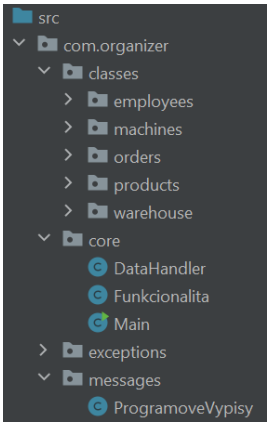
Emma Macháčová

1. (6 b) Uveďte fragment kódu z vášho projektu, na ktorom je vidieť súvis vnútornej logiky a používateľského rozhrania (nie nevyhnutne grafického). Vysvetlite, či je týmto dosiahnuté ich dostatočné oddelenie. Ak nie, vysvetlite, čo treba napraviť.

a) uvedenie relevantného kódu: 2 b

b) identifikácia súvisu vnútornej logiky a používateľského rozhrania: 2 b

c) vysvetlenie, či oddelenie vnútornej logiky a používateľského rozhrania postačuje a prípadná náprava: 2 b



V mojom projekte je vnútorná logika a používeteľské rozhranie – konzolová aplikácia – rozdelené pomocou balíkov a modalu MVC.

Messages (trieda ProgramoveVypisy) predstavuje rozhranie pre pouzívateľa – view, a jeho funkcionalitu zabezpečuje v balíku core (controller) trieda **Funkcionalita**.

```
case 4: // zobrazenie objednavky
    if (!(objednavka instanceof ObjednavkaZakaznika)) { // ak neeistuje
        System.out.println(ProgramoveVypisy.errorColor + ProgramoveVypisy.caseZakaznikNeexistuje + ProgramoveVypisy.white);
        break;
    }
    System.out.println(ProgramoveVypisy.caseZakaznikZobrazenie);
    objednavka.vypisObjednavku();
    break;
```

(využitie triedy **ProgramoveVypisy** triedou Funkcionalita)

O takéto oddelenie som sa snažila v celom kóde – všetky výpisy aj s formátovaním textu spravuje trieda ProgramoveVypisy, čím som chcela dosiahnuť prehľadnejší kód a jednoduchšiu úpravu používateľského rozhrania pri zmene funkcionality kódu.

```
do {
    System.out.print(ProgramoveVypisy.highlight + ProgramoveVypisy.run + ProgramoveVypisy.white);
    System.out.println(ProgramoveVypisy.exit);

    key = keyboard.nextInt();

    switch (key) {
        case 0:
            break;
        case 1: // operator
            prevadzka = Funkcionalita.caseOperator(prevadzka);
            break;
        case 2: // manazer
            prevadzka = Funkcionalita.caseManazer(prevadzka);
            break;
        case 3: // zamestnanec
            prevadzka = Funkcionalita.caseZamestnanec(prevadzka);
            break;
        case 4: // zakaznik
            prevadzka = Funkcionalita.caseZakaznik(prevadzka);
            break;
        default:
            break;
    }

} while (key != 0);
```

(hlavný cyklus v triede main využívajúci ProgramoveVypisy a volajuci triedu Funkcionalita)

Dostatočné oddelenie je zabezpečené zordelením do balíkov a správnym použitím enkapsulácie.

Emma Macháčová

2. (6 b) Vo vašom projekte ste možno použili alebo by ste mohli použiť jeden z návrhových vzorov Visitor, Observer, Composite alebo Strategy. Uveďte príslušnú implementáciu tohto vzoru (z projektu alebo novú – len nevyhnutný kód). Identifikujte roly vzoru v zmysle typov vo vašom projekte a ďalších typov, ktoré prípadne pridáte. Vyjadrite v terminológii vášho projektu dve najvýraznejšie protichodné sily, ktorých konflikt tento vzor rieši. Vysvetlite, ako vzor rieši tento konflikt.

a) uvedenie relevantného kódu: 3 b

b) identifikácia rolí: 1 b

c) identifikácia síl a ich konfliktu, vrátane jeho riešenia: 2 b

Vzor Visitor - účelom vzoru Visitor je definovať novú operáciu bez zavedenia úprav existujúcej objektovej štruktúry. Ide o pridanie novej funkcionality bez z meny existujúcich tried. Teda použijeme princíp otvorenosti a uzavretosti – nebudeme upravovať kód, ale budeme môcť ozšíroť funkcionalitu pridaním novej implementácie Visitora.

Konflikt spočíva v tom, že tento návrhový vzor potrebuje stálu údržbu pri pridávaní nových prvkov do štruktúry. Teda keď pridáme element musíme pridať Visitorovi novú metódu, ktorou ho spracuje.

Silná stránka je v oddelení algoritmu od tried v ktorých pracuje a uľahčuje pridávanie nivých operácií.

Implementácia:

```
// vytvorenie interface
public interface Visitable{
    public void accept(Visitor visitor);
}

// konkrétny element
public class Rukavice implements Visitable{
    private double cena;
    private double spotrebaMaterialu;

  // akceptovanie Visitora
  public void accept(Visitor vistor) {
        visitor.visit(this);
  }

  public double getCena() {
        return cena;
  }

  public double getSpotrebaMaterialu() {
        return spotrebaMaterialu;
  }
}
```

```
// pridanie metód do Visitora pre každý element
public interface Visitor{
 public void visit(Rukavice rukavice);

 // ostatné elementy
 public void visit(Sveter sveter);
 public void visit(Vesta vesta);
}
```

Emma Macháčová

3. (6 b) Na príklade prekonávania metód z vášho projektu vysvetlite, ako bol alebo nebol (stačí jedna z týchto možností) dodržaný Liskovej princíp substitúcie.

a) uvedenie príkladu prekonávania: 1 b

b) posúdenie dodržania Liskovej princípu substitúcie: 2 b

c) zdôvodnenie prostredníctvom predpokladov a dôsledkov: 3 b

Pri Liskovej princípe substitúcie sledujeme to, či objekty uvažovanej nadtriedy je možné nahradiť objektami jej podtried.

