Objektovo-orientované programovanie

Riadiaci softvér pre lokálnu handmade výrobu svetrov

Emma Macháčová

Meno cvičiaceho : Ing. Anna Považanová
Čas cvičení : štvrtok 9:00
Dátum vytvorenia : 03. marec 2021

Obsah

Zámer projektu

Softvér má slúžiť na manažovanie procesov, ktoré sú spojené s výrobou a následným predajom handmade svetrov lokálnou spoločnosťou.

Medzi tieto procesy patrí najmä správa inventáru spoločnosti, tvorba interných objednávok spojených so zabezpečovaním nákupu materiálu potrebného na výrobu, samotná výroba konečných produktov, generovanie časového harmonogramu výroby, rozdeľovanie úloh medzi zamestnancov a konečné odoslanie výrobku zákazníkovi. Tak isto je cieľom správa toku informácií medzi jednotlivými oddeleniami a kľúčovými zamestnancami. Zamestnanci pracujú na rôznych typoch pracovných pozícií, ich schopnosti a úlohy sa líšia a sú platení na základe ich pracovných výkonov. Výsledným produktom výrobného procesu je pletený výrobok, predovšetkým kus odevu (sveter), ale je možné vyrábať aj iné produkty (vesty, tašky, čiapky, rukavice..).

Úlohou softvér je priblížiť sa k optimálnej výrobe (všetky zakúpené materiály použité a všetky vyrobené svetre predané). Systém nebude brať do úvahy čas potrebný na dodanie objednaného materiálu (tovar je k dispozícii ihneď po objednaní), ani na čas potrebný na konečný export produktov zákazníkovi (sklad je uvoľnený momentom predaja). Koncom každého dňa systém vyhodnotí efektivitu výroby. Softvér nebude brať do úvahy prípadné meškanie zamestnanca alebo PN (všetci zamestnanci pracujú neustále).

Kľúčové slová

- Sveter – výsledný produkt výrobného procesu, má určitú veľkosť, strih, a iné vlastnosti závisiace od materiálu z ktorého je vytvorený
- Klbko – materiál potrebný pre vytvorenie výsledného produktu, má vlastnosti ako farbu, zloženie, gramáž, priemernú spotrebu, cenu za kus
- Sklad - priestor na uloženie tovaru, surovín, materiálu
- Zamestnanec – fyzická osoba, ktorá v konkrétnom pracovnoprávnom vzťahu vykonáva pre zamestnávateľa závislú prácu, určenú v závislosti od typu zamestnanca
- Výrobok – predmet kúpy a predaja (svetre, tašky, čiapky..)
- Vybavenie – nástroje potrebné na výrobný proces (rovné ihlice, kruhové ihlice, pletacie stroje..)
- Aplikácia – tvorí súčasť výrobku, všetko čo podľa povahy výrobku k nemu patrí a nemôže byť oddelené bez toho, že by sa tým znehodnotil (gombíky, nažehlovačky, spony, brmbolce)
- Objednávka – žiadosť o vyhotovenie a dodanie tovaru s dopredu vymienenými vlastnosťami
- Zákazník – objednávateľ, osoba, ktorá objednáva tovar na osobné účely (pre ktorú je zhotovená objednávka)

Kód

Dedenie

V projekte sa nachádzajú tieto hierarchie dedenia:

- Prvá hierarchia dedenia je v balíku **employees**, kde je nasledovné **trojvrstvové dedenie**:

  - **Zamestnanci**
    - **Zamestnanec** (implements Zamestnanci)
      - **Manazer** (extends Zamestnanec)
      - **ZamestnanecVyroby** (extends Zamestnanec)

- Ďalšia hierarchia dedenia je v balíku **products**, a to týmto spôsobom:

  - **Sveter**
    - **Vesta** (extends Sveter)

Rozhrania

V projekte používam rozhranie **Zamestnanci**, ako predpis pre všetky osoby kategórie zamestnanec (class Zamestnanec)

Polymorfizmus – prekonávanie

Prekonávanie metód nastáva v:

- class **Manazer**, metóda **odchodZPrace( )** okrem zmeny parametra vPraci, obsahuje metódu hromadnyOdchod( ), ktorá nastaví dochádzku všetkých podriadených zamestnancov manažéra na false, pretože ak nie je v práci vedúci manežér, jeho podriadení nemôžu pracovať

- class **Zamestnanec**, metódy **prichodDoPrace( )** a **odchodZPrace( )** – tieto metódy dedí z interface Zamestnanci, a sú doplnené o nastavenie parametra boolean vPraci na true alebo false

Preťažovanie metód

Preťažovanie metód nastáva v:

- class **Manazer**, metódy **vykonajDochadzku( )** pre zadanie dochádzky všetkých zamestnancov, a **vykonajDochadzku(String meno)** pre vykonanie dochádzky konkrétneho zamestnanca, pre uľahčenie práce a šetrenie času

Zapuzdrenie

Účelom zapuzdrenia je oddeliť rozhranie s kontraktom abstrakcie od jej implementácie. Trieda je „black box“, všetko je schované za public metódami. Použité sú modifikátory viditeľnosti (inštančné premenné sú private), prístup je spravovaný cez „gettre“ a „settre“.

