- 특정 클래스에 대해서 객체 인스턴스가 하나만 만들어 질 수 있도록 해주는 패턴
- 전역 변수를 사용할 때와 마찬가지로 객체 인스턴스를 어디서든지 액세스가능
- 처음부터 메모리에 올려두지 않고 필요할 때만 객체를 만들 수 있음.

고전적인 구현법

```
public class Singleton {
    private static Singleton uniqueInstance;
    Singleton 클래스에서만 클래스 인스턴스를 만들수 있음
    private Singleton() {}
                synchronized 를 추가하면 쓰레드안정. 한 스레드가 사용을 끝낼 때까지 다른쓰레드는 기다려야함.
    public static Singleton getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }
    // 기타 메소드
}
```

필요한 순간에 (getInstance()가 불리는 순간에) 생성

lazy initialization

↳ 클래스가 자원을 많이 차지하는 경우 유용

사용처

- 레지스트리 설정, 연결 풀, 스레드 풀

싱글턴 패턴 정의

: 해당 클래스의 인스턴스가 하나만 만들어지고, 어디서든지 그 인스턴스에 접근할 수 있도록 하기위한 패턴.

동기화 문제

효율저하!

- synchronized를 추가하면 해결.
- 사실 동기화가 꼭 필요한 시점은 이 메소드가 시작될때 뿐! (아직 인스턴스가 생성되지 않았을 때)
- 메서드를 동기화하면 효율이 100배정도 저하됨

동기화 문제 해결법

- 정적 초기화 부분에서 lazy initialization하지 않고 바로생성하는 것도 방법!

```
private static Singleton uniqueInstance = new Singleton();
```

↳ JVM에서 유일한 인스턴스를 생성하기 전에는 그 어떤 스레드도 uniqueInstance 정적 변수에 접근할 수 없음.

- DCL (Double-checking Locking)을 써서 getInstance()에서 동기화되는 부분 줄이기 오버헤드 줄임!!

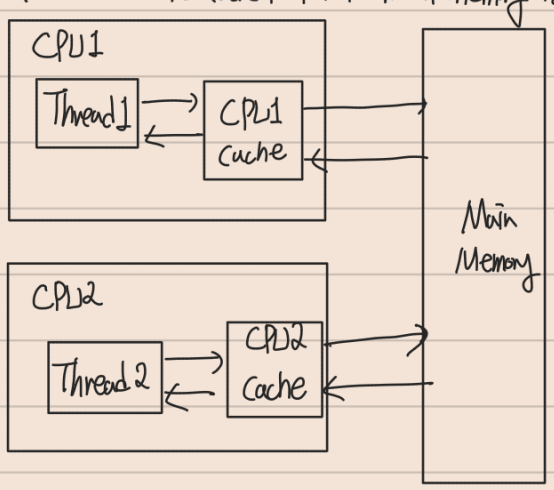
```
private volatile static Singleton uniqueInstance;
                ↳ multi-threading 쓰더라도 초기화 과정 올바르게 진행

public static Singleton getInstance() {
    if (uniqueInstance == null) {
        synchronized (Singleton.class) {
            if (uniqueInstance == null) {
                uniqueInstance = new Singleton();
            }
        }
```

처음에만 동기화 됨!

Volatile

- Java 변수를 Main memory에 저장하겠다고 명시하는 것
- read/write를 CPU cache가 아니라 main memory에!



Multi-thread 환경에서 Thread가 변수 값을 읽어올 때 CPU cache에 저장된 값이 다르다면 문제발생

- Multi Thread 환경에서 하나의 Thread만 read & write하고 나머지 Thread가 read하는 상황에서 가장 최신 값 보장.
  ↳ 여러 thread가 write하는 상황이라면 synchronized를 통해 read & write의 원자성을 보장해야 한다.

Q. 모든 메소드와 변수가 static으로 선언된 클래스?

- 자바에서 정적 초기화를 처리하는 방법 때문에 문제가 생길 수 있음. 초기화 순서와 관련된 문제는 복잡·미묘

- 클래스로더가 여러개라면 싱글턴 인스턴스가 여러개 만들어질 수 있음. 클래스로더를 직접지정해서 회피가능!
- 자바 1.2 이상이라면 가비지 컬렉터가 잡아먹는 버그 X
- private 생성자를 가진 Class는 확장할 수 없음

전역변수

- 기본적으로 객체에 대한 정적 레퍼런스
- lazy Initialization 불가
- 처음부터 끝까지 인스턴스를 가지고 있어야 함
- 간단한 객체에 대한 전역 레퍼런스를 자꾸 만들게 되어 네임 스페이스를 지저분하게 만드는 경향