1.

ChocolateBoiler 클래스는 수업시간에 다뤘던 What 클래스와는 다르게 생성자가 public이기 때문에 하위클래스를 생성하는 것이 가능하다. 하지만 만약 생성자를 private으로 구현한다면 extends를 통해 싱글톤 클래스를 상속 받는 것이 아니라 싱클톤 객체를 속성으로 갖는 추상클래스를 작성하고 하위클래스로 이를 상속받아 구현할 수 있다. 모든 하위클래스는 상속을 통해 상위 클래스의 유일한 싱글톤 객체를 갖으면서도 추상클래스를 구체화 함으로서 다형성을 갖을 수 있다.

예를 들어 초콜릿 공장이 있고 해당 공장에서 초콜렛을 녹이고 배합하면 이를 도매로 떼어다가 자신들이 원하는 틀에 넣고 굳혀서 파는 가게들이 있다고 가정하자. 해당 초콜릿 공장과 계약을 맺은 가게는 이 초콜릿 공장의 유일한 ChocolateBoiler 를 사용해 초콜릿을 녹이고 배합한다. 그리고 각자 자신의 가게로 녹인 초콜릿을 가져와 각 가게에서 원하는 모양으로 틀에 초콜릿을 넣고 다시 굳혀 판매하는 것이다. 이로서 하나의 ChocolateBoiler 를 사용한다는 싱글톤을 만족하면서 가게마다 자신의 초콜렛 틀을 사용해 서로 다른 모양의 초콜릿을 팔 수 있다는 다형성을 만족하게 된다. 이를 구현해보기 위해 아래와 같은 간단한 프로그램을 작성해보았다.

```
package singletone;

public abstract class ChocolateStore {

   ChocolateBoiler boilerSystem = ChocolateBoiler.getInstance(); //싱글톤 객체이므로 반드시 하나만 존재


   abstract void getChoco();      //ChocolateBoiler 에서 store 마다 필요한 양의 초콜렛 가져옴
   abstract void pourChoco();    //틀에 초콜릿 부음(store 마다 틀모양 다름)
   abstract void freezeChoco();  // 초콜릿 굳힘(store 에서 사용하는 틀마다 굳히는 시간 다름


}
```

먼저 ChocolateStore 라는 추상클래스를 만든다. 이 클래스에는 싱글톤으로 구현했던 ChocolateBoiler 인스턴스객체가 선언되어 있다. 그리고 3가지 추상메소드가 선언 되어있다. 각 메소드는 ChocolateBoiler에서 필요한 양의 초콜릿을 가져오고, 틀에 초콜릿을 붓고, 얼리는 동작을 구현하게 될 것이다.

```
package singletone;

public class StarChocolateStore extends ChocolateStore{

    void getChoco(){
        System.out.println("Get Chocolate 500L at " + boilerSystem +
".\n" );
    }

    void pourChoco(){
        System.out.println("Pour Chocolate in Star Shape Mold.\n");
    }
    void freezeChoco(){
        System.out.println("Freeze Chocolate 10 hour.\n");

    }

}
```

ChocolateStore를 상속받은 하위클래스이다. 이 가게는 별모양 초콜릿을 판매한다. 각 추상메소드는 별모양 초콜릿을 만드는 이 가게에 맞게 구현되어 있다. 이 가게는 별모양 초콜릿을 만들기 위해 초콜릿 보일러에서 500L의 초콜릿을 가져오고, 별모양 틀에 부어서 10시간 가량 얼린다.

```
package singletone;

public class SquareChocolateStore extends ChocolateStore {

    void getChoco(){
        System.out.println("Get Chocolate 1000L at " + boilerSystem +
".\n" );
    }

    void pourChoco(){
        System.out.println("Pour Chocolate in Square Shape Mold.\n");
    }
    void freezeChoco(){
        System.out.println("Freeze Chocolate 15 hour.\n");
    }

}
```

ChocolateStore를 상속받은 두번째 하위클래스이다. 이 가게는 네모 모양 초콜릿을 만들기 위해 초콜릿 보일러에서 1000L의 초콜릿을 가져오고, 네모 모양 틀에 부어서 15시간 가량 얼린다.

```
package singletone;

public class ChocolateController {
    public static void main(String args[]) {


        StarChocolateStore store1 = new StarChocolateStore();
        SquareChocolateStore store2 = new SquareChocolateStore();

        store1.getChoco();
```

```
        store1.pourChoco();
        store1.freezeChoco();
        System.out.println();
        store2.getChoco();
        store2.pourChoco();
        store2.freezeChoco();


    }
}
```

```
Get Chocolate 500L at singletone.ChocolateBoiler@3b07d329.

Pour Chocolate in Star Shape Mold.

Freeze Chocolate 10 hour.


Get Chocolate 1000L at singletone.ChocolateBoiler@3b07d329.

Pour Chocolate in Square Shape Mold.

Freeze Chocolate 15 hour.
```

다음은 main 함수에서 각 가게에 대한 인스턴스 객체를 만들고 각 메소드를 호출한 후 실행 결과이다. 서로 다른 가게가 생성되고 각자 다른 양, 다른 모양, 다른 시간 을 사용하여 초콜릿을 만들지만 초콜릿을 초콜릿 보일러에서 가져올 때 접근하는 초콜릿 보일러 객체는 동일함을 확인할 수 있다.

2.

Runnable 인터페이스를 상속받아 ChocoMaker 를 만들고 run() 함수를 오버라이딩 하여 그 안에 ChocolateBoiler.getInstance()를 호출한다. mian 함수에서는 ChocoMaker 객체를 만들고 이 객체를 인스턴스로 받는 스레드 2개를 생성하여 멀티스레드를 구현하다. 각 스레드의 run()이 실행되면서 ChocolateBoiler.getInstance()를 호출하게 되는데 싱글톤 패턴은 오로지 ChocolateBoiler 객체 하나만 생성되어야 한다. 하지만 다음과 같은 멀티스레드 실행결과 2개의 ChocolateBoiler 인스턴스 객체가 반환되는 것을 확인할 수 있었다.

```
package singletone;
```

```java
public class ChocoMaker implements Runnable{

    @Override
    public void run() {
        ChocolateBoiler.getInstance();

    }

}
```

```java
package singletone;

public class ChocolateController {
    public static void main(String args[]) {
        //ChocolateBoiler boiler = ChocolateBoiler.getInstance();
        //boiler.fill();
        //boiler.boil();
        //boiler.drain();

        // will return the existing instance
        //ChocolateBoiler boiler2 = ChocolateBoiler.getInstance();

        ChocoMaker maker = new ChocoMaker();    //Runnable 구현 객체
        Thread makeThread = new Thread(maker);  //Thread 생성
        Thread makeThread2 = new Thread(maker); //Thread 생성


        makeThread.start();
        makeThread2.start();


    }
}
```

```
singletone.ChocolateBoiler@198d03df
singletone.ChocolateBoiler@12dde2fc

Process finished with exit code 0
```

[멀티스레드 실행결과]

이를 해결하기 위해 ChocolateBoiler.getInstance()를 동기화 시켜주면 오롯이 1개의 ChocolateBoiler 인스턴스 객체가 반환되는 것을 확인할 수 있다.

```java
public static synchronized ChocolateBoiler getInstance(){ 멀티스레드 문제 해결
//synchronized
    if(onlyChoco == null){
        onlyChoco = new ChocolateBoiler();
```

```
    }
    return onlyChoco;
}
```

```
singletone.ChocolateBoiler@4974035b

Process finished with exit code 0
```

[동기화 시켰을 때 멀티스레드 실행 결과]