院系	专业	学号	姓名	时间
信息学院	计科	22920212204066	邓语苏	2023/05/01

实验一：Date,Random,Point2D综合应用

实验要求：

- [x] 创建一个Date对象，用Date的方法获取当前的时间time
- [x] 用String类提供的方法对获取到的时间time分割，并按照"yyyy-mm-dd-hh:mm”格式输出(yyyy表示四位年份，mm表示二位数字月份，dd表示二位日期，hh表示二位小时，最后一个mm表示二位分钟)
- [x] 设置种子数，创建一个Random类，生成两个浮点数字并输出
- [x] 生成20个整型数字：x1,X2,,X10;y1,y2,,y10
- [x] 用生成的x和y,按下标两两对应组合，创建10个Point2D对象（即(x,y)为一个点，(x2,y2)为一个点)
- [x] 实现对这10个点的KMeans聚类（至少聚2类），并用Point2D提供distance()的方法计算点之间的欧氏距离
  KMeans聚类算法思想参考网上资源

篇幅限制，只展示核心代码

用Date对象获取当前时间

```java
//导包
import java.text.SimpleDateFormat;
import java.util.Date;

        //创建一个Date对象
        Date date=new Date();

        //格式化时间
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd-hh:mm");
        String time=formatter.format(date);
        System.out.println(time);
```

分割时间

```java
        //分割时间
        String time2[]=time.split("[-:]");
        for(int i=0;i<time2.length;i++){//数组为.length，字符串为.length()
            System.out.println(time2[i]);
        }
```

设置种子+生成两个浮点数+生成20个整数

```java
        //设置种子
        int seed=sc.nextInt();
        Random rand=new Random(seed);

        //获得两个浮点数
        double a,b;
        a= rand.nextDouble(10);
        b= rand.nextDouble(10);
        System.out.println("a="+a+" b="+b);

        //生成20个随机整数
        for(int i=0;i<10;i++)
            x[i]= rand.nextInt(100);
        for(int i=0;i<10;i++)
            y[i]= rand.nextInt(100);
```

Point2D的UML图

成员变量/方法	含义	类型
px	x坐标	double
py	y坐标	double
belong	所属点	Point2D
public Point2D(int px,int py)	带参构造方法	
setX	设置x	
setY	设置y	
setBelong	设置归属点	
getX	获取x	double
getY	获取y	double
getBelong	获取归属点	Point2D
distance(Point2D p1,Point2D p2)	计算p1,p2点之间的欧氏距离	double

具体看附件---

Kmeans聚类（聚两类）

step1. 初始化

初始选择任两点作为聚类中心c1,c2，计算其他点与c1,c2的距离，离谁近归属于谁。

```
        //若聚两类
        Point2D c1=Point[0];
        Point2D c2=Point[1];
        while(true) {
            System.out.println("原始聚类中心为: x:" + c1.getX() + " y:" + c1.getY()
+ " x2:" + c2.getX() + " y2:" + c2.getY());
            int flag = 0;
            for (int i = 0; i < 10; i++) {//初始化
                double a, b;
                a = temp.distance(c1, Point[i]);
                b = temp.distance(c2, Point[i]);
                if (a < b) {
                    Point[i].setBelong(c1);
                } else {
                    Point[i].setBelong(c2);
                }
            }
```

step2.更新聚类中心

不断对同一个聚类的点取平均，得到一个更居中的聚类中心。若新中心相比较旧中心没有变化的话，聚类饱和，停止聚类

```
while(true){
```

...具体看附件

```
    double x1 = 0, y1 = 0, x2 = 0, y2 = 0;
    int sum1 = 0, sum2 = 0;
    for (int i = 0; i < 10; i++) {
        if (Point[i].getBelong() == c1) {
            x1 += Point[i].getX();
            y1 += Point[i].getY();
            sum1++;
        } else {
            x2 += Point[i].getX();
            y2 += Point[i].getY();
            sum2++;
        }
    }

    x1 /= (double) sum1;
    y1 /= (double) sum1;
    x2 /= (double) sum2;
    y2 /= (double) sum2;
    if (x1 != c1.getX() || y1 != c1.getY() || x2 != c2.getX() || y2 !=
c2.getY())
        flag = 1;

    c1.setX(x1);
    c1.setY(y1);
    c2.setX(x2);
    c2.setY(y2);

    if (flag == 0){
        System.out.println("聚类中心未更新");
        break;//退出循环
    }
    else
        System.out.println("更新后聚类中心为: x:" + c1.getX() + " y:" + c1.getY() +
" x2:" + c2.getX() + " y2:" + c2.getY());
}
```

