5. 싱글턴 패턴

세상에서 단 하나뿐인 특별한 객체

private 생성자?

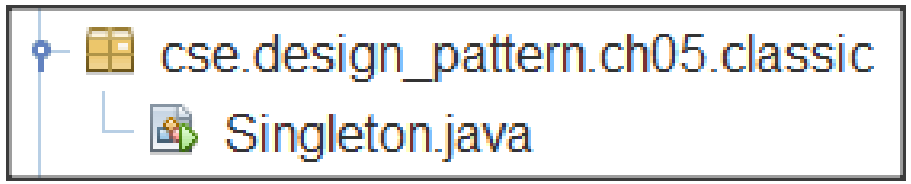
```
public MyClass {
  private MyClass() { }

  public static MyClass getInstance() {
    return new MyClass();
  }
}
```

외부에서 new MyClass() 라고 선언하여 객체 생성 불가함.

정적 메소드를 사용하여 객체 생성 가능 (예: MyClass.getInstance())

싱글턴 패턴

- Singleton Pattern
  - 문제: 정확하게 한 인스턴스만이 허용된다. 이것이 "singleton"이다. 객체는 전역적인 단일 접근점이 필요하다
  - 해결책: singleton을 반환하는 클래스의 정적 메소드를 정의한다.

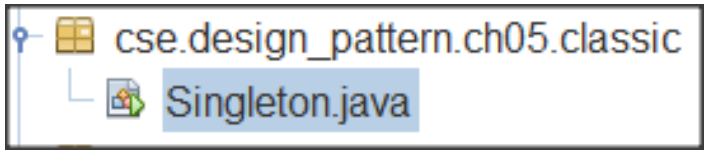


고전적인 싱글턴 패턴 구현법(classic)

```
5  public class Singleton {
6          private static Singleton uniqueInstance;
7
8          // other useful instance variables here
9
10         private Singleton() {}
11
12         public static Singleton getInstance() {
13                 if (uniqueInstance == null) {
14                         uniqueInstance = new Singleton();
15                 }
16                 return uniqueInstance;
17         }
18
19         // other useful methods here
20 }
```

- 문제점
  - 여러 개의 thread에서 동시에 getInstance() 메소드가 호출될 때 여러 개의 Singleton 객체가 생성될 수 있다.
  - 교재 p.217 & p.226 참조

5장: classic




```java
package cse.design_pattern.ch05.classic;

// NOTE: This is not thread safe!
public class Singleton {

    private static Singleton uniqueInstance;

    // other useful instance variables here
    private Singleton() {
        System.out.println("싱글턴 객체가 생성됩니다.");
    }

    public static Singleton getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }

    // other useful methods here

    /** 싱글턴 객체 생성 예 ...4 lines */
    public static void main(String[] args) {
        Singleton obj = Singleton.getInstance();

    }
}
```

