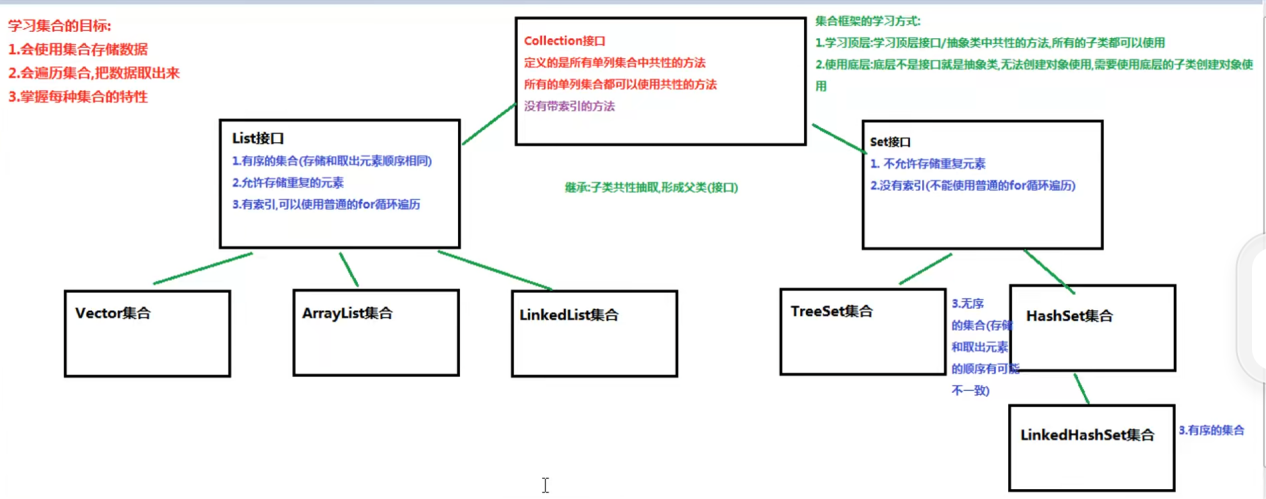
1. Collection集合

6.1 集合的概述Collection集合：  
集合概念：集合是java中提供的一种容器，可以用来存储多个数据。  
集合和数组都是容器，他们的区别：  
 数组的长度是固定的。集合的长度是可以改变的。  
 数组中存储的是同一类型的元素，可以存储基本数据类型值，集合存储的都是对象。而且对象的类型  
 不可以一致。在开发中一般当对象多的时候，使用集合进行存储。  
 例如：  
 数组：int[] arr = new int[10]; 存储基本数据类型  
 Student[] arr1 = new Student[3]; 存储对象  
 集合：ArrayList<Student> <String> <Integer> 都必须存储的是对象  
 6.2 集合框架



import java.util.ArrayList;  
import java.util.Collection;  
  
/\*  
集合框架：  
集合按照存储结构可以分为两大类：单排列集合java.util.Collection 、双排列结合java.util.Map  
Collection：单排列集合的根接口。  
  
java.util.Collection接口：  
 所有单列集合的最顶层的接口，里边定义了所有单列集合共性的方法。  
 任意的单列表集合都可以使用Collection接口中的方法。  
  
共性的方法：  
 public boolean add(E e):把给定的对象添加到当前集合中。  
 public boolean remove(E e):把给定的对象在当前集合中删除。  
 public boolean contains(E e):判断当前集合是否包含给定中的删除。  
 public boolean isEmpty():判断当前集合是否为空。  
 public int size():返回集合中元素的个数。  
 public Object[] toArray():把集合中的元素，存储到数组中。  
 public void clear():清空集合中所有的元素。  
 \*/  
public class DemoCollection {  
 public static void main(String[] args) {  
  
