中国矿业大学计算机学院

2019 级本科生课程报告

课程名称 Java 语言与网络编程

学生姓名 李 春 阳

学　　号 10193657

班　　级 信息安全 2019-1 班

任课教师 张 爱 娟

报告时间 2021.12.22

目录

1. 问题一

1.1 问题描述

输入三个字符串，分别
1 必须满足密码复杂性要求（认证需求）
2 满足身份证号码规范（15/18）
3 满足电子邮件规范

1.2 问题分析

1.2.1 正则表达式语法

在其他语言中,\\ 表示:**我想要在正则表达式中插入一个普通的(字面上的)反斜杠，请不要给它任何特殊的意义。**

在 Java 中，\\ 表示：**我要插入一个正则表达式的反斜线，所以其后的字符具有特殊的意义。**

所以，在其他的语言中（如 Perl），一个反斜杠 \ 就足以具有转义的作用，而在 Java 中正则表达式中则需要有两个反斜杠才能被解析为其他语言中的转义作用。也可以简单的理解在 Java 的正则表达式中，两个 \\ 代表其他语言中的一个 \，这也就是为什么表示一位数字的正则表达式是 **\\d**，而表示一个普通的反斜杠是 \\。

```
System.out.print("\\");      // 输出为 \
System.out.print("\\\\");    // 输出为 \\
```

字符	说明
\	将下一字符标记为特殊字符、文本、反向引用或八进制转义符。例如，**n** 匹配字符 **n**。**\n** 匹配换行符。序列 \\\\ 匹配 \\ ，**(** 匹配 **(**。
^	匹配输入字符串开始的位置。如果设置了 **RegExp** 对象的 **Multiline** 属性，^ 还会与"\n"或"\r"之后的位置匹配。
$	匹配输入字符串结尾的位置。如果设置了 **RegExp** 对象的 **Multiline** 属性，$ 还会与"\n"或"\r"之前的位置匹配。
*	零次或多次匹配前面的字符或子表达式。例如，zo* 匹配"z"和"zoo"。* 等效于 {0,}。
+	一次或多次匹配前面的字符或子表达式。例如，"zo+"与"zo"和"zoo"匹配，但与"z"不匹配。+ 等效于 {1,}。
?	零次或一次匹配前面的字符或子表达式。例如，"do(es)?"匹配"do"或"does"中的"do"。? 等效于 {0,1}。
{*n*}	*n* 是非负整数。正好匹配 *n* 次。例如，"o{2}"与"Bob"中的"o"不匹配，但与"food"中的两个"o"匹配。
{*n*,}	*n* 是非负整数。至少匹配 *n* 次。例如，"o{2,}"不匹配"Bob"中的"o"，而匹配"foooood"中的所有 o。"o{1,}"等效于"o+"。"o{0,}"等效于

