

# PRINCIPAL OF PROGRAMING LANGUAGE ASSIGNMENT 1

INSTRUCTOR : MR.LE MINH DUC

STUDENT : LE THI TRANG

CLASS : 3C14

ID :1301040213

DATE : 5/4/2016

LE THI TRANG – 1301040213

1. Write a brief report detailing what you did in and the results.

In previous task, I based on design specification of the program CoffeeTinGame in the lecture and created a new design and new program for me. In my program, I used two new procedures is:

(a) takeTwo: take out random two beans from many beans with two color blue or green, and I use the procedure Math.random of the Java class, it have we to take random two beans. Because, the program before was not take beans randomly.

(b) updateTin: if two beans are sample colour

throw them both away, put one blue beans back

else throw away the bean blue, put green bean back .

In this report, I will show all things about my work, my result, mid-condition and reason for the requirement.

2. Implement new CoffeeTinGame program using the specification in task 1.

```
package ppl_assignment;

import java.util.Arrays;

/** @author Le Trang
 */

public class PPL_assignment {
    public static void main(String[] args) {
        String Tin[] = {"blue","green","blue","blue"};
        int beanNum=Tin.length;
        System.out.printf("the tin of beans before:%s %n",Arrays.toString(Tin));
```

```

while(beanNum>1){

    int chosenBean[] = takeTwo(beanNum);

    System.out.printf("the tin after: %s %n",
Arrays.toString(chosenBean));

    Tin=updateTin(chosenBean, Tin,beanNum);

    beanNum--;

    System.out.printf("the tin after choosing: %s %n",
Arrays.toString(Tin));

    System.out.println();

}

System.out.printf("the tin after: %s %n", Arrays.toString(Tin));

System.out.println("the last bean:"+Tin[0]);


}

public static int[] takeTwo(int n){

    int i;

    int j;

    int a[] = new int[2];

    do{

        i=(int)(Math.random()*n);

        j=(int)(Math.random()*n);

        a[0]=i;

```

```
a[1]=j;

    }while(a[0]==a[1]);

    return a;

}

public static String[] updateTin(int a[], String [] tin,int beanNum){

    if(tin[a[0]]==tin[a[1]]){

        for(int i=a[0];i<tin.length-1;i++){

            tin[i]=tin[i+1];

        }

        tin[tin.length-1]="";

        for(int i=a[1];i<tin.length-1;i++){

            tin[i]=tin[i+1];

        }

        tin[beanNum-2]="blue";

    }

    else{

        int remove;

        if(tin[a[0]]=="green")

            remove = a[0];
```

```
else
```

```
    remove = a[1];
```

```
    for(int i=remove;i<tin.length-1;i++){
```

```
        tin[i]=tin[i+1];
```

```
    }
```

```
    tin[tin.length-1]="";
```

```
    }
```

```
    return tin;
```

