1. časť: Implementácia úlohy s návrhovým vzorom

Použite návrhový vzor Observer na realizáciu nasledovného príkladu:

Diagram

Schma implementcie lohy

- Otvoriť diagram samostatne

Pri implementácii dbajte na dodržiavanie názvov identifikátorov, ako sú uvedené v zadaní a diagrame. Všetky súbory vašej implementácie umiestňujte priamo do pripraveného balíka sk.tuke.kpi.oop.exams.

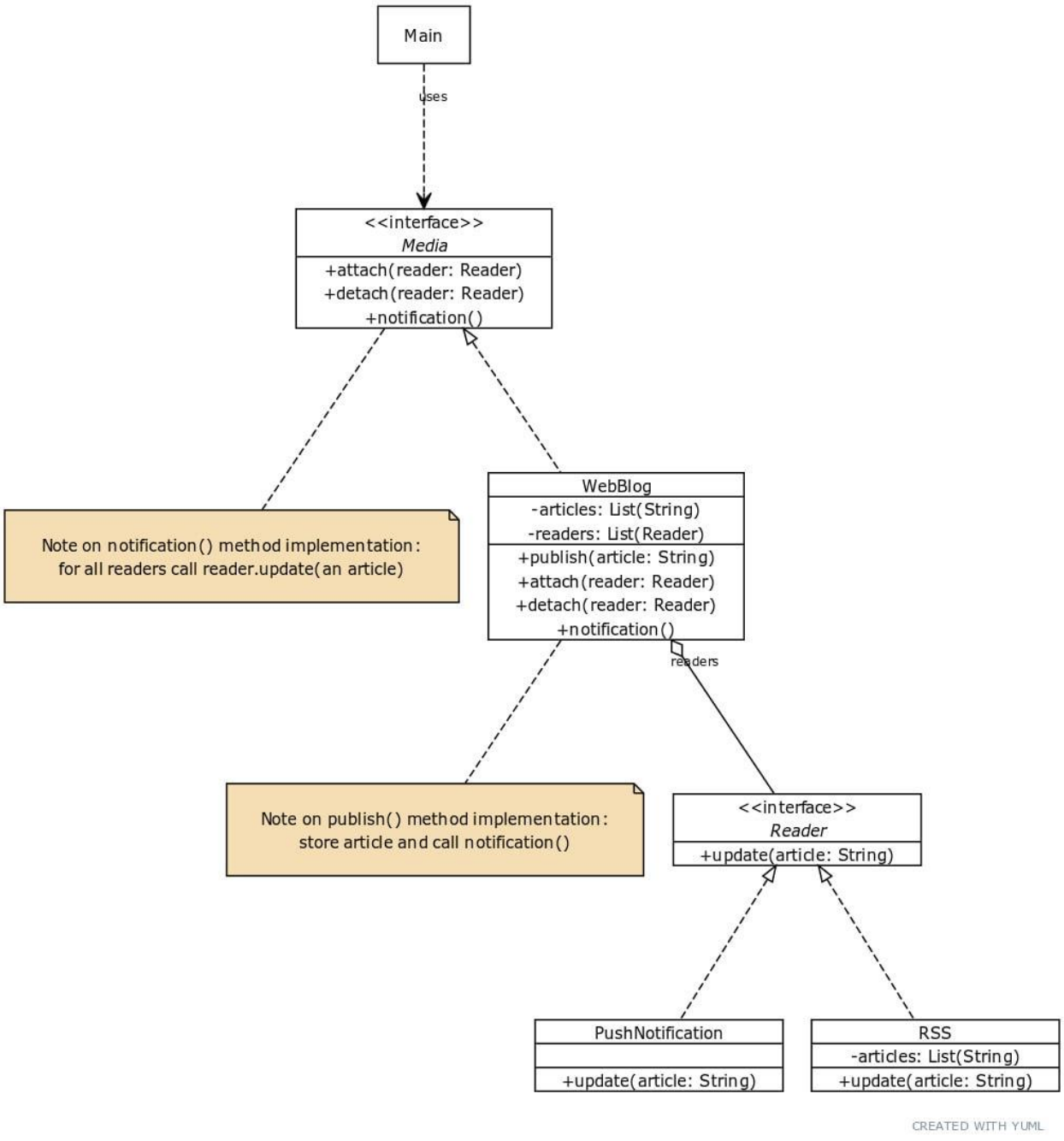
Webové blogy (WebBlog) umožňujú publikovať (metódou publish()) a udržiavať zoznam všetkých publikovaných článkov a podporujú sledovanie článkov (implementovaním všetkých metód rozhrania Media) čitateľmi. Čitateľ (Reader) je aktualizovaný (metódou upadte()) po publikovaní každého článku. Implementujte dva druhy čitateľov.