	"o*"。
{*n*,*m*}	*m* 和 *n* 是非负整数，其中 *n* <= *m*。匹配至少 *n* 次，至多 *m* 次。例如，"o{1,3}"匹配"fooooood"中的头三个 o。'o{0,1}' 等效于 'o?'。注意：您不能将空格插入逗号和数字之间。
?	当此字符紧随任何其他限定符（*、+、?、{*n*}、{*n*,}、{*n*,*m*}）之后时，匹配模式是"非贪心的"。"非贪心的"模式匹配搜索到的、尽可能短的字符串，而默认的"贪心的"模式匹配搜索到的、尽可能长的字符串。例如，在字符串"oooo"中，"o+?"只匹配单个"o"，而"o+"匹配所有"o"。
.	匹配除"\r\n"之外的任何单个字符。若要匹配包括"\r\n"在内的任意字符，请使用诸如"[\s\S]"之类的模式。
(*pattern*)	匹配 *pattern* 并捕获该匹配的子表达式。可以使用 **$0…$9** 属性从结果"匹配"集合中检索捕获的匹配。若要匹配括号字符 ()，请使用"\("或者"\)"。
(?:*pattern*)	匹配 *pattern* 但不捕获该匹配的子表达式，即它是一个非捕获匹配，不存储供以后使用的匹配。这对于用"or"字符 (|) 组合模式部件的情况很有用。例如，'industr(?:y|ies) 是比 'industry|industries' 更经济的表达式。
(?=*pattern*)	执行正向预测先行搜索的子表达式，该表达式匹配处于匹配 *pattern* 的字符串的起始点的字符串。它是一个非捕获匹配，即不能捕获供以后使用的匹配。例如，'Windows (?=95|98|NT|2000)' 匹配"Windows 2000"中的"Windows"，但不匹配"Windows 3.1"中的"Windows"。预测先行不占用字符，即发生匹配后，下一匹配的搜索紧随上一匹配之后，而不是在组成预测先行的字符后。
(?!*pattern*)	执行反向预测先行搜索的子表达式，该表达式匹配不处于匹配 *pattern* 的字符串的起始点的搜索字符串。它是一个非捕获匹配，即不能捕获供以后使用的匹配。例如，'Windows (?!95|98|NT|2000)' 匹配"Windows 3.1"中的 "Windows"，但不匹配"Windows 2000"中的"Windows"。预测先行不占用字符，即发生匹配后，下一匹配的搜索紧随上一匹配之后，而不是在组成预测先行的字符后。
x|*y*	匹配 *x* 或 *y*。例如，'z|food' 匹配"z"或"food"。'(z|f)ood' 匹配"zood"或"food"。
[*xyz*]	字符集。匹配包含的任一字符。例如，"[abc]"匹配"plain"中的"a"。
[^*xyz*]	反向字符集。匹配未包含的任何字符。例如，"[^abc]"匹配"plain"中"p"，"l"，"i"，"n"。
[*a-z*]	字符范围。匹配指定范围内的任何字符。例如，"[a-z]"匹配"a"到"z"范围内的任何小写字母。
[^*a-z*]	反向范围字符。匹配不在指定的范围内的任何字符。例如，"[^a-

	z]"匹配任何不在"a"到"z"范围内的任何字符。
\b	匹配一个字边界，即字与空格间的位置。例如，"er\b"匹配"never"中的"er"，但不匹配"verb"中的"er"。
\B	非字边界匹配。"er\B"匹配"verb"中的"er"，但不匹配"never"中的"er"。
\c*x*	匹配 *x* 指示的控制字符。例如，\cM 匹配 Control-M 或回车符。*x* 的值必须在 A-Z 或 a-z 之间。如果不是这样，则假定 c 就是"c"字符本身。
\d	数字字符匹配。等效于 [0-9]。
\D	非数字字符匹配。等效于 [^0-9]。
\f	换页符匹配。等效于 \x0c 和 \cL。
\n	换行符匹配。等效于 \x0a 和 \cJ。
\r	匹配一个回车符。等效于 \x0d 和 \cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等。与 [\f\n\r\t\v] 等效。
\S	匹配任何非空白字符。与 [^ \f\n\r\t\v] 等效。
\t	制表符匹配。与 \x09 和 \cI 等效。
\v	垂直制表符匹配。与 \x0b 和 \cK 等效。
\w	匹配任何字类字符，包括下划线。与"[A-Za-z0-9_]"等效。
\W	与任何非单词字符匹配。与"[^A-Za-z0-9_]"等效。
\x*n*	匹配 *n*，此处的 *n* 是一个十六进制转义码。十六进制转义码必须正好是两位数长。例如，"\x41"匹配"A"。"\x041"与"\x04"&"1"等效。允许在正则表达式中使用 ASCII 代码。
num	匹配 *num*，此处的 *num* 是一个正整数。到捕获匹配的反向引用。例如，"(.)\1"匹配两个连续的相同字符。
n	标识一个八进制转义码或反向引用。如果 *n* 前面至少有 *n* 个捕获子表达式，那么 *n* 是反向引用。否则，如果 *n* 是八进制数 (0-7)，那么 *n* 是八进制转义码。
nm	标识一个八进制转义码或反向引用。如果 *nm* 前面至少有 *nm* 个捕获子表达式，那么 *nm* 是反向引用。如果 *nm* 前面至少有 *n* 个捕获，则 *n* 是反向引用，后面跟有字符 *m*。如果两种前面的情况都不存在，则 *nm* 匹配八进制值 *nm*，其中 *n* 和 *m* 是八进制数字 (0-7)。
\nml	当 *n* 是八进制数 (0-3)，*m* 和 *l* 是八进制数 (0-7) 时，匹配八进制转义码 *nml*。
\u*n*	匹配 *n*，其中 *n* 是以四位十六进制数表示的 Unicode 字符。例如，\u00A9 匹配版权符号 (©)。

