Thinking in Java chapter4 笔记和习题

emacsun

目录

本章学习起来毫无压力，因为大多和 C/C++ 一脉相承。只记录几个和 C/C++ 不一样的点，以备复习时多加注意。

1 Java赋值

第四章给出带有一个整数成员的类的赋值，

```
1  //assignment with objects is a bit tricky
2  import static net.mindview.util.Print.*;
3
4  class Tank{
5      int level;
6  }
7
8  public class assignment{
9      public static void main(String[] args) {
10         Tank t1 = new Tank();
11         Tank t2 = new Tank();
12         t1.level = 47;
13         t1.level = 27;
14         print("1:t1.level:␣" + t1.level +
15               ",␣t2.level:␣" + t2.level);
16         t1 = t2;
17         print("2:t1.level:␣" + t1.level +
18               ",␣t2.level:" + t2.level);
19         t1.level = 33;
20         print("3:t1.level:␣" + t1.level +
21               ",␣t2.level:" + t2.level);
22         t2.level = 99;
23         print("4:t1.level:␣" + t1.level +
24               ",␣t2.level:" + t2.level);
25     }
26 }
```

在第16行执行结束之后， t1 和 t2 变成同一个对象，原先 t2 指向的对象会被garbage collector收走。如果不想 t2 被收走，则需要对对象的每个域进行赋值操作：

```
t1.level = t2.level;
```

接下来给出对应的 float 类型，也就是第四章练习2.

```
1  //assignment with objects is a bit tricky
2  import static net.mindview.util.Print.*;
3
4  class Tank{
5      float level;
6  }
7
8  public class assignment{
9      public static void main(String[] args) {
10         Tank t1 = new Tank();
11         Tank t2 = new Tank();
12         t1.level = 47.47f;
13         t1.level = 27.27f;
14         print("1:t1.level:␣" + t1.level +
15               ",␣t2.level:␣" + t2.level);
16         t1 = t2;
17         print("2:t1.level:␣" + t1.level +
18               ",␣t2.level:" + t2.level);
19         t1.level = 33.33f;
20         print("3:t1.level:␣" + t1.level +
21               ",␣t2.level:" + t2.level);
22         t2.level = 99.99f;
23         print("4:t1.level:␣" + t1.level +
24               ",␣t2.level:" + t2.level);
25     }
26 }
```

注意在赋值程序对应的 float 版本中，浮点数的赋值 t1.level = 47.47f 这个 47.47f 如果改成 47.47 在我的编译器 JSE8 上会出错。一定是有一个开关，可以允许这样的赋值，这个开关就是 cast 即类型转换。

Java 是强类型语言，默认的 int 变量初始化值是 0 ; long 变量初始化值是 0L ; float 变量初始化值是 0.0f ; double 初始化值是 0.0d.

当向一个函数传递对象的时候，也要注意传递的是引用，如下：

```
1  //assignment with objects is a bit tricky
2  import static net.mindview.util.Print.*;
3
4  class Letter{
5      char c;
6      float ff;
7  }
8
9  public class PassObject{
10     static void f(Letter y){
11         y.c = 'z';
12         y.ff = 12.3445f;
13     }
14     public static void main(String[] args) {
15         Letter x = new Letter();
16         x.c = 'a';
17         x.ff = 234.45f;
18         print("1:x.c␣" + x.c +
19               "x.ff␣" + x.ff);
20         f(x);
21         print("2:x.c␣" + x.c +
22               "x.ff␣" + x.ff);
23     }
24 }
```

输出是:

```
1:x.c ax.ff 234.45
2:x.c zx.ff 12.3445
```

可以看到当把 x 传递给 f() 时，传递给 f() 的是引用，在 f() 中执行的操作直接影响到 x 的值，这与 C/C++ 中不同。

2 操作符

```
1  import java.util.*;
2  import static net.mindview.util.Print.*;
3
4  public class MathOps
5  {
6      public static void main(String args[])
7      {
8          //create a seeded random number generator
9          Random rand = new Random(47);
10         int i,j,k;
11         j = rand.nextInt(100) +1;
12         k = rand.nextInt(100) +1;
13         i = j + k;
14         print("j␣(␣" + j +
15               "␣)␣+␣k␣(␣" + k +
16               "␣)␣=␣␣" + i);
17     }
18 }
```