实现效果

```
2023-04-26-09:58
2023
04
26
09
58
25
a=7.315948597820919 b=0.5372565982763688
x[i]:97 y[i]:81
x[i]:98 y[i]:70
x[i]:95 y[i]:26
x[i]:37 y[i]:7
x[i]:96 y[i]:54
```

```
x[i]:54 y[i]:82
x[i]:40 y[i]:61
x[i]:8 y[i]:70
x[i]:59 y[i]:14
x[i]:16 y[i]:34
原始聚类中心为: x:97.0 y:81.0 x2:98.0 y2:70.0
更新后聚类中心为: x:53.0 y:77.66666666666667 x2:63.0 y2:38.0
原始聚类中心为: x:53.0 y:77.66666666666667 x2:63.0 y2:38.0

(....中间省略若干行)

原始聚类中心为: x:29.500000000000002 y:61.75000000000002 x2:71.74999999999997
y2:25.250000000000007
更新后聚类中心为: x:29.500000000000007 y:61.75 x2:71.75 y2:25.25
原始聚类中心为: x:29.500000000000007 y:61.75 x2:71.75 y2:25.25
更新后聚类中心为: x:29.5 y:61.75 x2:71.75 y2:25.25
原始聚类中心为: x:29.5 y:61.75 x2:71.75 y2:25.25
聚类中心未更新
最终聚类中心为: x:29.5 y:61.75 x2:71.75 y2:25.25
```

实验二：设计一个汽车租赁程序

实验要求：

1. 创建三个类，分别为**Car、RentalCarCompany、RentalCarTester**
    - **Car** theCar，一个汽车类型的对象，存储了：租车人姓名**carRenter**，车号**carNum**，车品牌**carName**，车的类型**carType**，单日租金**rate**，租赁天数**days**
    - **RentalCarCompany** XMU，一个租赁公司类型的对象，存储了：汽车类型的对象数组**Car[] theCars**，公司的名字**name**，租赁总天数**totalDays**，总单日租金**totalRate**，多少车已租出**rentCnt**
    - **RentalCarTester** 测试类
2. 对**Car类**的要求

成员变量/方法	含义	类型
carRenter	租车人姓名	String
carNum	车号	int
carName	车品牌	String
carType	汽车类型	String
rate	单日租金	double
days	租车天数	int
Cars	默认构造方法	
Car(String,int ,...,int)	有参构造方法	
get类		
set类	设置各成员变量	
toString	返回所有信息	

3. 对**RentalCarCompany类**的要求

成员变量/方法	含义	类型
theCar	多个Car类型的数组	Car[]
name	公司名称	String
totalDays	总租赁天数	int
totalRate	总单日租金	double
rentCnt	租出车的总数	int
RentalCarCompany()	默认构造方法	
RentalCarCompany(String)	有参构造方法	
getName()	获取公司名字	String
setName()	设置公司名字	
addReservation()	将租出的Car信息录入Car[]数组中	
getTotalRentalSales	获取租出这些车将获得的收益	double
getAvgDays	计算平均租赁天数	double
getAvgRate	计算平均日租金	double
findReservation(int Num)	输入车号查找租赁信息	

4. 对**RentalCarTester类**(测试类)的要求

- [x] 创建**RentalCarCompany类**的对象
- [x] 将租车信息通过**addReservation()**方法录入

- [x] 用**findReservation()**方法查找有没有相应车的租赁信息
- [x] 调用**RentalCarCompany**中获取平均租赁时长、平均租金、总收益的方法获取这些信息

篇幅限制，只展示核心代码

Car

```
    //默认构造方法-设为默认值
    public Car(){
        carRenter="";
        carNum=0;
        carName="";
        carType="";
        rate=0;
        days=0;
    }
    //有参构造方法
    public Car(String carRenter,int carNum,String carName,String carType,double
rate,int days){
        this.carRenter=carRenter;
        this.carNum=carNum;
        this.carName=carName;
        this.carType=carType;
        this.rate=rate;
        this.days=days;
    }
    public String toString(){
        String s1="Car renter's name:"+carRenter+"\n"+"Car
number:"+carNum+"\n"+"Car name:"+carName+"\n"+"Car
type:"+carType+"\n"+"Rate:"+rate+"\n"+"Rented for:"+days+"days";
        return s1;
```