Manazer

- parametre
  - → **private** ArrayList<ZamestnanecVyroby> zamestnanci
  - → **private** ArrayList<ObjednavkaZakaznika> ulohy
  - → **private** int pocetZamestnancov
- get / set
  - → **public** float getMzda( )
  - → **public** String getMeno( )
  - → **public** ArrayList<ZamestnanecVyroby> getZamestnanci( )
  - → **public** int getPocetZamestnancov( )
  - → **public** ArrayList<ObjednavkaZakaznika> getUlohy( )
  - → **public** void setZamestnanci(..)
  - → **public** void setMeno(..)
  - → **public** void setMzda(..)
  - → **public** void setVPraci(..)
  - → **public** void setPocetZamestnancov(..)
  - → **publiv** void setUlohy(..)

Zamestnanec

- parametre
  - → **protected** String meno
  - → **protected** boolean vPraci
  - → **protected** float mzda
- get / set
  - → **public** String getMeno( )
  - → **public** float getMzda( )
  - → **public** void setMeno(..)
  - → **public** void setMzda(..)
  - → **public** void setVPraci(..)

ObjednavkaZakaznika

- parametre
  - → **private** boolean vybavena
  - → **private** ArrayList<PolozkaObjednavky> polozky
  - → **private** float cenaObjednavky
- get / set
  - → **public** ArrayList<PolozkaObjednavky> getPolozkaObjednavky( )
  - → **public** float getCenaObjednavky( )
  - → **public** void setPolozky(..)
  - → **public** void setVybavena(..)
  - → **public** void setCenaObjednavky(..)

- PolozkaObjednavky
  - parametre
    - → **private** String farba
    - → **private** String material
    - → **private** String velkost
    - → **private** String nazovPolozky
    - → **private** int damske
    - → **private** float cenaPolozky
  - get / set
    - → **public** String getVelkost( )
    - → **public** String getNazovPolozky( )
    - → **public** String getMaterial( )
    - → **public** String getFarba( )
    - → **public** int getDamske( )
    - → **public** float getCenaPolozky( )
    - → **public** void setVelkost(..)
    - → **public** void setMaterial(..)
    - → **public** void setFarba(..)
    - → **public** void setDamske(..)
    - → **public** void setNazovPolozky(..)
    - → **public** void setCenaPolozky(..)

- Material
  - parametre
    - → **private** static float cenaZaKlbko
    - → **private** String farba
    - → **private** String nazovMaterialu
  - get / set
    - → **public** static float getCenaZaKlbko( )
    - → **public** String getFarba( )
    - → **public** String getNazovMaterialu( )
    - → **public** static void setCenaZaKlbko(..)
    - → **public** void setFarba(..)
    - → **public** void setNazovMaterialu(..)

- **Sveter**
  - parametre
    - → **private** String velkost
    - → **private** boolean damske
    - → **private** int spotrebaKlbiek
    - → **private** float cenaMaterialu
    - → **private** Material materialSvetra
  - get / set
    - → **public** String getVelkost( )
    - → **public** float getCenaMaterialu( )
    - → **public** int getSpotrebaKlbiek( )
    - → **public**Material getMaterialSvetra( )
    - → **public** void setVelkost(..)
    - → **public** void setCenaMaterialu(..)
    - → **public** void setSpotrebaKlbiek(..)
    - → **public** void setDamske(..)
    - → **public** void setMaterialSvetra(..)

- **Vesta**
  - parametre
    - → **private** String velkost
    - → **private** boolean damske
    - → **private** int spotrebaKlbiek
    - → **private** float cenaMaterialu
    - → **private** Material materialVesty
  - get / set
    - → **public** String getVelkost( )
    - → **public** float getCenaMaterialu( )
    - → **public** int getSpotrebaKlbiek( )
    - → **public**Material getMaterialVesty( )
    - → **public** void setVelkost(..)
    - → **public** void setCenaMaterialu(..)
    - → **public** void setSpotrebaKlbiek(..)
    - → **public** void setDamske(..)
    - → **public** void setMaterialVesty(..)

- **Prevadzka**
  - parametre
    - → **private** ArrayList<Manazer> manazeri
    - → **private** int pocetZamestnancov
    - → **private** int pocetManazerov
    - → **private** String nazovPrevadzky
    - → **private** ArrayList<ObjednavkaZakanika> aktivneObjednavky
  - get / set (príslušné get a set metódy)

Agregácia

Agregácia nastáva v týchto prípadoch:

- Sveter
  - → private **Material** materialSvetra
- Vesta
  - → private **Material** materialVesty
- Prevadzka
  - → private ArrayList<Manazer> manazeri
  - → private ArrayList<ZamestnanecVyroby> zamestnanci
  - → private ArrayList<ObjednavkaZakaznika> aktivneObjednavky
- Manazer
  - → private ArrayList<ZamestnanecVyroby> zamestnanci
- ObjednavkaZakaznika
  - → private ArrayList<PolozkaObjednavky> polozky

Organizácia kódu, balíky (Model, View, Controller)

Projekt sa skladá z týchto balíkov:

- classes
  - employees
    - Manazer
    - Zamestnanci
    - Zamestnanec
    - ZamestnanecVyroby
  - orders
    - ObjednavkaZakaznika
    - PolozkaObjednavky
  - products
    - Material
    - Sveter
    - Vesta
  - Prevadzka
- core
  - DataHandler
  - Funkcionalita
  - Main
- messages
  - ProgramoveVypisy

Balík **classes** obsahuje všetky triedy, predstavuje Model. Balík **core** obsahuje hlavnú funkcionalitu programu, predstavuje Controller. Balík **messages** spravuje výstupy pre užívateľa, predstavuje View.

