3.1 问题1解析

```
tring  str1  =  new  String("1");
```

解析：首先此行代码创建了两个对象，在执行前会在常量池中创建一个"1"的对象，然后执行该行代码时new一个"1"的对象存放在堆区中；然后str1指向堆区中的对象；

```
    str1.intern();
```

解析：该行代码首先查看"1"字符串有没有存在在常量池中，此时存在则直接返回该常量，这里返回后没有引用接受他，【假如不存在的话在 jdk1.6中会在常量池中建立该常量，在jdk1.7以后会把堆中该对象的引用放在常量池中】

```
    String  str2  =  "1";
```

解析：此时"1"已经存在在常量池中，str2指向常量池中的对象；

```
    System.out.println(str1  ==  str2);    //结果是  false  or  true?
```

解析：str1指向堆区的对象，str2指向常量池中的对象，两个引用指向的地址不同，输入false；

```
    String  str3  =  new  String("2")  +  new  String("2");
```

解析：此行代码执行的底层执行过程是 首先使用StringBuffer的append方法将"2"和"2"拼接在一块，然后调用toString方法new出“22”；所以此时的“22”字符串是创建在堆区的；

```
    t3.intern();
```

解析：此行代码执行时字符串常量池中没有"22",所以此时在jdk1.6中会在字符串常量池中创建"22",而在jdk1.7以后会把堆中该对象的引用放在常量池中；

```
  String str4 = "22";
```

解析：此时的str4在jdk1.6中会指向方法区，而在jdk1,7中会指向堆区；

```
  System.out.println(str3 == str4); //结果是 false or true?
```

解析：很明显了 jdk1.6中为false 在jdk1.7中为true；

3.2 问题2解析

String str1 = "aaa";
解析：str1指向方法区；

String str2 = "bbb";
解析： str2 指向方法区

String str3 = "aaabbb";
解析：str3指向方法区

String str4 = str1 + str2;
解析：此行代码上边已经说过原理。str4指向堆区

String str5 = "aaa" + "bbb";
解析：该行代码重点说明一下，jvm对其有优化处理，也就是在编译阶段就会将这两个字符串常量进行拼接，也就是"aaabbb";所以他是在方法区中的；’

System.out.println(str3 == str4); // false or true
解析：很明显 为false， 一个指向堆 一个指向方法区

System.out.println(str3 == str4.intern()); // true or false
解析：jdk1.6中str4.intern会把“aaabbb”放在方法区，1.7后在堆区，所以在1.6中会是true 但是在1.7中是false

System.out.println(str3 == str5);// true or false
解析：都指向字符串常量区，字符串长常量区在方法区，相同的字符串只存在一份，其实这个地方在扩展一下，因为方法区的字符串常量是共享的，在两个线程同时共享这个字符串时，如果一个线程改变他会是怎么样的呢，其实这种场景下是线程安全的，jvm会将改变后的字符串常量在

字符串常量池中重新创建一个处理，可以保证线程安全

3.3 问题3解析

tring t1 = new String("2");
解析：创建了两个对象，t1指向堆区
String t2 = "2";

解析：t2指向字符串常量池

t1.intern();

解析:字符串常量池已经存在该字符串，直接返回；

System.out.println(t1 == t2); //false or true

解析：很明显 false

String t3 = new String("2") + new String("2");

解析：过程同问题1 t3指向堆区

String t4 = "22";

解析：t4 在1.6 和 1.7中指向不同

t3.intern();

解析： 字符串常量池中已经存在该字符串 直接返回

System.out.println(t3 == t4); //false or true

解析： 很明显为 false 指向不同的内存区

3.4 问题4解析

这个地方存在一个知识点。可能是个盲区，这次要彻底记住“

(1). 内存中有一个java基本类型封装类的常量池。这些类包括Byte, Short, Integer, Long, Character, Boolean。需要注意的是，Float和Double这两个类并没有对应的常量池。

(2).上面5种整型的包装类的对象是存在范围限定的；范围在-128~127存在在常量池，范围以外则在堆区进行分配。

(3). 在周志明的那本虚拟机中有这样一句话：包装类的“==”运行符在不遇到算术运算的情况下不会自动拆箱，以及他们的equals()方法不处理数据类型的关系，通俗的讲也就是 “==”两边如果有算术运算， 那么自动拆箱和进行数据类型转换处理，比较的是数值等不等能。

(4).Long的equals方法会先判断是否是Long类型。

(5).无论是Integer还是Long，他们的equals方法比较的是数值。

System.out.println(c == d)。

解析：由于常量池的作用，c与d指向的是同一个对象(注意此时的==比较的是对象，也就是地址，而不是数值)。因此为true

System.out.println(e == f)。
由于321超过了127，因此常量池失去了作用，所以e和f数值虽然相同，但不是同一个对象，以此为false。

System.out.println(c == (a+b))。
此时==两边有算术运算，会进行拆箱，因此此时比较的是数值，而并非对象。因此为true。

System.out.println(c.equals(a+b))
c与a+b的数值相等，为true。

System.out.pirnln(g == (a + b))
由于==两边有算术运算，所以比较的是数值，因此为true。

System.out.println(g.equals(a+b))。
Long类型的equal在比较是时候，会先判断a+b是否为Long类型，显然a+b不是，因此false