王新雨 0522 当日总结

1 学习内容

1　　类

类是相同或相似对象的一种抽象，是对象的一个模板，它描述一类对象的行为和状态。类是具有相同属性和方法（行为）的对象的集合

定义类:

```
public class 类名{
//定义属性部分（成员变量）
属性1的类型 属性1;
属性2的类型 属性2;
...
//定义方法部分
方法1;
方法2;
...
}
```

```
public class People{
    double height; //身高
    int age; //年龄
    int sex; // 性别

    void cry(){
        System.out.println("哭");
    }
    void laugh(){
        System.out.println("笑");
    }
    void print(){
        System.out.println("我的身高是"+height+"cm");
        System.out.println("我的年龄是"+age+"岁");
        if(this.sex==0)
            System.out.println("我是男性！");
        else
            System.out.println("我是女性！");
    }
}
```

2　　对象

创建：类名 对象名 = new 类名();

```
public class NewObject {
    public static void main(String[] args) {
        People LiLei = new People();
        LiLei.height =170;
        LiLei.age = 20;
        LiLei.sex = 1;
        LiLei.print();
    }
}
```

```
shiyanlou:project/ $ javac NewObject.java
shiyanlou:project/ $ java NewObject
我的身高是170.0cm
我的年龄是20岁
我是女性!
```

//引用对象属性

对象名.属性

//引用对象方法

对象名.方法

3　　构造方法

构造方法的名称与类名相同，且没有返回值。

```
public 构造方法名(){
//初始化代码
}
```

如果在定义类的时候没有写构造方法，系统会默认生成一个无参构造方法。当有指定的构造方法时，系统都不会再添加无参构造方法了。

构造方法的重载

4　　Static 静态成员或类成员

静态成员：被类的所有对象所共享。静态成员可以使用类名直接访问，也可以使用对象名进行访问

静态方法：被 static 修饰的方法是静态方法，静态方法不依赖于对象，不需要将类实例化便可以调用，由于不实例化也可以调用，所以不能有 this，也不能访问非静态成员变量和非静态方法。但是非静态成员变量和非静态方法可以访问静态方法

5　　Final:

final 关键字可以修饰类、方法、属性和变量:

final 修饰类，则该类不允许被继承，为最终类

final 修饰方法，则该方法不允许被覆盖（重写）

final 修饰属性：则该类的属性不会进行隐式的初始化（类的初始化属性必须有值）或在构造方法中赋值（但只能选其一）

final 修饰变量，则该变量的值只能赋一次值，即常量

6　　权限修饰符

访问修饰符	本类	同包	子类	其它
private	√			
默认	√	√		
protected	√	√	√	
public	√	√	√	√

Private< 默认包访问权限<protected<public

7　　封装:

只能通过规定的方法访问数据

隐藏类的实例细节，方便修改和实现。

如何实现类的封装:

（1）修改属性的可见性，在属性的前面添加修饰符(private)

（2）对每个值属性提供对外的公共方法访问，如创建 getter/setter（取值和赋

值） 方法，用于对私有属性的访问

（3）在 getter/setter 方法里加入属性的控制语句，例如我们可以加一个判断语句，对于非法输入给予否定。

```java
public class People {
    private double height;
    //setter
    public void setHeight(double h){
        height = h;
    }
    //getter
    public double getHeight(){
        return height;
    }
}
```

```java
public class NewObject {

    public static void main(String[] args) {
        People LiLei = new People();

        //setter方法为属性赋值
        LiLei.setHeight(170.0);

        //getter方法取属性值
        System.out.println("LiLei的身高是"+LiLei.getHeight());
    }
}
```

```
shiyanlou:project/ $ javac NewObject.java
shiyanlou:project/ $ java NewObject
LiLei的身高是170.0
```

8 This

this 关键字代表当前对象。使用 this.属性操作当前对象的属性，this.方法调用当前对象的方法。

9 继承

语法：class 子类 extends 父类

单继承 但可以实现多个接口

```java
public class Animal{
    public int legNum;

    public void bark(){
        System.out.println("叫！");
    }
}
```

```java
public class Dog extends Animal{

}
```

```java
public class Test{
    public static void main(String[] args) {
        Dog d = new Dog();
        d.legNum = 4;
        d.bark();
    }
}
```

```
shiyanlou:project/ $ javac Test.java
shiyanlou:project/ $ java Test
叫!
```