초콜릿 공장 (chocolate01)

```
public class ChocolateBoiler {
    private boolean empty;
    private boolean boiled;

    public ChocolateBoiler() {
        empty = true;
        boiled = false;
    }

    public void fill() {
        if (isEmpty()) {
            empty = false;
            boiled = false;
            // 보일러에 우유/초콜릿을 [illegible]
        }
    }

    public void drain() {
        if (!isEmpty() && is[illegible]
            // 끓인 재료를 다음 [illegible]로 넘김
            empty = true;
        }
    }

    public void boil() {
        if (!isEmpty() && !isBoiled()) {
            // 재료를 끓임
            boiled = true;
        }
    }

    public boolean isEmpty() {
        return empty;
    }

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

이 코드는 보일러가 비어있을 때만 돌아갑니다.

보일러가 비어있을 때만 재료를 집어 넣습니다. 원료를 가득 채우고 나면 empty와 boiled 플래그를 false로 설정합니다.

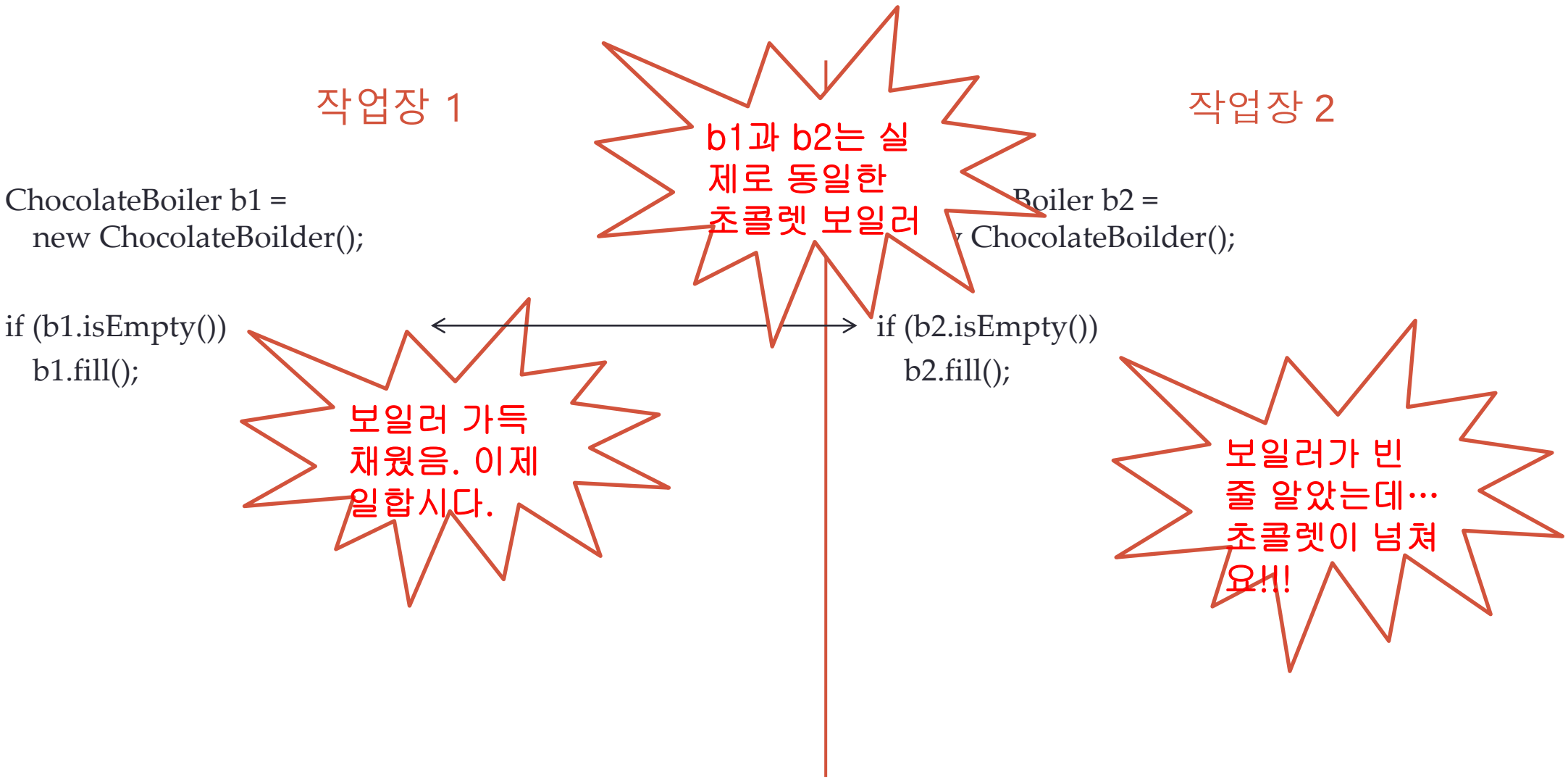
ChocolateBoiler 객체가 2개 이상 만들어지면?

[illegible]가 가득 차 있고(비어있지 않고), 다 끓[illegible] 상태에서만 보일러에 들어있는 재료를 다음 단계로 넘깁니다. 보일러를 다 비우고 나면 empty 플래그를 다시 true로 설정합니다.

보일러가 가득 차 있고 아직 끓지 않은 상태에서만 초콜릿과 우유가 혼합된 재료를 끓입니다. 재료가 다 끓고 나면 boiled 플래그를 true로 설정합니다.

p.209

초콜렛 공장에 보일러는 한 개!!!



5장: chocolate01

- ChocolateBoiler: 일반적인 객체 생성 방법 적용

ChocolateBoiler.java

