1) 예제 #1

```
1  package patterns.singleton;
2
3  import java.text.NumberFormat;
4
5  public class Example1 {
6
7      public static void main(String[] args) {
8          Runtime runtime = Runtime.getRuntime();
9
10         int processors = runtime.availableProcessors();
11         System.out.printf("available processors: %dWn", processors);
12
13         long freeMemory = runtime.freeMemory();
14         String s = NumberFormat.getInstance().format(freeMemory);
15         System.out.printf("free memory: %sWn", s);
16     }
17
18 }
```

출력

```
available processors: 12
free memory: 534,342,536
```

Runtime

Runtime 클래스는 Singleton 이다.

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Runtime.html

public static Runtime getRuntime()
  Returns the runtime object associated with the current Java application.

NumberFormat

NumberFormat 클래스는 Singleton이 아니다.

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/NumberFormat.html

public static final NumberFormat getInstance()
  Returns a general-purpose number format for the current default FORMAT locale.

protected NumberFormat()
  Sole constructor. (For invocation by subclass constructors, typically implicit.)

분석 실습 - factory method 사용 이유?

NumberFormat 객체를 생성할 때, getInstance() 메소드를 사용하도록 구현된 이유는?

생성자 대신 factory method를 사용하여 객체를 생성해야 하는 경우는?

2) 예제 #2

```
1  package singleton;
2
3  import java.io.Console;
4  import java.text.NumberFormat;
5
6  public class Example2 {
7
8      public static void main(String[] args) {
9          Runtime runtime = Runtime.getRuntime();
10         Console console = System.console();
11
12         int processors = runtime.availableProcessors();
13         System.out.printf("available processors: %dWn", processors);
14         console.printf("available processors: %dWn", processors);
15
16         long freeMemory = runtime.freeMemory();
17         String s = NumberFormat.getInstance().format(freeMemory);
18         System.out.printf("free memory: %sWn", s);
19         console.printf("free memory: %sWn", s);
20     }
21
22 }
```

eclipse 에서 실행하면, System.console() 메소드가 null 리턴하므로, 터미널(명령 프롬프트)에서 실행해야함.

출력

```
available processors: 12
available processors: 12
free memory: 534,342,504
free memory: 534,342,504
```

System.out

표준 출력
표준 출력은 redirect 될 수 있다.

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html

Console

화면 출력
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/Console.html

System 클래스의 console 메소드

public static Console console()

Returns the unique Console object associated with the current Java virtual machine, if any, otherwise null.

Console 클래스 private 생성자

private Console()

System.java

```
    private static volatile Console cons;

    public static Console console() {
         Console c;
         if ((c = cons) == null) {
             synchronized (System.class) {
                 if ((c = cons) == null) {
                     cons = c = SharedSecrets.getJavaIOAccess().console();
                 }
             }
         }
         return c;
    }
```

double checked locking 구현

분석 실습 - singleton?

System.out 객체를 singleton 패턴으로 구현하지 않은 이유는?

Console single instance를 리턴하는 메소드를, Console 클래스가 아니고 System 클래스에 구현한 이유는?

3) Double Checked Locking

```
1  class SingletonA {
2      private static SingletonA instance;
3
4      public static SingletonA getInstance() {
5          if (instance == null)
6              instance = new SingletonA();
7          return instance;
8      }
9  }
10
11 class SingletonB {
12     private static SingletonB instance;
13
14     public static synchronized SingletonB getInstance() {
15         if (instance == null)
16             instance = new SingletonB();
17         return instance;
18     }
19 }
20
21 class SingletonC {
22     private static SingletonC instance;
23
24     public static SingletonC getInstance() {
25         synchronized (SingletonC.class) {
26             if (instance == null)
27                 instance = new SingletonC();
28             return instance;
29         }
30     }
31 }
32
33 class SingletonD {
34     private static SingletonD instance;
35
36     public static SingletonD getInstance() {
37         synchronized (SingletonD.class) {
38             if (instance == null)
39                 instance = new SingletonD();
40         }
41         return instance;
42     }
43 }
44
45 class SingletonE {
46     private static SingletonE instance;
47
48     public static SingletonE getInstance() {
49         if (instance == null)
50             synchronized (SingletonE.class) {
51                 if (instance == null)
52                     instance = new SingletonE();
53             }
54         return instance;
55     }
56 }
57
58 class SingletonF {
59     private static volatile SingletonF instance;
60
61     public static SingletonF getInstance() {
62         if (instance == null)
63             synchronized (SingletonF.class) {
64                 if (instance == null)
65                     instance = new SingletonF();
66             }
67         return instance;
68     }
```

```
69  }
```