Prvý čitateľ (PushNotification) dostane výpis každého článku webového blogu, ktorý je práve publikovaný. Výpis realizujte výpisom článku na štandardný výstup.

Druhý čitateľ (RSS) dostane výpis o skupine posledných piatich publikovaných článkov. U tohto čitateľa použite zoznam, na priebežné udržiavanie aktualne publikovaných článkov. Keď sa zoznam naplní piatimi článkami, vykoná sa výpis a zoznam sa vymaže. Výpis realizujte výpisom pätice článkov na štandardný výstup.

V priloženom Main súbore vytvorte metódu main() (vstupný bod programu), v ktorej použijete vašu implementáciu nasledovne:

V programe (Main) vytvorte objekt webového blogu, ku ktorému pripojte (metódou attach()) obidva druhy čitateľov. Simulujte funkčnosť systému publikovaním rôznych článkov. Prvý čitateľ (PushNotification) musí vypísať článok hneď po jeho publikovaní. Druhý čitateľ (RSS) musí vypísať vždy skupinu piatich článkov až po publikovaní piateho z nich.



```
package sk.tuke.kpi.oop.exams;

import java.util.ArrayList;
import java.util.List;

// Rozhranie Media
public interface Media {
    void attach(Reader reader);
    void detach(Reader reader);
    void notification();
}

// Rozhranie Reader
public interface Reader {
    void update(String article);
}

// Trieda WebBlog implementujúca rozhranie Media
public class WebBlog implements Media {
    private List<String> articles;
    private List<Reader> readers;

    public WebBlog() {
        this.articles = new ArrayList<>();
        this.readers = new ArrayList<>();
    }
```

```
    public void publish(String article) {
        articles.add(article);
        notification(); // Zavoláme notifikáciu po publikovaní článku
    }

    @Override
    public void attach(Reader reader) {
        readers.add(reader);
    }

    @Override
    public void detach(Reader reader) {
        readers.remove(reader);
    }

    @Override
    public void notification() {
        for (Reader reader : readers) {
            reader.update(articles.get(articles.size() - 1)); // Posledný publikovaný článok
        }
    }
}

// Trieda PushNotification implementujúca rozhranie Reader
public class PushNotification implements Reader {
    @Override
```

```
    public void update(String article) {
        System.out.println("PushNotification: New article published -> " + article);
    }
}

// Trieda RSS implementujúca rozhranie Reader
public class RSS implements Reader {
    private List<String> articles;

    public RSS() {
        this.articles = new ArrayList<>();
    }

    @Override
    public void update(String article) {
        articles.add(article);
        if (articles.size() == 5) { // Ak je v zozname 5 článkov, vypíše ich
            System.out.println("RSS: Last 5 articles -> " + articles);
            articles.clear(); // Vymaže zoznam po výpise
        }
    }
}

// Trieda Main
public class Main {
    public static void main(String[] args) {
        WebBlog blog = new WebBlog();
```

```
        PushNotification pushNotification = new PushNotification();
        RSS rss = new RSS();

        blog.attach(pushNotification);
        blog.attach(rss);

        blog.publish("Article 1: Introduction to Observer Pattern");
        blog.publish("Article 2: Implementing Interfaces in Java");
        blog.publish("Article 3: Advanced Java Programming");
        blog.publish("Article 4: Understanding Streams");
        blog.publish("Article 5: Dependency Injection");
        blog.publish("Article 6: Design Patterns in Practice");
    }
}
```

```
package sk.tuke.kpi.oop.exams;

import java.util.List;
```

```
public interface Media {
    void attach(Reader reader);
    void detach(Reader reader);
    void notification();
}

public interface Reader {
    void update(String article);
}
```

```
package sk.tuke.kpi.oop.exams;

import java.util.ArrayList;
import java.util.List;

public class WebBlog implements Media {
```

```
private List<String> articles;
private List<Reader> readers;

public WebBlog() {
    this.articles = new ArrayList<>();
    this.readers = new ArrayList<>();
}

public void publish(String article) {
    this.articles.add(article);
    this.notification();
}

@Override
public void attach(Reader reader) {
    this.readers.add(reader);
}

@Override
public void detach(Reader reader) {
    this.readers.remove(reader);
}

@Override
public void notification() {
    for (Reader reader : this.readers) {
        reader.update(this.articles.get(this.articles.size() - 1));
```

```
        }
    }
}
```

