

Singleton Pattern

Prof. Jonathan Lee (李允中)

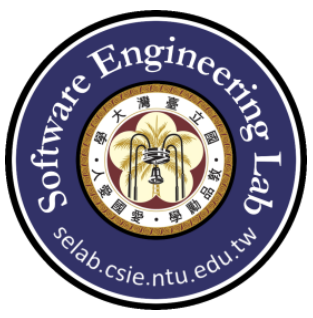
Department of CSIE

National Taiwan University



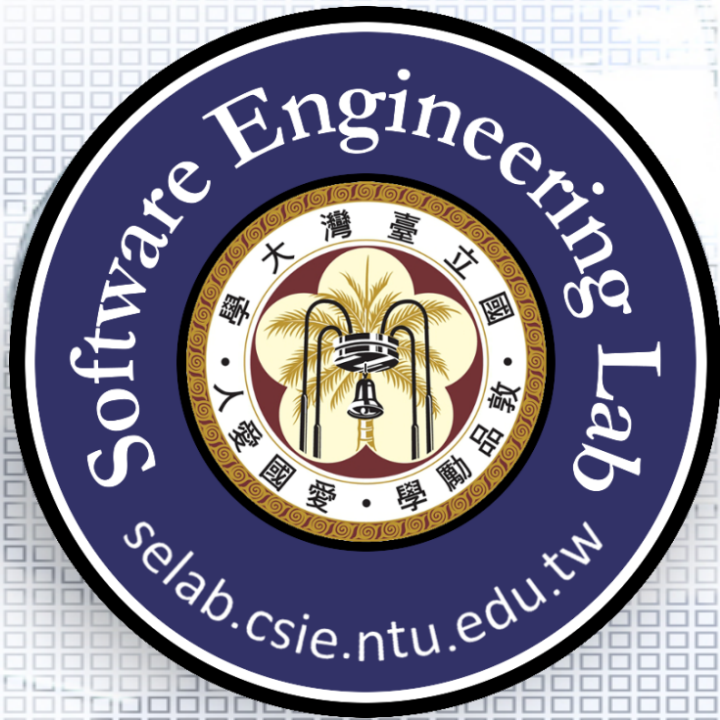
Design Aspect of Facade

the sole instance of a class



Outline

- ☐ Chocolate Boiler Requirements Statements
- ☐ Initial Design
- ☐ Recurrent Problems
- ☐ Intent
- ☐ Singleton Pattern Structure
- ☐ Singleton with Multi-threading Issues



Chocolate Boiler

Prof. Jonathan Lee (李允中)

Department of Computer Science and
Information Engineering
National Taiwan University



Requirements Statements

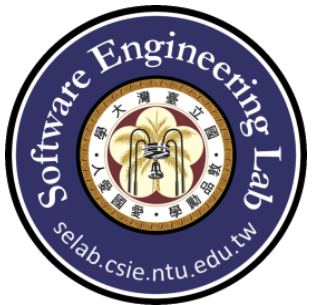
- ☐ A chocolate boiler is used to boil chocolate.
- ☐ Before boiling chocolate with the boiler, you have to make sure that the boiler is now empty and then fill chocolate in. Besides, you can't boil chocolate again while the chocolate has already been boiled.
- ☐ After boiling, drain out the boiled chocolate and make the boiler empty again.
- ☐ In order to prevent some unexpected situation, it is not allowed to have multiple instances of the chocolate boiler in the system.



Requirements Statements₁

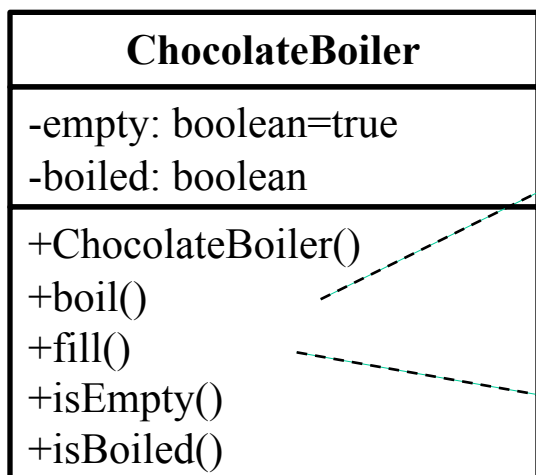
- ❑ A chocolate boiler is used to boil chocolate.

