Nowoczesne języki programowania obiektowego

Wzorce projektowe: SINGLETON

dr Aleksander Lamża

Wydział Informatyki i Nauki o Materiałach
Instytut Informatyki, Uniwersytet Śląski

Problem: tworzenie tylko jednej instancji obiektu

- dostęp do np. bazy danych, urządzeń itp.
- obsługa konfiguracji aplikacji
- obsługa logów
- obiekty wykorzystywane podczas logowania
- okna dialogowe
- ...

Przykład

```
public class Konfiguracja {
  public Konfiguracja() {
    ...
  }
  ...
}
```

```
public void zmienKonfPaskaNarzedzi() {
  ...
  Konfiguracja k = new Konfiguracja();
  ...
}
```

```
public void zmienKonfAplikacji() {
  ...
  Konfiguracja k = new Konfiguracja();
  ...
}
```

W czym tkwi problem?

```
public class Konfiguracja {
  public Konfiguracja() {
    ...
  }
  ...
}
```

W jaki sposób uniemożliwić tworzenie nowych obiektów?

Czy można zrobić coś takiego?

```
public class Konfiguracja {
  private Konfiguracja() {
    ...
  }
  ...
}
```

W jaki sposób utworzyć jakikolwiek obiekt tej klasy?

Wykorzystanie metody statycznej

```
public class Konfiguracja {
  private Konfiguracja() {
    ...
  }

  public static Konfiguracja pobierzInstancje() {
    return new Konfiguracja();
  }
  ...
}
```

```
...
 Konfiguracja k = Konfiguracja.pobierzInstancje();
...
```

Klasyczny singleton

```
public class Konfiguracja {
  private static Konfiguracja instancja = null;

  private Konfiguracja() { }

  public static Konfiguracja pobierzInstancje() {
    if (instancja == null)
      instancja = new Konfiguracja();
    return instancja;
  }
  ...
}
```

Klasyczny singleton - definicja

Wzorzec **Singleton**:

- zapewnia, że dana klasa będzie miała tylko i wyłącznie **jedną instancję obiektu**,
- zapewnia **globalny punkt dostępu** do tej instancji.

Przykład

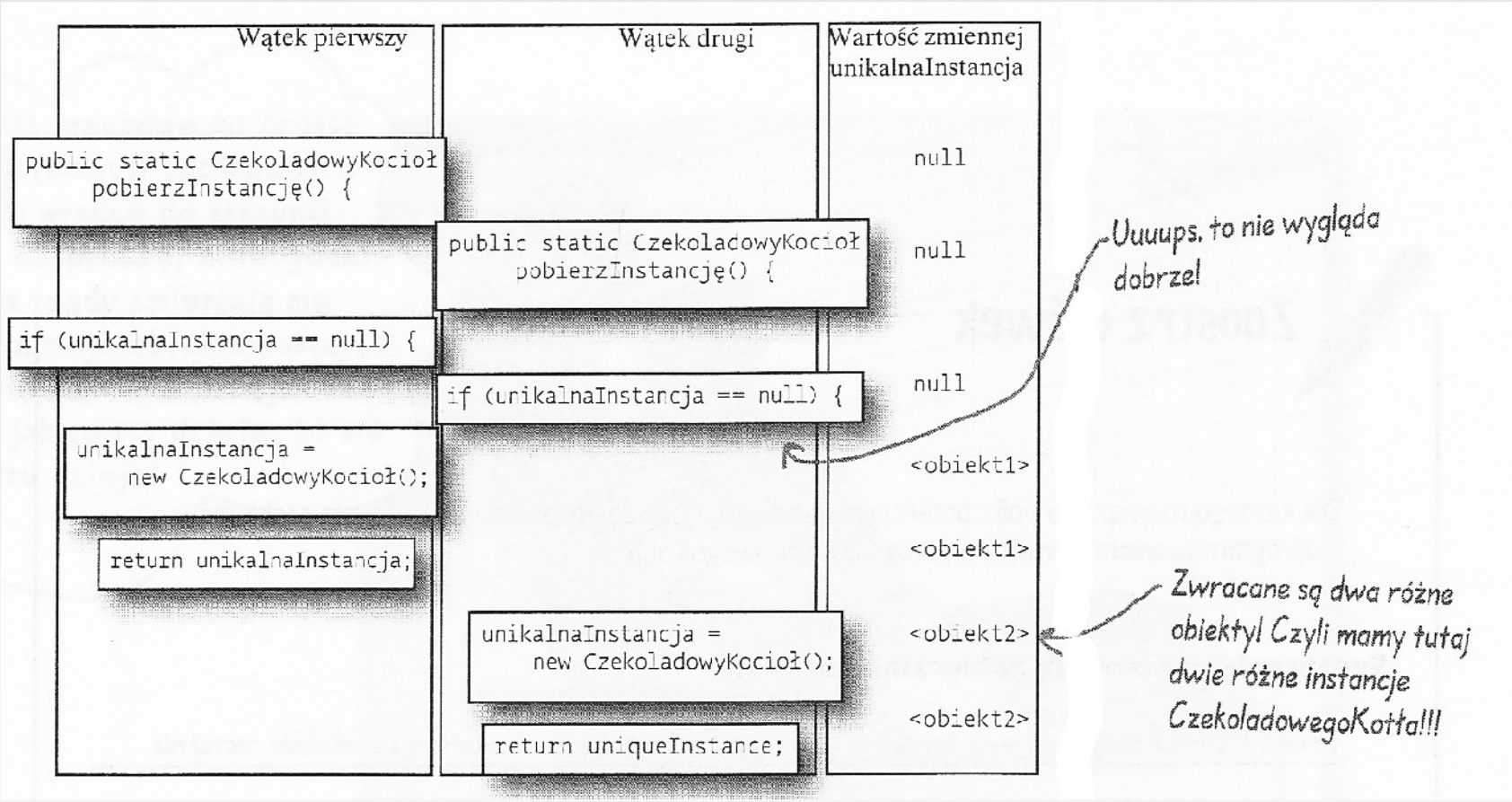
```
public class CzekoladowyKocioł {
    private boolean pusty;
    private boolean ugotowany;

    public CzekoladowyKocioł() {
        pusty = true;
        ugotowany = false;
    }

    public void napełniaj() {
        if (jestPusty()) {
            pusty = false;
            ugotowany = false;
            // napełniaj bojler mieszanką mleka i czekolady
        }
    }

    public void opróżniaj() {
        if (!jestPusty() && jestUgotowany()) {
            // opróżniaj bojler z ugotowanej mieszanki mleka i czekolady
            pusty = true;
        }
    }

    public void gotuj() {
        if (!jestPusty() && !jestUgotowany()) {
            // gotuj zawartość kotła
            ugotowany = true;
        }
    }

    public boolean jestPusty() {
        return pusty;
    }

    public boolean jestUgotowany() {
        return ugotowany;
    }
```