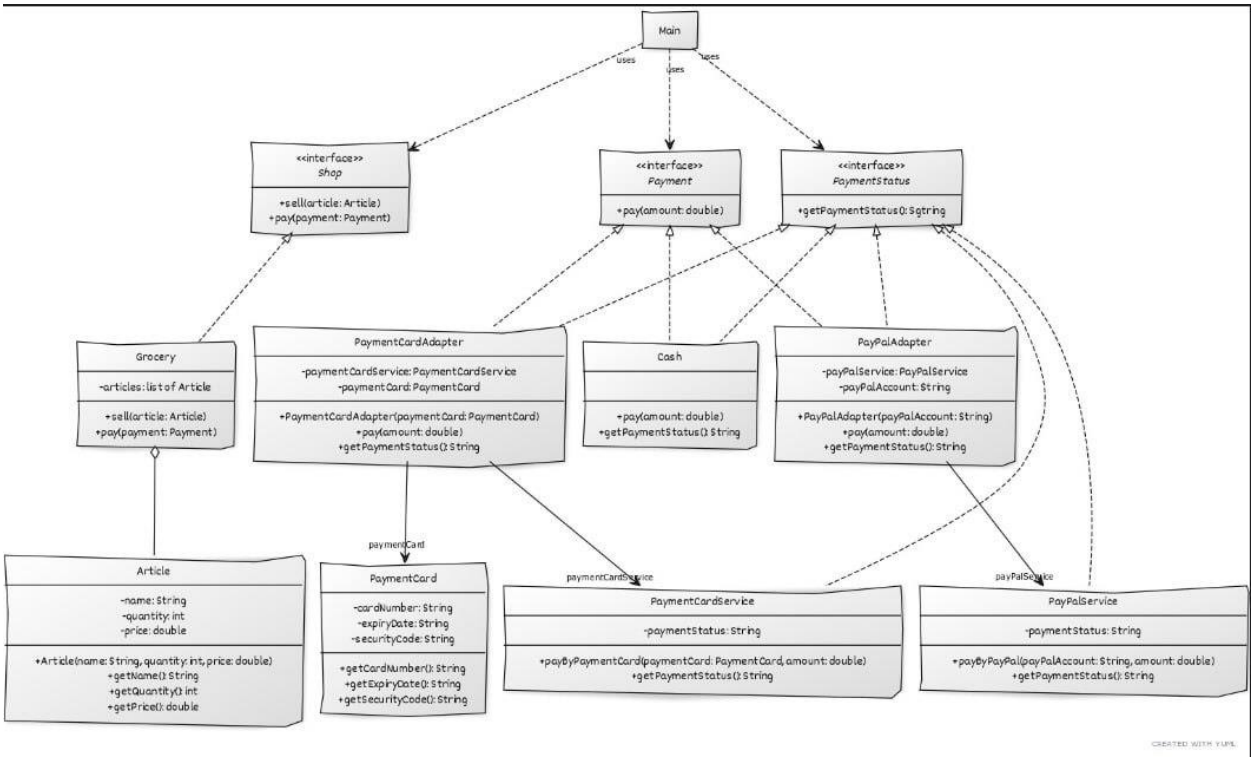
```
package sk.tuke.kpi.oop.exams;

public class PushNotification implements Reader {
    @Override
    public void update(String article) {
        System.out.println("PushNotification: " + article);
    }
}
```

```
package sk.tuke.kpi.oop.exams;
```

```
import java.util.ArrayList;
import java.util.List;

public class RSS implements Reader {
    private List<String> articles;

    public RSS() {
        this.articles = new ArrayList<>();
    }

    @Override
    public void update(String article) {
        this.articles.add(article);
        if (this.articles.size() == 5) {
            System.out.println("RSS: " + this.articles);
            this.articles.clear();
        }
    }
}
```

```
package sk.tuke.kpi.oop.exams;

public class Main {
    public static void main(String[] args) {
        WebBlog blog = new WebBlog();
        Reader pushNotification = new PushNotification();
        Reader rss = new RSS();
```

```
        blog.attach(pushNotification);
        blog.attach(rss);

        blog.publish("Article 1");
        blog.publish("Article 2");
        blog.publish("Article 3");
        blog.publish("Article 4");
        blog.publish("Article 5");
        blog.publish("Article 6");
    }
}
```




```
package sk.tuke.kpi.oop.exams;
```

```
import java.util.ArrayList;
import java.util.List;

interface Payment {
    void pay(double amount);
}

interface PaymentStatus {
    String getPaymentStatus();
}

class Cash implements Payment, PaymentStatus {
    private String paymentStatus;

    @Override
    public void pay(double amount) {
        this.paymentStatus = "Cash: " + amount;
        System.out.println(this.paymentStatus);
    }

    @Override
    public String getPaymentStatus() {
        return this.paymentStatus;
    }
}
```

```
class PayPalService implements PaymentStatus {
    public String paymentStatus;

    public void payByPayPal(String payPalAccount, double amount) {
        this.paymentStatus = "PayPal: " + payPalAccount + ", " + amount;
        System.out.println(this.paymentStatus);
    }

    @Override
    public String getPaymentStatus() {
        return paymentStatus;
    }
}

class PaymentCardService implements PaymentStatus {
    public String paymentStatus;

    public void payByPaymentCard(PaymentCard paymentCard, double amount) {
        this.paymentStatus = "Payment card: " + paymentCard.getCardNumber() + ", " +
                paymentCard.getExpiryDate() + ", " + paymentCard.getSecurityCode() + ", " +
amount;
        System.out.println(this.paymentStatus);
    }

    @Override
    public String getPaymentStatus() {
        return paymentStatus;
```

```
    }
}

class PayPalAdapter implements Payment, PaymentStatus {
    private final PayPalService payPalService;
    private final String payPalAccount;

    public PayPalAdapter(String payPalAccount) {
        this.payPalService = new PayPalService();
        this.payPalAccount = payPalAccount;
    }

    @Override
    public void pay(double amount) {
        payPalService.payByPayPal(payPalAccount, amount);
    }

    @Override
    public String getPaymentStatus() {
        return payPalService.getPaymentStatus();
    }
}

class PaymentCardAdapter implements Payment, PaymentStatus {
    private final PaymentCardService paymentCardService;
    private final PaymentCard paymentCard;
```