 // 创建集合对象，可以使用多态(接口指向对应的实现类)  
 Collection<String> coll = new ArrayList<>();  
 System.out.println(coll); // [] 重写了toString方法  
 /\*  
 public boolean add(E e):把给定的对象添加到当前集合中。  
 返回值是一个boolean值，一般都返回true，所有可以不用接收。  
 \*/  
 // 第一种接收返回值，返回值为boolean类型  
 boolean b1 = coll.add("张三");  
 // 第二种不接收值  
 coll.add("李四");  
 coll.add("赵六");  
 coll.add("田七");  
 System.out.println("b1:"+b1); // b1:true  
 System.out.println(coll); // [张三, 张三]  
  
 /\*  
 public boolean remove(E e):把给定的对象在当前集合中删除。  
 返回值是一个boolean值，集合中存在元素，删除元素，返回true  
 集合中不存在元素，删除失败，返回false  
 \*/  
 boolean b2 = coll.remove("赵六");  
 System.out.println("b2:"+b2); // b2:true  
 System.out.println(coll); // [张三, 李四, 田七]  
  
 boolean b3 = coll.remove("赵六");  
 System.out.println("b3:"+b3); // b3:false  
 System.out.println(coll); // [张三, 李四, 田七]  
  
 /\*  
 public boolean contains(E e):判断当前集合是否包含给定中的删除。  
 包含返回true  
 不包含返回false  
 \*/  
 boolean b4 = coll.contains("李四");  
 System.out.println("b4"+b4); // b4:true  
  
 boolean b5 = coll.contains("李四");  
 System.out.println("b5"+b5); // b5:false  
  
 // public boolean isEmpty():判断当前集合是否为空。集合为空返回为true，集合不为空返回false  
 boolean b6 = coll.isEmpty();  
 System.out.println("b6:"+b6); //b6:false  
  
 // public int size():返回集合中元素的个数。  
 int size = coll.size();  
 System.out.println("size:"+size); // size:3  
  
 // public Object[] toArray():把集合中的元素，存储到数组中。  
 Object[] arr = coll.toArray();  
 // 遍历一下数组  
 for (int i = 0; i < arr.length; i++) {  
 System.out.println(arr[i]); // 张三 李四 田七  
 }  
  
 // public void clear():清空集合中所有的元素。  
 coll.clear();  
 System.out.println(coll); // []  
 }  
  