在这段程序中，为了生成一个随机数，调用了 Random 这个 Class 。当没有指定随机数种子时，随机数种子是当前时间。

除了 nextInt() 这个方法外，还有其他方法，比如 nextfloat(), nextdouble() 详见这个类的方法总结。

```
1  import java.util.*;
2  import static net.mindview.util.Print.*;
3
4  public class MathOps
5  {
6      public static void main(String args[])
7      {
8          //create a seeded random number generator
9          Random rand = new Random(47);
10         int i,j,k;
11         j = rand.nextInt(100) +1;
12         k = rand.nextInt(100) +1;
13         i = j + k;
14         print("j␣(␣" + j +
15               "␣)␣+␣k␣(␣" + k +
16               "␣)␣=␣␣" + i);
17         float i,j,k;
18         j = rand.nextFloat(100) +1;
19         k = rand.nextFloat(100) +1;
20         i = j + k;
21         print("j␣(␣" + j +
22               "␣)␣+␣k␣(␣" + k +
23               "␣)␣=␣␣" + i);
24
25         double i,j,k;
26         j = rand.nextDouble(100) +1;
27         k = rand.nextDouble(100) +1;
28         i = j + k;
29         print("j␣(␣" + j +
30               "␣)␣+␣k␣(␣" + k +
31               "␣)␣=␣␣" + i);
32     }
```

```
33 }
```

上述代码会报错：错误信息是， `i,j,k` 已经定义了，这在 `C/C++` 中是允许的,但是在Java中却是不允许的。

看如下代码：

```
1  import java.util.*;
2  import static net.mindview.util.Print.*;
3  public class ExerciseCh0404
4  {
5      public static void main(String args[])
6      {
7          int distance=100;
8          int time=300;
9          float velocity;
10         velocity = (float) distance / time;
11         print("distance␣/␣time␣=␣" + velocity);
12     }
13 }
```

如果在第10行不加 `float` 显示的指示一下，结果输出就是:

```
distance / time = 0.00
```

如果加 `float` 指示转换除法结果为的类型就会把计算结果转换为浮点类型。

```
distance / time = 0.333333334
```

3 关系运算符

在Java中一切都是对象。所以针对对象的比较会带来一些意想不到的结果。看如下代码:

```
1  public class Equivalence
2  {
3      public static void main(String args[])
4      {
5          Integer n1 = new Integer(47);
6          Integer n2 = new Integer(47);
7          System.out.println(n1 == n2);
8          System.out.println(n1 != n2);
9      }
10 }
```

输出结果是：

```
false
true
```

确实surprise! 两个内容相同的整数(都是47)，但我们判断它们是否相等时，结果居然是 `false` 。这是因为在Java中== 这个关系运算符比较的不仅仅是内容，还比较 `reference` . 想要比较内容是否相等正确的做法是:使用方法 `equals()`

```
1  public class Equivalence
2  {
3      public static void main(String args[])
4      {
5          Integer n1 = new Integer(47);
6          Integer n2 = new Integer(47);
7          System.out.println(n1.equals(n2));
8      }
9  }
```

输出结果是：

```
true
```

方法 equals() 是 Integer 内置的一个方法，用来比较内容是否相等。同理， Float 类的对象也有 equals() 方法。

```
1 public class Equivalence
2 {
3     public static void main(String args[])
4     {
5         Float n1 = new Float(47.0456f);
6         Float n2 = new Float(47.0456f);
7         System.out.println(n1.equals(n2));
8     }
9 }
```

输出结果是:

```
true
```

这还没完，见如下代码：

```
1  class Value {
2      int i;
3  }
4
5  public class Equivalence
6  {
7      public static void main(String args[])
8      {
9          Value n1 = new Value();
10         Value n2 = new Value();
11         n1.i = n2.i = 100;
12         System.out.println(n1.equals(n2));
13     }
14 }
```

你认为结果是什么？会是 true 么？如果你认为是 true ，那么你就错了。结果是 false 。这是因为 equals() 函数默认比较的是 reference 。如果要 equals() 函数比较自定义对象的内容，那么你得重载 equals() 这个函数。那么，如果我照如下方法写代码呢？

```
1  class Value {
2      int i;
3  }
4
5  public class Equivalence
6  {
7      public static void main(String args[])
8      {
9          Value n1 = new Value();
10         Value n2 = new Value();
11         n1.i = n2.i = 100;
12         System.out.println(n1.i.equals(n2.i));
13     }
14 }
```

编译报错：

```
Equivalence.java:12: error: int cannot be dereferenced
        System.out.println(n1.i.equals(n2.i));
```

