Analýza a návrh informačných systémov II 7

objektovo orientovaný návrh

Peter Bednár

Návrhové vzory

Návrhové vzory

- Návrhové vzory sú ustálené riešenia často sa vyskytujúcich problémov pri návrhu softvéru

- Urýchľujú návrh a znižujú počet chýb pri implementácii
  - Poskytujú overenú paradigmu
  - Zohľadňujú aj problematické prípady, ktoré nemusí byť jednoduché identifikovať pri návrhu
  - Ak sú všeobecne známe pre vývojárov a architektov v tíme, zlepšujú čitateľnosť kódu

Návrhové vzory - rozdelenie

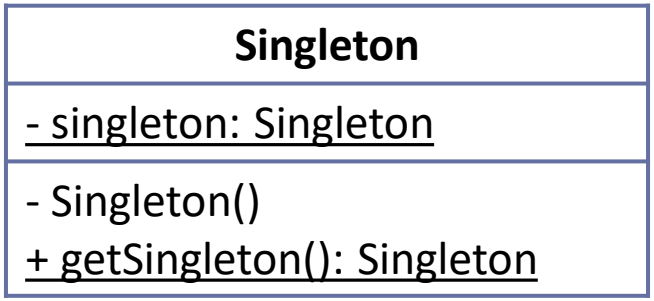
- Vzory pre vytváranie objektov
  - odstraňujú problémy pri priamom vytváraní objektov pomocou konštruktorov
- Štrukturálne vzory
  - organizujú viaceré objekty do väčších celkov, alebo tak aby poskytovali pridanú funkcionalitu
- Behaviorálne vzory
  - zaoberajú sa zlepšením flexibilnosti komunikácie medzi objektami (volaním metód)

Vzory pre vytváranie objektov

Singleton

- Zabezpečuje to, aby vždy existovala v aplikácii iba jedna inštancia danej triedy
- Definuje ako inštanciu vytvoriť a ako k nej bezpečne pristupovať

Singleton




```
public class Singleton {
    private static final Singleton singleton = new Singleton();

    private Singleton() {
        // inicializácia objektu
    }
    public static Singleton getSingleton() {
        return singleton;
    }
}
```

Singleton

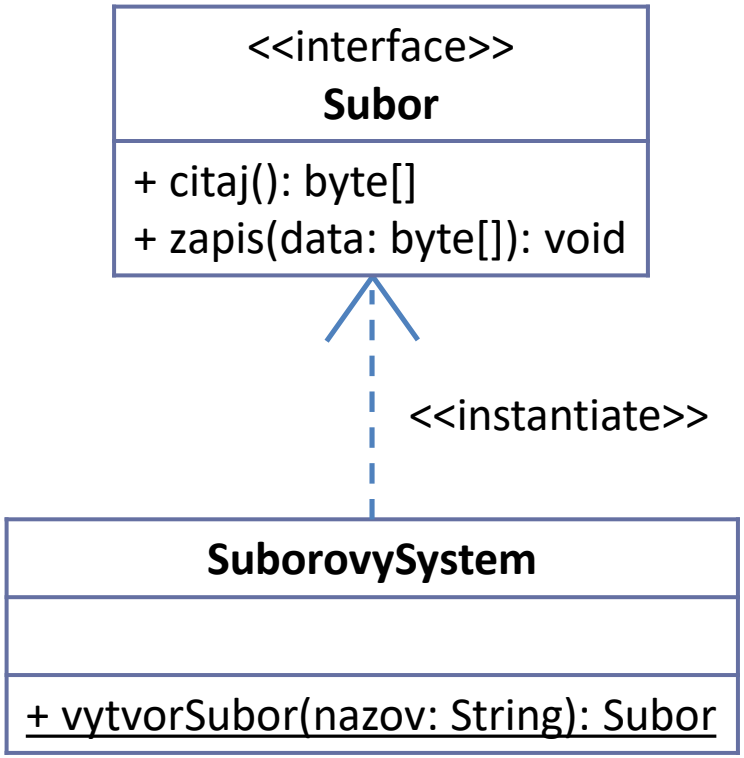
```
public class Singleton {
    private static Singleton singleton = null;

    private Singleton() {
        // inicializácia objektu
    }
    public static Singleton getSingleton() {
        if (singleton == null) {
            singleton = new Singleton();
        }
        return singleton;
    }
}
```