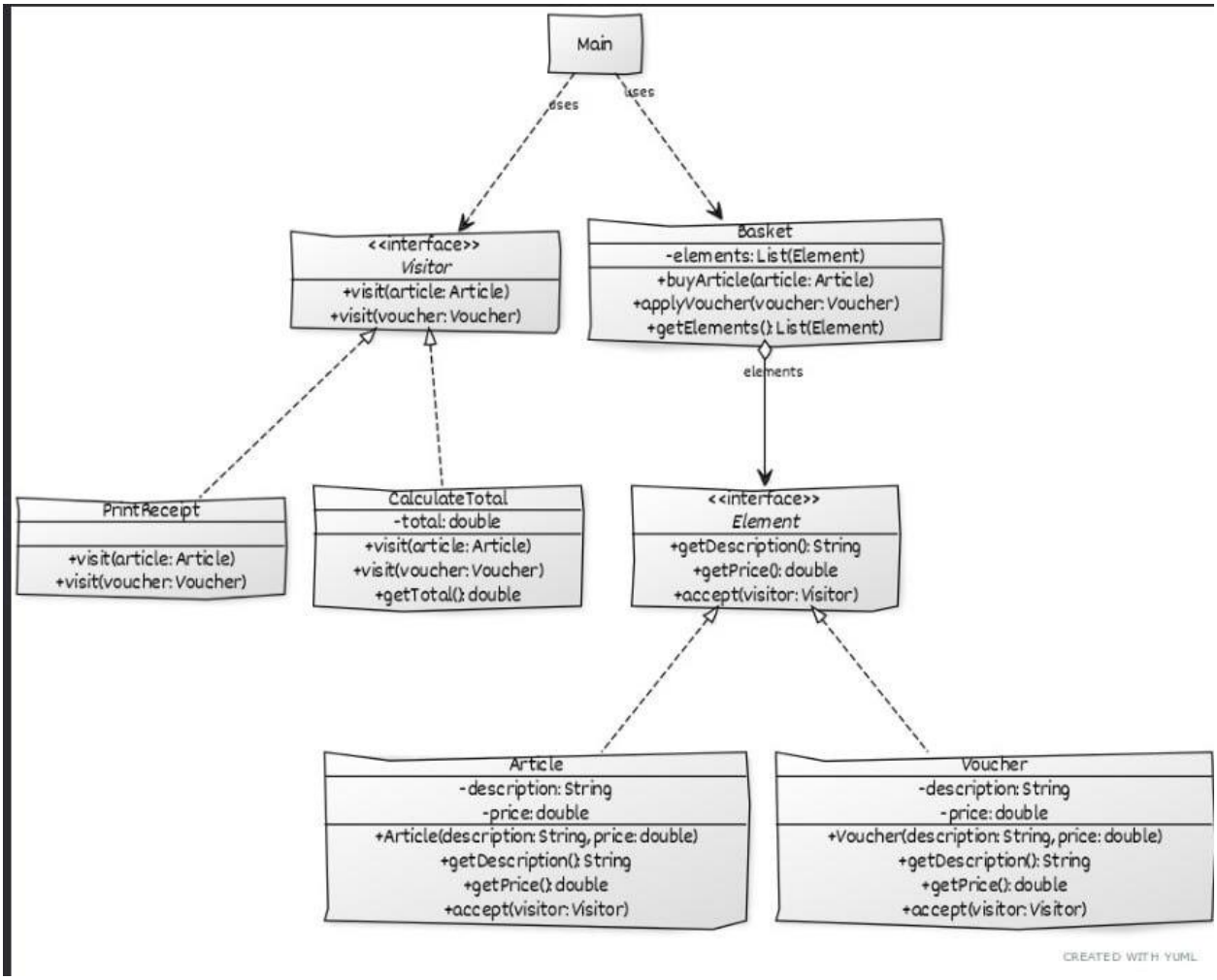
```
    public PaymentCardAdapter(PaymentCard paymentCard) {
        this.paymentCardService = new PaymentCardService();
        this.paymentCard = paymentCard;
    }

    @Override
    public void pay(double amount) {
        paymentCardService.payByPaymentCard(paymentCard, amount);
    }

    @Override
    public String getPaymentStatus() {
        return paymentCardService.getPaymentStatus();
    }
}

class PaymentCard {
    private final String cardNumber;
    private final String expiryDate;
    private final String securityCode;

    public PaymentCard(String cardNumber, String expiryDate, String securityCode) {
        this.cardNumber = cardNumber;
        this.expiryDate = expiryDate;
        this.securityCode = securityCode;
    }
```

```
    public String getCardNumber() {
        return cardNumber;
    }

    public String getExpiryDate() {
        return expiryDate;
    }

    public String getSecurityCode() {
        return securityCode;
    }
}

class Article {
    private final String name;
    private final int quantity;
    private final double price;

    public Article(String name, int quantity, double price) {
        this.name = name;
        this.quantity = quantity;
        this.price = price;
    }

    public String getName() {
        return name;
    }
```

```
    public int getQuantity() {
        return quantity;
    }

    public double getPrice() {
        return price;
    }
}

interface Shop {
    void sell(Article article);
    void pay(Payment payment);
}

class Grocery implements Shop {
    private final List<Article> articles;

    public Grocery() {