1.2.2 密码复杂性正则

密码包含大写字母、小写字母、数字、特殊符号中的至少三类，且长度在 8 到 20 之间。

```
^(?!([A-Z]*|[a-z]*|[0-9]*|[!-/:-@\[-`{-~]*|[A-Za-z]*|[A-Z0-9]*|[A-Z!-/:-@\[-`{-~]*|[a-z0-9]*|[a-z!-/:-@\[-`{-~]*|[0-9!-/:-@\[-`{-~]*)$)[A-Za-z0-9!-/:-@\[-`{-~]{8,20}$
```

思路：用零宽度前向负断言来排除只包含一类和只包含两类字符的简单密码

分解：

^：匹配行首

(?! ...)：表示零宽度前向负断言(zero-width negative lookahead)，断言紧接着后面的内容一定不能出现。

中间主体：表示字符串全部由任意一类字符组成，或者全部由任意两类字符组成。这里列举了所有的组合情况，一类的四种组合和两类的六种组合。和前面的负断言结合，含义发生反转，即表示整个字符串不能仅由一类字符组成，也不能仅由两类字符组成。

[A-Za-z0-9!-/:-@\[-`{-~]{8,20}：匹配整个字符串，限制只能出现这四类字符，且长度在 8 到 20 之间

$：匹配行尾

1.2.3 身份证号码正则

假设 18 位身份证号码:41000119910101123X　410001 19910101 123X

```
(^[1-9]\\d{5}(18|19|20)\\d{2}((0[1-9])|(10|11|12))(([0-2][1-9])|10|20|30|31)\\d{3}[0-9Xx]$)|(^[1-9]\\d{5}\\d{2}((0[1-9])|(10|11|12))(([0-2][1-9])|10|20|30|31)\\d{3}$)
```

