**面向对象的编程允许你从已经存在的类中定义新的类，这称为继承。**

**继承使得你可以定义一个通用的类（即父类），之后扩充该类为一个更加特定的类（即子类）。**

**关于继承需要注意的几点**

**1）和传统的理解不同，子类并不是父类的一个子集。实际上，一个子类通常比它的父类包含更多的信息和方法。**

**2）父类中的私有数据域在该类之外是不可访问的。因此，不能在子类中直接使用。但是，如果父类中定义了公共的访问器/修改器，那么可以通过这些公共的访问器/修改器来访问和修改它们。**

**3）不是所有的“是一种”关系都该用继承来建模。**

**4）继承是用来为“是一种”关系建模的。不要仅仅为了重用方法这个原因而盲目地扩展一个类。**

**5）某些程序设计语言是允许从几个类派生出一个子类的。这种能力称为多重继承。但是在java中不允许多重继承的。一个java类只可能直接继承自父类。这种限制称为单一继承。如果使用extends关键字来定义一个子类，它只允许有一个父类。然而，多重继承是可以通过接口来实现的。**

**super关键字**

**super指代父类，可以用于调用父类中的普通方法和构造方法。**

**调用父类的构造方法**

**构造方法用于构建一个类的实例。不同于属性和普通方法，父类的构造方法不会被子类继承。它们只能使用关键字super从子类的构造方法中调用。**

**super()或者super(parameters);**

**语句super()和super(parameters)必须出现在子类构造方法的第一行，这是显式调用父类构造方法的唯一方式。在子类中调用父类构造方法的名字会引起语法错误。**

**构造方法链**

**构造方法可以调用重载的构造方法或父类的构造方法。如果它们都没有被显式地调用，编译器就会自动地将super()作为构造方法的第一条语句。**

**在任何情况下，构造一个类的实例时，将会调用沿着继承链的所有父类的构造方法。当构造一个子类的对象时，子类构造方法会在完成自己的任务之前，首先调用它的父类的构造方法。如果父类继承自其它类，那么父类的构造方法又会在完成自己任务之前，调用它自己父类的构造方法。这个过程持续到沿着这个继承体系结构的最后一个构造方法被调用为止。这就是构造方法链（constructor chaining）。**

**一般情况下，最好能为每一个类提供一个无参构造方法，以便于对该类进行扩展，同时避免错误。**

**调用父类的方法**

**super看不仅可以引用父类的构造方法，也可以引用父类的方法。所用语法：**

**super.方法名（参数）;**

**方法重写**

**要重写一个方法，需要在子类中使用和父类一样的签名以及一样的返回值类型来对该方法进行定义。**

**子类从父类中继承方法。有时，子类需要修改父类中定义的方法的实现，这称为方法重写（method overriding）。**

**注意：**

**1）仅当实例方法是可访问时，它才能被覆盖。因为私有方法在它的类本身以外是不能访问的，所以它不能被覆盖。如果子类中定义的方法在父类中是私有的，那么这两个方法没有关系。**

**2）与实例方法一样，静态方法也能被继承。但是，静态方法不能被覆盖。如果父类中定义的静态方法在子类中被重新定义，那么在父类中定义的静态方法将被隐藏。可以使用语法：父类名.静态方法名 调用隐藏的静态方法。**

**方法重写与重载**

**重载意味着使用同样的名字但是不同的签名来定义多个方法。**

**重写意味着在子类中提供一个对方法的新的实现。**

**注意：**

**1）方法重写发生在继承而不相关的类中；方法重载可以发生在同一个类中，也可以发生在由于继承而不相关的类中。**

**2）方法重写具有同样的签名和返回值类型；方法重载具有同样的名字，但是不同的参数列表。**

**为了避免错误，可以使用一个特殊的java语法，称为重写标注（override annotation），在子类的方法签名前面放一个@Override。**

**Object类与其toString方法**

**java中的所有类都继承自java.lang.Object类。**

**如果在定义一个类时没有指定继承性，那么这个类的父类就被默认为时Object。**

**toString方法**

**toString方法的签名是：**

**public String toString()**

**调用一个对象的toString()会返回一个描述该对象的字符串。默认情况下，它返回一个由该对象所属的类名、at符号@以及该对象十六进制形式的内存地址组成的字符串。**

**通常应该重写这个toString方法，这样，它可以返回一个代表该对象的描述性字符串。**

**也可以传递一个对象来调用System.out.println(Object)或者System.out.print(object)。这等价于调用System.out.println(object.toString())或者System.out.print(object.toString())。因此，可以使用System.out.println(loan）来替换System.out.println(loan.toString())。**

