接口是对**动作的抽象**，而抽象类是对**根源的抽象**。比如男人，女人这两个类，那我们可以为这两个类设计一个更高级别的抽象类——人。对于接口，我们可以坐着吃饭，可以站着吃饭，可以用筷子吃饭，可以用叉子吃饭，甚至可以用手抓着吃饭，那么可以把这些吃饭的动作抽象成一个接口——吃饭。所以一个类只能继承一个抽象类，而可以实现多个接口。

# 接口

1. 接口指定了一组实现类必须要实现的方法，如果不能全部实现，那么必须是abstract class。
2. 接口是任何实现该接口的父类，可以将类的实例赋值给接口类型的变量，但是不可以new一个接口的实例。
3. 接口可以包含静态**方法**，所有的方法默认是public abstract的；
4. 接口所有**变量**默认是public **static** final的（为什么是static的？因为接口指定的是行为，不是状态，在**接口中没有实例变量**）；
5. 接口可以包含**默认方法**，实现类可以继承或者覆盖该默认方法（默认方法是指声明为default的方法，如果没有声明为default，那么子类必须要实现，而default方法如果在子类中没有实现，将采用接口中定义的）；
6. 实现类必须将接口**方法**声明为public（如果没有声明为public，默认是default，会报错）；
7. 类型T的变量不需要转型就能够赋值给类型S的变量，那么这S是T的父类。所以从父类转为子类才需要强制转型；
8. 接口中的变量都是static final，所以在子类中不可以重新对其复制，但是可以直接拿来使用。

## 继承接口

一个接口可以extends另一个接口，在原本接口的基础上增加额外的方法。

interface **Closeable**{

void **close**();

}

interface **Channel** extends **Closeable**{

void **open**();

}

public class **Test** implements **Closeable**, **Channel**{

public void **close**() { }

public void **open**() { }

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

Test t = new **Test**();

t.**close**();t.**open**();

Closeable c = (Closeable) t;

c.**close**(); *//实例c无法调用open方法*

Channel cc = (Channel) t;

cc.**open**();

}

}

实现了Channel接口的类必须实现两个方法，而且它的对象可以转成两个接口类型。

## 静态方法和默认方法

早期的java所有接口方法是没有方法体的，但是现在的接口中可以添加两种有**具体实现**的方法：静态方法和默认方法。

### 默认方法

给接口的方法提供默认实现，那么需要给这样的方法加上default修饰符标签。实现了接口的类可以重写覆盖接口中方法的默认实现，也可以直接继承接口中的实现。

默认方法的一个重要用途是**接口演化**。

public class **Bag** implements **Collection**

在java8，Collection接口增加了一个stream方法。如果stream方法不是默认方法，那么Bag类将无法编译，因为它必须实现接口的stream方法。给接口添加一个非默认的方法会导致源代码无法兼容。

如果没有重新编译类，只是直接使用包含该类的旧JAR文件。即使缺失方法，类依然会加载，程序依然能构造Bag实例，但是如果Bag的实例调用了stream方法，那么会发生AbstractMethodError异常。

而采用默认方法可以解决这些问题，Bag类可以重新编译，即使没有重新编译，在调用到stream方法是，会执行Collection接口默认的stream方法。

### 静态方法

静态方法，只能通过接口名调用，不可以通过实现类的类名或者实现类的对象调用。default方法，只能通过接口实现类的对象来调用。

## 实现多个接口

如果一个类实现了两个接口，

1. 两个接口定义了同样的static方法，这个类不会产生错误，因为只能通过接口类调用接口中的静态方法；
2. 两个接口定义了同样的default方法。由于default方法在接口中是有具体的实现的，编译器没有办法决定谁该覆盖谁。所以只能在子类中重写解决歧义；或者其中一个接口是默认方法，另一个不是默认方法，这样也一样会冲突；
3. 两个接口定义了同样的public abstract方法。这样是没有冲突的，因为要么在子类实现public的方法，要么不实现，该类作为abstract class。

interface **Closeable**{

default boolean **close**(){return true;}

}

interface **Channel**{

default boolean **close**(){return false;}

default boolean **open**(){return false;}

}

public class **Test** implements **Closeable**, **Channel**{

public boolean **close**() { //open方法没有冲突，所以可以不重写；但是close方法有冲突，必须在子类中重写解决冲突

return Closeable.super.**close**();

}

}

如果一个类继承了一个类并实现了一个接口中有同样的方法，这种情况下只关心父类的方法，忽视来着接口的默认方法。

## 接口的使用

### Comparable

一个对象数组，排序算法需要重复比较数组中的元素。不同的类比较元素的规则是不同的，但是排序算法只应该调用类提供的比较方法，只要所有的类就比较的时候提供的方法达成一致，那么排序算法就能开始工作。这个在排序时对象之间进行比较方法就可以是一个接口，所有需要比较的对象继承这个接口并且实现比较的方法，就可以对这些对象进行排序。

如果一个类想启用对象排序，那么就应该实现Comparable接口。

public class **Test**{

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

Employee[] employees = new Employee[3];

employees[0] = new **Employee**(20);

employees[1] = new **Employee**(10);

employees[2] = new **Employee**(30);

Arrays.**sort**(employees);

for(Employee e : employees){

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

}

}

static class **Employee** implements **Comparable**<Employee>{

private int id;

public **Employee**(int id){this.id = id;}

@Override

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

return this.id - o.id;

}

@Override

public String **toString**() {

return "Employee{" + "id=" + id + '}';

}

}

}

### Comparator

假如想根据字符串的长度而不是根据字典顺序对字符串排序，但是String类我们是无法修改的。

Arrays.sort方法和Collections.sort方法都提供了一个可以接收Comparator实例作为第二个参数的版本。

要按照长度比较字符串，定义一个实现Comparator<String>的类。

public class **LengthComparator** implements **Comparator**<String> {

@Override

public int **compare**(String o1, String o2) {

return o1.**length**() - o2.**length**();

}

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

String[] names = {"tom", "alice", "fred"};

Arrays.**sort**(names, new **LengthComparator**());

out.**println**(Arrays.**toString**(names));

}

}