^开头

[1-9] 第一位 1-9 中的一个　　4

\\d{5} 五位数字　　10001（前六位省市县地区）

(18|19|20)　　19（现阶段可能取值范围 18xx-20xx 年）

\\d{2}　　91（年份）

((0[1-9])|(10|11|12))　　01（月份）

(([0-2][1-9])|10|20|30|31)01（日期）

\\d{3} 三位数字　　123（第十七位奇数代表男，偶数代表女）

[0-9Xx] 0123456789Xx 其中的一个 X（第十八位为校验值）

1.2.4 电子邮件正则

只判断@和.

```
\w+@\w+[.]\w+
```

1.3 代码展示

```
1.    import java.util.Scanner;
2.
3.    public class RegularExercise {
4.        private static final String Password = "^(?!([A-Z]*|[a-
z]*|[0-9]*|[!-/:-@\\[-`{-~]*|[A-Za-z]*|[A-Z0-9]*|[A-Z!-/:-@\\[-`{-
~]*|[a-z0-9]*|[a-z!-/:-@\\[-`{-~]*|[0-9!-/:-@\\[-`{-~]*)$)[A-Za-z0-
9!-/:-@\\[-`{-~]{8,20}$";
5.        private static final String identification_number = "(^[1-
9]\\d{5}(18|19|20)\\d{2}((0[1-9])|(10|11|12))(([0-2][1-
9])|10|20|30|31)\\d{3}[0-9Xx]$)|" +
6.                "(^[1-9]\\d{5}\\d{2}((0[1-9])|(10|11|12))(([0-2][1-
9])|10|20|30|31)\\d{3}$)";
7.        private static final String Email = "\\w+@\\w+[.]\\w+";
8.        boolean is_Complex_Password(String text){
9.            return text.matches(Password);
10.       }
11.
12.       boolean is_Complex_identification_number(String text){
13.           return text.matches(identification_number);
14.       }
15.
16.       boolean is_Complex_Email(String text){
17.           return text.matches(Email);
18.       }
19.
20.       public static void main(String[] args) {
21.           Scanner scan = new Scanner(System.in);
22.           RegularExercise R = new RegularExercise();
23.           String text = "Li010209@";
24.           System.out.println(R.is_Complex_Password(text));
25.           text = "142225200102095037";
26.           System.out.println(R.is_Complex_identification_number(t
ext));
27.           text = "lichunyang@163.com";
28.           System.out.println(R.is_Complex_Email(text));
29.           while (true) {
30.               System.out.println("**************************");
31.               System.out.println("请选择判断类型：");
32.               System.out.println("[0]退出");
33.               System.out.println("[1]判断密码是否复杂");
34.               System.out.println("[2]判断身份证号码规范");
35.               System.out.println("[3]判断电子邮件规范");
36.               int op = scan.nextInt();
37.               String str = "";
38.               switch (op) {
39.                   case 0 :
40.                       System.out.println("感谢使用");
41.                       System.exit(0);
42.
43.                   case 1 :
44.                       System.out.println("请输入密码：");
45.                       str = scan.next();
```

```
46.                         System.out.println(R.is_Complex_Password(st
r));
47.                         break;
48.
49.                     case 2 :
50.                         System.out.println("请输入身份证：");
51.                         str = scan.next();
52.                         System.out.println(R.is_Complex_identificat
ion_number(str));
53.                         break;
54.
55.                     case 3 :
56.                         System.out.println("请输入电子邮件：");
57.                         str = scan.next();
58.                         System.out.println(R.is_Complex_Email(str))
;
59.                         break;
60.
61.
62.                 }
63.             }
64.         }
65.
66.     }
```

1.4 运行结果

```
"C:\Program Files\java\jdk-13.0.2\bin\java.exe" "-javaagent:D:\Program Files\JetBrains\IntelliJ IDEA 2020.1\lib\idea_rt.jar=13421:D:\Program Files\JetBrains\IntelliJ IDEA 2020.1\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\lenovo\Desktop\JAVA\作业 4\code\out\production\code RegularExercise
true
true
true
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
1
请输入密码:
psadasd
false
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
```

```
[2]判断身份证号码规范
[3]判断电子邮件规范
1
请输入密码:
pasdas123213
false
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
1
请输入密码:
pasdas@sad11
true
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
2
请输入身份证:
1231244325352
false
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
2
请输入身份证:
142225200102095037
true
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
3
```

```
请输入电子邮件:
asd@asdkjd
false
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
3
请输入电子邮件:
2132142@qq.com
true
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
0
感谢使用

进程已结束，退出代码 0
```

```
"C:\Program Files\java\jdk-13.0.2\bin\java.exe" "-javaagent:D:\Program Files\JetBrains\Inte
true
true
true
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
1
请输入密码:
psadasd
false
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
1
请输入密码:
pasdas123213
false
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
1
请输入密码:
pasdas@sad11
true
**************************
```

```
[2]判断身份证号码规范
[3]判断电子邮件规范
2
请输入身份证:
142225200102095037
true
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
3
请输入电子邮件:
asd@asdkjd
false
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
3
请输入电子邮件:
2132142@qq.com
true
**************************
请选择判断类型:
[0]退出
[1]判断密码是否复杂
[2]判断身份证号码规范
[3]判断电子邮件规范
0
感谢使用
```