Inicializácia až pri prvom prístupe (tzv. *lazy* inicializácia)

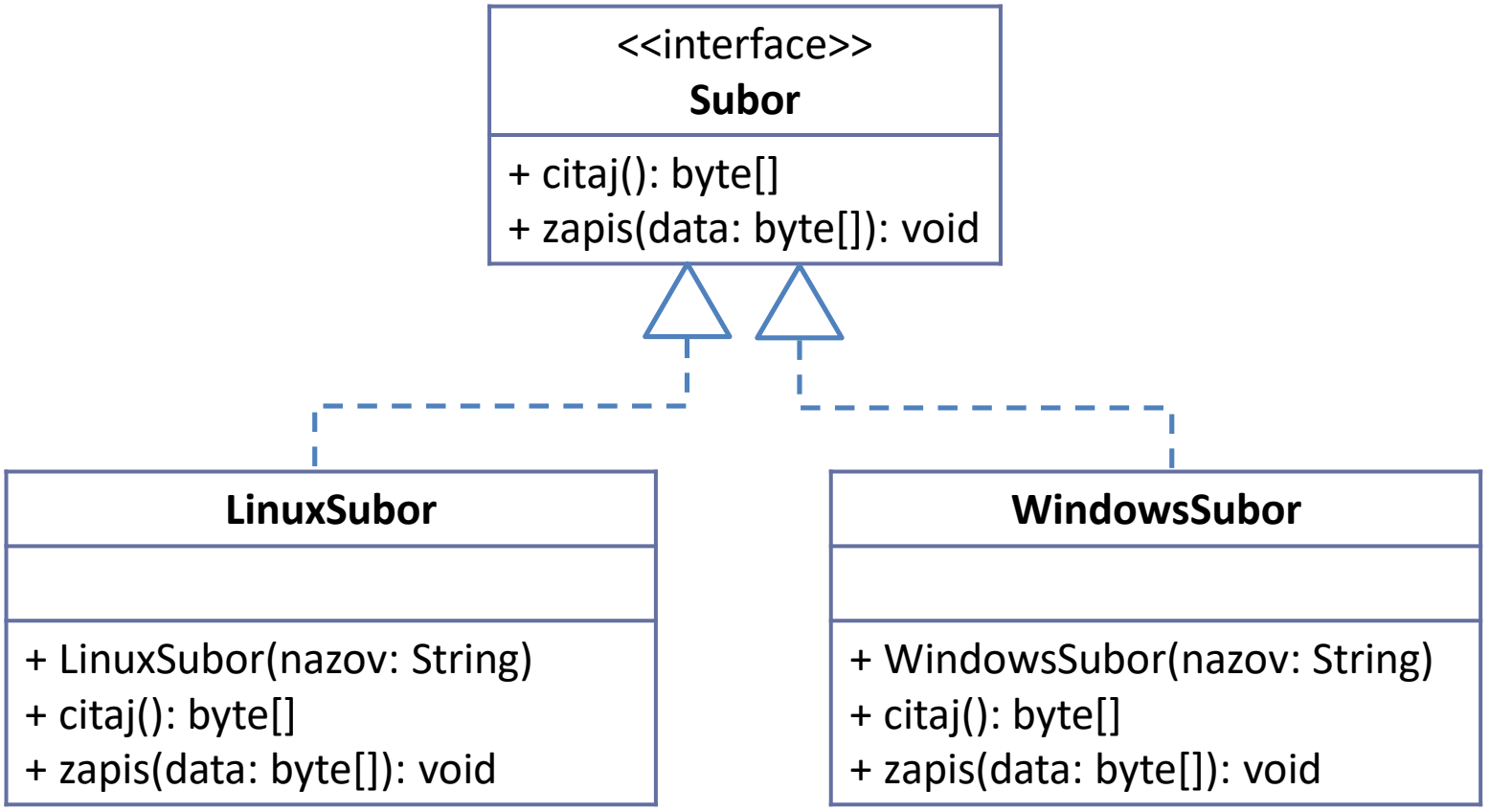
Factory

- Poskytuje spôsob ako jednotne vytvoriť objekty bez toho, aby sme priamo špecifikovali implementujúcu triedu
- Klient pracuje iba s rozhraním vytváraných objektov, nie je dôležité pre neho vedieť, ktorá konkrétna trieda ich implementuje

