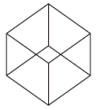
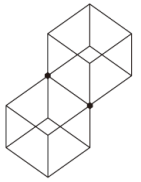
6. 싱글턴패턴



JAVA 객체 지향 디자인 패턴

UML과 GoF 디자인 패턴 핵심 10가지로 배우는

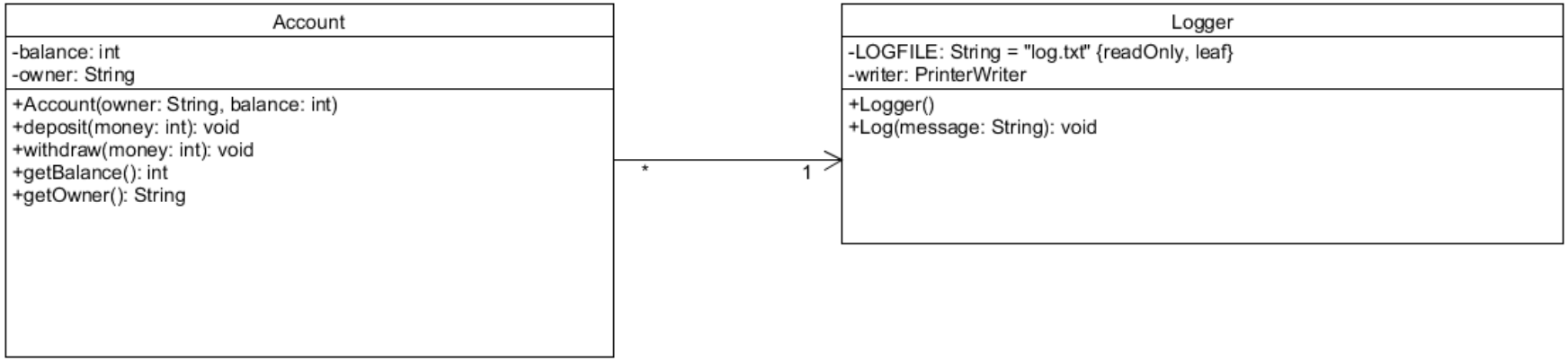


학습목표

- 싱글턴 패턴 이해하기
  - Eager Initialization
  - Lazy Initialization
- 다중 쓰레드 환경에서 싱글턴 패턴
  - Synchronized
  - DCL(Double Checked Locking)
  - Initialization on demand holder idiom

Logger 만들기

- 공통 로그 파일에 모든 사용자 계좌의 입금/출금의 발생 내역을 기록
- 설계



Logger 클래스

```
public class Logger {
    private final String LOGFILE = "log.txt";
    private PrintWriter writer;
    public Logger() {
        try {
            FileWriter fw = new FileWriter(LOGFILE);
            writer = new PrintWriter(fw, true);
        } catch (IOException e) {}
    }
    public void log (String message) {
        SimpleDateFormat formatter= new SimpleDateFormat("yyyy-MM-dd 'at' HH:mm:ss z");
        Date date = new Date(System.currentTimeMillis());
        writer.println(formatter.format(date) + ":" + message);
    }
}
```

Account 클래스

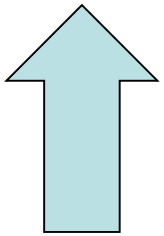
```
public class Account {
    private String owner;
    private int balance;
    private Logger myLogger;

    public Account(String owner, int balance) {
        this.owner = owner;
        this.balance = balance;
        this.myLogger = new Logger();
    }

    public String getOwner() { return owner; }
    public int getBalance() { return balance; }

    public void deposit(int money) {
        myLogger.log("owner" + " : " +this.getOwner() + " deposit " + money);
        balance += money;
    }
    public void withdraw(int money) {
        if (balance >= money) {
            myLogger.log("owner" + " : " +this.getOwner() + " withdraw " + money);
            balance -= money;
        }
    }

}
```

Main 클래스

```
public class Main {
    public static void main(String[] args) {
        Account acct1 = new Account("insang1", 1000000);
        acct1.deposit(20000);
        Account acct2 = new Account("insang2", 2000000);
        acct2.withdraw(5000);
    }
}
```