Zmiana klasy CzekoladowyKociol na singleton

```
public class CzekoladowyKocioł {
    private boolean pusty;
    private boolean ugotowany;

    public CzekoladowyKocioł() {
        pusty = true;
        ugotowany = false;
    }

    public void napełniaj() {
        if (jestPusty()) {
            pusty = false;
            ugotowany = false;
            // napełniaj bojler mieszanką mleka i czekolady
        }
```

```
public class CzekoladowyKocioł {
    private boolean pusty;
    private boolean ugotowany;

    private static CzekoladowyKocioł unikalnaInstancja;

    private CzekoladowyKocioł() {
        pusty = true;
        ugotowany = false;
    }

    public static CzekoladowyKocioł pobierzInstancję() {
        if (unikalnaInstancja == null) {
            unikalnaInstancja = new CzekoladowyKocioł ();
        }
        return unikalnaInstancja;
    }

    public void napełniaj() {
        if (jestPusty()) {
            pusty = false;
            ugotowany = false;
            // napełniaj bojler mieszanką mleka i czekolady
        }
    }
    // reszta kodu klasy CzekoladowyKocioł
}
```

Problemy z wielowątkowością



Rozwiązanie 1. Synchronizacja

```
public class Konfiguracja {
  private static Konfiguracja instancja = null;

  private Konfiguracja() { }

  public static synchronized Konfiguracja pobierzInstancje() {
    if (instancja == null)
      instancja = new Konfiguracja();
    return instancja;
  }
  ...
}
```

Rozwiązanie 2. Podwójne blokowanie

```
public class Konfiguracja {
  private static volatile Konfiguracja instancja = null;

  private Konfiguracja() { }

  public static synchronized Konfiguracja pobierzInstancje() {
    if (instancja == null) {
      synchronized (Konfiguracja.class) {
        if (instancja == null)
          instancja = new Konfiguracja();
      }
    }
    return instancja;
  }
  ...
}
```

Rozwiązania 3. Tworzenie obiektu z wyprzedzeniem

```
public class Konfiguracja {
  private static Konfiguracja instancja = new Konfiguracja();

  private Konfiguracja() { }

  public static synchronized Konfiguracja pobierzInstancje() {
    if (instancja == null)
      instancja = new Konfiguracja();
    return instancja;
  }
  ...
}
```