Task1.变量和数据类型

1..基本数据类型

- **整型（4种）及其范围**

  `byte`：1字节，范围 -128 ~ 127

  `short`：2字节，范围 -32,768 ~ 32,767

  `int`：4字节，范围 -2,147,483,648 ~ 2,147,483,647

  `long`：8字节，范围 -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
- **字符型（1种）**

  `char`
- **浮点型（2种）**

  `float`

  `double`
- **布尔型（1种）**

  `boolean`

2.整型范围

整型（4种）及其范围

- `byte`：1字节，范围 -128 ~ 127
- `short`：2字节，范围 -32,768 ~ 32,767
- `int`：4字节，范围 -2,147,483,648 ~ 2,147,483,647
- `long`：8字节，范围 -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807

3.显隐式转换

显式转换：用(类型)强制指定进行转换

隐式转换：系统自动转换

```
int a=4;
char c='0';
int b=a+c;
//请回答这个过程涉及到的是隐式类型转换还是显式类型转换，b的值是多少，为什么会是这个值。
```

涉及的是隐式类型转换

b的值是52

char类型的c变量底层储存的是Unicode编码值，'0'编码值是48

所以b = 4 + 48 = 52

4.包装类，引用类型和基本数据类型缓存池

```
Integer x = new Integer(18);
Integer y = new Integer(18);
System.out.println(x == y);

Integer z = Integer.valueOf(18);
Integer k = Integer.valueOf(18);
System.out.println(z == k);

Integer m = Integer.valueOf(300);
Integer p = Integer.valueOf(300);
System.out.println(m == p);
```

输出结果为：

```
false
true
false
```

1.x与y

```
Integer x = new Integer(18);
Integer y = new Integer(18);
System.out.println(x == y);
```

此处`==`比较的是x与y的地址值，虽然x和y都是值为18的Integer包装类，但是两者为两个不同的对象，所属的地址不同

故比较结果为false

2.z与k

```
Integer z = Integer.valueOf(18);
Integer k = Integer.valueOf(18);
System.out.println(z == k);
```

`Integer.valueOf()`方法做了内存优化，当在 [-128, 127] 范围内时，会从缓存池返回已经存在的对象引用

故虽然`z`和`k`为两个不同名称的变量，但是都是指向相同地址

故比较结果为true

3.m与p

```
Integer m = Integer.valueOf(300);
Integer p = Integer.valueOf(300);
System.out.println(m == p);
```

`Integer.valueOf()`方法做了内存优化，当在 [-128, 127] 范围内时，会从缓存池返回已经存在的对象引用

但此时值为300，已经超出了[-128,127]的范围，所以会创建两个相同值但不同地址的对象

故比较结果为false

Task2.运算符

5.

```
int a = 5 ;
int b = 7 ;
int c= (++a) + (b++);
System.out.println( c );
System.out.println(a+" "+b);
```

1. **`++a`**：前置自增，先让 `a` 加 1，再参与运算
2. **`b++`**：后置自增，先用 `b` 的当前值，再执行自增
3. **计算 `c`**：`c = 6 + 7 = 13`。

输出结果:

```
13
6 8
```

6.

`int`和`float`都是以二进制的方式储存的

`int`

- 4字节，首位存储数值的正负，其余31位表示数值
- 负数由补码表示

`float`

- 4字节,首位存储数值的正负，8位指数位(表示范围),23 位尾数位(小数部分，表示精度)

两个正数相加超过了数据类型的最大值，就会出现溢出，会绕回该数据类型的最小值重新开始

Task3.数据结构

见 `package uestc.glimmerJava3`