ |
| protected | √ | √ | √ |  |
| default | √ | √ |  |  |
| private | √ |  |  |  |

## 1.5复用具体实现（has-a关系）

（1）组合：将一个类的对象作为另一个类的成员变量。

Class Couple(){

private Children children = new Children();

}

（2）注意事项：在建立新类时，首先应该考虑组合，不应该一味的使用继承，使继承体系过于复杂。

## 1.6继承（类型层次结构，继承体系）

父类-子类

基类-导出类

超类-继承类

1. 父类的修改，直接影响到子类，耦合性太强。解耦的最佳方式，实现接口！！！
2. 一个基类型包含其所有导出类共有的特性和行为。类型不仅仅只是描述了作用于一个对象集合的约束条件，同时还有与其他类型之间的关系。两个类型可以有相同的特性和行为，但是一个类型比另外一个类型有更多的特性和行为（或者以不同方式处理信息）。
3. 父类和子类产生差异的方法:@子类添加新方法@子类复写父类的方法，以不同的方式实现某一个功能。

## 1.6.1“是一个”（1）和“像一个”（2）的区别

（1）子类不添加新的方法，仅覆盖。（is-a）【纯粹替代，替代原则】

（2）子类添加父类没有的自己特有的方法（非覆盖方法）。（is-like-a）

注意：多态的情况：父类的引用a指向子类的对象，a只可以调用父类有的接口，而不可以调用子类的特有的接口。

|  |
| --- |
| 2.面向对象编程中两种对象组合方式——[is-a](是一个#_1.6.1) 和 [has-a](#_1.5复用具体实现（has-a关系）)：  (1).is-a组合：一个类继承具有相似功能的另一个类，根据需要在所继承的类基础上进行扩展。  优点：具有共同属性和方法的类可以将共享信息抽象到父类中，增强代码复用性，同时也是多态的基础。  缺点：子类中扩展的部分对父类不可见，另外如果共性比较少的时候使用继承会增加冗余代码。  (2).has-a组合：has-a组合是在一个类中引用另一个类作为其成员变量。  优点：可扩展性和灵活性高。在对象组合关系中应优先考虑has-a组合关系。  缺点：具有共性的类之间看不到派生关系。 |

## 1.7伴随多态的可互换对象。（多态+考题）

——（即父类的引用指向子类的对象）

Eg.List list = new ArrayList();

List.add(…);

1. 在编译期，编译器是无法准确找到哪段代码被执行，通过一段特殊的代码代替绝对地址的调用，在运行时期，才能够确定代码的位置。
2. 后期绑定(面向过程的叫法，java中称为动态绑定)，运行时才知道哪段代码被执行。

前期绑定（面向过程的叫法，java中称为静态绑定），编译期就已经加载到内存。

方法绑定：一个方法被调用时该方法关联其方法体的过程。

1. 动态绑定，将函数调用和真正被执行的代码关联起来。编译期间，真正被调用的对象不知道被调用，运行期间，才明确那个对象的函数被调用。

注意：

|  |
| --- |
| * 动态绑定的范畴只是方法，不包含成员变量，子类同名的变量是不会覆盖父类的变量的；eg. Peson p = new Ling(); p.name = ? 注意这时输出的是person的name值，不是Ling的name值。 * java中的变量都是静态绑定的； * java中的构造方法 以及private,static,final类方法的调用都是静态绑定的。 |