10 Super

super 关键字在子类内部使用，代表父类对象。

访问父类的属性 super.属性名

访问父类的方法 super.bark()

子类构造方法需要调用父类的构造方法时，在子类的构造方法体里最前面的位置：super()

11 方法重载和重写

方法重载是指在一个类中定义多个同名的方法，但要求每个方法具有不同的参数的类型或参数的个数。

方法中的参数列表必须不同。比如：参数个数不同或者参数类型不同。

重载的方法中允许抛出不同的异常

可以有不同的返回值类型，但是参数列表必须不同

可以有不同的访问修饰符

方法重写：子类可以继承父类的方法，但如果子类对父类的方法不满意，想在里面加入适合自己的一些操作时，就需要将方法进行重写。并且子类在调用方法中，优先调用子类的方法。方法名称、参数列表和返回类型与父类被重写方法相同，重写方法不能使用比被重写的方法更严格的访问权限。

```java
public class Animal{
    public int legNum;

    public void bark(){
        System.out.println("动物叫！ ");
    }
}
```

```java
public class Dog extends Animal{
    public void bark() {
        System.out.println("汪！ 汪！ 汪！ ");
    }
}
```

```java
public class Test{
    public static void main(String[] args) {
        Animal a = new Animal(); // Animal 对象
        Dog d = new Dog();   // Dog 对象

        Animal b = new Dog(); // Dog 对象,向上转型为Animal类型

        a.bark();
        d.bark();
        b.bark();
    }
}
```

```
shiyanlou:project/ $ javac Test.java
shiyanlou:project/ $ java Test
动物叫!
汪! 汪! 汪!
汪! 汪! 汪!
```

12 多态

多态是指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同

而采用多种不同的行为方式。

动态绑定（在执行期间判断所引用对象的实际类型，根据其实际的类型调用其相应的方法。）

13 向上转型

父类引用指向子类对象。

如果定义了一个指向子类对象的父类引用类型，那么它除了能够引用父类中定义的所有属性和方法外，还可以使用子类强大的功能。但是对于只存在于子类的方法和属性就不能获取。

```
1  class Animal {
2      public void bark() {
3          System.out.println("动物叫！");
4      }
5  }
6  class Dog extends Animal {
7      public void bark() {
8          System.out.println("汪、汪、汪！");
9      }
10     public void dogType() {
11         System.out.println("这是什么品种的狗？");
12     }
13 }
14 public class Test {
15     public static void main(String[] args) {
16         Animal a = new Animal();
17         Animal b = new Dog();
18         Dog d = new Dog();
19
20         a.bark();
```

```
shiyanlou:project/ $ javac Test.java
shiyanlou:project/ $ java Test
动物叫！
汪、汪、汪！
汪、汪、汪！
这是什么品种的狗？
```

14 多态实现条件：

继承 重写 向上转型

Java 中多态的实现方式：继承父类进行方法重写，抽象类和抽象方法，接口实现。

15 抽象类：

仅有声明而没有方法体。抽象方法声明语法：

```
abstract void f();  //f()方法时抽象方法
```

1. 用 abstract 修饰符定义抽象类
2. 用 abstract 修饰符定义抽象方法，只用声明，不需要实现
3. 包含抽象方法的类就是抽象类
4. 抽象类中可以包含普通的方法，也可以没有抽象方法
5. 抽象类的对象不能直接创建，通常是定义引用变量指向子类对象。

```java
public class CellPhone extends TelePhone{
    public void call(){
        System.out.println("Cell phone call");
    }
    public void message(){
        System.out.println("Cell phone message");
    }
    public static void main(String[] args){
        TelePhone cp = new CellPhone();
        cp.call();
        cp.message();
    }
}
```

```
shiyanlou:project/ $ javac CellPhone.java
shiyanlou:project/ $ java CellPhone
Cell phone call
Cell phone message
```

16 接口

```
修饰符 interface 接口名称 [extends 其他的接口名] {
        // 声明变量
        // 抽象方法
}
```

注意点：在Java8中

- 接口不能用于实例化对象
- 接口中方法只能是抽象方法、default 方法、静态方法
- 接口成员是 static final 类型
- 接口支持多继承

在Java9中，接口可以拥有私有方法和私有静态方法，但是只能被该接口中的 default 方法和静态方法使用。

```java
interface Animal {
        int y = 5;
        public void eat();
        public void travel();
}
public class Cat implements Animal{
    public void eat(){
        System.out.println("Cat eats");
    }
    public void travel(){
        System.out.println("Cat travels");
    }
    public static void main(String[] args) {
        Cat cat = new Cat();
        cat.eat();
        cat.travel();
    }
}
```

```
shiyanlou:project/ $ javac Cat.java
shiyanlou:project/ $ java Cat
Cat eats
Cat travels
```

