고급객체지향 프로그래밍
강의노트 #07

Singleton Pattern

조용주
ycho@smu.ac.kr

싱글턴 패턴 (Singleton Pattern)

- 목적
  - Ensure a class only has one instance, and provide a global point of access to it.
  - 클래스가 한 개의 인스턴스만을 만들 수 있도록 하고, 어디서나 생성된 인스턴스에 접근할 수 있도록 함

디자인 패턴 요소

요소	설명
이름	싱글턴 (Singleton)
문제	여러 객체가 생성되면 상태 관리가 어려움
해결방안	객체 생성자를 중앙 관리
결과	객체가 1 개라서 일관된 상태

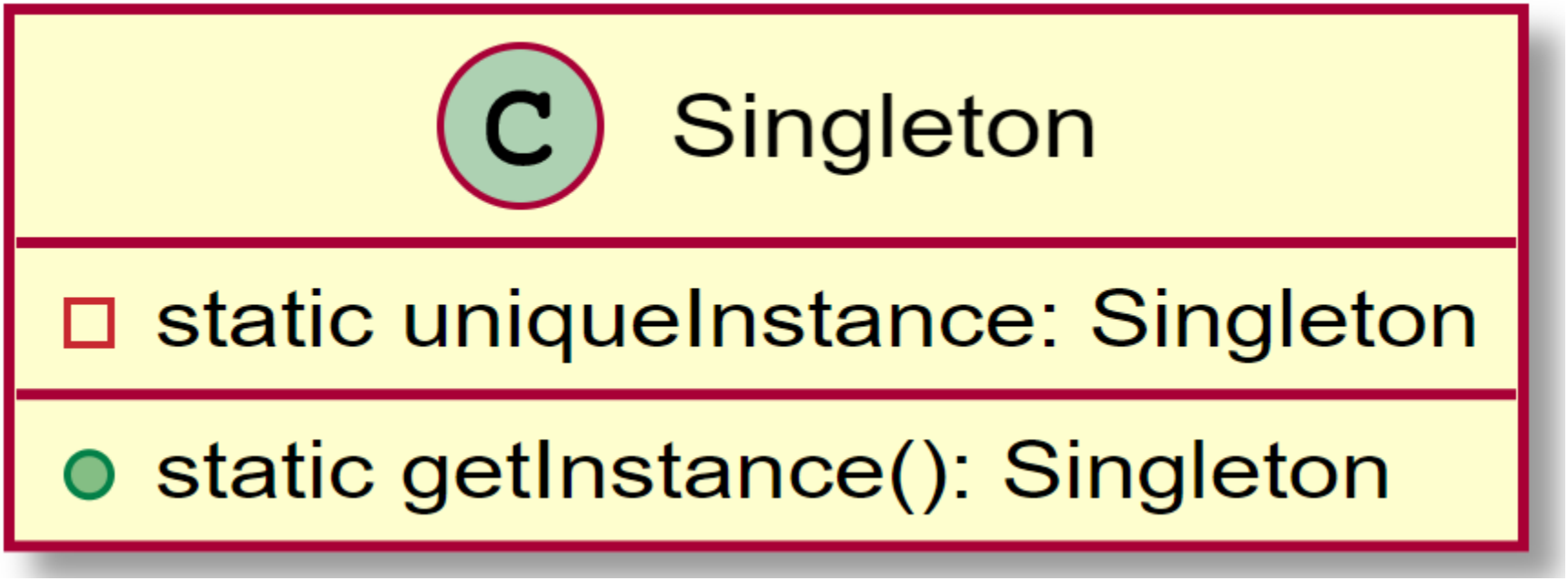
고전적 싱글턴 패턴 구현법

```
public class Singleton {
  // Singleton  클래스의 유일한 인스턴스를 저장
  private static Singleton uniqueInstance;
  //  기타 멤버 변수
  private Singleton() { }

  public static Singleton getInstance() {
    if (uniqueInstance == null) {
      uniqueInstance = new Singleton();
    }
    return uniqueInstance;
  }
  //  기타 메소드
}
```