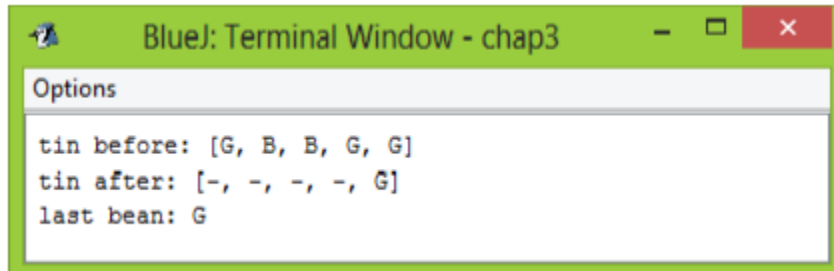
```
}
```

```
}
```



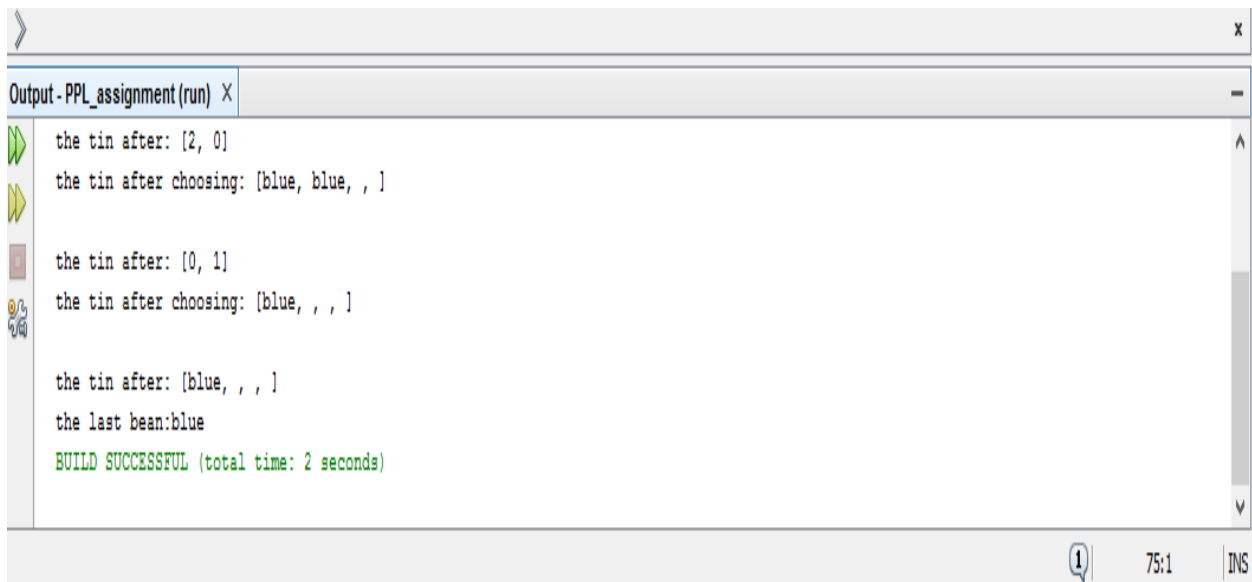
This is the output of each program

- Program CoffeeTinGame in the lecture have a function tinGame run the output below



```
BlueJ: Terminal Window - chap3
Options
tin before: [G, B, B, G, G]
tin after: [-, -, -, -, G]
last bean: G
```

- In my program, CoffeeTinGame have more two functions is takeTwo and updateTin (include updateTin), and we have output



```
Output - PPL_assignment (run) X
the tin after: [2, 0]
the tin after choosing: [blue, blue, , ]
the tin after: [0, 1]
the tin after choosing: [blue, , , ]
the tin after: [blue, , , ]
the last bean:blue
BUILD SUCCESSFUL (total time: 2 seconds)
```

In two outputs, we have tin before, tin after and last bean, the tin after different between two outputs because in my program two beans was take randomly. There for, the last bean of two outputs is the same.

3. Add suitable mid-condition to the code of program

```

public class ass1 {

    /**
     * main procedure
     * @effects:
     *     initialise a coffee tin
     *     display the tin content
     *     { @link CoffeeTinGame#takeTwo(int,String[]) }: choose 2 beans
randomly
     * { @link CoffeeTinGame#updateTin(int,String[]) }: update the content of
the tin
     */

    public static void main(String[] args) {
        // initialize a tin with 6 beans

        String Tin[] = {"blue","green","blue","blue","green","blue"};

        int beanNum=Tin.length;// beanNum la so bean ban dau trong tin

        System.out.printf("the tin of beans before: %s %n",
Arrays.toString(Tin));

        // P(beanNum) = true if n >= 2

        // false if n = 1

        // beanNum = number of bean

        while(beanNum>1){

```



```

int chosenBean[] = takeTwo(beanNum);//chosenBean = int[a,b]
^a!=b^a<beanNum^b<beanNum(a and b is index of chosen bean)

    System.out.printf("the tin after: %s %n",
Arrays.toString(chosenBean));

    Tin=updateTin(chosenBean, Tin,beanNum);//update tin

    beanNum--;//decrease the number of bean remains 1

    System.out.printf("the tin after choosing: %s %n",
Arrays.toString(Tin));

    System.out.println();

} // beanNum=1

System.out.printf("the tin after: %s %n", Arrays.toString(Tin));

System.out.println("the last bean:"+Tin[0]);

}

/**

    * pick randomly two beans from the tin \

    * @requires: tin has more than 1 bean

    * @effects:

    *         take randomly two separated beans in tin

    *         return index of 2  chosen beans in the tin

    */

