**父类中定义了相关子类中的共同行为。接口可以用于定义类的共同行为（包括非相关的类）。**

**抽象类**

**抽象类不可以用于创建对象。抽象类可以包含抽象方法，这些方法将在具体的子类中实现。**

**抽象类和常规类很像，但是不能使用new操作符创建它的实例。抽象方法只有定义而没有实现。它的实现由子类提供。一个包含抽象方法的类必须声明为抽象类。**

**抽象类的构造方法定义为protected，因为它只被子类使用。创建一个具体子类的实例时，它的父类的构造方法被调用以初始化父类中定义的数据域。**

**注意：**

**1）抽象方法不能包含在非抽象类中。如果抽象父类的子类不能实现所有的抽象方法，那么子类也必须定义为抽象类的。换句话说，在抽象类扩展的非抽象子类中，必须实现所有的抽象方法。还要注意到，抽象方法是非静态的。**

**2）抽象类是不能使用new操作符初始化的。但是，仍然可以定义它的构造方法，这个构造方法在它的子类的构造方法中调用。例如，GeometricObject类的构造方法在Circle类和Rectangle类中调用。**

**3）包含抽象方法的类必须是抽象的。但是可以定义一个不包含抽象方法的抽象类。在这种情况下，不能使用new操作符创建该类的实例。这种类是用来定义新子类的基类的。**

**4）子类可以覆盖父类的方法并将它定义为abstract。这是很少见的，但是它在当父类的实现方法实现在子类中变得无效时是很有用的。在这种情况下，子类必须定义为abstract。**

**5）即使子类的父类是具体的，这个子类也可以是抽象的。例如，Object类是具体的，但是它的子类如GeometricObject可以是抽象的。**

**6）不能使用new操作符从一个抽象类创建一个实例，但是抽象类可以用作一种数据类型。因此，下面的语句创建一个元素是GeometricObject类型的数组是正确的：**

**GeometricObject[] objects=new GeometricObject[10];**

**然后可以创建一个GeometricObject的实例，并将它的引用赋值给数组，如下**

**objects[0]=new Circle();**

**实例学习：Calendar和GregorianCalendar**

**GregorianCalendar是抽象类Calendar的一个具体子类。**

**一个java.util.Date的实例表示以毫秒为精度的特定时刻。java.util.Calendar是一个抽象的基类，可以提取出详细的日历信息，例如，年、月、日、小时、分钟和秒。Calendar类的子类可以实现特定的日历系统，例如，公历、农历和犹太历。**