Factory – príklad (1)



Factory – príklad (2)



Factory – príklad (3)

```
public class SuborovySystem {

    private String system;
    ...

    public static Subor vytvorSubor(String nazov) {
        if (system.equals("Windows")) {
            return new WindowsSubor(nazov);
        }
        if (system.equals("Linux")) {
            return new LinuxSubor(nazov);
        }
        throw new IllegalStateException("nepodporovaný systém");
    }
}
```

Lazy initialization

- Neskorá inicializácia – vytvorenie objektu až keď je prvý krát potrebný
- Výhodné ak vytvorenie nového objektu vyžaduje veľa výpočtových zdrojov
  - Iba ak je možné zdieľať objekty pre tie isté parametre pri vytváraní

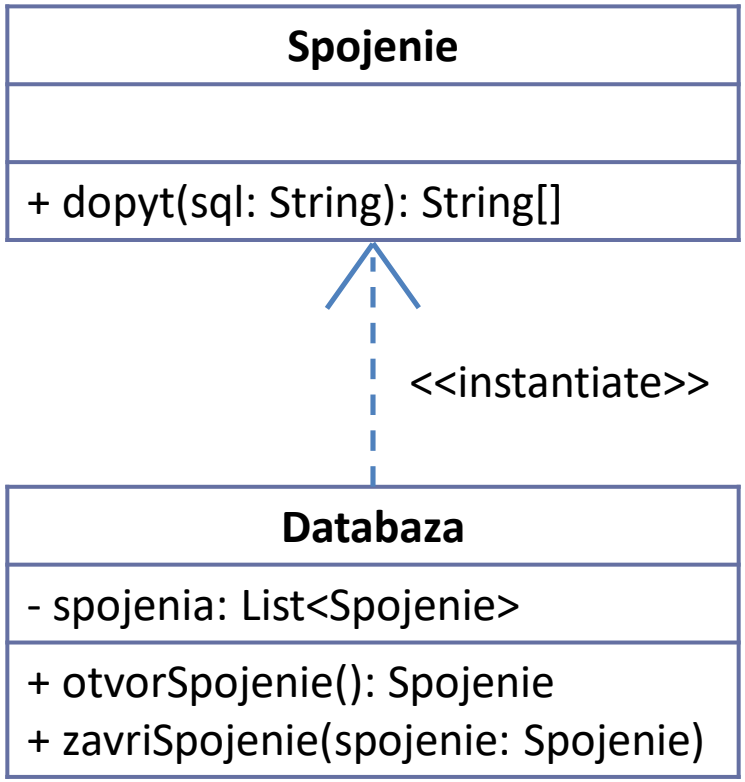
Lazy initialization – príklad

```
public class Kluc {

    private byte[] data;
    private Kluc(byte[] data) { this.data = data; }

    private static Map<String, Kluc> kluce = new HashMap<>();

    public static Kluc vygenerujKluc(String heslo) {
        if (kluce.contains(heslo)) {
            return kluce.get(heslo);
        } else {
            // zložitý výpočet dát z hesla
            kluc = new Kluc(data);
            kluce.put(heslo, kluc);
            return kluc;
        }
    }
}
```

Object pool

- Podobne ako pri lazy inicializácii, výhodné ak vytvorenie objektu vyžaduje veľa zdrojov
- Na začiatky sa vytvorí množina objektov, ktoré sú znovupoužiteľné
  - Keď už klient objekt nevyužíva, vráti ho pre ďalšie použitie
  - Výhodné pre zdieľaný prístup k externým zdrojom (napr. k databáze)