실행 결과: log.txt

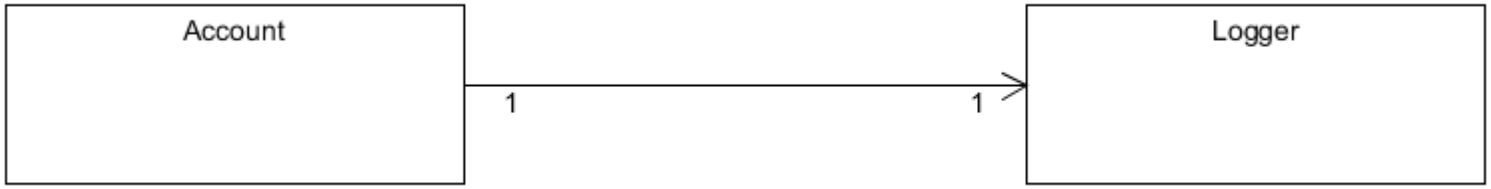
```
2020-07-21 at 10:50:33 KST : owner :
insang2 withdraw 5000
```



Insang1의 입금 내역이 누락

문제점

- Account 인스턴스가 생성될 때 마다 새로운 Logger 인스턴스 생성



- 해결책
  - 모든 Account 인스턴스가 하나의 Logger 인스턴스를 공유하도록 설계

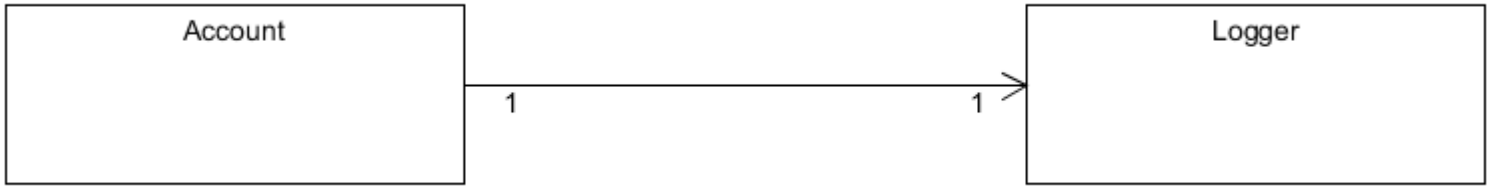
Accout 클래스-revised

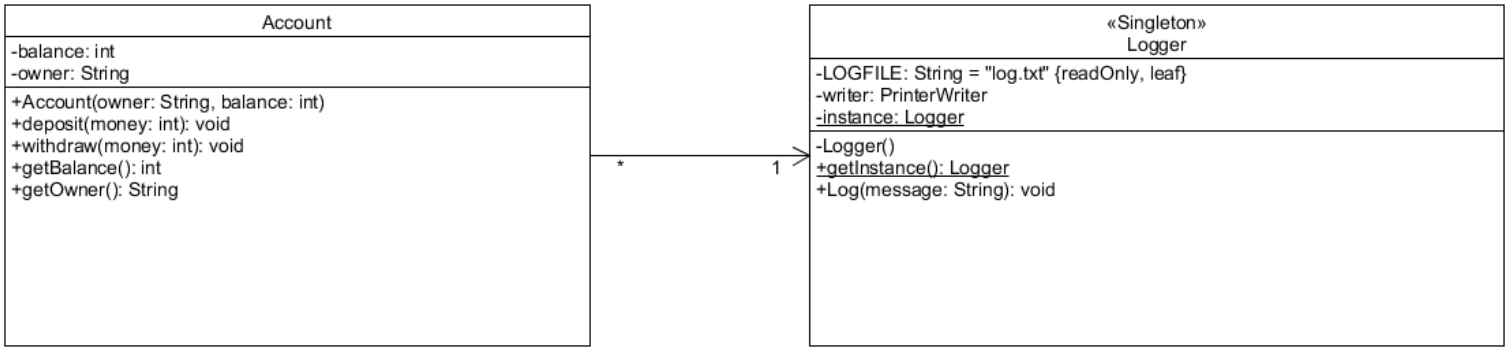
```
public class Account {
    private String owner;
    private int balance;
    private Logger myLogger;

    public Account(String owner, int balance) {
        this.owner = owner;
        this.balance = balance;
    }

    public String getOwner() { return owner; }
    public int getBalance() { return balance; }

    public void deposit(int money) {
        myLogger.log("owner" + " : " +this.getOwner() + " deposit " + money);
        balance += money;
    }
    public void withdraw(int money) {
        if (balance >= money) {
            myLogger.log("owner" + " : " +this.getOwner() + " withdraw " + money);
            balance -= money;
        }
    }
    public void setMyLogger(Logger myLogger) {
        this.myLogger = myLogger;
    }
}
```