}

1. Iterator迭代器

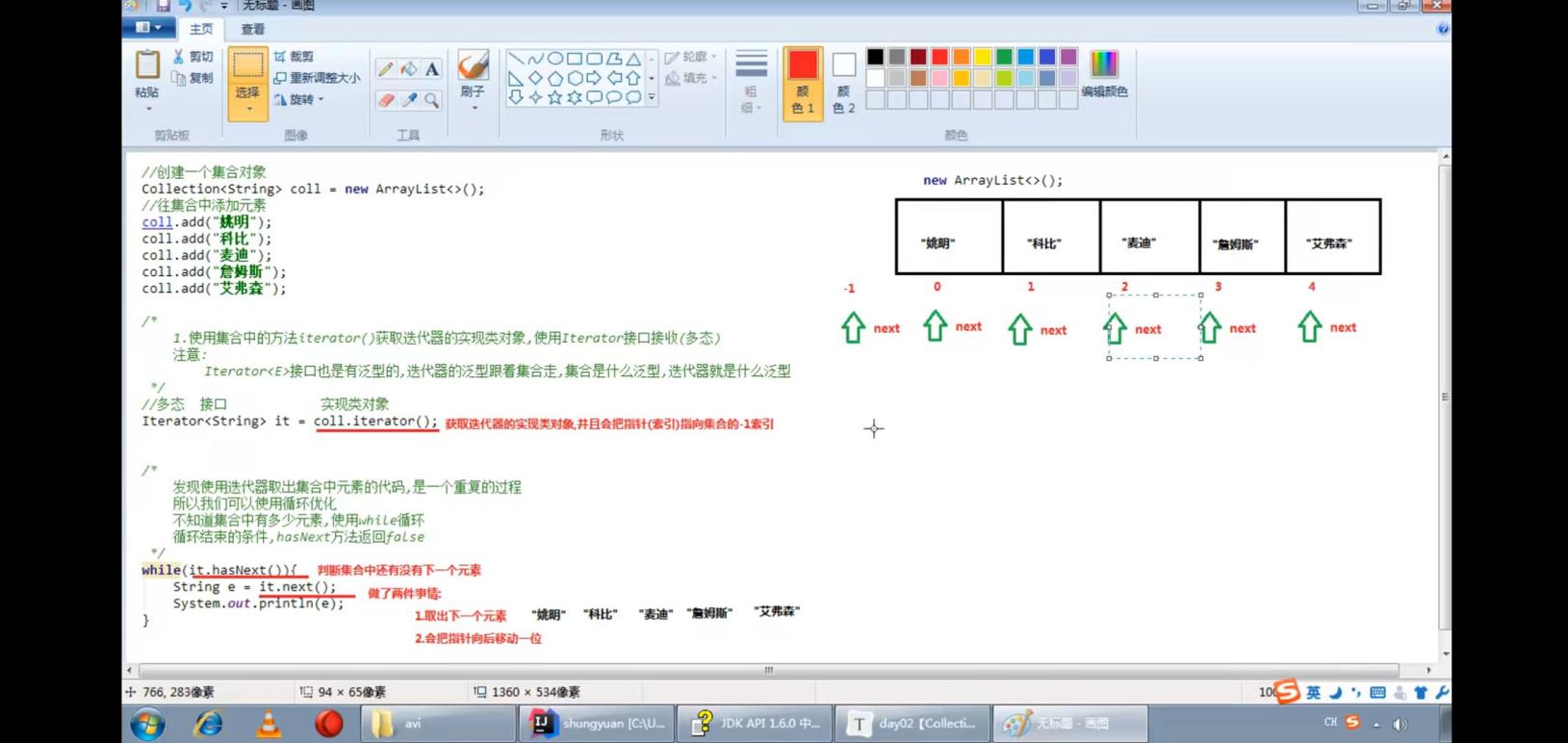
7.1 Iterator 接口

在程序开发中，经常需要遍历集合中的所有元素。针对这种需求，JDK专门提供了一个接口

java.util.Iterator。

package demo01;  
  
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.Iterator;  
  
/\*  
java.util.Iterator接口:迭代器（对集合进行遍历）  
有两个常见的方法  
 boolean hasNext() 如果仍有元素可以迭代，则返回 true  
 判断集合中还有没有下一个元素，有就返回true，没有则返回false  
 E next() 返回迭代器的下一个元素。  
 取出集合中的下一个元素。  
 Iterator迭代器，是一个接口，我们无法直接使用，需要使用Iterator接口的实现类对象，获取实现类的方式比较特殊。、  
 Collection接口中有一个方法，叫iterator()，这个方法的返回值就是迭代器的实现类对象。  
 Iterator<E> iterator() 返回在此 collection 的元素上进行迭代的迭代器。  
 迭代器的使用步骤（重点）：  
 1.使用集合中的方法 iterator 获取迭代器的实现类对象，使用Iterator接口接收（多态）  
 2.使用Iterator接口中的方法hasNext判断还没有下一个元素  
 3.使用Iterator接口中的方法next取出集合中的下一个元素。  
  
 \*/  
public class Demo01Iterator {  
 public static void main(String[] args) {  
 // 创建一个集合对象  
 Collection<String> coll = new ArrayList<>();  
 // 往集合中添加元素  
 coll.add("姚明");  
 coll.add("科比");  
 coll.add("麦迪");  
  
 /\*  
 1.使用集合中的方法iterator()获取迭代器的实现类对象，使用Iterator接口接收（多态）  
 注意：  
 Iterator<E> 接口也是有泛型的，迭代器的泛型跟着集合走，集合是什么泛型，迭代器就是什么泛型。  
  
 \*/  
 // 多态 接口 实现类对象  
 Iterator<String> it = coll.iterator();  
  