ChocolateBoiler
+ChocolateBoiler() +boil()



Requirements Statements₂

- ❑ Before boiling chocolate with the boiler, you have to make sure that the boiler is now empty and then fill chocolate in. Besides, you can't boil chocolate again while the chocolate has already been boiled.



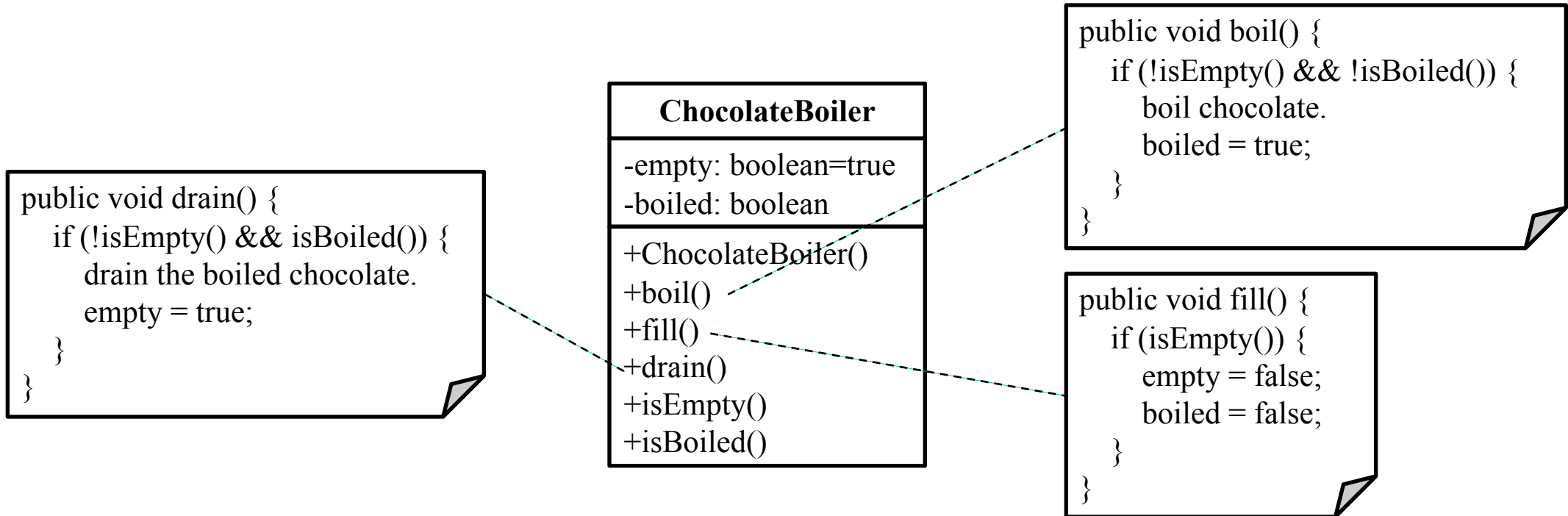
```
public void boil() {  
    if (!isEmpty() && !isBoiled()) {  
        boil chocolate.  
        boiled = true;  
    }  
}
```

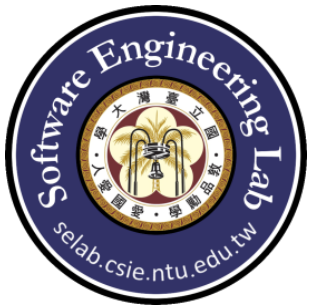
```
public void fill() {  
    if (isEmpty()) {  
        empty = false;  
        boiled = false;  
    }  
}
```




Requirements Statements₃

- ❑ After boiling, drain out the boiled chocolate and make the boiler empty again.