고전적 싱글턴 패턴 구현법

- 클래스 다이어그램



사례 1 – 초콜릿 공장

- 초콜릿 공장에서 초콜릿을 끓이는 장치 (초콜릿 보일러) 를 컴퓨터로 제어함
- 이 보일러는 초콜릿과 우유를 받아서 끓이고 초코바를 만드는 단계로 넘겨줌
- 초코홀릭 (Choc-O-Holic) 사의 최신형 보일러를 제어하는 클래스를 보임
  - 500 갤런의 재료를 그냥 흘려 버리거나 보일러가 가득 차 있는 상태에서 새로운 원료를 붓는다거나 빈 보일러에 불을 지핀다거나 하는 실수를 하지 않도록 주의를 기울인 것을 확인할 수 있음

사례 1 – 초콜릿 공장

```
public class ChocolateBoiler {
  private boolean empty;
  private boolean boiled;
  public ChocolateBoiler() {
    empty = true; //  보일러가 비어있을 때만  동작
    boiled = false;
  }
  public boolean isEmpty() {
    return empty;
  }
  public boolean isBoiled() {
    return boiled;
  }
  public void fill() {
    if (isEmpty()) { //  비어있을 때에만 재료 넣음
      empty = false;
      boiled = false;
      //  보일러에 우유 / 초콜릿 혼합 재료 넣음
  }
```

사례 1 – 초콜릿 공장

```
  // 보일러가 가득 차있고, 다 끓여진 상태에서만 보일러
  // 에 있는 재료를 다음 단계로 넘기고 보일러를 비움
  public void drain() {
    if (!isEmpty() && isBoiled()) {
      // 끓인 재료를 다음 단계로 넘김
      empty = true;
    }
  }
  // 보일러가 가득 차있고, 아직 끓지 않은 상태면 끓임
  public void boil() {
    if (!isEmpty() && !isBoiled()) {
      // 재료를 끓임
      boiled = true;
    }
  }
}
```

싱글턴 버전의 초콜릿 보일러 코드

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

  private ChocolateBoiler() {
    empty = true; //  보일러가 비어있을 때만  동작
    boiled = false;
  }

  public static ChocolateBoiler getInstance() {
    if (uniqueInstance == null) {
      uniqueInstance = new ChocolateBoiler();
    }
    return uniqueInstance;
  }
  //  나머지 멤버 함수 코드
}
```

Thread-safe 버전의 싱글턴

- 여러 개의 스레드에서 앞에서 작성한 코드가 사용되면 문제가 발생할 수 있음
- 이를 해결하려면 getinstance() 함수에 동기화 시키는 코드를 넣어야 함

```
public class Singleton {
  private static Singleton uniqueInstance;

  private Singleton() { }
  public static synchronized Singleton getInstance() {
    if (uniqueInstance == null) {
      uniqueInstance = new Singleton();
    }
    return uniqueInstance;
  }
  //  나머지 멤버 함수 코드
}
```

Thread-safe 버전의 싱글턴

- 문제 ?
  - 비효율적일 수 있음 (느려질 수 있음)
- 해결 방법
  - getInstance() 의 속도가 크게 영향 미칠 정도가 아니면 그냥 둠
  - 인스턴스를 필요할 때 생성하지 말고 , 프로그램 시작될 때 생성

```
public class Singleton {
   private static Singleton inst = new Singleton();
   private Singleton() { }
   public static Singleton getInstance() {
     return inst;
   }
   //  나머지 멤버 함수 코드
}
```

Thread-safe 버전의 싱글턴

- DCL(Double-checking Locking)을 사용해서 getInstance() 함수에서 동기화되는 부분을 줄이기 (Java5 이후에서만 사용 가능)
  - 인스턴스가 생성되어 있는지 확인 후, 생성되어 있지 않았을 때만 동기화를 시킬 수 있음

Thread-safe 버전의 싱글턴

```
public class Singleton {
  private volatile static Singleton inst;
  private Singleton() { }
  public static Singleton getInstance() {
    if (inst == null) {
      synchronized (Singleton.class) {
        if (inst == null) {
          inst = new Singleton();
        }
      }
    }
    return inst;
  }
  //  나머지 멤버 함수 코드
}
```

디자인 패턴 요소

요소	설명
이름	싱글턴 (Singleton)
문제	여러 객체가 생성되면 상태 관리가 어려움
해결방안	객체 생성자를 중앙 관리
결과	객체가 1 개라서 일관된 상태

고전적 싱글턴 패턴 구현법

- 클래스 다이어그램