RentalCarCompany

```
public class RentalCarCompany {
    ...较简单，略过，详细看附件

    //这些车将获得的受益--Car[]数组中所有受益的和
    public double getTotalSales(){
        double sum=0;
        for(int i=0;i<rentCnt;i++){
            sum+=theCars[i].getRate()*(double)theCars[i].getDays();
        }
        return sum;
    }
    //平均租赁时长
    public double getAvgDays(){
        return  (double)totalDays/(double)rentCnt;
    }
    //平均租金
    public double getAvgRate(){
        return  (double)totalRate/(double)rentCnt;
```

```
    }

    //查找有没有对应车号的相应车的租赁信息
    public void findReservation(int Num){
        int flag=0;
        for(int i=0;i<rentCnt;i++){
            if(Num==theCars[i].getCarNum()){
                System.out.println(theCars[i].toString());
                flag=1;
                break;
            }
        }
        if(flag==0)
            System.out.println("Could not find reservation for this car number
"+Num);
    }
    //录入信息
    public void addReservation(){
        theCars[rentCnt]=new Car();
        Scanner sc=new Scanner(System.in);

        String renter;
        System.out.print("Car renter's name:");
        renter=sc.nextLine();
        theCars[rentCnt].setCarRenter(renter);

        ...省略

        //组多少天
        int days;
        System.out.print("Rented for:");
        days=sc.nextInt();
        theCars[rentCnt].setDays(days);

        totalRate+=theCars[rentCnt].getRate();
        totalDays+=theCars[rentCnt].getDays();
        rentCnt++;
    }

}
```

RentalCarTester

较简单，略过

实现效果

```
Rental Car Company:XMU
Car renter's name:Andy
Car number:1
Car name:Nissan
Car type:compact
Rate: ￥200
```

```
Rented for:6

Car renter's name:Taylor
Car number:2
Car name:Mazda
Car type:mid
Rate: ￥100
Rented for:5

Car renter's name:Joe
Car number:3
Car name:Chevy
Car type:SUV
Rate: ￥450.99
Rented for:7

Car renter's name:Jane
Car number:4
Car name:Ford
Car type:convertible
Rate: ￥320.99
Rented for:4

Found reservation for car number: 1
Car renter's name:Andy
Car number:1
Car name:Nissan
Car type:compact
Rate:200.0
Rented for:6days

Average days rented out is :5.5
Average rate is: ￥267.995
Total rental income is: ￥6140.89

Found reservation for car number: 200
Could not find reservation for this car number 200

Process finished with exit code 0
```

实验三：智能钱包

描述：

普通钱包只能手动存取钱，这十分麻烦，现在我们设计了一个智能钱包，其中可以存储1,5,10,20,100元的纸币，在你需要取钱时，智能钱包可以自动计算出是否能取出对应数值的纸币。

对Wallet要求：

成员变量/方法	含义	类型	附加要求
value	存储面额为1,5,10,20,100的张数	int[5]	
w	{1,5,10,20,100}	int[5]	
c	进位所需要的张数	int[4]	
Wallet()	默认构造方法，value全部置0		
Wallet(x,y,z,w,k)	带参构造方法		
deposit(int facevalue)	存钱，facevalue为一张纸币		
balance()	获取钱包总金额	int	
withdraw(int amount)	取出amount金额的钱		1. 当无法取出amout时，返回最多能给的钱 2.从小到大取钱
toString	返回Wallet当前所有的信息	String	

篇幅限制，只展示核心代码

desposit

```
//存钱
    public void deposit(int facevalue){
        switch (facevalue){
            case 1:{
                value[0]++;
                break;
            }
            case 5:{
                value[1]++;
                break;
            }
            case 10:{
                value[2]++;
                break;
            }
            case 20:{
                value[3]++;
                break;
            }
            case 100:{
                value[4]++;
                break;
            }
            default:break;
        }
```

```
    }
```