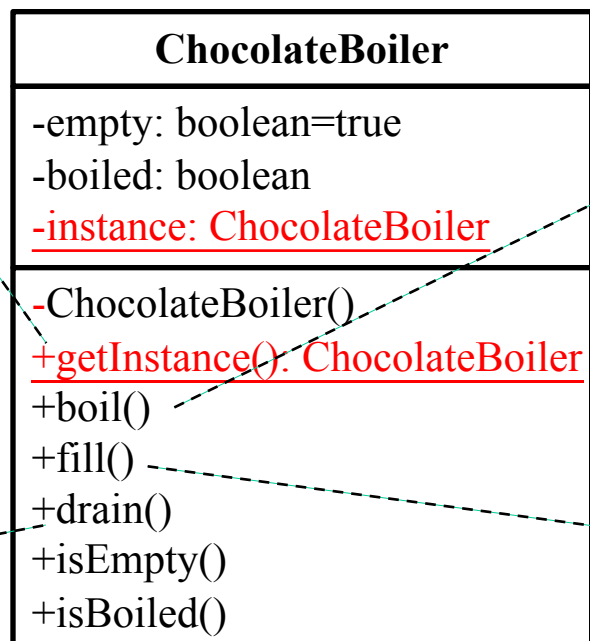


Requirements Statements₄

- ❑ In order to prevent some unexpected situation, it is not allowed to have multiple instances of the chocolate boiler in the system.

```
public static ChocolateBoiler getInstance() {  
    if (instance == null) {  
        instance = new ChocolateBoiler();  
    }  
    return instance;  
}
```

```
public void drain() {  
    if (!isEmpty() && isBoiled()) {  
        drain the boiled chocolate.  
        empty = true;  
    }  
}
```



```
public void boil() {  
    if (!isEmpty() && !isBoiled()) {  
        boil chocolate.  
        boiled = true;  
    }  
}
```

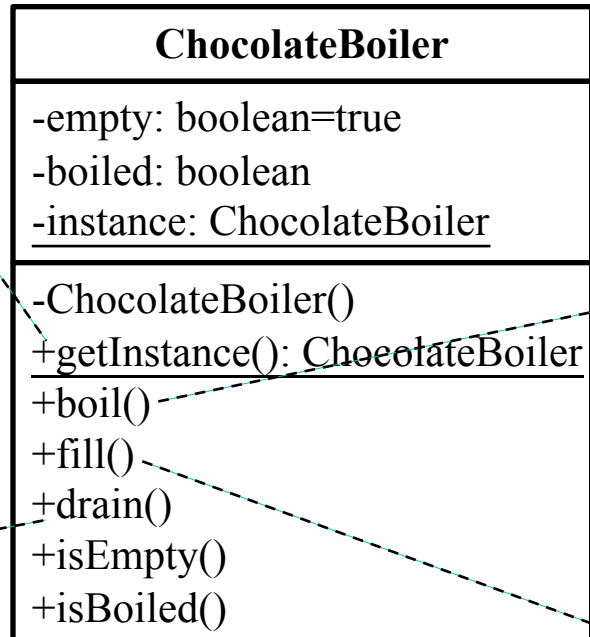
```
public void fill() {  
    if (isEmpty()) {  
        empty = false;  
        boiled = false;  
    }  
}
```



Initial Design

```
public static ChocolateBoiler getInstance() {  
    if (instance == null) {  
        instance = new ChocolateBoiler();  
    }  
    return instance;  
}
```

```
public void drain() {  
    if (!isEmpty() && !isBoiled()) {  
        drain the boiled chocolate.  
        empty = true;  
    }  
}
```



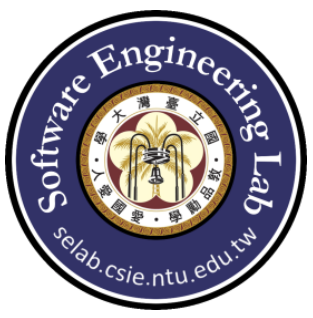
```
public void boil() {  
    if (!isEmpty() && !isBoiled()) {  
        boil chocolate.  
        boiled = true;  
    }  
}
```

```
public void fill() {  
    if (isEmpty()) {  
        empty = false;  
        boiled = false;  
    }  
}
```



Recurrent Problem

- ☐ It's important for some classes to have exactly one instance and ensure that the instance is easily accessible.
- ☐ A global variable makes an object accessible, but it doesn't keep you from instantiating multiple objects.