**多态**

**多态意味着父类的变量可以指向子类对象。**

**面向对象程序设计的三大支柱是封装、继承和多态。**

**继承关系使一个子类继承父类的特征，并且附加一些新特性。子类是它的父类的特殊化，每个子类的实例都是父类的实例，但是反过来就不成立。**

**简单来说，多态意味着父类型的变量可以引用子类型的对象。**

**动态绑定**

**方法可以在沿着继承链的多个类中实现。JVM决定运行时调用哪个方法。**

**Object o=new GeometricObject();**

**System.out.println(o.toString());**

**o调用了哪个toString呢？**

**一个变量必须被声明为某种类型。变量的这个类型称为它的声明类型（declared type）。这里，o的声明类型是Object。一个引用类型变量可以是一个null值或者是一个对声明类型实例的引用。实例可以使用声明类型或它的子类型的构造方法创建。**

**变量的实际类型（actual type）是被变量引用的对象的实际类。这里，o的实际类型是GeometricObject，因为o指向使用new GeometricObject（）创建的对象。o调用哪个toString（）方法由o的实际类型决定。这称为动态绑定。**

**匹配方法的签名和绑定方法的实现是两个不同的问题。引用变量的声明类型决定了编译时匹配哪个方法。在编译时，编译器会根据参数类型、参数个数和参数顺序找到匹配的方法。一个方法可能在沿着继承链的多个类中实现。java虚拟机在运行时动态绑定方法的实现，这是由变量的实际类型决定的。**

**对象转换和instanceof运算符**

**对象的引用可以类型转换为对另外一种对象的引用，这称为对象转换。**

**总是可以将一个子类的实例转换为一个父类的变量，称为向上转换（upcasting），因为子类的实例永远是它的父类的实例。当把一个父类的实例转换为它的子类变量（称为向下转换（downcasting））时，必须使用转换记号“（子类名）”进行显式转换，向编译器表明你的意图。**

**为使转换成功，必须确保要转换的对象是子类的一个实例。如果父类对象不是子类的一个实例，就会出现一个运行异常ClassCastException。例如，如果一个对象不是Student的实例，它就不能转换成Student类型的变量。因此，一个好的经验是，在尝试转换之前确保该对象是另一个对象的实例。可以利用运算符instanceof实现。**

**Object myObject=new Circle();**

**if(myObject instanceof Circle){**

**System.out.println("the circle diameter is"+((Circle)myObject).getDiameter());**

**}**

**为什么进行类型转换？变量myObject被声明为Object。声明类型决定了在编译时匹配哪个方法。使用myObject.getDiameter()会引起一个编译错误，因为Object类没有getDiameter方法。编译器无法找到和myObject.getDiameter()匹配的方法。所以，有必要将myObject转换成Circle类型，来告诉编译器myObject也是Circle的一个实例。**