        this.articles = new ArrayList<>();
    }

    @Override
    public void sell(Article article) {
        articles.add(article);
        System.out.println("Added article: " + article.getName() + " (" + article.getQuantity()
+ " pcs)");
```

```
    }

    @Override
    public void pay(Payment payment) {
        double totalAmount = articles.stream()
                .mapToDouble(article -> article.getPrice() * article.getQuantity())
                .sum();
        payment.pay(totalAmount);
        articles.clear();
    }
}

public class Main {
    public static void main(String[] args) {
        Grocery grocery = new Grocery();

        grocery.sell(new Article("Milk", 1, 1.50));
        grocery.sell(new Article("Bread", 1, 2.00));

        Cash cash = new Cash();
        grocery.pay(cash);
        System.out.println("Payment status: " + cash.getPaymentStatus());

        PayPalAdapter payPalAdapter = new PayPalAdapter("user@paypal.com");
        grocery.sell(new Article("Cheese", 1, 3.00));
        grocery.pay(payPalAdapter);
```

```
        System.out.println("Payment status: " + payPalAdapter.getPaymentStatus());


        PaymentCard paymentCard = new PaymentCard("1234-5678-9012-3456", "12/25",
"123");
        PaymentCardAdapter cardAdapter = new PaymentCardAdapter(paymentCard);
        grocery.sell(new Article("Juice", 1, 2.50));
        grocery.pay(cardAdapter);
        System.out.println("Payment status: " + cardAdapter.getPaymentStatus());


        // Проверка всех методов, чтобы избежать предупреждений
        cash.getPaymentStatus();
        payPalAdapter.getPaymentStatus();
        cardAdapter.getPaymentStatus();
    }
}
```