```java
package cse.design_pattern.ch05.chocolate01;

public class ChocolateBoiler {

    private boolean empty;
    private boolean boiled;

    public ChocolateBoiler() {
        empty = true;
        boiled = false;
        System.out.println("ChocolateBoiler 생성자가 실행됩니다.");
    }

    public void fill() {
        if (isEmpty()) {
            empty = false;
            boiled = false;
            // fill the boiler with a milk/chocolate mixture
            System.out.println("보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.");
        }
    }
```

```
23      public void drain() {
24          if (!isEmpty() && isBoiled()) {
25              // drain the boiled milk and chocolate
26              empty = true;
27              System.out.println("차있으며 끓고 있으므로 이제 비웁니다.");
28          }
29      }
30
31      public void boil() {
32          if (!isEmpty() && !isBoiled()) {
33              // bring the contents to a boil
34              boiled = true;
35              System.out.println("보일러에 초콜렛과 우유가 있으므로 끓입니다.");
36          }
37      }
38
39      public boolean isEmpty() {
40          return empty;
41      }
42
43      public boolean isBoiled() {
44          return boiled;
45      }
46  }
```

ChocolateController.java

```
1   package cse.design_pattern.ch05.chocolate01;
2
3   public class ChocolateController {
4
5       public static void main(String args[]) {
6           new Thread(() -> {
7               ChocolateBoiler boiler = new ChocolateBoiler();
8               System.out.println("객체 정보: " + boiler);
9               boiler.fill();
10              boiler.boil();
11              boiler.drain();
12          }).start();
13
14          new Thread(() -> {
15              ChocolateBoiler boiler = new ChocolateBoiler();
16              System.out.println("객체 정보: " + boiler);
17              boiler.fill();
18              boiler.boil();
19              boiler.drain();
20          }).start();
21
22      }
23  }
```

실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---
ChocolateBoiler 생성자가 실행됩니다.
객체 정보: cse.design_pattern.ch05.chocolate01.ChocolateBoiler@136283e3
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.
ChocolateBoiler 생성자가 실행됩니다.
객체 정보: cse.design_pattern.ch05.chocolate01.ChocolateBoiler@74cd8833
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.
```

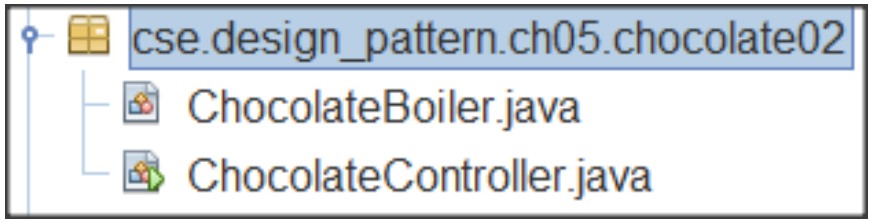
해결 방법 (chocolate02)

- 싱글턴 클래스!!!

```
public class ChocolateBoiler {
    private boolean empty;
    private boolean boiled;

    private static ChocolateBoiler uniqueInstance;

    private ChocolateBoiler() {
        empty = true;
        boiled = false;
    }

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

    public void fill() {
        if (isEmpty()) {
            empty = false;
            boiled = false;
            // 기타 코드...
        }
    }
    // 기타 코드...
}
```

5장: chocolate02

- ChocolateBoiler: 싱글턴 패턴 적용하여 유일한 객체 하나만 생성



ChocolateBoiler.java

```
1   package cse.design_pattern.ch05.chocolate02;
2
3   public class ChocolateBoiler {
4
5       private boolean empty;
6       private boolean boiled;
7       private static ChocolateBoiler uniqueInstance;
8
9       private ChocolateBoiler() {
10          empty = true;
11          boiled = false;
12          System.out.println("ChocolateBoiler 생성자가 실행됩니다.");
13      }
14
15      /** 싱글턴 패턴을 적용하여 초콜렛 보일러의 유일한 객체를 반환하는 메
20      public static ChocolateBoiler getInstance() {
21          if (uniqueInstance == null) {
22              System.out.println("₩n1. 초콜렛 보일러의 유일한 객체를 생성");
23              uniqueInstance = new ChocolateBoiler();
24          }
25          System.out.println("2. 초콜렛 보일러 객체를 반환");
26          return uniqueInstance;
27      }
```

```
29      public void fill() {
30          if (isEmpty()) {
31              empty = false;
32              boiled = false;
33              // fill the boiler with a milk/chocolate mixture
34              System.out.println("보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.");
35          }
36      }
37
38      public void drain() {
39          if (!isEmpty() && isBoiled()) {
40              // drain the boiled milk and chocolate
41              empty = true;
42              System.out.println("차있으며 끓고 있으므로 이제 비웁니다.");
43          }
44      }
45
46      public void boil() {
47          if (!isEmpty() && !isBoiled()) {
48              // bring the contents to a boil
49              boiled = true;
50              System.out.println("보일러에 초콜렛과 우유가 있으므로 끓입니다.");
51          }
52      }
53
54      public boolean isEmpty() {
55          return empty;
56      }
57
58      public boolean isBoiled() {
59          return boiled;
60      }
61  }
```