withdraw

算法思想：

先从大到小取，再将大面额的钱，用钱包中剩余的小面额替代。即可实现从小到大取。若直接从小到大取，可能会出现例如￥1*3, ￥5*1，要求取出6元，根据从小到大的规则，先取出3元，剩下一张5元的，无法取出，失败的情况。

```
     //取钱
    public void withdraw(int amount) {
        int[] value_temp=value.clone();
        int temp=amount,x;
        for(int i=4;i>=0;i--){
            x=temp/w[i];
            if(value_temp[i]>=x){
                temp-=x*w[i];
                value_temp[i]-=x;
            }
        }
        if(temp==0){
            System.out.println("success to withdraw "+amount+" yuan!");
        }else{
            x=amount-temp;
            System.out.println("can only withdraw "+x+" yuan!");
            return;
        }

        //每位用到的张数
        int[] use=value.clone();
        for(int i=4;i>=0;i--) {
            use[i] = value[i] - value_temp[i];
            if (i == 4)
                continue;
            //若高位有用到
            if (use[i + 1] != 0) {
                x = value_temp[i] / c[i];
                if (use[i + 1] >= x) {//若用到的比可替代的多
                    value_temp[i + 1] += x;
                    value_temp[i] -= x*c[i];
                } else {
                    value_temp[i + 1] += use[i + 1];
                    value_temp[i] -= use[i]*c[i];
                }
                use[i] = value[i] - value_temp[i];
            } else continue;
        }
        value=value_temp;
    }
```

```
7 2 3 1 1
balance: ￥1 * 7,￥5 * 2,￥10 * 3,￥20 * 1,￥100 * 1 total: ￥167
```

```
存入请按1,取出请按2,终止服务请按0
1
5
deposit 5 yuan:
balance: ￥1 * 7,￥5 * 3,￥10 * 3,￥20 * 1,￥100 * 1 total: ￥172
2
25
try to withdraw 25 yuan:
success to withdraw 25 yuan!
balance: ￥1 * 2,￥5 * 1,￥10 * 2,￥20 * 1,￥100 * 1 total: ￥147
2
22
try to withdraw 22 yuan:
success to withdraw 22 yuan!
balance: ￥1 * 0,￥5 * 1,￥10 * 0,￥20 * 1,￥100 * 1 total: ￥125
2
22
try to withdraw 22 yuan:
can only withdraw 20 yuan!
balance: ￥1 * 0,￥5 * 1,￥10 * 0,￥20 * 1,￥100 * 1 total: ￥125
```

实验四、找出智能钱包中纸笔的种类和数目

描述：虽然我们有了一个智能钱包对象，但是没有任何方法可以直接获取到其中的纸币种类和数目。请设计一个方法，通过利用智能钱包对象提供的公共方法(withdraw,deposit,balance)来间接的探查其中的纸币种类和数日。

要求：

- [x] 在探查前后，需要保证智能钱包对象中纸币的种类和数目不变
- [x] 在保证结果正确的情况下，尽可能减少调用withdraw和deposit
- [x] 请在一个测试类中实现算法，并给出以下三个测试用例的结果，以及调用withdraw的次数和deposit的次数

算法思想：

由于withdraw方法是从小到大取，若想要得到钱包中的每种面额的张数，只能从小到大取，不然大的面额的前会用小面额替代，所得到的大面额张数偏小。所调用的withdraw的次数和deposit的次数均为8次

```
public static void getDetail(Wallet E){
        int[] value={0,0,0,0,0};
        int sum=0;

        //从小到大取
        int y=E.withdraw(1);
        while(y==1){
            value[0]++;
            sum+=y;
            y=E.withdraw(1);
        }
        y=E.withdraw(5);
        while(y==5){
            value[1]++;
```

```
            sum+=y;
            y=E.withdraw(5);
        }
        y=E.withdraw(10);
        while(y==10){
            value[2]++;
            sum+=y;
            y=E.withdraw(10);
        }
        y=E.withdraw(20);
        while(y==20){
            value[3]++;
            sum+=y;
            y=E.withdraw(20);
        }
        y=E.withdraw(100);
        while(y==100){
            value[4]++;
            sum+=y;
            y=E.withdraw(100);
        }

        System.out.println("￥1 * "+value[0]+",￥5 * "+value[1]+",￥10 *
"+value[2]+",￥20 * "+value[3]+",￥100 * "+value[4]+" total: "+"￥"+sum);


        //将取出的重新放入
        while(value[0]!=0){
            E.deposit(1);
            value[0]--;
        }
        while(value[1]!=0){
            E.deposit(5);
            value[1]--;
        }
        while(value[2]!=0){
            E.deposit(10);
            value[2]--;
        }
        while(value[3]!=0){
            E.deposit(20);
            value[3]--;
        }
        while(value[4]!=0){
            E.deposit(100);
            value[4]--;
        }
    }
```

测试用例的结果:

```
create a wallet:
7 2 3 1 1
balance: ￥1 * 7,￥5 * 2,￥10 * 3,￥20 * 1,￥100 * 1 total: ￥167
不使用toString函数查看Wallet:
```

```
￥1 * 7,￥5 * 2,￥10 * 3,￥20 * 1,￥100 * 1 total: ￥167
create a wallet:
2 7 0 3 2
balance: ￥1 * 2,￥5 * 7,￥10 * 0,￥20 * 3,￥100 * 2 total: ￥297
不使用toString函数查看Wallet:
￥1 * 2,￥5 * 7,￥10 * 0,￥20 * 3,￥100 * 2 total: ￥297
create a wallet:
17 12 4 5 6
balance: ￥1 * 17,￥5 * 12,￥10 * 4,￥20 * 5,￥100 * 6 total: ￥817
不使用toString函数查看Wallet:
￥1 * 17,￥5 * 12,￥10 * 4,￥20 * 5,￥100 * 6 total: ￥817
```