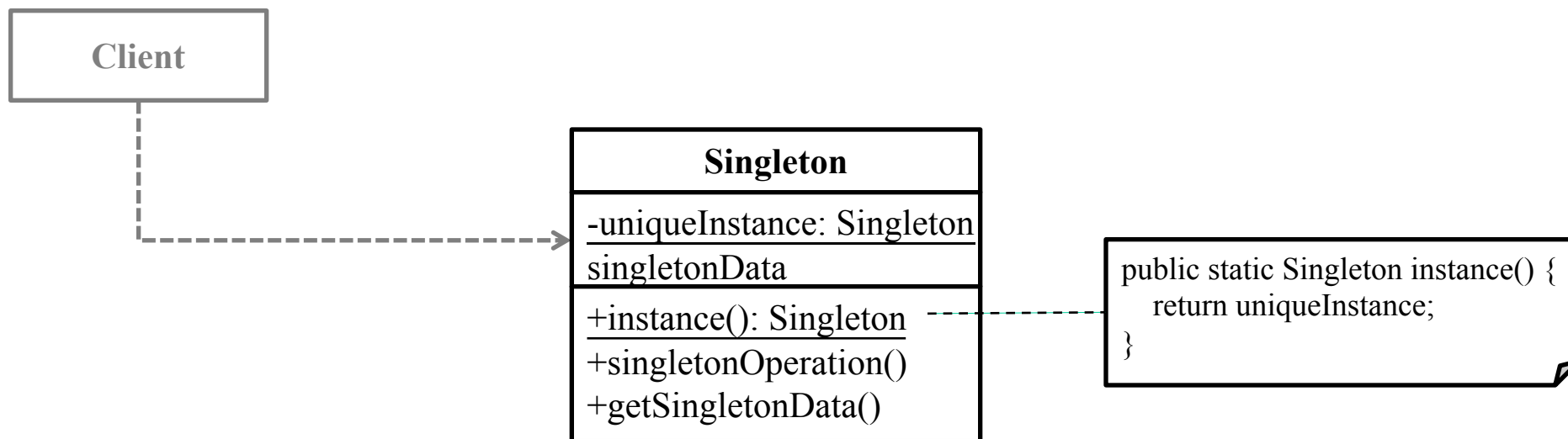


Intent

- ☐ Ensure a class only has one instance, and provide a global point of access to it.

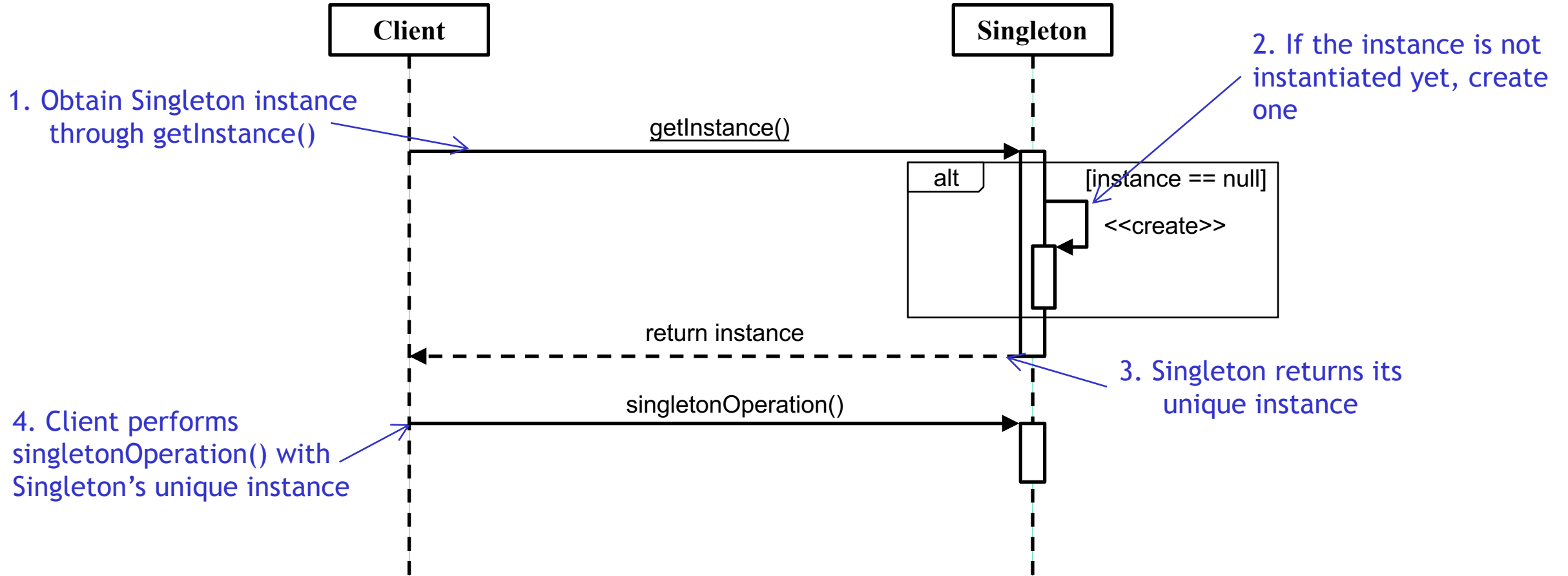


Singleton Structure₁





Singleton Structure₂





Singleton Structure₃

	Instantiation	Use	Termination
Singleton	Singleton creates itself and make sure there is only one instance.	Client gets the unique instance of Singleton from Singleton, and performs its operation(s).	Don't Care