ChocolateController.java

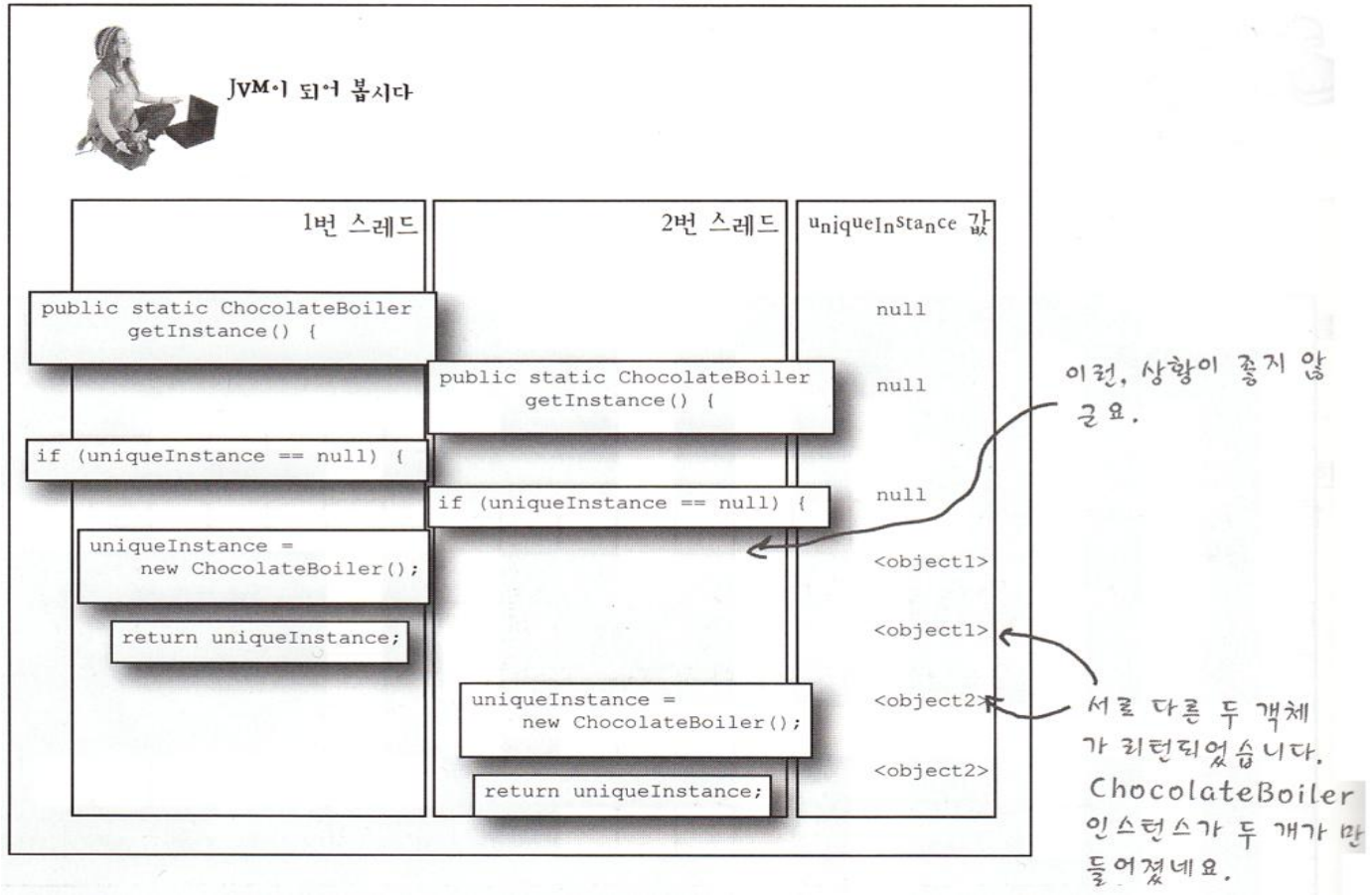
```java
package cse.design_pattern.ch05.chocolate02;

public class ChocolateController {

    public static void main(String args[]) {
        ChocolateBoiler boiler = ChocolateBoiler.getInstance();
        System.out.println("객체 정보: " + boiler);
        boiler.fill();
        boiler.boil();
        boiler.drain();

        // will return the existing instance
        ChocolateBoiler boiler2 = ChocolateBoiler.getInstance();
        System.out.println("객체 정보: " + boiler2);
        boiler2.fill();
        boiler2.boil();
        boiler2.drain();
    }
}
```