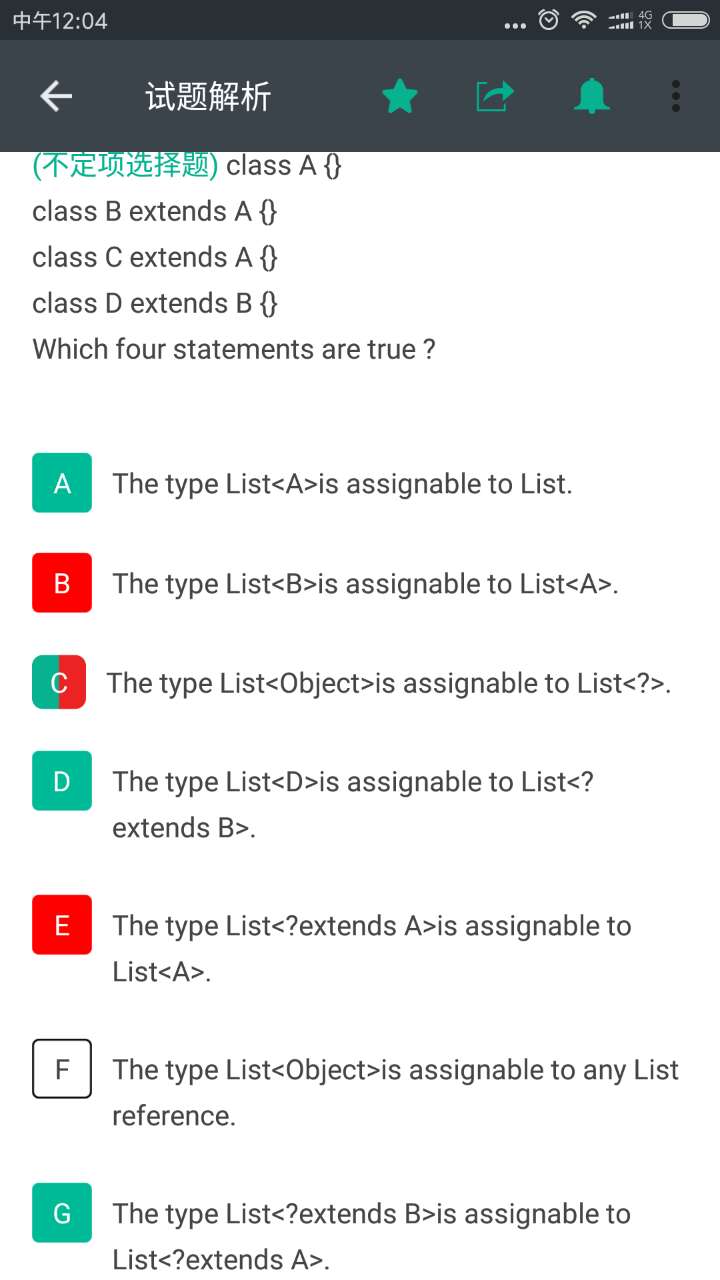
**补充：**

<http://blog.csdn.net/sureyonder/article/details/5569617> [JAVA动态绑定的内部实现机制](http://blog.csdn.net/sureyonder/article/details/5569617)

<http://blog.csdn.net/zdwzzu2006/article/details/5979923> 静态绑定和动态绑定（good）

java动态绑定和向上转型链接

1. 多态的表现形式
   1. 方法参数：父类接收子类的对象。
   2. 创建对象：父类的引用指向子类的对象。



答案：a,c,d

考点：

1.泛型如果明确指定了仅接受某一个类型的数据，那么即使是子类对象，也无法存储。(b,e错误)如果泛型定义的是一个范围，那么符合这个范围的对象都可以存储。（d，g对）

2.?代表可接受任何类型（c对）

3.如果集合没有指定类型，那么可接受任何数据。（a对）

## 1.8 单根继承体系

——（单继承原则+所有的类最终的继承自单一的基类Object）

相比于C++的多继承，java只支持类的单继承。优点：

（1）单继承可以确保所有的对象拥有某种共同的特性，这样对于JVM虚拟机对所有的类进行系统级的操作将提供方便，所有的java对象可以方便地在内存堆栈中创建，传递参数也变的更加方便简单。Eg.toStirng;getclass()方法保证对象运行可以知道类的信息。不需要在每个都写上这个方法，提高复用性。

（2）java的单继承使得实现垃圾回收器功能更加容易，因为可以确保JVM知道所有对象的类型信息。Eg。每个类都有finalize().



## 1.9容器（collection,map,堆栈，队列）

消息队列（？？？）

Java设置多个容器的原因：

（1）不同容器可以提供不同的接口和外部行为。（即不同的类型有不同的方法。可是实现功能不一样）

（2）不同容器对于某些操作的效率差异很大。

Eg. ArrayList 查询快（有索引）LinkedList 增删快（链表，后一个记录前一个的位置）

建议：我们可以在一开始使用LinkedList构建程序，而在优化系统性能时改用Arraylist.

接口List带来的抽象，把在容器之间的转换时对代码的影响下降到最低。（即使用多态）

## 1.9.1参数化类型机制（泛型）

——编译器自动定制 作用于特定类型(eg.String)的 类(eg.容器)

（eg.编译期定制了一个容器，规定只能存储和取出string类型的对象）

有一下两种表现形式：

1. javaSE5之前，Map内部默认存储Object,所以存储数据时默认将对象向上转型，取出数据时，为了可以调用原对象特有的方法，需要将数据向下强转（应该存储数据是，记住了数据的类型，取出的时候才可以强转。），这一个过程有安全问题。
2. javaSE5泛型：编译时期规定只可以接收某一个类型数据。
3. 注意：map只可以存储对象，数字可以自动装箱。

|  |
| --- |
| Map<String,String> parmaMap = **new** HashMap<String,String>();  parmaMap.put("one", "haha1");  parmaMap.put("two", "haha2"); |

|  |
| --- |
| 6.类型转换：  Java中有两种常见的类型转换：向上类型转换(upcast)和向下类型转换(downcast)：  (1).向上类型转换(upcast)：  向上类型转换是将子类对象强制类型转换为父类类型，经典用法是面向对象的多态特性。向上类型转换时，子类对象的特性将不可见，只有子类从父类继承的特性仍然保持可见，向上类型转换时编译器会自动检查是否类型兼容，通常是安全的。  (2).向下类型转换：  向下类型转换是将父类类型强制转换为子类类型，转换过后父类中不可见的子类特性又恢复可见性，向下类型转换时，编译器无法自动检测是否类型兼容，往往会产生类型转换错误的运行时异常，通常不安全。 |

1.10对象的创建（动态内存分配）和生命周期和内存释放（垃圾回收机制）

（1）创建方式

1.堆栈（栈，stack）:先进先出，控制指针上下移动即可。用完即可释放（方法内变量，方法参数）

2.堆（heap）:new出来的对象。只有运行时才可以知道对象的数量，类型，生命周期。动态分配。通过垃圾回收器释放内存。