Singleton

Cel:

Zapewnienie, że klasa ma tylko jedną instancję i dostarczenie globalnego dostępu do niej.

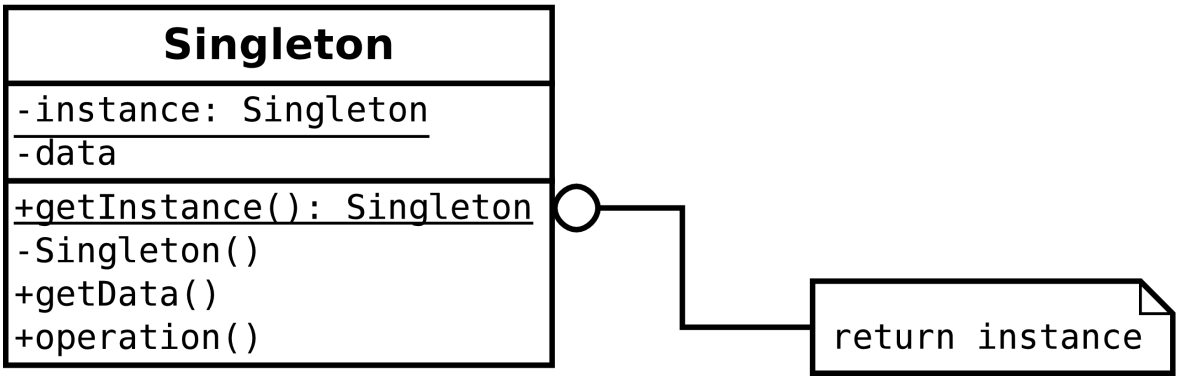
Przykład:

- Niekiedy ważne jest, aby tworzyć tylko jedną instancję jakiejś klasy.
- Globalne zmienne zapewnią dostęp do obiektu, ale nie zapewnią, że będzie to jedyna instancja klasy. Lepiej niech sama klasa pilnuje tworzenia swoich instancji i daje do nich dostęp.

Zastosowanie:

- Gdy musi istnieć dokładnie jedna instancja klasy i musi być ona łatwo dostępna dla klientów.
- Gdy jedyna instancja powinna być rozszerzalna przez dziedziczenie i klienci powinni mieć możliwość jej używania bez modyfikacji swego kodu.

Struktura:




```
public class Singleton {
   private static final Singleton instance = new Singleton();
   private Singleton() { }
   public static Singleton getInstance() {
      return instance;
   }
   // ...
}
```

Składniki:

Singleton

- definiuje operację Instance, która daje klientom dostęp do jej jedynej instancji (operacja klasowa, czyli statyczna metoda)
- może być odpowiedzialny za stworzenie swej jedynej instancji

Zależności:

Klienci mają dostęp do instancji Singletona jedynie przez jego operację getInstance.

Konsekwencje:

1. kontrolowany dostęp do jedynej instancji (ponieważ klasa Singleton zamyka w sobie jedyną instancję, może ona ściśle kontrolować to jak i kiedy klienci do niej sięgają)
2. ograniczenie przestrzeni nazw (ulepszenie względem zmiennych globalnych, zapobiega zaśmiecaniu przestrzeni nazw globalnymi zmiennymi, które miałyby przechowywać jedyną instancję)
3. pozwala na modyfikację operacji i reprezentacji (możemy dziedziczyć z klasy Singletona i udostępnić aplikacji instancję tej rozszerzonej klasy, nawet w czasie wykonania programu)

4. pozwala na zmienną liczbę instancji (możemy kontrolować liczbę instancji klasy zmieniając jedynie metodę getInstance)
5. bardziej elastyczne od operacji klasowych (metody statyczne nie pozwalają na dziedziczenie czy zmianę liczby instancji)

Implementacja

1. Zapewnienie jedynej instancji: konstruktory niedostępne dla klienta.
2. Dostęp do instancji dzięki metodom statycznym.
3. Późna inicjalizacja (zwłaszcza, gdy do jego stworzenia potrzebne dodatkowe informacje):

```
public class Singleton {
   private static final Singleton instance = null;
   private Singleton() { }
   public static Singleton getInstance() {
      if (instance == null)
         instance = new Singleton();
      return instance;
   }
   // ...
}
```

4. ...inne rozwiązanie późnej inicjalizacji:

```
public class Singleton {
   private Singleton() { }
   private static class SingletonHolder {
      static final Singleton INSTANCE = new Singleton();
   }

   public static Singleton getInstance() {
     return SingletonHolder.INSTANCE;
   }
}
```

5. W środowisku wielowątkowym problem wyścigu i niezamierzonego stworzenia kilku obiektów, rozwiązanie:

```
public class Singleton {
    private volatile static Singleton singleton;
    private Singleton() { }

    public static synchronized Singleton getInstance() {
        if(instance==null) {
            instance = new Singleton();

        }
        return instance;
    }
}
```

6. inne, lepsze rozwiązanie:

```
public class Singleton {
    private volatile static Singleton instance;
    private Singleton() { }

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

7. Dziedziczenie z klasy Singletona – zmienna przechowująca instancję klasy Singletona powinna zostać zainicjalizowana instancją jego klasy pochodnej.

Powiązania:

- Wiele innych wzorców projektowych może zostać zaimplementowanych z użyciem Singletona – Abstract Factory, Builder, Prototype.