introduction to Java

Lists

1.关于赋值的小练习--海象之谜：

```
Walrus a = new Walrus(1000, 8.3);
Walrus b;
b = a;
b.weight = 5;
System.out.println(a);
System.out.println(b);
```

最终a和b都会被改成5，因为对象的赋值是将a，b同时指向该内存块，代表同一块内存
与之不同的是：

```
int x = 5;
int y;
y = x;
x = 2;
System.out.println("x is: " + x);
System.out.println("y is: " + y);
```

将x赋值给y之后，改变x的值并不会再改变y的值

2.primitive type

java 语言不提供让你知道变量具体内存地址的方法，与 C 不同，这节省了我们对内存的管理 java 声明一个变量后，并**不赋初值**（primitive type），因此当你想使用你所声明的变量时，必须先赋予其初值。
java 的赋值操作是将 y 的 bit 复制给 x
8种基本类型：**byte, short, int, long, float, double, boolean, char**

3.引用类型

java的存储，比如 int x，x 在内存中的地址是 200-250, 那么 java 会创建一个盒子存储 x 的第一位数据，也就是 200
当我们声明一个引用类型的变量时，java 会自动分配一个 64bit 的盒子，无论你储存任何类型的变量
例如

```
Walrus someWalrus;
someWalrus = new Walrus(1000, 8.3);
```

第一行创建一个 someWalrus 盒子，储存大小64bit
第二行通过 **new** 关键字创建实例，大小为 **double+int** 为 96bit，**new** 返回其第一个数据的地址,存在

someWalrus 里面，这也就解释了为何 64bit 大小的盒子却存下了 96bit 大小的实例,因为它只是存储其地址而非真实的数据值

```
Walrus someWalrus; someWalrus = null;
```

假如我们将 someWalrus 赋值为 null，则 someWalrus 储存64个0

对于此，假如一个box notation（someWalrus)值全为0，则代表null
假如一个box notation的值非0,代表其通过箭头指向一个对象实例

综上所述，当我们执行这段代码时：

```
Walrus a = new Walrus(1000, 8.3);
Walrus b;
b = a;
b.weight = 5;
```

创建盒子a,储存new Walrus(weight1000,size8.3)的地址，创建盒子b,暂未赋值，但非null，b=a，表示将a储存的内存bit信息复制给b，因此a与b同时指向Walrus(weight1000,size8.3)，所以改变b.weight也会改变a,这就是引用类型和primitive type的区别。

4.参数传递

```
public static double average(double a,double b)
{
return (a+b)/2;
}
```

在main函数里面:

```
double x=5.5;
double y=10.5
double avg=average(x,y);
```

在传递参数时同样是遵循Golden Rule of Equal，即average函数创建两个新的盒子x,y，这两个x,y 有他自己的作用域，在传递参数时只是将main里面的x,y复制到average里面的x,y。

5.一个加深理解的练习：

```
    public static void main(String[] args) {
           Walrus walrus = new Walrus(3500, 10.5);
           int x = 9;

           doStuff(walrus, x);
           System.out.println(walrus);
```

```
        System.out.println(x);
    }

    public static void doStuff(Walrus W, int x) {
        W.weight = W.weight - 100;
        x = x - 5;
    }
```

请判断打印出的x与walrus的值

解决问题的关键是理解java在储存primitive type类型变量与reference type类型变量的区别，后者的盒子储存的是walrus实例对象的地址，指向walrus（data存在其中），而前者则直接储存其值data
因此，无论是函数参数传递还是赋值操作，本质上都是将盒子里面存的bit复制一份放到新盒子里面去

所以本题中，

```
public static void doStuff(Walrus W,int x);
```

首先创建了两个空盒子W和x,在**main**中执行doStuff(walrus,x)时，将main中的walrus和x盒子中的值复制一份传递给doStuff里面去了，我们知道walrus里面存储的是64bit的地址，而x则储存的是32bit的值，因此这样做的结果是:

doStuff里面的W和main里面的walrus是存放的同一个地址，因此均指向Walrus(3500,10.5),而由于main里面的x只是储存的值而非地址，在函数传参时拷贝过去的就是x的值，而函数自己也有一个作用域，因此在函数里面对x进行减法不影响main里面的x

最终输出是

```
Walrus(3400,10.5)
```

x=9 不变
同样是引用类型的还有数组

```
int [] a;
a=new int[] {1,2,3};
```

第一行是创建了一个大小为64bit的盒子，第二行创建一个大小为3X32bit的盒子储存数据，然后通过new关键字返回首地址，=赋值给第一个盒子，也即是第一个盒子储存数组的首地址
整个过程是：声明---实例化---赋值
现在a是唯一存储实例对象{1,2,3}的地址的盒子，如果我们此时令
`a=new int[] {4,5,6};`
那么最开始的{1,2,3}的地址就丢失了，我们再也找不到这个对象，将会**Garbage collector**被作为垃圾回收

LinkedList, SLList, DLList

Doubly Linked List (Double Sentinel)

Doubly Linked List (Circular Sentinel)

Java allows us to defer type selection until declaration

X^N