**为什么没有一开始就把myObject定义为Circle类型呢？为了能够进行通用程序设计，一个好的经验是把变量定义为父类型，这样，它就可以接受任何子类型的值。**

**对象成员访问运算符（.）优先于类型转换运算符。使用圆括号保证在点运算符之前进行转换，例如**

**((Circle)object).getArea();**

**对基本类型值进行转换不同于对对象引用进行转换。转换基本类型值返回一个新的值。例如**

**int age=45；**

**byte newAge=(byte)age;//A new value is assigned to newAge**

**而转换一个对象引用不会创建一个新的对象，例如**

**Object o=newCircle();**

**Circle c=(Circle)o; //no new object is created**

**现在，引用变量o和c指向同一个对象。**

**Object类的equals方法**

**如同toString方法，equals(Object)方法是定义在Object类中的另一个有用的方法。**

**Object类中equals方法的默认实现是：**

**public boolean equals（Object obj）{**

**return （this==obj）;**

**}**

**这个实现使用==运算符检测两个引用变量是否指向同一个对象。因此，应该在自己的客户类中重写这个方法，以检测两个不同的对象是否具有相同的内容。**

**equals方法在javaAPI的许多类中被重写，比如java.lang.String和java.util.Date，用于比较两个对象的内容是否相等。**

**比较运算符==用来比较两个基本数据类型过得值是否相等，或者判断两个对象是否具有相同的引用。如果想让equals方法能够判断两个对象是否具有相同的内容，可以定义这些对象的类时，重写Circle类中的equals方法。**

**运算符==比equals方法的功能强大些，因为==运算符可以检测两个引用变量是否指向同一个对象。**

**在子类中，使用签名equals（SomeCalssName obj）（例如：equals（Circle c））重写equals方法是一个错误，应该使用equals（Object obj）。**

**ArrayList类**

**ArrayList对象可以用于存储一个对象列表，可以存储不限定个数的对象。**

**ArrayList是一种泛型类，具有一个泛型类型E。创建一个ArrayList时，可以指定一个具体的类型来替换E。例如**

**ArrayList<String> cities=new ArrayList<>();**

**ArrayList位于java.util包中，第一行导入该包。**

**add(o:E):void**

**add(index:int,o:E):void**

**clear():void**

**contains(o:Object):boolean**

**get(index:int):E**

**indexOf(o:Object):int**

**isEmpty():boolean**

**lastIndexOf(o:Object):int**

**remove(o:Object):boolean**

**size():int**

**set(index:int,o:E):E**

**一旦创建了一个数组，它的大小就确定下来了。可以使用方括号访问数组元素。当创建ArrayList后，它的大小为0,。如果元素不在数组列表中，就不能使用get(index)和set(index,element)方法。**

**向数组列表中添加、插入和删除元素时比较容易的，而想数组中添加、插入和删除元素是比较复杂的。为了实现这些操作，必须编写代码操纵这个数组。**

**可以使用java.util.Arrays.sort(array)方法来对一个数组排序。如果要对一个数组列表排序，使用java.util.Collections(arrayList)方法。**

**ArrayList<int> list=new ArrayList<>();//错误的**

**因为存储在ArrayList中的元素必须是一种对象。不能使用基本数据类型，然而，可以使用Integer对象。**

**ArrayList的大小是灵活的，所以无须提前给定它的大小。而创建数组时，必须给定大小。**

**ArrayList包含许多有用的方法。比如，可以使用contains方法来测试某个元素是否在列表中。如果使用数组，则需要编写额外代码来实现该方法。**

**可以在数组里使用foreach循环来遍历元素。数组列表中的元素也可以使用foreach循环来进行遍历，语法：**

**for(elementType element:arrayList){**

**}**

**对于列表有用的方法**

**java提供了方法，用于从数组创建列表、对列表排序、找到列表中的最大和最小元素，以及打乱一个列表。**

**我们经常需要从一个对象数组中创建一个数组列表，或者相反。可以使用循环来实现，但更容易的方法是使用javaAPI中的方法。**

**//从数组中创建一个数组列表**

**String[] array={"red","green","blue"};**

**ArrayList<String> list=new ArrayList<>(Arrays.asList(array));**

**Arrays类中的静态方法asList返回一个列表，该列表传递给ArrayList的构造方法用于创建一个ArrayList。反过来，可以使用下面代码从一个数组列表来创建一个对象数组。**

**String[] array={"red","green","blue"};**

**ArrayList<String> list=new ArrayList<>(Arrays.asList(array));**

**String[] array1=new String[list.size()];**

**list.toArray(array1);**

**调用list.toArray(array1)将list中的内容复制到array1中。**

**如果列表中的元素是可比较的，比如整数、双精度浮点数或者字符串，则可以使用java.util.Collections类中的静态的sort方法来对元素进行排序。**

**可以使用java.util.Collections类中的静态的max和min方法来返回列表中的最大和最小元素。**

**可以使用java.util.Collections类中的静态的shuffle方法来随机打乱列表的元素。**

**protected数据和方法**

**一个类中受保护成员可以从子类中访问。**

**经常需要允许子类访问定义在父类中的数据域或方法，但不允许非子类访问这些数据域和方法。可以使用关键字protected完成该功能。**

**修饰符private、protected和public都称为可见性修饰符（visibility modifier）或可访问性修饰符（accessibility modifier）,因为它们指定如何访问类和类的成员。**

**可见性递增顺序：**

**私有-->默认（无修饰符）-->被保护-->公共成员**

**子类可以重写它的父类的protected方法，并把它的可见性改为public。但是子类不能削弱父类中定义的方法的可访问性。例如：如果一个方法在父类中定义为public，在子类中也必须为public。**

**protected修饰符来防止方法和数据被不同包的非子类访问。**

**防止扩展和重写**

**一个被final修饰的类和方法都不能被扩展。被final修饰的数据域是一个常数。**

**修饰符public、protected、private、static、abstract以及final可以用在类和类的成员（数据和方法上），只有final修饰符还可以用在方法中的局部变量上，方法内的最终局部变量就是常量。**