Main 클래스-revised

```
public class Main {
    public static void main(String[] args) {
        Logger logger = new Logger();
        Account acct1 = new Account("insang1", 1000000);
        acct1.setMyLogger(logger);
        acct1.deposit(20000);
        Account acct2 = new Account("insang2", 2000000);
        acct2.setMyLogger(logger);
        acct2.withdraw(5000);
    }
}
```

문제점


Account
1
1
Logger


Account
-balance: int
-owner: String
+Account(owner: String, balance: int)
+deposit(money: int): void
+withdraw(money: int): void
+getBalance(): int
+getOwner(): String
*
1
«Singleton»
Logger
-LOGFILE: String = "log.txt" {readOnly, leaf}
-writer: PrinterWriter
-instance: Logger
-Logger()
+getInstance(): Logger
+Log(message: String): void

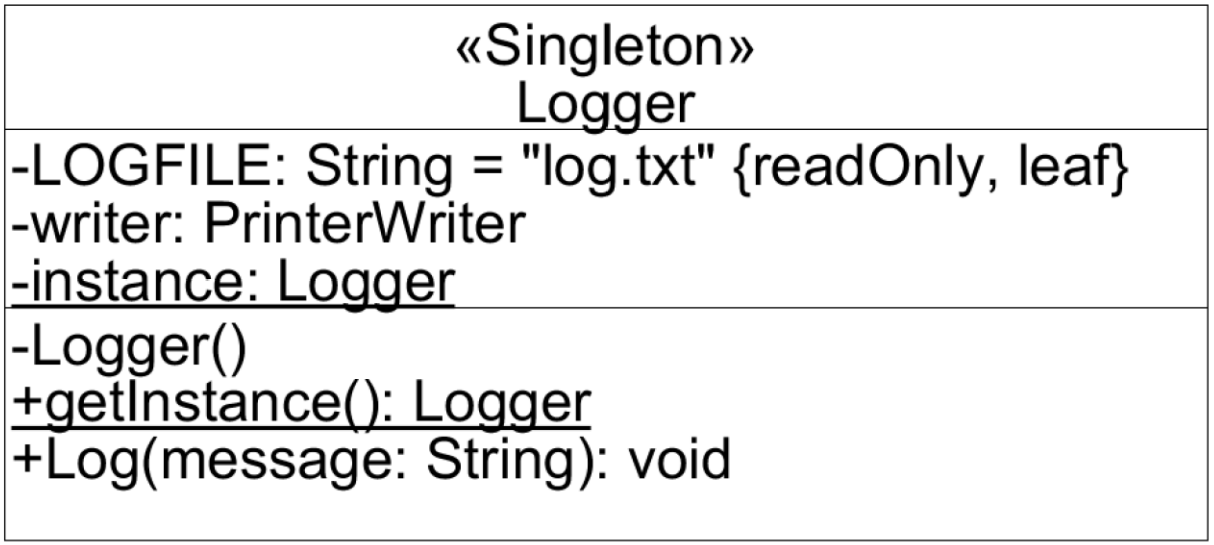
문제점

- 여러 Logger 인스턴스를 생성하지 못하게 하는 수단을 제공하지 못한다.
- 외부에서 제공받아야 한다.
- 예

```
public class Main {
    public static void main(String[] args) {
        Logger logger1 = new Logger();
        Account acct1 = new Account("insang1", 1000000);
        acct1.setMyLogger(logger1);
        acct1.deposit(20000);
        Account acct2 = new Account("insang2", 2000000);
        Logger logger2 = new Logger();
        acct2.setMyLogger(logger2);
        acct2.withdraw(5000);
    }
}
```

해결책

❖ 클래스가 하나의 인스턴스만을 가지도록 설계



- static 변수 instance 선언
- 생성자를 private으로
- Logger의 (유일한) 인스턴스를 생성 및 반환하는 getInstance() 메소드 정의