2. 问题二

2.1 问题描述

要求:

利用鼠标事件启动 3 个线程分别在三个窗口中同时绘制动态图形（图形自选）

2.2 问题分析

2.2.1 java 多线程

java 给多线程编程提供了内置的支持。 一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。

多线程是多任务的一种特别的形式，但多线程使用了更小的资源开销。

这里定义和线程相关的另一个术语 - 进程：一个进程包括由操作系统分配

的内存空间，包含一个或多个线程。一个线程不能独立的存在，它必须是进程的一部分。一个进程一直运行，直到所有的非守护线程都结束运行后才能结束。

多线程能满足程序员编写高效率的程序来达到充分利用 CPU 的目的。

2.2.2 一个线程的生命周期

新建状态:

使用 new 关键字和 Thread 类或其子类建立一个线程对象后，该线程对象就处于新建状态。它保持这个状态直到程序 start() 这个线程。

就绪状态:

当线程对象调用了 start()方法之后，该线程就进入就绪状态。就绪状态的线程处于就绪队列中，要等待 JVM 里线程调度器的调度。

运行状态:

如果就绪状态的线程获取 CPU 资源，就可以执行 run()，此时线程便处于运行状态。处于运行状态的线程最为复杂，它可以变为阻塞状态、就绪状态和死亡状态。

阻塞状态:

如果一个线程执行了 sleep（睡眠）、suspend（挂起）等方法，失去所占用资源之后，该线程就从运行状态进入阻塞状态。在睡眠时间已到或获得设备资源后可以重新进入就绪状态。可以分为三种:

等待阻塞：运行状态中的线程执行 wait() 方法，使线程进入到等待阻塞状态。

同步阻塞：线程在获取 synchronized 同步锁失败(因为同步锁被其他线程占用)。

其他阻塞: 通过调用线程的 sleep() 或 join() 发出了 I/O 请求时，线程就会进入到阻塞状态。当 sleep() 状态超时，join() 等待线程终止或超时，或者 I/O 处理完毕，线程重新转入就绪状态。

死亡状态:

一个运行状态的线程完成任务或者其他终止条件发生时，该线程就切换到终止状态。

2.2.3 创建一个线程

Java 提供了三种创建线程的方法：

通过实现 Runnable 接口；

通过继承 Thread 类本身；

通过 Callable 和 Future 创建线程。

通过实现 Runnable 接口来创建线程

创建一个线程，最简单的方法是创建一个实现 Runnable 接口的类。

为了实现 Runnable，一个类只需要执行一个方法调用 run()，声明如下：

```
public void run()
```

你可以重写该方法,重要的是理解的 run() 可以调用其他方法,使用其他类,并声明变量,就像主线程一样。

在创建一个实现 Runnable 接口的类之后,你可以在类中实例化一个线程对象。

Thread 定义了几个构造方法,下面的这个是我们经常使用的:

Thread(Runnable threadOb,String threadName);

这里,threadOb 是一个实现 Runnable 接口的类的实例,并且 threadName 指定新线程的名字。

新线程创建之后,你调用它的 start() 方法它才会运行。

void start();

2.3 代码展示

MultiProgress

```
1.    package MultiProgress;
2.
3.    public class MultiProgress {
4.        public static void main(String[] args) {
5.            myFrame[] myThread = new myFrame[3];
6.            System.out.println(myThread.length);
7.            myThread[0] = new huitu1();
8.            myThread[1] = new huitu2();
9.            myThread[2] = new huitu3();
10.           for (int i = 0; i < myThread.length; i++) {
11.               myThread[i].setTitle("第 " + (i + 1) + " 个图形");
12.               myThread[i].setBounds((1 + i) * 300, 300, 300, 300)
;
13.               myThread[i].setVisible(true);
14.           }
15.       }
16.   }
```