 /\*  
 发现使用迭代器取出结合中的代码，是一个重复的过程  
 所以我们可以使用循环优化  
 不知道集合中有多少元素，使用while循环  
 循环结束的条件，hasNext方法false  
 \*/  
 while(it.hasNext()){  
 String e = it.next();  
 System.out.println(e);  
 }  
 // 2.使用Iterator接口中的方法hasNext判断还有没有下一个元素  
 boolean b = it.hasNext();  
 System.out.println(b); // true  
  
 // 3.使用Iterator接口中的方法next取出集合中的下一个元素。  
 String s= it.next();  
 System.out.println(s); // 姚明  
  
 b = it.hasNext();  
 System.out.println(b); // true  
 s = it.next();  
 System.out.println(s); // 科比  
  
 b = it.hasNext();  
 System.out.println(b); // true  
 s = it.next();  
 System.out.println(s); // 麦迪  
  
 b = it.hasNext();  
 System.out.println(b); // 没有元素，返回false  
 s = it.next();  
 System.out.println(s); // 没有元素，在取出元素会抛出NoSuchElementException没有元素异常  
  
 }  
}

7 .2 迭代器的实现原理



8.1 增强for

增加for循环（也称for each）JDk1.5以后出来的一个高级for循环，专门用来遍历数组和集合的。他的内部原理其实是个Iterator迭代器，所以在遍历的过程中，不能对集合中的元素进行增删操作。