실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---

1. 초콜렛 보일러의 유일한 객체를 생성
ChocolateBoiler 생성자가 실행됩니다.
2. 초콜렛 보일러 객체를 반환
객체 정보: cse.design_pattern.ch05.chocolate02.ChocolateBoiler@15db9742
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.
2. 초콜렛 보일러 객체를 반환
객체 정보: cse.design_pattern.ch05.chocolate02.ChocolateBoiler@15db9742
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.
```

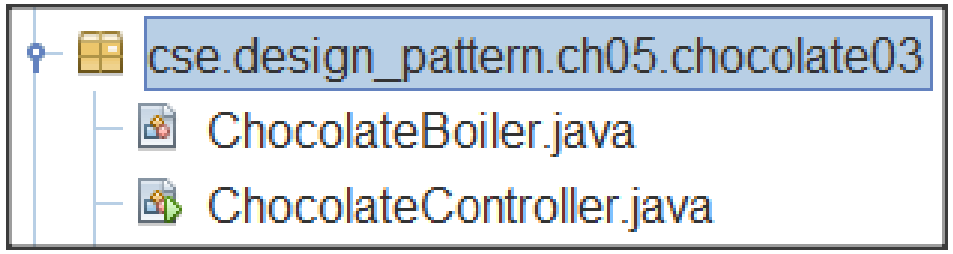
여전히 문제가... (chocolate03)

- 여전히 초콜렛 보일러 객체가 2개 이상 생성 가능하네요!!!



5장: chocolate03

- ChocolateBoiler: 싱글턴 패턴 적용 시에서 다중 스레드에서 동작하면 싱글턴이 아닌 여러 개의 객체 생성 가능



ChocolateBoiler.java

```java
package cse.design_pattern.ch05.chocolate03;

public class ChocolateBoiler {

    private boolean empty;
    private boolean boiled;
    private static ChocolateBoiler uniqueInstance;

    private ChocolateBoiler() {
        empty = true;
        boiled = false;
        System.out.println("ChocolateBoiler 생성자가 실행됩니다.");
    }

    /** 싱글턴 패턴을 적용하여 초콜렛 보일러의 유일한 객체를 반환하는 메
    public static ChocolateBoiler getInstance() {

        if (uniqueInstance == null) {
            try {
                System.out.println("₩n1초 간 지연됩니다.");
                Thread.sleep(1000);
            } catch (InterruptedException ex) {
                System.err.println(ex);
            }
            System.out.println("₩n1. 초콜렛 보일러의 유일한 객체를 생성");
            uniqueInstance = new ChocolateBoiler();
        }
        System.out.println("2. 초콜렛 보일러 객체를 반환");
        return uniqueInstance;
    }
```

```
37      public void fill() {
38          if (isEmpty()) {
39              empty = false;
40              boiled = false;
41              // fill the boiler with a milk/chocolate mixture
42              System.out.println("보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.");
43          }
44      }
45
46      public void drain() {
47          if (!isEmpty() && isBoiled()) {
48              // drain the boiled milk and chocolate
49              empty = true;
50              System.out.println("차있으며 끓고 있으므로 이제 비웁니다.");
51          }
52      }
53
54      public void boil() {
55          if (!isEmpty() && !isBoiled()) {
56              // bring the contents to a boil
57              boiled = true;
58              System.out.println("보일러에 초콜렛과 우유가 있으므로 끓입니다.");
59          }
60      }
61
62      public boolean isEmpty() {
63          return empty;
64      }
65
66      public boolean isBoiled() {
67          return boiled;
68      }
69  }
```

ChocolateController.java