**public class TestCalendar {**

**public static void main(String[] args) {**

**// Construct a Gregorian calendar for the current date and time**

**Calendar calendar = new GregorianCalendar();**

**System.out.println("Current time is " + new Date());**

**System.out.println("YEAR:\t" + calendar.get(Calendar.YEAR));**

**System.out.println("MONTH:\t" + calendar.get(Calendar.MONTH));**

**System.out.println("DATE:\t" + calendar.get(Calendar.DATE));**

**System.out.println("HOUR:\t" + calendar.get(Calendar.HOUR));**

**System.out.println("HOUR\_OF\_DAY:\t" +**

**calendar.get(Calendar.HOUR\_OF\_DAY));**

**System.out.println("MINUTE:\t" + calendar.get(Calendar.MINUTE));**

**System.out.println("SECOND:\t" + calendar.get(Calendar.SECOND));**

**System.out.println("DAY\_OF\_WEEK:\t" +**

**calendar.get(Calendar.DAY\_OF\_WEEK));**

**System.out.println("DAY\_OF\_MONTH:\t" +**

**calendar.get(Calendar.DAY\_OF\_MONTH));**

**System.out.println("DAY\_OF\_YEAR: " +**

**calendar.get(Calendar.DAY\_OF\_YEAR));**

**System.out.println("WEEK\_OF\_MONTH: " +**

**calendar.get(Calendar.WEEK\_OF\_MONTH));**

**System.out.println("WEEK\_OF\_YEAR: " +**

**calendar.get(Calendar.WEEK\_OF\_YEAR));**

**System.out.println("AM\_PM: " + calendar.get(Calendar.AM\_PM));**

**// Construct a calendar for September 11, 2001**

**Calendar calendar1 = new GregorianCalendar(2001, 8, 11);**

**System.out.println("September 11, 2001 is a " +**

**dayNameOfWeek(calendar1.get(Calendar.DAY\_OF\_WEEK)));**

**}**

**public static String dayNameOfWeek(int dayOfWeek) {**

**switch (dayOfWeek) {**

**case 1: return "Sunday";**

**case 2: return "Monday";**

**case 3: return "Tuesday";**

**case 4: return "Wednesday";**

**case 5: return "Thursday";**

**case 6: return "Friday";**

**case 7: return "Saturday";**

**default: return null;**

**}**

**}**

**}**

**set（int field,value)方法用来设置一个域。例如，可以使用calendar.set(Calendar.DAY\_OFMONTH,1)将calendar设置为当月的第一天。**

**add(field,value)方法为某个特定域增加指定的量。例如，add(Calendar.DAY\_OF\_MONTH,5)给日历的当前时间加五天。**

**为了获得一个月中的天数，使用calendar.getActualMaximum(Calendar.DAY\_OF\_MONTH)方法。**

**可以通过调用canlendar.setTime(data)为calendar设置一个用Date对象表示的时间，通过调用calendar.getTime()获取时间。**

**接口**

**接口是一种与类相似的结构，只包含常量和抽象方法。**

**使用正确的接口，可以指明这些对象是可比较的、可食用的，以及可克隆的。**

**由于接口中所有的数据域都是public static final而且所有的方法都是public abstract，所以java允许忽略这些修饰符。**

**Comparable接口**

**Comparable接口定义了compareTo方法，用于比较对象。**

**由于所有Comparable对象都有compareTo方法，如果对象是Comparable接口类型的实例的话，javaAPI中java.util.Arrays.sort(Object[])方法就可以使用compareTo方法来对数组中的对象进行比较和排序。**

**public class SortComparableObjects {**

**public static void main(String[] args) {**

**String[] cities = {"Savannah", "Boston", "Atlanta", "Tampa"};**

**java.util.Arrays.sort(cities);**

**for (String city: cities)**

**System.out.print(city + " ");**

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

**BigInteger[] hugeNumbers = {new BigInteger("2323231092923992"),**

**new BigInteger("432232323239292"),**

**new BigInteger("54623239292")};**

**java.util.Arrays.sort(hugeNumbers);**

**for (BigInteger number: hugeNumbers)**

**System.out.print(number + " ");**

**}**

**}**

**不能使用sort方法来对一个Rectangle对象数组进行排序，因为Rectangle类没有实现接口Comparable。然而，可以定义一个新的Rectangle类来实现Comparable。例如**

**public class ComparableRectangle extends Rectangle**

**implements Comparable<ComparableRectangle> {**

**/\*\* Construct a ComparableRectangle with specified properties \*/**

**public ComparableRectangle(double width, double height) {**

**super(width, height);**

**}**

**@Override // Implement the compareTo method defined in Comparable**

**public int compareTo(ComparableRectangle o) {**

**if (getArea() > o.getArea())**

**return 1;**

**else if (getArea() < o.getArea())**

**return -1;**

**else**

**return 0;**

**}**

**@Override // Implement the toString method in GeometricObject**

**public String toString() {**

**return "Width: " + getWidth() + " Height: " + getHeight() +**

**" Area: " + getArea();**

**}**

**}**

**现在可以使用sort方法来对ComparableRectangle对象数组进行排序了，例如**

**public class SortRectangles {**

**public static void main(String[] args) {**

**ComparableRectangle[] rectangles = {**

**new ComparableRectangle(3.4, 5.4),**

**new ComparableRectangle(13.24, 55.4),**

**new ComparableRectangle(7.4, 35.4),**

**new ComparableRectangle(1.4, 25.4)};**

**java.util.Arrays.sort(rectangles);**

**for (Rectangle rectangle: rectangles) {**

**System.out.print(rectangle + " ");**

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

**}**

**}**

**}**

**接口提供通用程序设计的另一种形式。**

**Object类包含equals方法，它的目的就是为了让Object类的子类来覆盖它，以比较对象的内容是否相同。**

**假设Object类包含一个类似于Comparable接口中所定义的compareTo方法，那么sort方法可以用来比较一组任意的对象。Object类中是否应该包含一个compareTo方法尚有争论。**

**由于在Object类中没有定义compareTo方法，所以java中定义了Comparable接口，以便能够对两个Comparable接口的实例对象进行比较。强烈建议compareTo应该与equals保持一致。也就是说，对于两个对象o1和o2，应该确保当且仅当o1.equals(o2)为true时o1.compareTo(o2)==0成立。**