Eager Initialization: Logger 클래스

```
public class Logger {
    private final String LOGFILE = "log.txt";
    private PrintWriter writer;
    private static Logger instance = new Logger();
    private Logger() {
        try {
            FileWriter fw = new FileWriter(LOGFILE);
            writer = new PrintWriter(fw, true);
        } catch (IOException e) {}
    }

    public static Logger getInstance() { return instance; }
    public void log (String message) {
        SimpleDateFormat formatter= new SimpleDateFormat("yyyy-MM-dd 'at' HH:mm:ss z");
        Date date = new Date(System.currentTimeMillis());
        writer.println(formatter.format(date) + " : " + message);
    }
}
```

Account 클래스

```
public class Account {
    private String owner;
    private int balance;
    private Logger myLogger;

    public Account(String owner, int balance) {
        this.owner = owner;
        this.balance = balance;
        this.myLogger = Logger.getInstance();
    }

    public String getOwner() { return owner; }
    public int getBalance() { return balance; }

    public void deposit(int money) {
        myLogger.log("owner" + " : " +this.getOwner() + " deposit " + money);
        balance += money;
    }
    public void withdraw(int money) {
        if (balance >= money) {
            myLogger.log("owner" + " : " +this.getOwner() + " withdraw " + money);
            balance -= money;
        }
    }

}
```

Main 클래스

```
public class Main {
    public static void main(String[] args) {
        Account acct1 = new Account("insang1", 1000000);
        acct1.deposit(20000);
        Account acct2 = new Account("insang2", 2000000);
        acct2.withdraw(5000);
    }
}
```

- 실행결과

```
2020-07-21 at 11:32:25 KST : owner : insang1 deposit 20000
2020-07-21 at 11:32:25 KST : owner : insang2 withdraw 5000
```

실습

```
public class Main {
    public static void main(String[] args) {
        Account acct1 = new Account("insang1", 1000000);
        acct1.deposit(20000);
        Account acct2 = new Account("insang2", 2000000);
        acct2.withdraw(5000);
        acct2.withdraw(3000);
        acct1.withdraw(5000);
    }
}
```

- 실행결과

```
2020-07-21 at 11:38:05 KST : owner : insang1 deposit 20000
2020-07-21 at 11:38:05 KST : owner : insang2 withdraw 5000
2020-07-21 at 11:38:05 KST : owner : insang2 withdraw 3000
2020-07-21 at 11:38:05 KST : owner : insang1 withdraw 5000
```

문제점

- 클래스 로딩 시점에 초기화되어 인스턴스가 필요하지 않는 경우에도 생성
- 해결책
  - Lazy Initialization
  - 인스턴스가 필요할 때 생성

Lazy Initialization

```
public class Logger {
    private final String LOGFILE = "log.txt";
    private PrintWriter writer;
    private static Logger instance;
    private Logger() {
        try {
            FileWriter fw = new FileWriter(LOGFILE);
            writer = new PrintWriter(fw, true);
        } catch (IOException e) {}
    }

    public static Logger getInstance() {
        if (instance == null) instance = new Logger();
        return instance;
    }
    public void log (String message) {
        SimpleDateFormat formatter= new SimpleDateFormat("yyyy-MM-dd 'at' HH:mm:ss z");
        Date date = new Date(System.currentTimeMillis());
        writer.println(formatter.format(date) + " : " + message);
    }
}
```

다중 쓰레드 환경

```
public class User extends Thread {
    public User(String name) { super(name); }
    public void run() {
        Random r = new Random();
        Account acct = new Account(Thread.currentThread().getName(), r.nextInt(1000000));
        if (r.nextBoolean()) acct.withdraw(r.nextInt(acct.getBalance()));
        else acct.deposit(r.nextInt(acct.getBalance()));
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        User[] users = new User[10];
        for (int i = 0; i < 10; i++) {
            users[i] = new User("insang"+i);
            users[i].start();
        }
    }
}
```