**格式：for（元素的数据类型 变量 : Collection集合or数组）{**

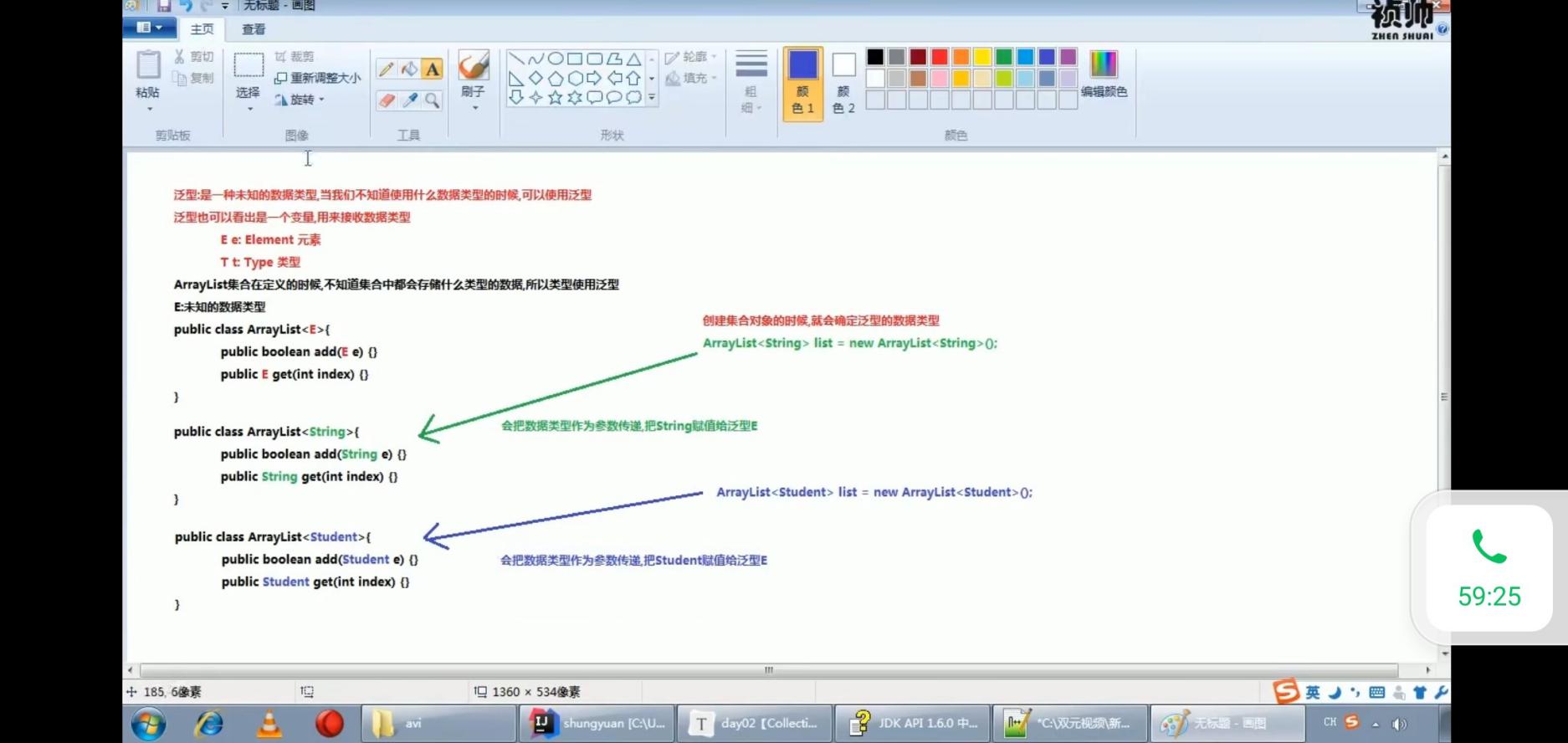
**// 写操作代码**

**}**

它用于遍历Collection和数组。通常只进行遍历元素，不要在遍历的过程中对集合元素进行增删操作。

package demo01;  
  
 import java.util.ArrayList;  
  
/\*  
增强的for循环：底层使用的也是迭代器，使用for循环的格式，简化了迭代器的书写。  
是JDK1.5以后出现的新特征。  
 Collection<E> extends Iterable<E> ：所有的单列集合都可以使用增强的for  
 public interface Iterable<T> ：实现这个接口允许对象成为 ”foreach“ 语句的目标。  
  
 增强for循环：用来遍历集合和数组。  
  
 格式：  
 for(集合/数组的类型 变量名 ：集合名/数组名){  
 sout(变量名);  
 }  
 注意：  
 新for循环必须有被遍历的目标。目标只能是Collection或者是数组。新式for仅仅作为遍历操作出现。  
 \*/  
public class Demo01For {  
 public static void main(String[] args) {  
 demo01();  
 demo02();  
 }  
  
 // 使用增强的 for 循环遍历数组  
 private static void demo01(){  
 int[] arr = {1,2,3,4,5};  
 for (int i : arr){  
 System.out.print(i); // 12345  
 }  
 }  
  
 // 使用增强的for循环遍历集合  
 private static void demo02(){  
 ArrayList<String> list = new ArrayList<>();  
 list.add("aaa");  
 list.add("bbb");  
 list.add("ccc");  
 list.add("ddd");  
  
 for (String s : list){  
 System.out.print(s); // aaabbbcccddd  
 }  
 }  
}

9.1 泛型的概念



package demo01;  
  
import java.util.ArrayList;  
import java.util.Iterator;  
  
public class Demo01Generic {  
 public static void main(String[] args) {  
 show01();  
 show02();  
 }  
 /\*  
 创建集合对象，不使用泛型  
 好处：  
 集合不使用泛型，默认的类型就是Object类型，可以存储任意类型的数据。  
 弊端：  
 不安全，会引发异常  
 \*/  
  
 private static void show01(){  
 ArrayList list = new ArrayList();  
 list.add("abc");  
 list.add(1);  
  
 // 使用迭代器遍历list集合  
 // 获取迭代器  
 Iterator it = list.iterator();  
 // 使用迭代器中的方法hasNext和next遍历集合。  
 while(it.hasNext()){  
 // 取出元素也是Object类型  
 Object obj = it.next();  
 System.out.println(obj); // abc 1  
  
 /\*  
 想要使用String类特有的方法，length获取字符串的长度；不能使用 多态 Object obj = "abc";  
 需要向下转型  
 会抛出ClassCastException类型转换异常，不能把Integer类型转换为String类型。  
 \*/  
 String s = (String)obj;  
 System.out.println(s.length()); // abc ClassCastException(1不可以转换为String)  
 }  
 }  
  