Návrhový vzor Singelton

Tento vzor je tvorený triedou Prevadzka, ktorá sa stará o to, aby jej inštancia existovala iba jedenkrát.

```
class Prevadzka implements Serializable {

        private static Prevadzka prevadzka = null;

        private Prevadzka (  ) { ......}

        public static Prevadzka getInstance(  ) {
                if (prevadzka == null) { prevadzka = new Prevadzka(  ); }
                return prevadzka;
        }
}
```

Funkcionalita

Funkcionalita programu sa delí podľa typu osoby, ktorá program spúšťa – každá má vlastné oprávnenia.

Switch: caseOperator

„Operátor“ je osoba zodpovedná za existenciu prevádzky, je akoby „vyššou mocou“, ktorá má najväčšie oprávnenia.

1. Vytvorenie prevádzky

Ako prvé pred prácou s programom je potrebné vytvoriť inštanciu prevádzky, preto že bez nej nemôžu byť spustiteľné ďalšie prvky funkcionality. Za celý beh programu existuje iba jediná inštancia prevádzky.

2. Zamestnanie manažérov

Je to ďalší dôležitý krok – ak prevádzka nemá manažérov, nemôže mať ani zamestnancov výroby, a teda nie je schopná vykonávať výrobný proces. Je možné zamestnať ľubovoľný počet manažérov.

Ak prevádzka neexistuje, nie je možné zamestnať manažérov.

3. Zamestnanie zamestnancov výroby

Pre každého z manažérov je možné zamestnať ľubovoľný počet jemu podriadených zamestnancov výroby. Toto oprávnenie má aj sám manažér.

Ak nie sú zamestnaní žiadni manažéri, nie je možné zamestnať ani zamestnancov výroby.

4. Kontrola pracoviska

Ide o hlásenie dochádzky celého personálu prevádzky – manažérov a im podriadených zamestnancov.

Ak nie sú zamestnaní žiadni manažéri, nie je možné vykonať kontrolu prevádzky.

Switch: caseManazer

Pre prístup k oprávneniam manažéra je potrebné identifikovať sa ako jeden z existujúcich manažérov. Musia teda už byť vytvorení manažéri v caseOperator. Tým, že sa manažér prihlási, sa mu automaticky vykoná dochádzka do práce.

1. Vykonanie dochádzky podriadených zamestnancov

Táto funkcia umožní manažérovi zadať dochádzku všetkým jeho podriadeným zamestnancom.

Pre vykonanie dochádzky zamestnancov musia existovať zamestnanci.

2. Vykonanie dochádzky konkrétneho zamestnanca

Táto funkcia tiež slúži na zadanie dochádzky, ale v prípade ak ide iba o konkrétneho zamestnanca, aby sa šetril manažérovi čas tým, že nemusí robiť celú dochádzku odznova.

3. Zamestnanie nových zamestnancov výroby

Manažér má oprávnenie zamestnať pod seba (ďalších) sebe podriadených zamestnancov výroby.

4. Vykonanie hlásenia o zamestnancoch a dochádzke

Funkcia vypíše meno manažéra a všetkých jemu podriadených zamestnancov, spolu s tým, či sa nachádzajú v práci, alebo nie.

Switch: caseZakaznik

Pre využitie oprávnení zákazníka je potrebné, aby už bola vytvorená prevádzka a zamestnaní manažéri.

1. Pridanie výrobku do objednávky

Pri vytváraní / pridávaní položiek do objednávky sa postupne program spýta na druh a parametre želaného produktu. Pokiaľ program zadaný vstup nerozpozná ako možný druh produktu, do objednávky nebude pridaný a zákazníka na to upozorní.

2. Potvrdenie objednávky

Pri potvrdzovaní objednávky program zobrazí výslednú čiastku, a spýta sa, či si zákazník vážne praje objednávku odoslať. Po potvrdení sa objednávka automaticky presunie medzi aktívne objednávky do prevádzky.

Pre potvrdenie objednávky musí objednávka existovať, teda musí byť pridaná aspoň jedna položka objednávky.

3. Zobrazenie aktuálnej objednávky

Funkcia zobrazí všetky položky objednávky, spolu s ich cenou a s výslednou cenou celej objednávky.

Pre zobrazenie objednávky musí objednávka existovať, teda musí byť pridaná aspoň jedna položka objednávky.

UML diagram tried