Singleton

Jedináčik

Zabezpečuje, aby trieda mala iba jednu inštanciu, ku ktorej poskytuje globálny prístupový bod

Ensure a class only has one instance, and provide a global point of acces to it.

Návrhové vzory

Motivácia

- Často sa stáva, že niektoré triedy majú mať práve jednu inštanciu v systéme:
  - Schránka (copy - paste).
  - Súborový systém.
  - Manažér okien.

Aplikovateľnosť

- Jedináčika je výhodné použiť, keď:
  - Je potrebná **práve jedna** inštancia triedy, ktorá musí byť dostupná z jediného prístupového bodu.
  - Keď je možné jedinú inštanciu rozšíriť o potomkov a klienti by mali byť schopní ju použiť bez nutnosti modifikovať jej kód.

Implementácia

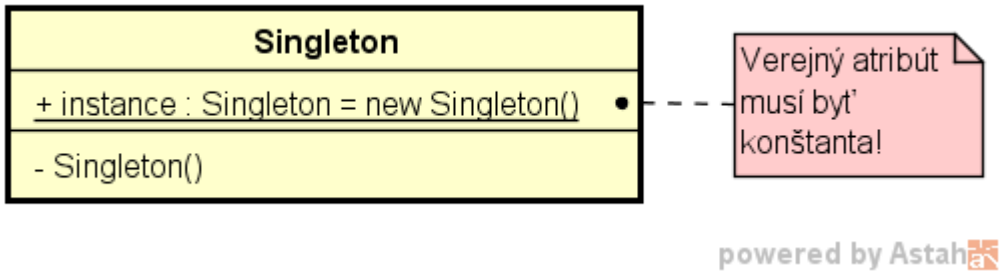
- Existuje niekoľko spôsobov implementácie.
- Podľa spôsobu implementácie sa líši spôsob prístupu, zakaždým je však prístupovým bodom trieda.
- Ako by ste implementovali jedináčika:
  - ktorý nemá potomkov?
  - tak, aby sa vytvorila konkrétna inštancia potomka?
  - aby bola zaručená unikátnosť pri využití paralelných výpočtových prostriedkov?

Implementácia – *Library class*

```
public class Math {
        // Documentation

        public static final double E = 2.71828;
        public static final double PI = 3.14159;

        public static double sin() {}
        public static double cos() {}

        public static double min() {}
        public static double max() {}

        public static double round() {}

}
```

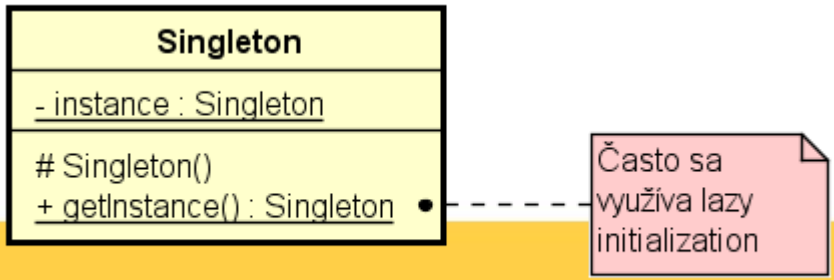
Implementácia – verejná konštanta

```
public class Singleton {

        public static final Singleton instance = new Singleton();

        private Singleton() {
                // initialization
        }
}
```