실행결과

```
2020-07-21 at 12:22:20 KST : owner : insang7 withdraw 177027
2020-07-21 at 12:22:20 KST : owner : insang7 deposit 3855
```

```
public class Logger {
    private final String LOGFILE = "log.txt";
    private PrintWriter writer;
    private static Logger instance;
    private Logger() {
        try {
            FileWriter fw = new FileWriter(LOGFILE);
            writer = new PrintWriter(fw, true);
        } catch (IOException e) {}
    }

    public static Logger getInstance() {
        if (instance == null) instance = new Logger();
        return instance;
    }
    public void log (String message) {
        System.out.println(this.toString());
        SimpleDateFormat formatter= new SimpleDateFormat("yyyy-MM-dd 'at' HH:mm:ss z");
        Date date = new Date(System.currentTimeMillis());
        writer.println(formatter.format(date) + " : " + message);
    }
}
```

```
Logger@368f0890
Logger@5a6b83b0
Logger@40147f91
Logger@37ce62b5
Logger@91b17d7
Logger@91b17d7
Logger@21eca24f
Logger@692db0ef
Logger@13f938d6
Logger@5ab1f63f
Logger@4831b27e
```

문제점

- 인스턴스가 여러 개 생긴다.
- 시나리오 (쓰레드 경합)
  1. Logger 인스턴스가 아직 생성되지 않았을 때 스레드 1이 getInstance 메서드의 if문을 실행해 이미 인스턴스가 생성되었는지 확인한다. 현재 instance 변수는 null인 상태다.
  2. 만약 스레드 1이 생성자를 호출해 인스턴스를 만들기 전 스레드 2가 if문을 실행해 instance 변수가 null인지 확인한다. 현재 null이므로 인스턴스를 생성하는 코드, 즉 생성자를 호출하는 코드를 실행하 게 된다.
  3. 스레드 1도 스레드 2와 마찬가지로 인스턴스를 생성하는 코드를 실행하게 되면 결과적으로 instance 클래스의 인스턴스가 2개 생성된다.

해결책: 동기화

❖ Synchronized로 동기화하는 방법

```
public class Logger {
    private final String LOGFILE = "log.txt";
    private PrintWriter writer;
    private static Logger instance;
    private Logger() {
        try {
            FileWriter fw = new FileWriter(LOGFILE);
            writer = new PrintWriter(fw, true);
        } catch (IOException e) {}
    }

    public synchronized static Logger getInstance() {
        if (instance == null) instance = new Logger();
        return instance;
    }
    public void log (String message) {
        System.out.println(this.toString());
        SimpleDateFormat formatter= new SimpleDateFormat("yyyy-MM-dd 'at' HH:mm:ss z");
        Date date = new Date(System.currentTimeMillis());
        writer.println(formatter.format(date) + " : " + message);
    }
}
```

실행 결과

```
2020-07-21 at 12:36:29 KST : owner : insang9 withdraw 641171
2020-07-21 at 12:36:29 KST : owner : insang7 withdraw 251924
2020-07-21 at 12:36:29 KST : owner : insang8 withdraw 54978
2020-07-21 at 12:36:29 KST : owner : insang4 deposit 166095
2020-07-21 at 12:36:29 KST : owner : insang0 withdraw 346992
2020-07-21 at 12:36:29 KST : owner : insang9 deposit 155343
2020-07-21 at 12:36:29 KST : owner : insang2 withdraw 53567
2020-07-21 at 12:36:29 KST : owner : insang7 deposit 37570
2020-07-21 at 12:36:29 KST : owner : insang2 deposit 1543
2020-07-21 at 12:36:29 KST : owner : insang6 withdraw 444378
2020-07-21 at 12:36:29 KST : owner : insang3 withdraw 48199
2020-07-21 at 12:36:29 KST : owner : insang6 deposit 44340
2020-07-21 at 12:36:29 KST : owner : insang5 withdraw 401474
2020-07-21 at 12:36:29 KST : owner : insang8 deposit 214776
2020-07-21 at 12:36:29 KST : owner : insang3 deposit 58173
2020-07-21 at 12:36:29 KST : owner : insang0 deposit 27950
2020-07-21 at 12:36:29 KST : owner : insang1 withdraw 116286
2020-07-21 at 12:36:29 KST : owner : insang5 deposit 404652
2020-07-21 at 12:36:29 KST : owner : insang1 deposit 260831
```

```
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
Logger@3bea1121
```

DCL(Double Checked Locking)

- Synchronized를 이용하는 방식은 성능적으로 효율적이지 않다.