**Cloneable接口**

**Cloneable接口给出了一个可克隆的对象。**

**接口包括常量和抽象方法，但是Cloneable接口是一个特殊情况。在java.lang包中的Cloneable接口的定义如下**

**package java.lang;**

**public interface Cloneable{**

**}**

**这个接口是空的。一个带空体的接口称为标记接口（marker interface）。一个比较接口既不包括常量也不包括方法。它用来表示一个类拥有某些特定的属性。实现Cloneable接口的类标记为可克隆的，而且它的对象可以使用在Object类中定义的clone()方法克隆。**

**为了定义一个自定义类来实现Cloneable接口，这个类必须覆盖Object类中的clone()方法。例如**

**public class House implements Cloneable, Comparable<House> {**

**private int id;**

**private double area;**

**private java.util.Date whenBuilt;**

**public House(int id, double area) {**

**this.id = id;**

**this.area = area;**

**whenBuilt = new java.util.Date();**

**}**

**public int getId() {**

**return id;**

**}**

**public double getArea() {**

**return area;**

**}**

**public java.util.Date getWhenBuilt() {**

**return whenBuilt;**

**}**

**@Override /\*\* Override the protected clone method defined in**

**the Object class, and strengthen its accessibility \*/**

**public Object clone() throws CloneNotSupportedException {**

**return super.clone();**

**}**

**@Override // Implement the compareTo method defined in Comparable**

**public int compareTo(House o) {**

**if (area > o.area)**

**return 1;**

**else if (area < o.area)**

**return -1;**

**else**

**return 0;**

**}**

**}**

**House类实现在Object类中定义的clone方法，方法头是**

**protected native Object clone() throws CloneNotSupportedException;**

**关键字native表明这个方法不是用java写的，但它是JVM针对自身平台实现的。关键字protected限定方法只能在同一个包内或在其子类中访问。由于这个原因，House类必须覆盖该方法并将它的可见性修饰符改为public，这样，该方法就可以在任何一个包中使用。**

**因为Object类针对自身平台实现的clone方法完成了克隆对象的任务，所以，在House类中的clone方法只要简单调用super.clone即可。在Object类中定义的clone方法会抛出CloneNotSupportedException异常。**

**House house1=new House(1,1444.4);**

**House house2=(House)house1.clone();**

**house1和house2是两个内容相同的不同的对象。Object类中的clone方法将原始对象的每个数据域复制给目标对象。如果一个数据域是基本类型，复制的就是它的值。如果一个数据域是对象，复制的就是该域的引用。因此，尽管house1==house2假，但是house1.whenBuile==house2.whenBuilt为真。这称为浅复制而不是深复制，这意味着如果数据域是对象类型，那么复制的是对象的引用，而不是它的内容。**

**如果希望进行深复制，将clone方法改为：**

**public Object clone() {**

**// Perform a shallow copy**

**House houseClone = new House(id, area);**

**// Deep copy on whenBuilt**

**houseClone.whenBuilt = new java.util.Date();**

**houseClone.getWhenBuilt().setTime(whenBuilt.getTime());**

**return houseClone;**

**}**

**house1.whenBuilt==house.whenBuilt将为false。house1和house2包含两个不同的Date对象。**

**接口与抽象类**

**一个类可以实现多个接口，但是只能继承一个父类。**

**所有的类共享同一个根类Object，但是接口没有共同的根。与类相似，接口也可以定义一种类型。一个接口类型的变量可以引用任何实现该接口的类的实例。如果一个类实现了一个接口，那么这个接口就类似于该类的一个父亲。可以将接口当做一种数据类型使用，将接口类型的变量转换为它的子类，反过来也可以。**

**类名是一个名词。接口名可以是形容词或名词。**

**抽象类和接口都是用来明确多个对象的共同特征的。那么该如何确定在什么情况下应该使用接口，什么情况下应该使用类呢？一般来说，清晰描述父子关系的枪的“是一种”的关系应该用类建模。弱的“是一种”关系也称为类属关系，它表明对象拥有某种属性，可以用接口来建模。**

**通常，推荐使用接口而非抽象类，因为接口可以定义非相关类共有的父类型。接口比类更加灵活。**

**5451127769624268DF0AAB4DBBB6FC88213BE3CE**

**在java中，接口被认为是一种特殊的类。就像常规类一样，每个接口都被编译为独立的字节码文件。**