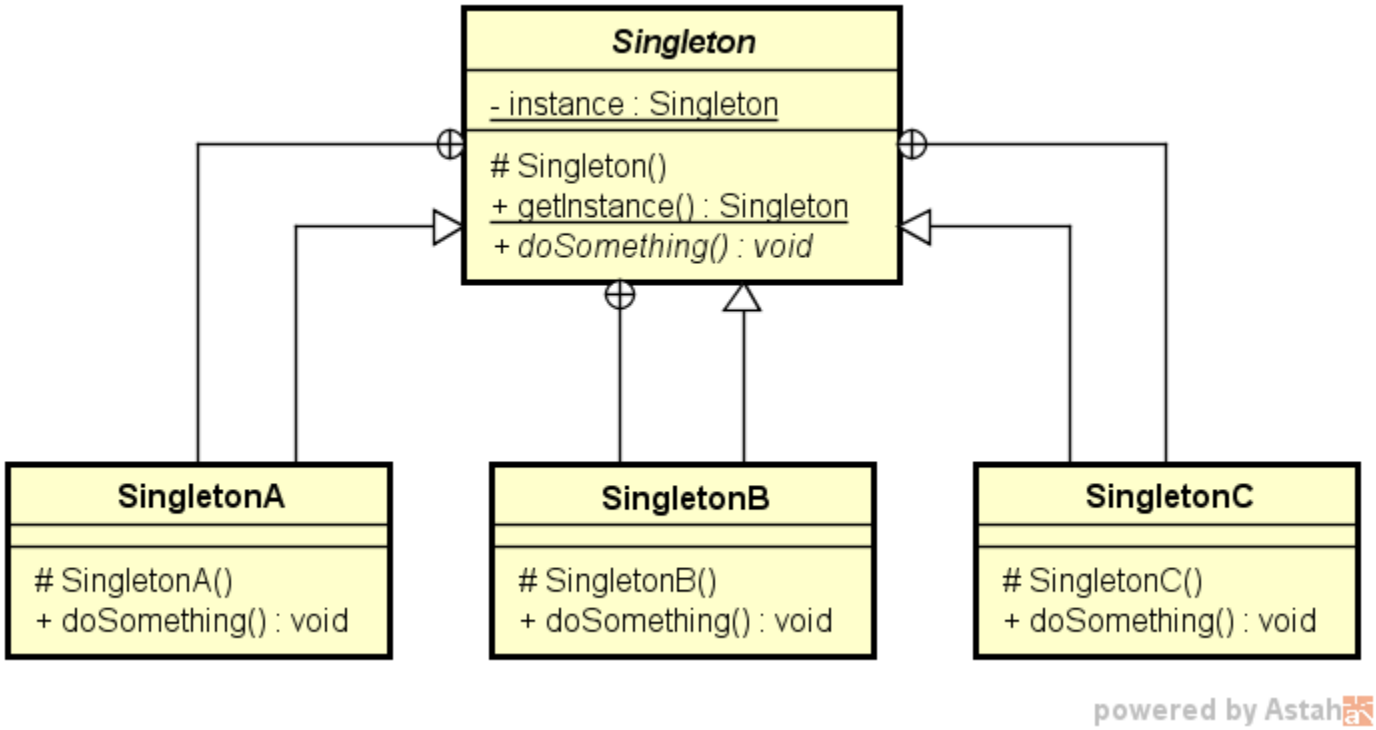


Implementácia – *Lazy initialization*

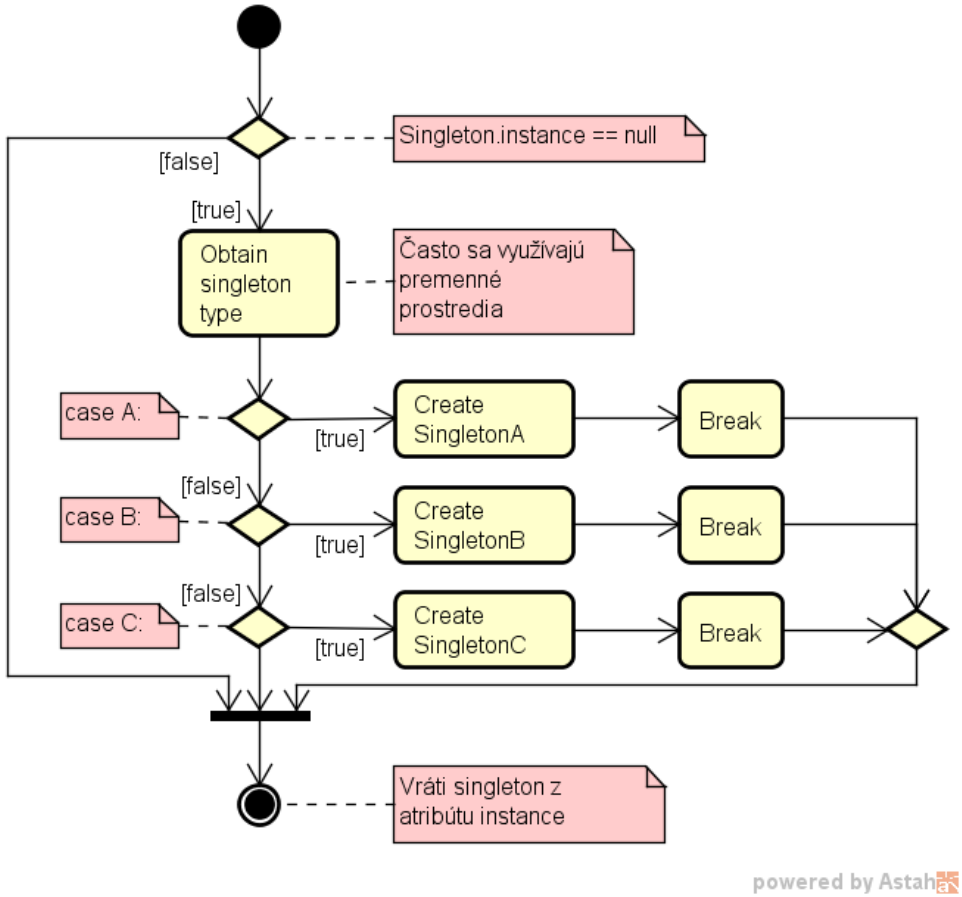
```
public class Singleton {

        private static Singleton instance;

        protected Singleton() {
                // initialization
        }

        public Singleton getInstance() {
                if (instance == null)
                        instance = new Singleton();
                return insatnce;
        }
}
```



Implementácia – niekoľko potomkov



Implementácia – niekoľko potomkov



Implementácia – viacvláknová aplikácia

Vlákno A	Vlákno B
`public Singleton getInstance() {` `  if (instance == null)`	

	`public Singleton getInstance() {`
	`  if (instance == null)`
	`  instance = new Singleton();`
	`  return insatnce;`
	`}`

`  instance = new Singleton();`
`  return insatnce;`
`}`	

Implementácia – viacvláknová aplikácia

```
public class Singleton {

        private static volatile Singleton instance;

        protected Singleton() {
                // initialization
        }

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

Dôsledky

- Kontroluje prístup k inštancii, keďže k nej má priamy prístup.
- Môže definovať niekoľko potomkov, pričom aplikácia sa môže ľahko rozhodnúť, ktorého potomka využije.
- Flexibilnejší ako statická trieda („*Library class*“, „*Utility*“), keďže prípadní potomkovia, ktorí s výhodou uplatnia polymorfizmus.
- Je ho možné relatívne jednoducho rozšíriť tak, aby bola možnosť použiť konečný počet inštancií.
- Namiesto globálnych premenných vždy použite singleton!

Príbuzné vzory

- *Abstract factory*
- *Builder*
- *Prototype*

Upozornenie

- Tieto študijné materiály sú určené výhradne pre študentov predmetu 5II132 Návrhové vzory (Design Patterns) na Fakulte riadenia a informatiky Žilinskej univerzity v Žiline.
- Reprodukovanie, šírenie (i častí) materiálov bez písomného súhlasu autora nie je dovolené.

Ing. Michal Varga, PhD.
Katedra informatiky
Fakulta riadenia a informatiky
Žilinská univerzita v Žiline
Michal.Varga@fri.uniza.sk