V mojom projekte je použité prekonávanie napríklad v tomto prípade:

V balíku **machines**, všetky stroje obsahujú metódu **uprav( ),** ktorá sa správa podľa toho, o aký konkrétny stroj ide. Napríklad trieda NaparovaciStrojNaZehlednie dedí od triedy StrojNaZehlenie metódu vyzehli, okrem ktorej pri volaní metódy uprav využije aj metódu napar.

```
public Sveter uprav (Sveter sveter) {
    sveter = vyzehli(sveter);
    return sveter;
}

public Vesta uprav (Vesta vesta) {
    vesta = vyzehli(vesta);
    return vesta;
}
```

(metóda uprav v triede StrojNaZehlenie)

```
public Sveter uprav (Sveter sveter) {

    sveter = Naparovac.napar(sveter);
    sveter = vyzehli(sveter);

    return sveter;
}

Params: vesta – vytvorená vesta, ktorá sa má strojom upraviť
Returns: upravená vesta (naparená a vyžehlená)
public Vesta uprav (Vesta vesta) {

    vesta = Naparovac.napar(vesta);
    vesta = vyzehli(vesta);

    return vesta;
}
```

(metóda uprav v triede NaparovaciStrojNaZehlenie)

Tým pádom je možné nahradiť objekt nadriedy jej podtriedou, nie však naopak, nakoľko podtrieda je rozšírená a funkcionalitu, ktorú nadtrieda neobsahuje.

Emma Macháčová

4. (6 b) Vysvetlite na príklade z vášho projektu, ktorá vlastnosť jazyka AspectJ, ktorú Java neposkytuje, by vám prišla vhod pri jeho realizácii a prečo.

a) uvedenie relevantného príkladu (kódom alebo opisne): 2 b

b) pomenovanie vlastnosti jazyka AspectJ, ktorú Java neposkytuje: 1 b

c) zdôvodnenie relevantnosti použitia identifikovanej vlastnosti jazyka AspectJ: 3 b

Problematika pracujúceho objektu – Java nepodporuje priamo prenesenie odkazu na metódu, ktorou ju možno spustiť.

Využíva sa na to implementovanie rozhrania Runnable a metódy run.

V triede SijaciStroj používam preddefinované funkčné rozhranie Runnable na odkazovanie na statickú metódu. Rovnako sa pri tom využívajú vlákna (threads) pri vytváraní produktov.

```
public Odev vyrobOdev(String velkost, boolean damske, String material, String typ){
    if(typ == null){
        return null;
    }
    if(typ.toLowerCase().equals("sveter")){
        Thread t = new Thread (ZamestnanecVyroby::ThreadStatus);
        t.start();
        Sveter sveter = new Sveter(velkost, damske, material);
        sveter.setVyslednaCena(2*(sveter.getCenaMaterialu()));
        return sveter;
    }
    if (typ.toLowerCase().equals("vesta")){
        Thread t = new Thread (ZamestnanecVyroby::ThreadStatus);
        t.start();
        Vesta vesta = new Vesta(velkost, damske, material);
        vesta.setVyslednaCena(2*(vesta.getCenaMaterialu()));
        return vesta;
    }
    return null;
```

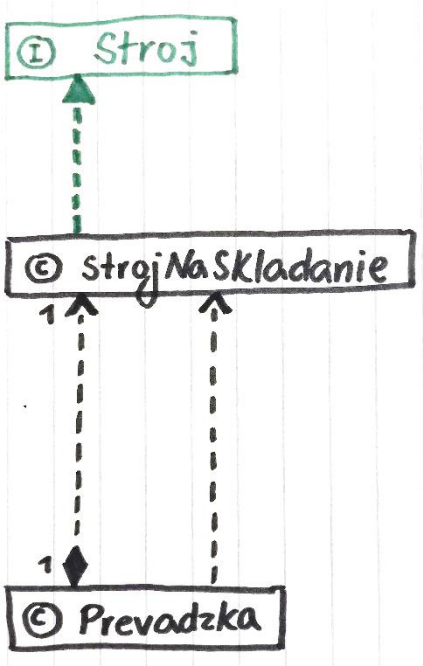
Emma Macháčová

5. (6 b) Diagramom tried v UML vyjadrite vzťah vybraného rozhrania z vášho projektu, triedy, ktorá ho implementuje, a triedy ktorá ho používa. V triedach a v rozhraní stačí uviesť len nevyhnutné prvky z hľadiska otázky. Diagram vysvetlite.

a) uvedenie relevantného kódu: 1 b

b) správnosť notácie UML: 2 b

c) vysvetlenie diagramu: 3 b



- Rozhranie: **Stroj**,

```
public interface Stroj {
```

- trieda, kt. ho implementuje: **StrojNaSkladanie**,

```
public class StrojNaSkladanie implements Stroj {
```

- trieda, kt. ho používa: **Prevádzka**.

```
// poskladanie
this.strojNaSkladanie.uprav(this.getManazeri().get
```

(v metóde odosliDoSkladu)

Rozhranie Stroj predstavuje predpis pre triedu StrojNaSkladanie, a trieda modelu Singleton Prevadzka vytvára inštanciu triedy StrojNaSkladanie, ktorú využíva.