4 逻辑运算符

Java语言中对逻辑云算法的操作也不同于 C/C++. 在 C/C++ 中，非零值都是 true ,只有零会被当成 false . 但是在Java中却不是：在Java中，不能把非零值或者零当做 boolean 来处理， boolean 值只有两种 true 或者

false . 看代码：

```
1  import java.util.*;
2  import static net.mindview.util.Print.*;
3
4  public class Bool
5  {
6      public static void main(String args[])
7      {
8          Random rand = new Random(47);
9          int i = rand.nextInt(100);
10         int j = rand.nextInt(100);
11         print("i␣=␣" + i );
12         print("j␣=␣" + j );
13         print("i␣<␣j␣is␣" + (i<j));
14         print("i␣>␣j␣is␣" + (i>j));
15         print("i␣<=␣j␣is␣" + (i<=j));
16         print("i␣>=␣j␣is␣" + (i>=j));
17         print("i␣==␣j␣is␣" + (i==j));
18         print("i␣!=␣j␣is␣" + (i!=j));
19         print("i␣&&␣j␣is␣" + (i&&j));
20     }
21 }
```

这个代码的第19行会报错，因为 && 这样的逻辑运算符只针对 boolean 类型，在 C/C++ 中这样的操作是允许的。但是在 Java 中却不是允许的。所以 Java 是比 C/C++ 更强的有类型语言，在写代码的时候最好把类型转化这样的写清楚，不要指望系统自动去转换，这样的代码也更好懂。

5 Java中的数值表示

前面说过 Java 是一种强类型语言，甚至比 C/C++ 还要强类型。因此在数值表示的时候要格外小心，否则在编译阶段就不会通过。这对于写出健壮的代码是非常有用的。看代码：

```
1  import static net.mindview.util.Print.*;
2
3  public class ExerciseCh0408
4  {
5      public static void main(String args[])
6      {
7          byte i1 = 0X80;
8          print("i1␣=␣" + Integer.toBinaryString(i1));
9          print("i1␣=␣" + Long.toBinaryString(i1));
10     }
11 }
```

上面的代码对一个 byte 类型的变量赋值 0x80 这在边一阶段就会报错：

```
error: incompatible types: possible lossy conversion from int to byte
        byte i1 = 0X80;
```

提醒我，从 int 到 byte 的类型转换会出错。因为 0x80 是 int 范围内的数， byte 表示不了这个数值。 byte 只能表示 -0x80 。

如果把 0x80 改成 -0x80 则会看到输出:

```
i1 = 11111111111111111111111110000000
i1 = 11111111111111111111111111111111111111111111111111111110000000
```

注意方法 Integer.toBinaryString 把 byte 变换成 int 来处理, Long.toBinaryString 把 byte 变换成 long 来处理而 Byte 类却没有方法 toBinaryString() 不知道 Java 是怎么想的。 -0x80 表示的是 -128 这个数是 Byte 能表示的最小的数。在Java中所有的数都是有符号数。

6 指数表示

在脚本语言 MATLAB 以及 C/C++ 中 aEb 表示 $a \times 10^b$。在 Java 中，也有这种指数表示，不过由于 Java 是强类型语言，所以还是稍有不同。看代码：

```
1  public class Exponents
2  {
3      public static void main(String args[])
4      {
5          float expfloat = 1.25e-43f;
6          float expFloat = 1.25E-43f;
7          System.out.println(expfloat);
8          System.out.println(expFloat);
9          double expdouble = 47e47d;
10         double expDouble = 47e47;
11         System.out.println(expDouble);
12     }
13 }
```

从这段代码中可以知道： e 和 E 是通用的。还有一些隐藏的信息： aEb 在 Java 中默认是 Double 类型。所以代码第9行的 47e47d 的 d 是可选的，第10行就去掉了。进而第5行和第6行的 f 是必须的，这里的 1.25e-43f 相当于 (float) 1.25e-43 或者 (float) 1.25e-43D .

7 位操作

虽然没有 C/C++ 中位操作灵活，但是Java中也有对应的位操作，看代码：

```
1  import static net.mindview.util.Print.*;
2  public class ExerciseCh0410
3  {
4      public static void main(String args[])
5      {
6          int a = 0xBBBB;
7          int b = 0x5555;
8          print("a␣=␣" + Integer.toBinaryString(a));
9          print("b␣=␣" + Integer.toBinaryString(b));
10         print("a␣&=␣b␣" + Integer.toBinaryString(a&=b));
11         print("a␣^=␣b␣" + Integer.toBinaryString(a^=b));
12         print("a␣|=␣b␣" + Integer.toBinaryString(a|=b));
13     }
14 }
```

输出为:

```
a = 1011101110111011
b = 101010101010101
a &= b 1000100010001
a ^= b 100010001000100
a |= b 101010101010101
```