17 内部类

内部类提供了更好的封装，可以把内部类隐藏在外部类之内，不允许同一个包中的其他类访问该类

内部类的方法可以直接访问外部类的所有数据，包括私有的数据

内部类所实现的功能使用外部类同样可以实现，只是有时使用内部类更方便

内部类允许继承多个非接口类型

```java
public class People{
    private String name = "wangxinyu";
    public class Student{
        String ID = "21617033";
        public void stuInfo(){
            System.out.println("访问外部类中的name: " + name);
            System.out.println("访问内部类中的ID: " + ID);
        }
    }
    public static void main(String[] args){
        People a = new People();
        People.Student b = a.new Student();

        b.stuInfo();
    }
}
```

```
shiyanlou:project/ $ javac People.java
shiyanlou:project/ $ java People
访问外部类中的name：wangxinyu
访问内部类中的ID：21617033
```

定义成员内部类后，必须使用外部类对象来创建内部类对象，即 内部类 对象名 = 外部类对象.new 内部类();

18 静态内部类 嵌套类

1. 静态内部类不能直接访问外部类的非静态成员，但可以通过 new 外部类().成员 的方式访问
2. 如果外部类的静态成员与内部类的成员名称相同，可通过类名.静态成员访问外部类的静态成员；如果外部类的静态成员与内部类的成员名称不相同，则可通过成员名直接调用外部类的静态成员
3. 创建静态内部类的对象时，不需要外部类的对象，可以直接创建 内部类 对象名= new 内部类();

```java
public class People{
    private String name = "wangxinyu";
    static String ID = "1234567890";
    public static class Student{
        String ID = "21617033";
        public void stuInfo(){
            System.out.println("访问外部类中的name: " + (new People().name
            System.out.println("访问内部类中的ID: " + ID);
            System.out.println("访问外部类中的ID: " + People.ID);
        }
    }
    public static void main(String[] args){
        People a = new People();
        People.Student b = new Student();
        b.stuInfo();
    }
```

```
shiyanlou:project/ $ javac People.java
shiyanlou:project/ $ java People
访问外部类中的name：wangxinyu
访问内部类中的ID：21617033
访问外部类中的ID：1234567890
```

19 局部内部类

局部内部类，是指内部类定义在方法和作用域内。

```java
public class People{
    public void peopleInfo() {
        final String sex = "man";  //外部类方法中的常量
        class Student {
            String ID = "20151234"; //内部类中的常量
            public void print() {
                System.out.println("访问外部类的方法中的常量sex：" + sex);
                System.out.println("访问内部类中的变量ID:" + ID);
            }
        }
        Student a = new Student();  //创建方法内部类的对象
        a.print();//调用内部类的方法
    }
    public void peopleInfo2(boolean b) {
        if(b){
            final String sex = "man";  //外部类方法中的常量
            class Student {
                String ID = "20151234"; //内部类中的常量
                public void print() {
                    System.out.println("访问外部类的方法中的常量sex：" + s
```

```
shiyanlou:project/ $ javac People.java
shiyanlou:project/ $ java People
定义在方法内：==========
访问外部类的方法中的常量sex：man
访问内部类中的变量ID:20151234
定义在作用域内：==========
访问外部类的方法中的常量sex：man
访问内部类中的变量ID:20151234
```

20 匿名内部类

匿名内部类是不能加访问修饰符的。new 匿名类，这个类是要先定义的,如果不先定义，编译时会报错该类找不到。

匿名内部类只能使用一次，但使用匿名内部类还有个前提条件：必须继承一个父类或实现一个接口。

```java
public class Outer {
    public Inner getInner(final String name, String city) {
        return new Inner() {
            private String nameStr = name;
            public String getName() {
                return nameStr;
            }
        };
    }

    public static void main(String[] args) {
        Outer outer = new Outer();
        Inner inner = outer.getInner("Inner", "NewYork");
        System.out.println(inner.getName());
    }
}
interface Inner {
    String getName();
}
```

```
shiyanlou:project/ $ javac Outer.java
shiyanlou:project/ $ java Outer
Inner
```