Object pool – príklad (1)



Object pool – príklad (2)

```
public class Databaza {
    private List<Spojenie> spojenia = new LinkedList<>();
    public Databaza() {
        for (int i = 0; i < 10; i++) spojenia.add(new Spojenie());
    }
    public Spojenie otvorSpojenie() {
        if (spojenie.isEmpty()) {
            return new Spojenia();
        } else {
            Spojenie spojenie = spojenia.get(0);
            spojenie.remove(0);
            return spojenie
        }
    }
    public void zavriSpojenie(Spojenie spojenie) {
        spojenia.add(spojenie);
    }
}
```

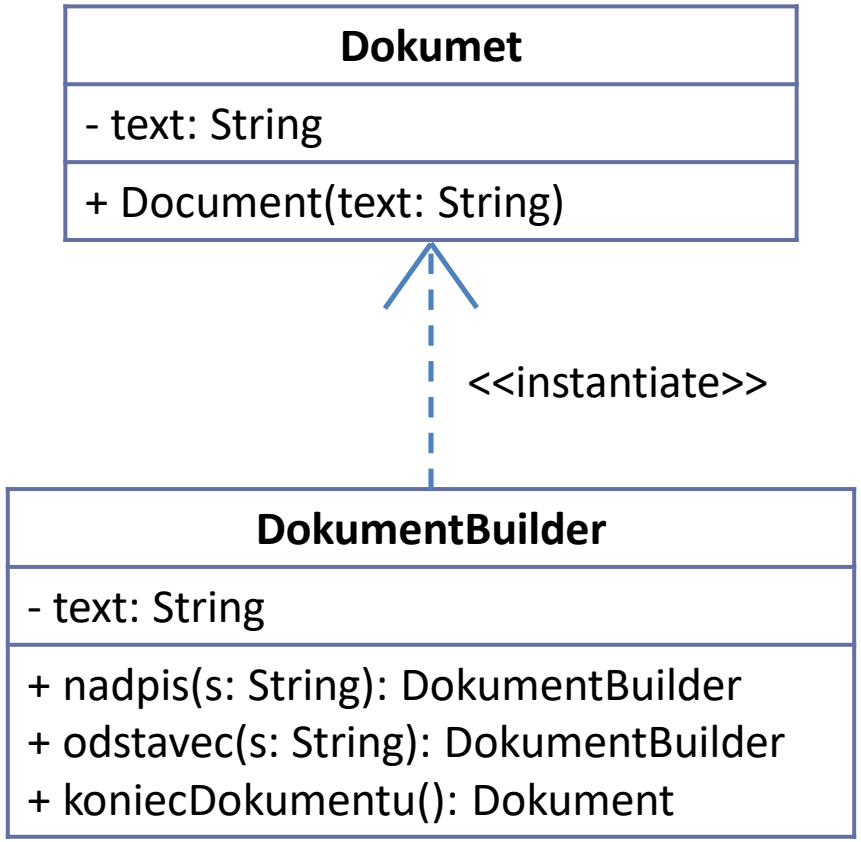
Object pool – príklad (3)

```
public static void main(String args[]) {
    Databaza db = new Databaza();

    ...
    Spojenie spojenie = null;
    try {
        spojenie = db.otvorSpojenie();
        spojenie.dopyt("SELECT * FROM tabulka");
        ...
    } finally {
        if (spojenie != null) {
            db.zavriSpojenie(spojenie);
        }
    }
}
```

Builder

- Flexibilné vytváranie zložitých objektov
- Builder je trieda ktorá poskytuje:
  - Metódy pre inicializáciu jednotlivých častí komplexného objektu
  - Metódu ktorá vráti výsledný objekt po skončený inicializácie

Builder – príklad (1)



Builder – príklad (2)

```
public class DokumentBuilder {
    private String text = "";

    public DokumentBuilder nadpis(String s) {
        text += ("*** " + s + " ***\n");
        return this;
    }
    public DokumentBuilder odstavec(String s) {
        text += (s + "\n");
        return this;
    }
    public Dokument koniecDokumentu() {
        Dokument dokument = new Dokument(text);
        text = "";
        return dokument;
    }
}
```