```java
package cse.design_pattern.ch05.chocolate03;

public class ChocolateController {

    public static void main(String args[]) {
        new Thread(() -> {
            ChocolateBoiler boiler = ChocolateBoiler.getInstance();
            System.out.println("객체 정보: " + boiler);
            boiler.fill();
            boiler.boil();
            boiler.drain();
        }).start();

        // TODO  500msec 정지시 초콜렛 보일러 객체 두 개가 생성됨.
        //          > 1000msec 정지시에는 하나만 생성됨.
        try {
            Thread.sleep(500);
        } catch (InterruptedException ex) {
            System.err.println(ex);
        }

        new Thread(() -> {
            ChocolateBoiler boiler = ChocolateBoiler.getInstance();
            System.out.println("객체 정보: " + boiler);
            boiler.fill();
            boiler.boil();
            boiler.drain();
        }).start();
    }
}
```

실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---

1초 간 지연됩니다.

1초 간 지연됩니다.

1. 초콜렛 보일러의 유일한 객체를 생성
ChocolateBoiler 생성자가 실행됩니다.
2. 초콜렛 보일러 객체를 반환
객체 정보: cse.design_pattern.ch05.chocolate03.ChocolateBoiler@74cd8833
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.

1. 초콜렛 보일러의 유일한 객체를 생성
ChocolateBoiler 생성자가 실행됩니다.
2. 초콜렛 보일러 객체를 반환
객체 정보: cse.design_pattern.ch05.chocolate03.ChocolateBoiler@58458a33
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.
```

다른 ChocolateBoiler 객체 생성 확인!

보완책1: Thread Safe Version

- getInstance() 메소드를 동기화시키면 다중쓰레딩 관련 문제가 해결됨.
- 단점: 속도 지연 가능성 있음.

```
3  public class Singleton {
4          private static Singleton uniqueInstance;
5
6          // other useful instance variables here
7
8          private Singleton() {}
9
10         public static synchronized Singleton getInstance() {
11                 if (uniqueInstance == null) {
12                         uniqueInstance = new Singleton();
13                 }
14                 return uniqueInstance;
15         }
16
17         // other useful methods here
18 }
```

ChocolateBoiler (thread_safe)

```
15  /**
16   * 싱글턴 패턴을 적용하여 초콜렛 보일러의 유일한 객체를 반환하는 메소드.
17   * 스레드 적용시 문제점을 보기 위하여 Thread.sleep() 사용함.
18   *
19   * @return 초콜렛 보일러의 유일한 객체를 반환
20   */
21  public static synchronized ChocolateBoiler getInstance() {
22
23      if (uniqueInstance == null) {
24          try {
25              System.out.println("₩n1초 간 지연됩니다.");
                Thread.sleep(1000);
27          } catch (InterruptedException ex) {
28              System.err.println(ex);
29          }
30          System.out.println("₩n1. 초콜렛 보일러의 유일한 객체를 생성");
31          uniqueInstance = new ChocolateBoiler();
32      }
33      System.out.println("2. 초콜렛 보일러 객체를 반환");
34      return uniqueInstance;
35  }
```

실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---

1초 간 지연됩니다.

1. 초콜렛 보일러의 유일한 객체를 생성
ChocolateBoiler 생성자가 실행됩니다.
2. 초콜렛 보일러 객체를 반환
2. 초콜렛 보일러 객체를 반환
객체 정보: cse.design_pattern.ch05.thread_safe.ChocolateBoiler@60bc58ca
객체 정보: cse.design_pattern.ch05.thread_safe.ChocolateBoiler@60bc58ca
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
```

보완책2: 인스턴스 미리 생성

- 인스턴스를 필요할 때 생성하지 말고
  (늦은 초기화; lazy initialization)
  프로그램 시작시 미리 생성
  (적극적 초기화; eager initialization)