同时，在上面的例子中，当所在的方法的形参需要在内部类里面使用时，该形参必须为 final。这里可以看到形参 name 已经定义为 final 了，而形参 city 没有被使用则不用定义为 final。

21 包

```
package 包名
//注意：必须放在源程序的第一行，包名可用"."号隔开
```

22 Arrays

方法	描述
<T> List<T> asList(T... a)	返回由指定数组构造的 List
void sort(Object[] a)	对数组进行排序
void fill(Object[] a, Object val)	为数组的所有元素都赋上相同的值
boolean equals(Object[] a, Object[] a2)	检查两个数组是否相等
int binarySearch(Object[] a, Object key)	对排序后的数组使用二分法查找数据

```
1  import java.util.Arrays;
2  import java.util.Random;
3
4  public class ArraysDemo{
5      public static void main(String[] args){
6          int[] arr = new int[10];
7          Arrays.fill(arr, 8);
8          System.out.println("fill: "+Arrays.toString(arr));
9          Random random = new Random(47);
10         for(int i=0;i<10;i++){
11             arr[i] = random.nextInt(101);
12         }
13         System.out.println("Random后:"+Arrays.toString(arr));
14         arr[5] = 50;
15         Arrays.sort(arr);
16         System.out.println("Sort后:"+Arrays.toString(arr));
17         int j = Arrays.binarySearch(arr,50);
18         System.out.println("50索引位置:"+j);
```

```
shiyanlou:project/ $ javac ArraysDemo.java
shiyanlou:project/ $ java ArraysDemo
fill: [8, 8, 8, 8, 8, 8, 8, 8, 8, 8]
Random后:[67, 62, 17, 100, 15, 20, 95, 65, 18, 22]
Sort后:[15, 17, 18, 22, 50, 62, 65, 67, 95, 100]
50索引位置:4
equals:true
```

23

```
1  import java.util.Arrays;
2  public class ArraysTest{
3      public static void main(String[] args){
4          int[] arr = {6, 17, 92, 32, 58, 22, 84, 66, 36, 33};
5          int[] arr2 = {6, 17, 92, 32, 58, 22, 84, 66, 36, 33};
6          Arrays.sort(arr);
7          System.out.println("sorded:"+Arrays.toString(arr));
8          System.out.println("33位置:"+Arrays.binarySearch(arr,33));
9          int i =0;
10         for(;i<arr2.length;i++){
11             if(arr2[i]==33){
12                 System.out.println("33位置不排序:"+i);
13                 break;
14             }
15         }
16     }
17 }
```

```
shiyanlou:project/ $ javac ArraysTest.java
shiyanlou:project/ $ java ArraysTest
sorded:[6, 17, 22, 32, 33, 36, 58, 66, 84, 92]
33位置:4
33位置不排序:9
```

24 StringBuilder

构造方法:

构造方法	说明
StringBuilder()	构造一个其中不带字符的 StringBuilder，其初始容量为 16 个字符
StringBuilder(CharSequence seq)	构造一个 StringBuilder，它包含与指定的 CharSequence 相同的字符
StringBuilder(int capacity)	构造一个具有指定初始容量的 StringBuilder
StringBuilder(String str)	并将其内容初始化为指定的字符串内容

常用方法:

方法	返回值	功能描述
insert(int offsetm,Object obj)	StringBuilder	在 offsetm 的位置插入字符串 obj
append(Object obj)	StringBuilder	在字符串末尾追加字符串 obj
length()	int	确定 StringBuilder 对象的长度
setCharAt(int index,char ch)	void	使用 ch 指定的新值设置 index 指定的位置上的字符
toString()	String	转换为字符串形式
reverse()	StringBuilder	反转字符串
delete(int start, int end)	StringBuilder	删除调用对象中从 start 位置开始直到 end 指定的索引（end-1）位置的字符序列
replace(int start, int end, String str)	StringBuilder	使用一组字符替换另一组字符。将用替换字符串从 start 指定的位置开始替换，直到 end 指定的位置结束

```java
public class StringBuilderTest {
    public static void main(String[] args){
        StringBuilder s = new StringBuilder("I ");
        s.append("love ");
        s.insert(7,"you");
        System.out.println(s.toString());
        s.setCharAt(2,'L');
        System.out.println(s.toString());

        System.out.println(s.replace(1,2,"ab").toString());
        s.reverse();
        System.out.println(s.toString());
        System.out.println(s.length());

    }
}
```

```
shiyanlou:project/ $ java StringBuilderTest
I love you
I Love you
IabLove you
uoy evoLbaI
11
```