myFrame

```
1.    package MultiProgress;
2.
3.    import javax.swing.*;
4.    import java.awt.*;
5.    import java.awt.event.MouseAdapter;
6.    import java.awt.event.MouseEvent;
7.
8.    public class myFrame extends JFrame implements Runnable {
9.        int i = 0;
10.       Thread t;
11.       public myFrame() {
12.           this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13.           this.addMouseListener(new MouseAdapter() {
14.               @Override
15.               public void mouseClicked(MouseEvent e) {
16.                   super.mouseClicked(e);
```

```
                    startT();
                }
            });
        }

        public void startT() {
            t = new Thread(this);
            t.start();
        }

        @Override
        public void run() {
            for (i = 0; i < 20; i++) {
                try {
                    repaint();
                    Thread.sleep(200);
                } catch (Exception e) {
                    System.out.println(e.toString());
                }
            }
        }

    }
```

huitu1

```
    package MultiProgress;

    import java.awt.*;

    public class huitu1 extends myFrame {

        public void paint(Graphics g) {
            super.paint(g);
            if (i != 0)
                g.setColor(Color.RED);
    //          g.drawOval(50 + i * 10, 50 + i * 10, 30, 30);
    //          super.paint(g);
    //          g.drawOval(50, 10, 30, 30);
    //          g.drawRect(100, 100, 60, 60);
            g.drawString("love",50 + i * 10,50 + i * 10);
        }
    }
```

huitu2

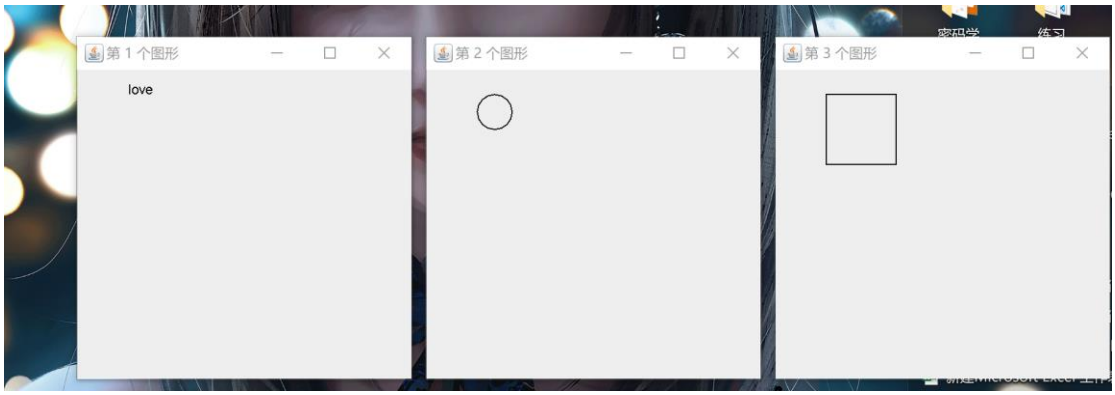
```
    package MultiProgress;

    import java.awt.*;

    public class huitu2 extends myFrame {

        public void paint(Graphics g) {
            super.paint(g);
```