Singleton with Multi-threading Issues₁

- ❑ **Issue:** In a multi-threading situation, if more than one thread request Singleton's instance, it may result in multiple instances which violates Singleton's intent.

Thread A
Thread B
Thread A
Thread B

```
public static ChocolateBoiler getInstance() {  
    if(instance == null) {  
        instance = new ChocolateBoiler();  
    }  
    return instance;  
}
```

Thread A has created new instance, and next, Thread B will also create another instance.

Both Thread A and B pass the guard condition to create Singleton's instance.

ChocolateBoiler
-empty: boolean -boiled: boolean <u>-instance: ChocolateBoiler</u>
<u>-ChocolateBoiler()</u> <u>+getInstance(): ChocolateBoiler</u> +boil() +fill() +drain() +isEmpty() +isBoiled()



Singleton with Multi-threading Issues₂

❑ Solution 1: Synchronizing the getInstance() method

This keyword in Java permits only one thread to enter the method at a time.

```
public static synchronized ChocolateBoiler getInstance() {  
    if (instance == null) {  
        instance = new ChocolateBoiler();  
    }  
    return instance;  
}
```

However, if the instance has been created already, it doesn't make sense to allow only one thread to obtain the existing instance at the a time.

ChocolateBoiler
-empty: boolean -boiled: boolean <u>-instance: ChocolateBoiler</u>
-ChocolateBoiler() <u>+getInstance(): ChocolateBoiler</u> +boil() +fill() +drain() +isEmpty() +isBoiled()



Singleton with Multi-threading Issues₃

❑ Solution 2: Eager creation

The instance is created when loading the ChocolateBoiler class, so the getInstance() method just return the early created instance to client.

```
public static ChocolateBoiler getInstance() {  
    return instance;  
}
```

It is a tradeoff to apply this approach if a Singleton instance occupies a huge amount of memory and is not needed so often.

ChocolateBoiler	
-empty: boolean	
-boiled: boolean	
-instance: ChocolateBoiler = new ChocolateBoiler()	
-ChocolateBoiler()	
+getInstance(): ChocolateBoiler	
+boil()	
+fill()	
+drain()	
+isEmpty()	
+isBoiled()	



Singleton with Multi-threading Issues₄

❑ Solution 3: Synchronizing the getInstance() method with double-checked locking

This statement in Java permits only one thread to enter this block at a time. If the instance is not created yet, the critical creation block allows only one thread to enter.

If the instance has been created, just return the existing instance without any synchronization.

Check nullity again in order to prevent creating instance more than once.

```
public static ChocolateBoiler getInstance() {  
    if (instance == null) {  
        synchronized (ChocolateBoiler.class) {  
            if (instance == null) {  
                instance = new ChocolateBoiler();  
            }  
        }  
    }  
    return instance;  
}
```

ChocolateBoiler
-empty: boolean -boiled: boolean -instance: ChocolateBoiler
-ChocolateBoiler() +getInstance(): ChocolateBoiler +boil() +fill() +drain() +isEmpty() +isBoiled()



ChocolateBoiler

```
public class ChocolateBoiler {
    private boolean empty = true;
    package-private more... (%%F1) boiled = false;
    private static ChocolateBoiler instance = new ChocolateBoiler();

    private ChocolateBoiler(){
    }

    public static ChocolateBoiler getInstance(){
        return instance;
    }

    public void boil() {
        if( !isEmpty() && !isBoiled() ){
            System.out.println("Boil chocolate");
            boiled = true;
        }
    }

    public void fill(){
        if(isEmpty()){
            System.out.println("Fill chocolate");
            empty = false;
            boiled = false;
        }
    }

    public void drain(){
        if(!isEmpty() && isBoiled()){
            System.out.println("Drain the boiled chocolate");
            empty = true;
        }
    }

    public boolean isEmpty(){ return empty; }

    public boolean isBoiled(){ return boiled; }
}
```