25 Calendar

```
1   port java.text.DateFormat;
2   port java.text.SimpleDateFormat;
3   port java.util.Calendar;
4   port java.util.Date;
5
6   blic class CalendarDemo {
7     public static void main(String[] args) {
8         System.out.println("完整显示日期时间：");
9         // 字符串转换日期格式
10        DateFormat fdate = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
11        String str = fdate.format(new Date());
12        System.out.println(str);
13
14        // 创建 Calendar 对象
15        Calendar calendar = Calendar.getInstance();
16        // 初始化 Calendar 对象，但并不必要，除非需要重置时间
17        calendar.setTime(new Date());
```

```
shiyanlou:project/ $ javac CalendarDemo.java
shiyanlou:project/ $ java CalendarDemo
完整显示日期时间：
2019-05-22 13:59:09
年： 2019
月： 4
分钟： 59
今年的第 142天
本月的第 22天
三小时以后的时间： Wed May 22 16:59:09 UTC 2019
2019-05-22 16:59:09:730
2019-05-22 13:59:09:796
时间比较：-1
时间比较：1
时间比较：0
2019-05-22 13:59:09:796
2019-05-22 13:59:09:796
时间比较：0
```

26 Date

Date 类表示日期和时间，里面封装了操作日期和时间的方法。Date 类经常用来获取系统当前时间。

构造方法	说明
Date()	构造一个 Date 对象并对其进行初始化以反映当前时间
Date(long date)	构造一个 Date 对象，并根据相对于 GMT 1970 年 1 月 1 日 00:00:00 的毫秒数对其进行初始化

```
1   import java.text.SimpleDateFormat;
2   import java.util.Date;
3
4   public class DateDemo {
5       public static void main(String[] args) {
6           String strDate, strTime;
7           Date objDate = new Date();
8           System.out.println("今天的日期是：" + objDate);
9           long time = objDate.getTime();
10          System.out.println("自1970年1月1日起以毫秒为单位的时间（GMT）：" +
11          strDate = objDate.toString();
12          //提取 GMT 时间
13          strTime = strDate.substring(11, (strDate.length() - 4));
14          //按小时、分钟和秒提取时间
15          strTime = "时间：" + strTime.substring(0, 8);
16          System.out.println(strTime);
17          //格式化时间
18          SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd H
19          System.out.println(formatter.format(objDate));
```

```
shiyanlou:project/ $ javac DateDemo.java
shiyanlou:project/ $ java DateDemo
今天的日期是：Wed May 22 14:02:58 UTC 2019
自1970年1月1日起以毫秒为单位的时间（GMT）：1558533778472
时间：14:02:58
2019-05-22 14:02:58
```

27 Math

方法	返回值	功能描述
sin(double numvalue)	double	计算角 numvalue 的正弦值
cos(double numvalue)	double	计算角 numvalue 的余弦值
acos(double numvalue)	double	计算 numvalue 的反余弦
asin(double numvalue)	double	计算 numvalue 的反正弦
atan(double numvalue)	double	计算 numvalue 的反正切
pow(double a, double b)	double	计算 a 的 b 次方
sqrt(double numvalue)	double	计算给定值的正平方根
abs(int numvalue)	int	计算 int 类型值 numvalue 的绝对值，也接收 long、float 和 double 类型的参数
ceil(double numvalue)	double	返回大于等于 numvalue 的最小整数值
floor(double numvalue)	double	返回小于等于 numvalue 的最大整数值
max(int a, int b)	int	返回 int 型 a 和 b 中的较大值，也接收 long、float 和 double 类型的参数
min(int a, int b)	int	返回 a 和 b 中的较小值，也可接受 long、float 和 double 类型的参数
rint(double numvalue)	double	返回最接近 numvalue 的整数值

round(T arg)	arg 为 double 时返回 long，为 float 时返回 int	返回最接近arg的整数值
random()	double	返回带正号的 double 值，该值大于等于 0.0 且小于 1.0

```java
public class MathDemo {
    public static void main(String[] args) {
        System.out.println(Math.abs(-12.7));
        System.out.println(Math.floor(-12.7));
        System.out.println(Math.ceil(-12.7));
        System.out.println(Math.rint(-12.7));
        System.out.println(Math.round(-12.7));
        System.out.println(Math.random());
        System.out.println("sin30 = " + Math.sin(Math.PI / 6));
        System.out.println("6*arcsin (0.5) = " +(6* Math.asin(0.5)));
    }
}
```

```
shiyanlou:project/ $ javac MathDemo.java
shiyanlou:project/ $ java MathDemo
12.7
-13.0
-12.0
-13.0
-13
0.1765669855827139
sin30 = 0.49999999999999994
6*arcsin (0.5) = 3.1415926535897936
```