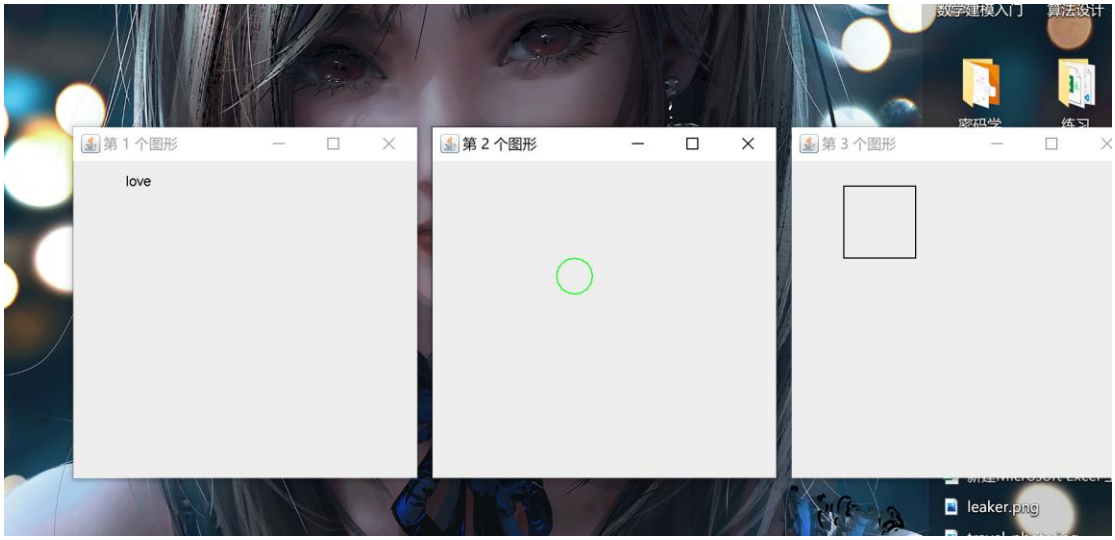
```
            if (i != 0)
                g.setColor(Color.GREEN);
            g.drawOval(50 + i * 10, 50 + i * 10, 30, 30);
    //          super.paint(g);
    //          g.drawOval(50, 10, 30, 30);
    //          g.drawRect(100, 100, 60, 60);
    //          g.drawString("",50 + i * 10,50 + i * 10);
        }
    }
```