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

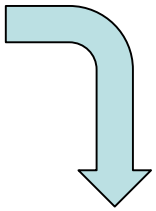
실행 결과

```
2020-07-21 at 12:53:27 KST : owner : insang1 withdraw
313279
2020-07-21 at 12:53:27 KST : owner : insang8 deposit 349798
2020-07-21 at 12:53:27 KST : owner : insang6 deposit 50237
2020-07-21 at 12:53:27 KST : owner : insang4 deposit 125567
2020-07-21 at 12:53:27 KST : owner : insang0 withdraw 61037
2020-07-21 at 12:53:27 KST : owner : insang2 deposit 25390
2020-07-21 at 12:53:27 KST : owner : insang9 withdraw 85186
2020-07-21 at 12:53:27 KST : owner : insang7 withdraw
801482
2020-07-21 at 12:53:27 KST : owner : insang0 deposit 78802
2020-07-21 at 12:53:27 KST : owner : insang1 deposit 109310
2020-07-21 at 12:53:27 KST : owner : insang3 withdraw 17105
2020-07-21 at 12:53:27 KST : owner : insang5 deposit 3950
2020-07-21 at 12:53:27 KST : owner : insang9 deposit 332917
2020-07-21 at 12:53:27 KST : owner : insang7 deposit 3542
2020-07-21 at 12:53:27 KST : owner : insang3 deposit 16960
```

```
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
Logger@5746ac53
```

Looks good, 그러나~

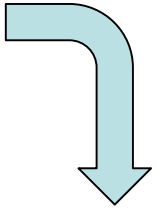
```
public static Logger getInstance() {
        if (instance == null) {
            synchronized(Logger.class) {
                if (instance == null)
                    instance = new Logger();
            }
        }
        return instance;
}
```



1. Logger 인스턴스를 위한 메모리 할당
2. 생성자를 통한 초기화
3. 할당된 메모리를 instance 변수에 할당

명령어 reorder를 통한 최적화 수행

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



1. Logger 인스턴스를 위한 메모리 할당
2. 할당된 메모리를 instance 변수에 할당
3. 생성자를 통한 초기화

문제점

- 미완성 인스턴스를 사용
- 시나리오
  1. Logger 인스턴스가 아직 생성되지 않았을 때 쓰레드 1이 getInstance 메서드의 첫번째 if문을 실행해 이미 인스턴스가 생성되었는지 확인한다. 현재 instance 변수는 null인 상태다.
  2. Logger 인스턴스를 위한 메모리 할당
  3. 할당된 메모리를 instance 변수에 할당
  4. 쓰레드 2가 getInstance 메소드의 첫번째 if문 실행하여 instance 변수가 성정된 것을 확인하고 생성된(그러나 아직 초기화 되지 않은) 인스턴스를 반환 받음
  5. 쓰레드 2가 로깅하려고 하면 문제가 발생