thread safe 하지 않은 구현을 모두 선택하시오.

다중 스레드 환경에서 매우 비효율적인 구현을 모두 선택하시오.

성능 개선

```
class SingletonG {
    private static volatile SingletonG instance;

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

thread safe 한가?

성능이 조금 개선된 이유는?

4) C# Console

```
1  using System;
2
3  namespace Hello_World
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World!");
10             Console.Write("Enter your name: ");
11
12             String name = Console.ReadLine();
13             Console.Write("Good day, " + name);
14         }
15     }
16 }
```

C# 언어의 Console 클래스
  화면 입출력 메소드들이 전부 static 이다.

분석 실습 - singleton vs static method

아래의 예제 #3A GlobalDataA 클래스는 singleton 패턴으로 구현되었다.
  예제 #3B GlobalDataB 클래스는 모든 멤버들이 static 이다.

두 구현의 장단점을 비교하시오.

5) 예제 #3A

```
1  package singleton;
2
3  class GlobalDataA {
4      private int[] data;
5      private int count;
6      private static GlobalDataA instance = null;
7
8      private GlobalDataA() {
9          data = new int[1000];
10         count = 0;
11     }
12
13     public static GlobalDataA getInstance() {
14         if (instance == null)
15             instance = new GlobalDataA();
16         return instance;
17     }
18
19     public void addData(int value) {
20         data[count] = value;
21         ++count;
22     }
23
24     public int getData(int index) {
25         return data[index];
26     }
27
28     public int getCount() {
29         return count;
30     }
31 }
32
33 public class Example3a {
34     public void doSomething1() {
35         GlobalDataA a = GlobalDataA.getInstance();
36         a.addData(123);
37         a.addData(456);
38         a.addData(789);
39     }
40
41     public void doSomething2() {
42         GlobalDataA a = GlobalDataA.getInstance();
43         for (int i = 0; i < a.getCount(); ++i) {
44             int value = a.getData(i);
45             System.out.println(value);
46         }
47     }
48
49     public static void main(String[] args) {
50         var client = new Example3a();
51         client.doSomething1();
52         client.doSomething2();
53     }
54 }
```

GlobalDataA - singleton 패턴 구현

6) 예제 #3B

```
1  package singleton;
2
3  class GlobalDataB {
4      private static int[] data;
5      private static int count;
6
7      static {
8          data = new int[1000];
9          count = 0;
10     }
11
12     public static void addData(int value) {
13         data[count] = value;
14         ++count;
15     }
16
17     public static int  getData(int index) {
18         return data[index];
19     }
20
21     public static int  getCount() {
22         return count;
23     }
24 }
25
26 public class Example3b {
27     public void doSomething1() {
28         GlobalDataB.addData(123);
29         GlobalDataB.addData(456);
30         GlobalDataB.addData(789);
31     }
32
33     public void doSomething2() {
34         for (int i=0; i < GlobalDataB.getCount(); ++i) {
35             int value = GlobalDataB.getData(i);
36             System.out.println(value);
37         }
38     }
39
40     public static void main(String[] args) {
41         var client = new Example3b();
42         client.doSomething1();
43         client.doSomething2();
44     }
45 }
```

GlobalDataB - static 메소드들로 구현