Builder – príklad (3)

```
DokumentBuilder builder = new DokumentBuilder();

Dokument dokument1 = builder.
    nadpis("Prvá kapitola").
    odstavec("Text ...").
    odstavec("Text ...").
    nadpis("Druhá kapitola").
    odstavec("Text ...").
    koniecDokumentu();

Dokument dokument2 = builder.
    nadpis("Kapitola 1").
    odstavec("Text ...).
    koniecDokumentu();
```

Štrukturálne vzory I

Marker

- Prázdne rozhranie ktoré reprezentuje metadáta o kóde
- V Jave sú implementované ako anotácie
  - Preddefinované v jazyku na bežné použitie, napr. @Override
  - Je možné si definovať vlastné

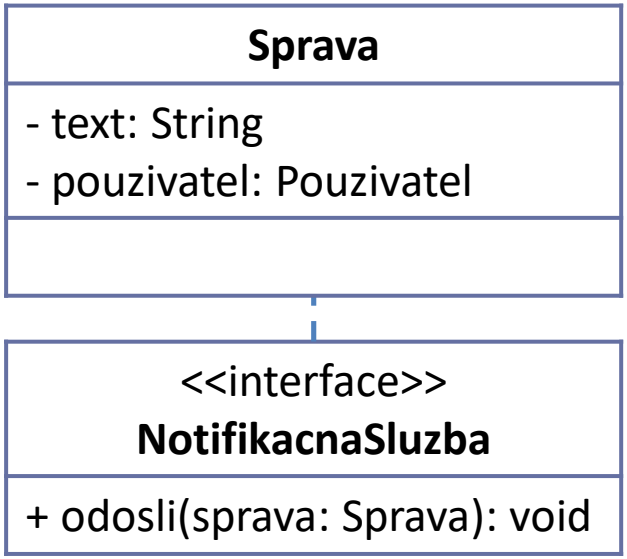
Marker – príklad

```
public @interface Nedokoncene {
}

public @interface Autor {
    String meno();
    String email();
}

@Nedokoncene
@Autor(meno="Peter Bednár", email="peter.bednar@tuke.sk")
public class MojaTrieda {
    ...
    @Nedokoncene
    public void vypocitaj() {
        ...
    }
}
```

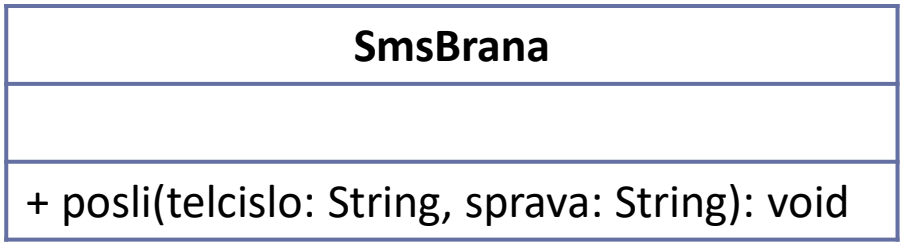
Adapter

- Umožňuje zmeniť rozhranie, ktoré implementuje objekt na iné rozhranie ktoré očakáva klient.
- Je možné dosiahnuť lepšiu znovupoužitelnosť kódu