Input / Output

Input:

```
[Boil_chocolate_step]
```

```
...
```

Output:

```
//if [Boil_chocolate_step] is Fill
```

```
Fill chocolate
```

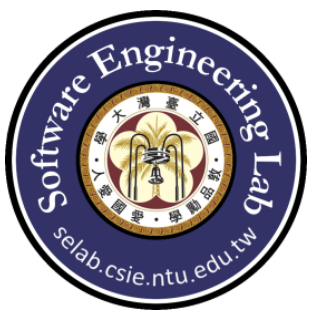
```
//if [Boil_chocolate_step] is Boil
```

```
Boil chocolate
```

```
//if [Boil_chocolate_step] is Drain
```

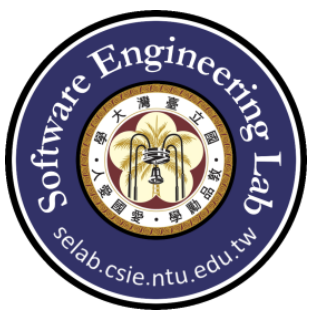
```
Drain the boiled chocolate
```

```
...
```



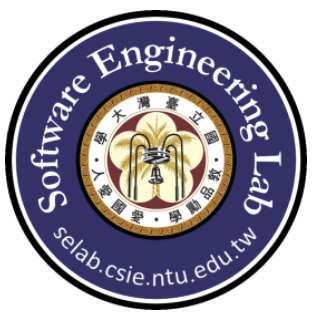
Test cases

- ☐ TestCase 1: Basic correct three operation
- ☐ TestCase 2: Random 100 operation



Test case1

Sample1.in		Sample1.out	
1	Fill	1	Fill chocolate
2	Boil	2	Boil chocolate
3	Drain	3	Drain the boiled chocolate



Test case2

```
Sample2.in
1 Fill 43 Boil 85 Fill
2 Fill 44 Drain 86 Fill
3 Drain 45 Fill 87 Fill
4 Boil 46 Fill 88 Drain
5 Fill 47 Boil 89 Boil
6 Fill 48 Fill 90 Fill
7 Boil 49 Fill 91 Boil
8 Drain 50 Boil 92 Fill
9 Fill 51 Drain 93 Drain
10 Drain 52 Drain 94 Fill
11 Fill 53 Drain 95 Boil
12 Fill 54 Drain 96 Boil
13 Drain 55 Drain 97 Boil
14 Boil 56 Drain 98 Drain
15 Boil 57 Boil 99 Drain
16 Drain 58 Boil 100 Drain
17 Drain 59 Drain
18 Boil 60 Drain
19 Drain 61 Fill
20 Drain 62 Drain
21 Fill 63 Drain
22 Drain 64 Drain
23 Drain 65 Fill
24 Boil 66 Fill
25 Boil 67 Drain
26 Fill 68 Fill
27 Boil 69 Fill
28 Boil 70 Boil
29 Boil 71 Drain
30 Drain 72 Drain
31 Boil 73 Drain
32 Boil 74 Fill
33 Drain 75 Fill
34 Fill 76 Drain
35 Drain 77 Fill
36 Fill 78 Drain
37 Fill 79 Drain
38 Boil 80 Drain
39 Fill 81 Drain
40 Drain 82 Drain
41 Drain 83 Boil
42 Fill 84 Drain
```

```
Sample2.out
1 Fill chocolate
2 Boil chocolate
3 Drain the boiled chocolate
4 Fill chocolate
5 Boil chocolate
6 Drain the boiled chocolate
7 Fill chocolate
8 Boil chocolate
9 Drain the boiled chocolate
10 Fill chocolate
11 Boil chocolate
12 Drain the boiled chocolate
13 Fill chocolate
14 Boil chocolate
15 Drain the boiled chocolate
16 Fill chocolate
17 Boil chocolate
18 Drain the boiled chocolate
19 Fill chocolate
20 Boil chocolate
21 Drain the boiled chocolate
22 Fill chocolate
23 Boil chocolate
24 Drain the boiled chocolate
25 Fill chocolate
26 Boil chocolate
27 Drain the boiled chocolate
28 Fill chocolate
29 Boil chocolate
30 Drain the boiled chocolate
```