### JAVA 第一阶段—DAY10-JAVA案例

1. 编写抽象类员工类Employee和抽象方法，并定义对应的子类讲师Lecturer。

* 参考答案

// 1、抽象类

public abstract class Employee {

public abstract void work();

}

// 2、子类

public class Lecturer extends Employee {

@Override

public void work() {

System.out.println("讲师在讲课");

}

}

// 3、测试类

public class EmployeeTest {

public static void main(String[] args) {

Lecturer lt= new Lecturer();

lt.work();

}

}

1. 定义A、B两个接口及对应方法，编写一个实现类同时实现这两个接口，并重写对应的方法。

* 参考答案

// 接口A

public interface InterfaceA {

// 1、常量

int A=1;

// 2、抽象方法

public abstract void work();

// 3、静态方法

public static void sleep() {

System.out.println("睡觉了A");

}

// 4、默认方法

public default void eat() {

System.out.println("吃饭了A");

}

}

// 接口B

public interface InterfaceB {

// 1、常量

int A=2;

// 2、抽象方法

public abstract void work();

// 3、静态方法

public static void sleep() {

System.out.println("睡觉了B");

}

// 4、默认方法

public default void eat() {

System.out.println("吃饭了B");

}

}

// 实现类

public class MoreInterfaceTest implements InterfaceA,InterfaceB {

// 必须实现所有的接口方法

@Override

public void work() {

System.out.println("该工作了");

}

// 实现的接口默认方法重名，必须重写一次

@Override

public void eat() {

System.out.println("重写的接口的eat()默认方法");

}

public static void main(String[] args) {

// 1、实现多接口时的接口常量使用

System.out.println(InterfaceA.A);

System.out.println(InterfaceB.A);

// 2、接口抽象方法调用

MoreInterfaceTest mit=new MoreInterfaceTest();

mit.work();

// 3、调用接口的静态方法

InterfaceA.sleep();

InterfaceB.sleep();

// 4、调用默认方法

mit.eat();

}

}

1. 分别使用Comparable和Comparator两种比较器的方式完成上述Student类型数组的比较排序。

* 参考答案

// 学生类

class Student implements Comparable<Student> {

private String name;

private int age;

public Student() {

}

public Student(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

@Override

public String toString() {

return "Student [name=" + name + ", age=" + age + "]";

}

@Override

public int compareTo(Student o) {

return this.age - o.age;

}

}

// 自定义比较器类

class MyComparator implements Comparator<Student> {

@Override

public int compare(Student o1, Student o2) {

return o1.getName().charAt(0)-o2.getName().charAt(0);

}

}

// 测试类

public class CompareTest {

public static void main(String[] args) {

// 初始化学生类数组

Student[] str=new Student[5];

str[0]=new Student("tom",22);

str[1]=new Student("kitty",18);

str[2]=new Student("Jacklove",23);

str[3]=new Student("uzi",23);

str[4]=new Student("tom",17);

// 打印原生数组

System.out.println(Arrays.toString(str));

// 使用比较器默认排序

Arrays.sort(str);

// 排序后再次打印

System.out.println(Arrays.toString(str));

// 使用比较器排序

Arrays.sort(str, new MyComparator());

// 排序后再次打印

System.out.println(Arrays.toString(str));

}

}