```

```

public static int[] takeTwo(int n){ // ham tra ve mang cac so nguyen la
kieu int []

    int i;

    int j;

    int a[] = new int[2]; // khai bao mang so nguyen co 2 phan tu

    // take randomly two bean

    // Loop variant:

    //i= math.random(0 to n(number of beans remain))

    //j= math.random(0 to n)

    // true if i!=j

    // false if i=j

    do{

        i=(int)(Math.random()*n);

        j=(int)(Math.random()*n);

        a[0]=i; // phan tu dau tien chua so thu tu cua bean dau tien mk chon
trong mang

        a[1]=j;

        }while(a[0]==a[1]); //chose two bean again if the chosen indices is
the same

    return a;

}

```

```
/**
```

```
 * update the tin after taking two beans
```

```
 * @modifies: tin
```

```
 * @effects:
```

```
 *         remove the marked bean:
```

```
 *         if two beans have the same color
```

```
 *             remove both
```

```
 *             n=n-2
```

```
 *             add a blue bean at the end of the array
```

```
 *             n=n+1
```

```
 *         if only one is blue
```

```
 *             remove the blue one
```

```
 *             n=n-1
```

```
 *         return tin after updating
```

```
 */
```

```
public static String[] updateTin(int a[], String [] tin, int beanNum){
```

```
    if(tin[a[0]]==tin[a[1]]){//tin[a[0]] = tin[a[1]] ->remove both
```

```
        for(int i=a[0];i<tin.length-1;i++){
```

```
            //a[0]<i<tin.length-1=>tin[i]=tin[i+1]
```

```
            tin[i]=tin[i+1];
```

```
        }
```

```

tin[tin.length-1]="";//number of bean decrease 1

    for(int i=a[1];i<tin.length-1;i++){

        tin[i]=tin[i+1];

    }

    tin[beanNum-2]="blue";//and one blue back
}

else{//tin[a[0]] != tin[a[1]] ->remove blue one

    int remove;

    if(tin[a[0]]=="blue")//tin[a[0]] = blue ^ tin[a[1]] = green ->
remove tin a[0]

        remove = a[0];

    else

        remove = a[1];

    for(int i=remove;i<tin.length-1;i++){

        tin[i]=tin[i+1];

    }

    tin[tin.length-1]="";

}

return tin;

```

#### 4. Explain why mid-condition are correct with regards to the code

When we implement the specifications there is a very simple technique for reasoning. Midcondition covers the idea of pre and post-conditions by using logical that are supposed to hold at points in the middle of the computation not just at the beginning or end. They are written as comments in the middle of the code. Each mid-condition is supposed to hold whenever program control passes through that point at least provided that the procedure was called correctly with the pre-condition holding.

To prove a mid-condition we do not look at all the computation that has gone, but rather at the preceding program statement and the mid-condition just before that.

In takeTwo function we have a WHILE loop, a mid-condition here call the loop invariant and I put it before the loop

```
//loop invariant  
do (something)  
while (test)
```



Because, there are two things of reaching the invariant point, such as invariant is established initially, we continue looping then execution of the body will ensure that the invariant still holds next time around. For the CoffeeTinGame, the invariant is established by definition of  $p_0$ . It is reestablishment that is important. At the end, the payoff is that invariant holds, but the loop test fails (finish looping). In the pseudocode of the coffee tin game we have line 4 states the range of  $\text{tin.length}$ , which follows from the loop test being true, and the accumulated effect on variable  $i$  from the first iteration up to the iteration immediately before the current iteration, i.e.  $n*1$ ,  $n$  is incremented by one and  $i$  is the sum of the elements of the array  $a$  from 0 to  $n-1$ . In line 6, mid-condition follows mid-condition 4 and the variable assignment at line 5, which adds at element  $a[n]$  of the sum. At the end of CoffeeTinGame  $\text{tin}$  exist with two beans is still  $p_0$  and there is one bean left. Finally, this combination will show us exactly what is colour of the last bean

5. Explain why procedure  $\text{tinGame}$  still satisfies the given post-condition.

The post-condition is the final mid-condition. By this stage we know that by the time the program returns it must have set up the post-condition. Post-condition summarises behavior of program which include proc. invocations for non-trivial tasks.

A post-condition will usually want to compare the values before and after and this is where the special notation comes in.

$\langle \text{post} \rangle \ (p_0 = \text{even} \wedge \text{one blue left}) \vee (p_0 = \text{odd} \wedge \text{one green left})$  To specify that a variable does not change, for example  $p_0 = \text{even}$  says that does not  $p_0 = \text{even}$  does not change, value on return value on entry.