28 练习

```java
public class MathTest {
    public static void main(String[] args) {
        double a = Math.random();
        double b = Math.random();
        System.out.println("a: "+a+"    b:"+b);
        System.out.println("较大: "+Math.max(a,b));
    }
}
```

```
shiyanlou:project/ $ javac MathTest.java
shiyanlou:project/ $ java MathTest
a: 0.011946016911698387    b:0.8133825452785273
较大: 0.8133825452785273
```

29 System

标准输入，标准输出和错误输出流;
访问外部定义的属性和环境变量;
加载文件和库的方法;
以及用于快速复制数组的实用方法
System 不可以被实例化，只可以使用其静态方法。

```java
import java.util.Arrays;

public class SystemDemo {
    public static void main(String[] args) {
        int[] a = {7, 8, 9, 10, 11};
        int[] b = {1, 2, 3, 4, 5, 6};
        //从数组a的第二个元素开始，复制到b数组的第三个位置 复制的元素长度为
        System.arraycopy(a, 1, b, 2, 3);
        //输出结果
        System.out.println(Arrays.toString(b));
        System.out.println("当前时间：" + System.currentTimeMillis());
        System.out.println("java版本信息：" + System.getProperty("java
        //运行垃圾收集器
        System.gc();
        //退出
        System.exit(0);
    }
}
```

```
shiyanlou:project/ $ javac SystemDemo.java
shiyanlou:project/ $ java SystemDemo
[1, 2, 8, 9, 10, 6]
当前时间：1558534931060
java版本信息：1.8.0_191
```

30 练习

```java
import java.util.Arrays;

public class SystemTest {
    public static void main(String[] args) {
        System.out.println("目录："+System.getProperty("java.home"));
        int[] a = {7, 8, 9, 10, 11,2,7};
        int[] b = {1, 2, 3, 4, 5, 6,8,9};
        System.arraycopy(a,3,b,2,3);
        System.out.println(Arrays.toString(b));
    }
}
```

```
shiyanlou:project/ $ javac SystemTest.java
shiyanlou:project/ $ java SystemTest
目录：/usr/lib/jvm/java-8-oracle/jre
[1, 2, 10, 11, 2, 6, 8, 9]
```

31 Random

```java
import java.util.Random;

public class RandomDemo {
    public static void main(String[] args) {
        Random random = new Random(47);
        //随机生成一个整数 int范围
        System.out.println(random.nextInt());
        //生成 [0,n] 范围的整数  设n=100
        System.out.println(random.nextInt(100 + 1));
        //生成 [0,n) 范围的整数  设n=100
        System.out.println(random.nextInt(100));
        //生成 [m,n] 范围的整数  设n=100 m=40
        System.out.println((random.nextInt(100 - 40 + 1) + 40));
        //随机生成一个整数 long范围
        System.out.print(random.nextLong());
        //生成[0,1.0)范围的float型小数
        System.out.println(random.nextFloat());
        //生成[0,1.0)范围的double型小数
```

```
shiyanlou:project/ $ javac RandomDemo.java
shiyanlou:project/ $ java RandomDemo
-1172028779
62
93
63
29552893544413037710.18847865
0.4170137422770571
```

32　　练习

```java
import java.util.Random;
import java.util.Scanner;
public class RandomTest {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int m = in.nextInt();
        int n = in.nextInt();
        if(m>n){
            System.out.println("输入错误");
            return;
        }
        Random random = new Random(47);
        int k = random.nextInt(n-m+1)+m;
        System.out.println("随机数: "+k);
    }
}
```

```
shiyanlou:project/ $ javac RandomTest.java
shiyanlou:project/ $ java RandomTest
5
100
随机数: 79
```

2 总结

今天完成了实验 3 面向对象和实验 4 常用类部分，许多常用类都需要在平常敲代码中更加熟练。
明日计划：泛型集合 异常 以及 Linux 基础部分内容。时间充裕继续往下。