 /\*  
 创建集合对象，使用泛型  
 好处：  
 1.避免了类型转换的麻烦，存储的是什么类型，取出的就是什么类型。  
 2.把运行期异常（代码运行之后会抛出异常），提升到了编译期（写代码的时候会报错）  
 弊端：  
 泛型是什么类型，只能存储什么类型的数据。  
 \*/  
 private static void show02(){  
 ArrayList<String> list = new ArrayList<>();  
 list.add("abc");  
 // list.add(1); 会直接报错，因为1不属于String类型  
  
 // 使用迭代器遍历list集合  
 // 迭代器的泛型跟着集合类型走  
 Iterator<String> iterator = list.iterator();  
 while (iterator.hasNext()){  
 String s = iterator.next();  
 System.out.println(s); // abc  
 }  
 }  
}

9.2 泛型的定义和使用（类）

泛型，可以灵活地将数据类型应用到不用的类、方法、接口当中。将数据类型作为参数进行传递。

1. 定义和使用含有泛型的类

定义格式：

修饰符 class 类名<代表泛型的变量> { }

例如：API种的ArrayList集合：

class ArrayList<E>{

public boolean add(E e) { }

public E get(int Index) { }

...

}

定义一个含有泛型的类：

package demo02;  
/\*  
定义一个含有泛型的类，模拟ArrayList集合  
泛型是一个未知的数据类型，当我们不确定什么什么数据类型的时候，就可以用泛型。  
泛型可以接收任意的数据类型，可以使用Integer、String、Student...  
创建对象的时候确定泛型的数据类型。  
 \*/  
public class Demo03<E> {  
 private E name;  
  
 public E getName() {  
 return name;  
 }  
  
 public void setName(E name) {  
 this.name = name;  
 }  
}

主函数：

package demo02;  
  
public class Demo03Main {  
 public static void main(String[] args) {  
 // 不写泛型默认为Object类型  
 Demo03 gc = new Demo03();  
 gc.setName("只是字符串");  
 Object obj = gc.getName(); // 只是字符串 且为Object类型  
  
 // 创建泛型Demo03对象，泛型使用Integer类型  
 Demo03<Integer> gc01 = new Demo03<>();  
 gc01.setName(1);  
  
 Integer name = gc01.getName();  
 System.out.println(name); // 1  
  
 // 创建一个泛型Demo03对象，泛型使用String类型  
 Demo03<String> gc02 = new Demo03<>();  
 gc02.setName("王蓉");  
  
 String name1 = gc02.getName();  
 System.out.println(name1); // 王蓉  
 }  
}

1. 定义和使用含有泛型的方法定义

定义格式：

修饰符 <代表泛型的变量> 返回值类型 方法名(参数){ }

类函数：

package demo02;  
  
/\*  
定义含有泛型的方法：泛型定义在方法的修饰符和返回值类型之间。  
格式：  
 修饰符 <泛型> 返回值类型 方法名(参数列表(使用泛型)){  
 方法体；  
 }  
  
 含有泛型的方法，在调用方法的时候确定泛型的数据类型。  
 传递什么类型的参数，泛型就是什么类型。  
 \*/  
public class Demo02Main {  
 // 定义一个含有泛型的方法  
 public <M> void method01(M m){  
 System.out.print(m);  
 }  
  
 // 定义一个含有泛型的静态方法  
 public static <S> void method02(S s){  
 System.out.print(s);  
 }  
}

主函数：

package demo02;  
  
/\*  
测试含有泛型的方法  
 \*/  
public class Demo01 {  
 public static void main(String[] args) {  
 // 创建一个Demo02Main对象  
 Demo02Main gm = new Demo02Main();  
  
 /\*  
 调用含有泛型的方法method01  
 传递什么类型，泛型就是什么类型  
 \*/  
 gm.method01(10); // 10  
 gm.method01("abc"); // abc  
 gm.method01(8.8); // 8.8  
 gm.method01(true); // true  
  