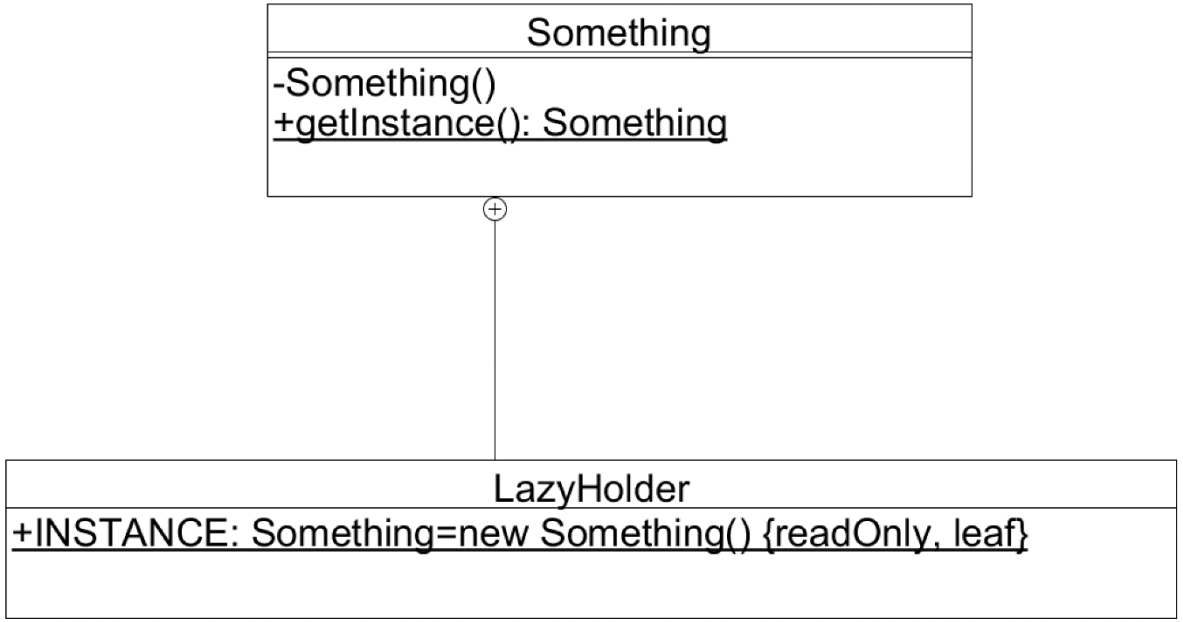
해결책

- Volatile를 사용하여 명령어 reorder 금지

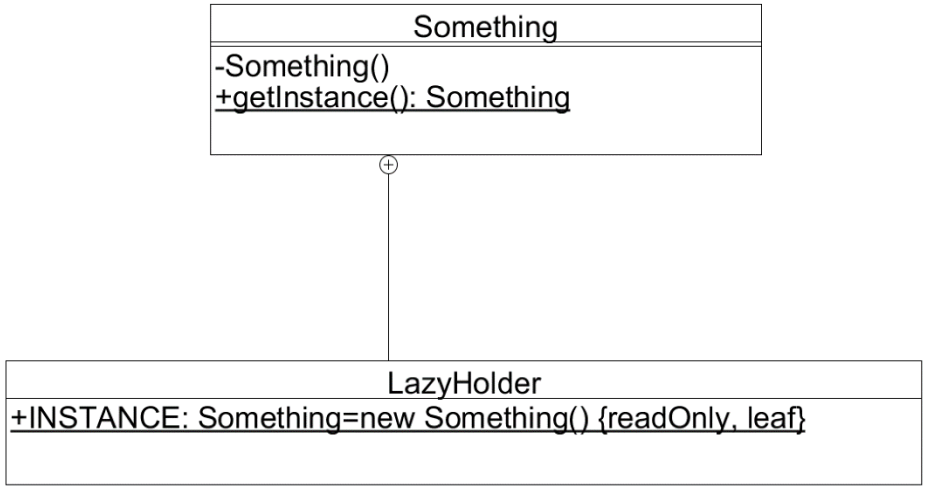
```
public class Logger {
    private final String LOGFILE = "log.txt";
    private PrintWriter writer;
    private volatile static Logger instance;
    private Logger() {
        try {
            FileWriter fw = new FileWriter(LOGFILE);
            writer = new PrintWriter(fw, true);
        } catch (IOException e) {}
    }
    …
}
```

Initialization on demand holder idiom

- Demand(또는 Lazy) Holder 방식은 가장 많이 사용되는 싱글턴 구현 방식.
- volatile 이나 synchronized 키워드 없이도 동시성 문제를 해결



Initialization on demand holder idiom




```
public class Something {
    private Something() {
    }

    private static class LazyHolder {
        public static final Something INSTANCE = new Something();
    }

    public static Something getInstance() {
        return LazyHolder.INSTANCE;
    }
}
```

Logger 클래스

```
public class Logger {
    private final String LOGFILE = "log.txt";
    private PrintWriter writer;
    private static Logger instance;
    private Logger() {
        try {
            FileWriter fw = new FileWriter(LOGFILE);
            writer = new PrintWriter(fw, true);
        } catch (IOException e) {}
    }

    private static class LazyHolder {
        public static final Logger INSTANCE = new Logger();
    }

    public static Logger getInstance() { return LazyHolder.INSTANCE; }

    public void log (String message) {
        System.out.println(this.toString());
        SimpleDateFormat formatter= new SimpleDateFormat("yyyy-MM-dd 'at' HH:mm:ss z");
        Date date = new Date(System.currentTimeMillis());
        writer.println(formatter.format(date) + " : " + message);
    }
}
```