```
 3 public class Singleton {
 4         private static Singleton uniqueInstance = new Singleton();
 5
 6         private Singleton() {}
 7
 8         public static Singleton getInstance() {
 9                 return uniqueInstance;
10         }
11 }
```

ChocolateBoiler (eager_init)

```
6   public class ChocolateBoiler {
7
8       private boolean empty;
9       private boolean boiled;
10      private static ChocolateBoiler uniqueInstance = new ChocolateBoiler();
11
12      private ChocolateBoiler() {
13          empty = true;
14          boiled = false;
15          System.out.println("ChocolateBoiler 생성자가 실행됩니다.");
16      }
17
18      /** 싱글턴 패턴을 적용하여 초콜렛 보일러의 유일한 객체를 반환하는 메소드 ...
24      public static ChocolateBoiler getInstance() {
25
26          System.out.println("초콜렛 보일러 객체를 반환");
27          return uniqueInstance;
28      }
```

실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---
ChocolateBoiler 생성자가 실행됩니다.
초콜렛 보일러 객체를 반환
객체 정보: cse.design_pattern.ch05.eager_init.ChocolateBoiler@74cd8833
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.
초콜렛 보일러 객체를 반환
객체 정보: cse.design_pattern.ch05.eager_init.ChocolateBoiler@74cd8833
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.
```

Keyword “volatile” in JAVA

- 서로 다른 쓰레드에 의하여 동시에 수정 가능한 변수에 사용
- 레지스터에 변수를 캐싱하여 사용하는 것이 아니라 매번 주기억장치에서 새 값을 읽어오도록 컴파일러에게 경고함.
- 다른 쓰레드에서 지역 변수는 볼 수 없으므로, 지역 변수를 volatile로 지정할 필요는 없음.

보완책3: Double-Checked Locking

- Double-checked locking idiom

```
 8 public class Singleton {
 9         private volatile static Singleton uniqueInstance;
10
11         private Singleton() {}
12
13         public static Singleton getInstance() {
14                 if (uniqueInstance == null) {
15                         synchronized (Singleton.class) {
16                                 if (uniqueInstance == null) {
17                                         uniqueInstance = new Singleton();
18                                 }
19                         }
20                 }
21                 return uniqueInstance;
22         }
23 }
```

Double-Checked Locking

ChocolateBoiler (dcl)

```
10      private volatile static ChocolateBoiler uniqueInstance;
11
12      private ChocolateBoiler() {...5 lines }
17
18      /** 싱글턴 패턴을 적용하여 초콜렛 보일러의 유일한 객체를 반환하는 메소드 ...6 lin
24      public static ChocolateBoiler getInstance() {
25
26          if (uniqueInstance == null) {
27              try {
28                  System.out.println("₩ngetInstance: null checking - 1초간 지연");
29                  Thread.sleep(1000);
30              } catch (InterruptedException ex) {
31              }
32              synchronized (ChocolateBoiler.class) {
33                  if (uniqueInstance == null) {
34                      try {
35                          System.out.println("₩ngetInstance: null checking - 1초간 지연");
                            Thread.sleep(1000);
37                      } catch (InterruptedException ex) {
38                      }
39                      uniqueInstance = new ChocolateBoiler();
40                  }
41              }
42          }
43          System.out.println("초콜렛 보일러 객체를 반환");
44          return uniqueInstance;
45      }
```

실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---

getInstance: null checking - 1초간 지연

getInstance: null checking - 1초간 지연

getInstance: null checking - 1초간 지연
ChocolateBoiler 생성자가 실행됩니다.
초콜렛 보일러 객체를 반환
초콜렛 보일러 객체를 반환
객체 정보: cse.design_pattern.ch05.dcl.ChocolateBoiler@51b50f87
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.
객체 정보: cse.design_pattern.ch05.dcl.ChocolateBoiler@51b50f87
보일러가 비어 있으므로 초콜렛과 우유를 채웁니다.
보일러에 초콜렛과 우유가 있으므로 끓입니다.
차있으며 끓고 있으므로 이제 비웁니다.
```

Double-Checked Locking

- Also known as “double-checked locking optimization”
- Designed to reduce the overhead of acquiring a lock by first testing the locking criteria (the 'lock hint') in an unsafe manner.
- 다중쓰레드 환경에서 늦은 초기화를 구현할 때 잠금 오버헤드(locking overhead)를 줄여주기 위하여 사용됨.