 // 静态方法，通过类名.方法名(参数)可以直接使用。  
 Demo02Main.method02("静态方法"); // 静态方法  
 Demo02Main.method02(1); // 1  
 }  
}

1. 含有泛型的接口

定义格式：

修饰词 interface接口名 <代表泛型的变量> { }

例如：

public interface Demo03<E> {  
 public abstract void add(E e);  
   
 public abstract E getE();  
}

9.3 泛型通配符

概念：当前使用泛型类或者接口时，传递的数据中，泛型类型不确定，可以通过通配符<?>表示，但是一旦使用泛型的通配符后，只能使用Object类中的共性方法，集合中元素自身方法无法使用。

通配符基本使用

泛型的通配符：不知道使用什么类型来接收的时候，此时可以使用？，？表示未知通配符。

此时只能接受数据，不能往集合中存储数据。

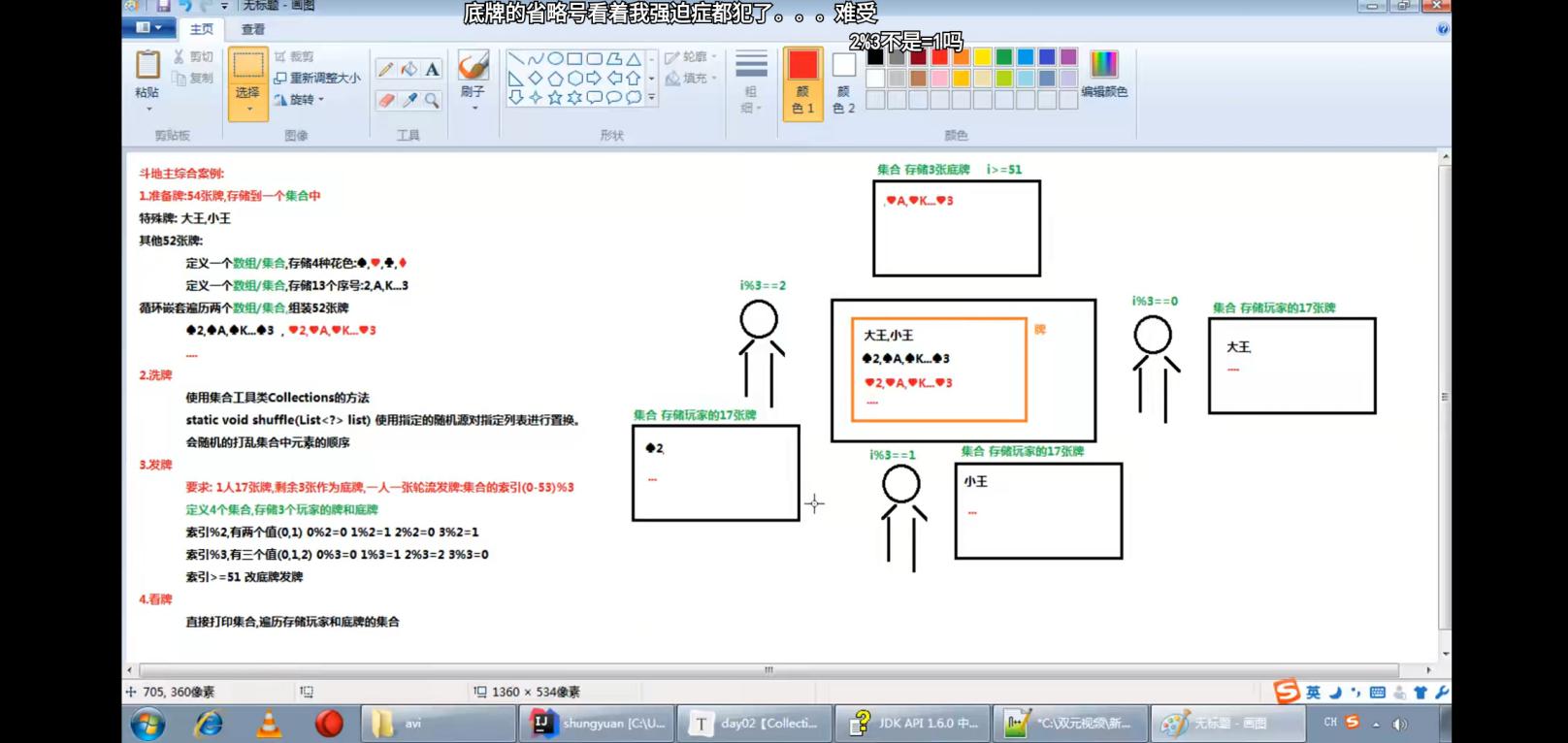
4 例题：集合综合案例

题目：按照斗地主的规则，完成洗牌发牌的动作。

具体规则：

使用5张牌打乱顺序，三个玩家参与游戏，三人交替摸牌，每人17张牌，最后三张留作底牌。

4—1 案例分析



4—2 代码解决

package demo02;  
  
import java.util.ArrayList;  
import java.util.Collections;  
  
/\*  
斗地主案例：  
 1.准备牌  
 2.洗牌  
 3.发牌  
 4.看牌  
 \*/  
public class Demo02Text {  
 public static void main(String[] args) {  
 // 1.准备牌  
 // 定义一个存储54张牌的ArrayList集合，泛型使用String  
 ArrayList<String> poker = new ArrayList<>();  
 // 定义两个数组，一个数组存储牌的花色，一个数组存储牌的序号。  
 String[] colors = {"♠", "♥", "♣", "♦"};  
 String[] numbers = {"2", "A", "K", "Q", "J", "10", "9", "8", "7", "6", "5", "4", "3"};  
 // 先把大王和小王存储到poker集合中  
 poker.add("大王");  
 poker.add("小王");  
 // 循环嵌套遍历两个数组，组装52张牌  
 for (String number:numbers){  
 for (String color : colors) {  