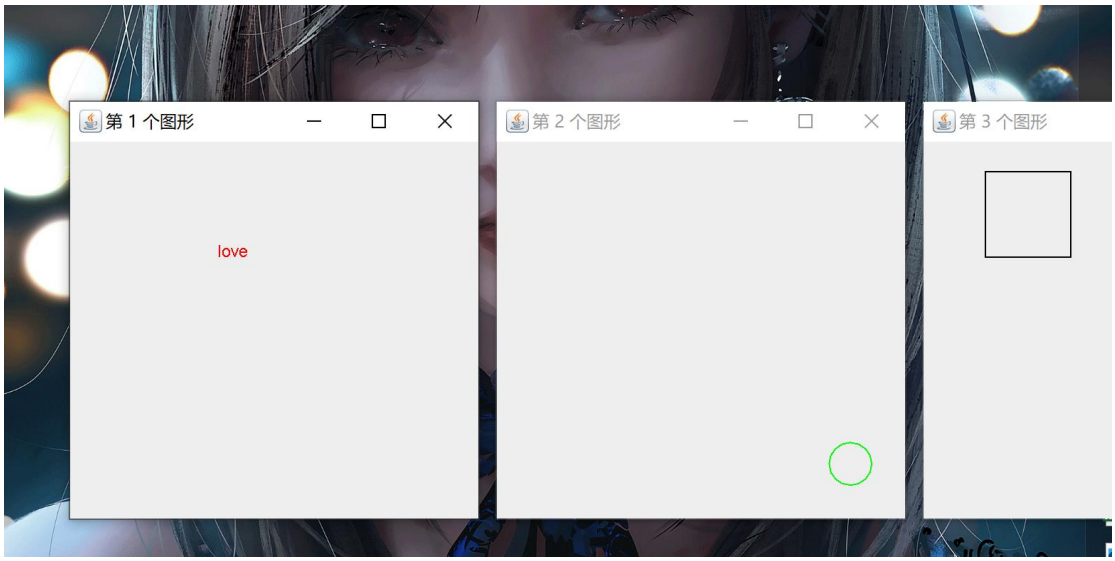
huitu3

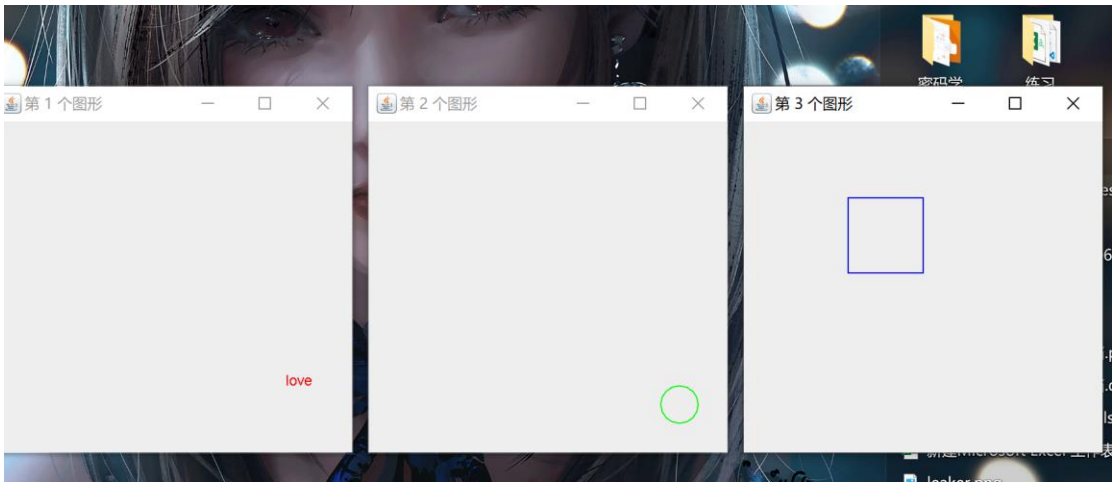
```
package MultiProgress;

import java.awt.*;

public class huitu3 extends myFrame {

    public void paint(Graphics g) {
        super.paint(g);
        if (i != 0)
            g.setColor(Color.BLUE);
    //          g.drawOval(50 + i * 10, 50 + i * 10, 30, 30);
    //          super.paint(g);
    //          g.drawOval(50, 10, 30, 30);
        g.drawRect(50 + i * 10, 50 + i * 10,60, 60);
    //          g.drawString("",50 + i * 10,50 + i * 10);
    }
}
```

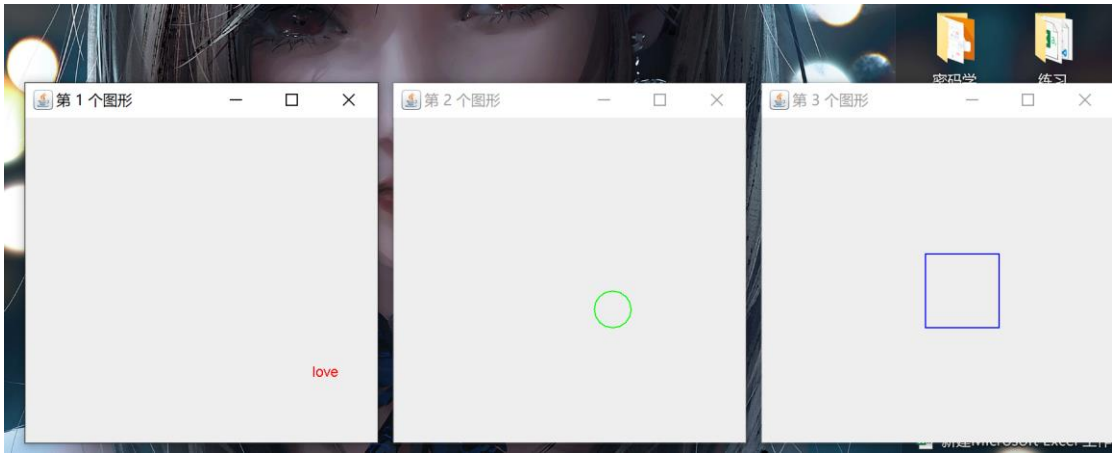
2.4 运行结果




第 1 个图形
love
第 2 个图形
第 3 个图形
leaker.png


第 1 个图形
love
第 2 个图形
第 3 个图形


第 1 个图形
love
第 2 个图形
第 3 个图形


第 1 个图形
love
第 2 个图形
第 3 个图形