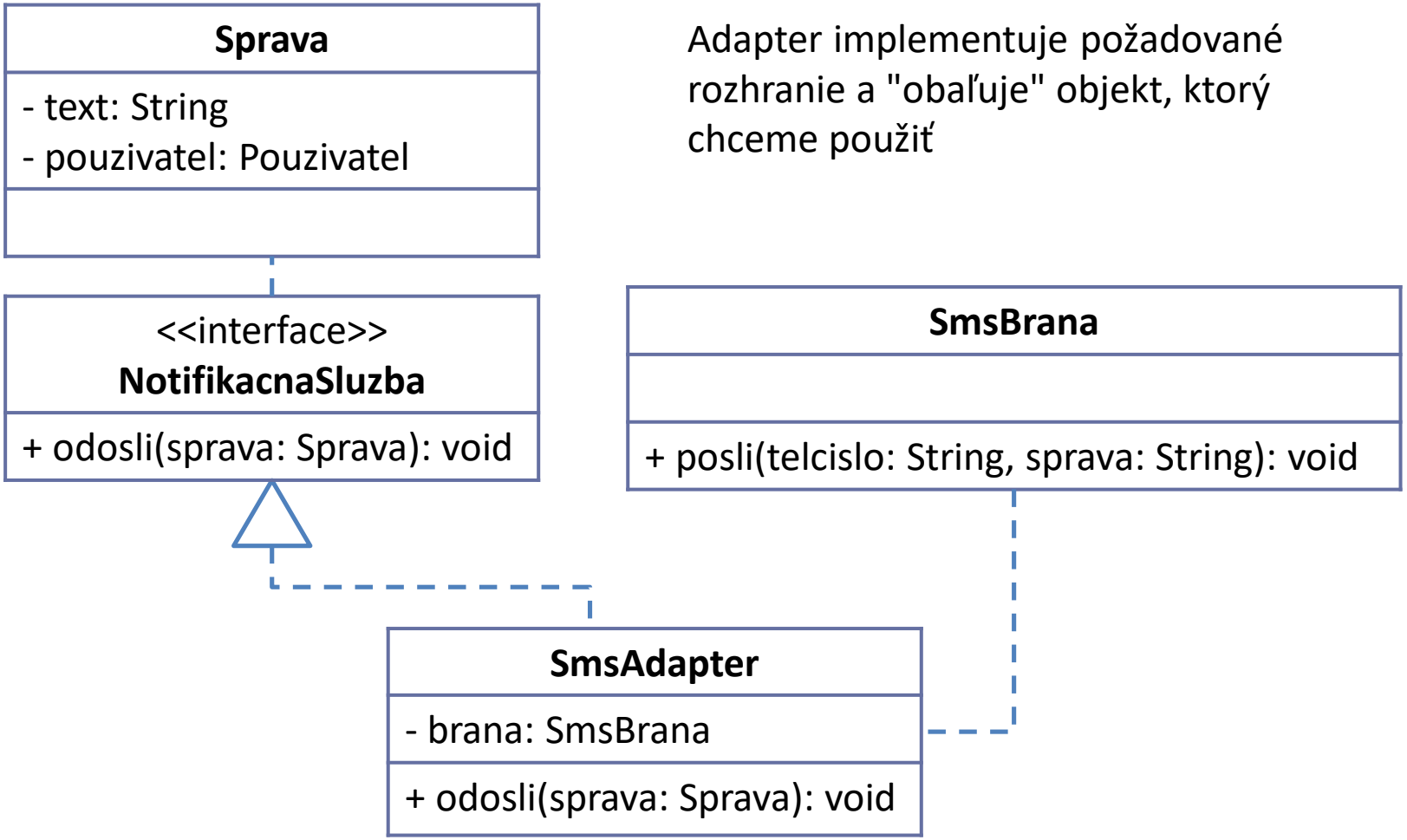
Adapter – príklad (1)



Chceme použiť SMS bránu pre notifikácie. V našej aplikácii sa používa notifikačná služba reprezentovaná rozhraním NotifikacnaSluzba



Adapter – príklad (2)



Adapter implementuje požadované rozhranie a "obaľuje" objekt, ktorý chceme použiť

Adapter – príklad (3)

```
public class SmsAdapter implements NotifikacnaSluzba {

    private SmsBrana brana = new SmsBrana();

    @Override
    public void odosli(Sprava sprava) {
        String text = sprava.text;
        String tel  = sprava.pouzivatel.getTelCislo();
        brana.posli(tel, text);
    }

}
```

Zhrnutie

- Vzory pre vytváranie objektov
  - Singleton
  - Factory
  - Lazy initialization
  - Object pool
  - Builder
- Štrukturálne vzory I
  - Marker
  - Adapter