 // System.out.print(color+number); //♠2♥2♣2♦2  
 // 把组装好的牌存储到poker集合中  
 poker.add(color+number);  
 }  
 }  
 System.out.println(poker); //[大王, 小王, ♠2, ♥2,...]  
  
 /\*  
 2.洗牌  
 使用集合的工具类Collections中的方法  
 static void shuffle(List<?> List)  
 使用默认随机源对指定列表进行置换。  
 随机打乱54张牌的顺序。  
 \*/  
 Collections.shuffle(poker); // API中自带的方法  
 // System.out.println(poker); // 打乱顺序后的poker数组  
  
 /\*  
 3.发牌  
 \*/  
 // 定义4个集合，存储玩家的牌和三张底牌  
 ArrayList<String> player01 = new ArrayList<>();  
 ArrayList<String> player02 = new ArrayList<>();  
 ArrayList<String> player03 = new ArrayList<>();  
 ArrayList<String> dipai = new ArrayList<>();  
  
 /\*  
 遍历poker集合，获取每一张牌  
 使用poker集合的索引%3给3个玩家轮流洗牌  
 剩余3张牌给底牌  
 注意：  
 先判断底牌(i>=51),否则牌就发没了  
 \*/  
 for (int i = 0; i < poker.size(); i++) {  
 // 获取每一张牌  
 String p = poker.get(i);  
 //轮流发牌  
 if(i>=51){  
 // 改底牌发牌  
 dipai.add(p);  
 }else if(i%3==0){  
 // 给玩家1发牌  
 player01.add(p);  
 }else if(i%3==1){  
 // 给玩家2发牌  
 player02.add(p);  
 }else if(i%3==2){  
 // 给玩家3发牌  
 player03.add(p);  
 }  
 }  
 // 4.看牌  
 System.out.println("刘德华"+player01);  
 System.out.println("周润发"+player02);  
 System.out.println("周星驰"+player03);  
 System.out.println("底牌"+dipai);  
 }  
}