Main
uses
uses
<<interface>>
Visitor
+visit(article: Article)
+visit(voucher: Voucher)
Basket
-elements: List(Element)
+buyArticle(article: Article)
+applyVoucher(voucher: Voucher)
+getElements(): List(Element)
elements
PrintReceipt
+visit(article: Article)
+visit(voucher: Voucher)
CalculateTotal
-total: double
+visit(article: Article)
+visit(voucher: Voucher)
+getTotal(): double
<<interface>>
Element
+getDescription(): String
+getPrice(): double
+accept(visitor: Visitor)
Article
-description: String
-price: double
+Article(description: String, price: double)
+getDescription(): String
+getPrice(): double
+accept(visitor: Visitor)
Voucher
-description: String
-price: double
+Voucher(description: String, price: double)
+getDescription(): String
+getPrice(): double
+accept(visitor: Visitor)
CREATED WITH YUML

Skúška z predmetu Objektovo orientované programovanie

Termín: 17. 1. 2025 10.00

1. časť: Implementácia úlohy s návrhovým vzorom

Použite návrhový vzor Builder na realizáciu nasledovného príkladu:

Diagram

Schma implementcie lohy

▸ Otvoriť diagram samostatne

Pri implementácii dbajte na dodržiavanie názvov identifikátorov, ako sú uvedené v zadaní a diagrame. Všetky súbory vašej implementácie umiestňujte priamo do pripraveného balíka sk.tuke.kpi.oop.exams.

V Charlieho továrni na čokoládu je riadiace centrum nových chutí (NewTastesCenter) zodpovedné za výrobu exkluzívnych čokolád s rôznymi náplňami a príchuťami (Chocolate). Každú tabuľku čokolády (Chocolate) tvorí niekoľko základných prvkov:

základná čokoládová hmota (base),

náplň (filling), a

posýpka (sprinkle).

Na reprezentáciu jednotlivých prvkov objektu čokolády použite textové reťazce. Charlieho továreň na čokoládu využíva na vytvorenie čokolád skladacie triedy, ktoré

implementujú rozhranie Builder deklarujúce operácie na vyskladanie čokolády z jednotlivých častí. Skladacie triedy si uchovávajú informáciu o každej vytvorenej časti a objekt čokolády je vytvorený z týchto častí, až keď sú všetky z nich vytvorené skladacou triedou príslušnými operáciami.

Čokoládu „Vanilla Dream“ (VanillaDream) tvorí:

základná čokoládová hmota reprezentovaná reťazcom „cocoa butter with cream“,

náplň reťazcom „vanilla pod“,

posýpka „almond flakes“.

Čokoláda „Cherry Secret“ (CherrySecret) nech má

základnú čokoládovú hmotu reprezentovanú reťazcom „cocoa mass with 80 % of cocoa“,

náplň reťazcom „cherry puree“,

posýpku reťazcom „chilli pepper flakes“.

Riadiace centrum nových chutí Charlieho továrne (NewTastesCenter) implementuje rozhranie Director deklarujúce operáciu na zostavenie čokolády pomocou objektu typu Builder.

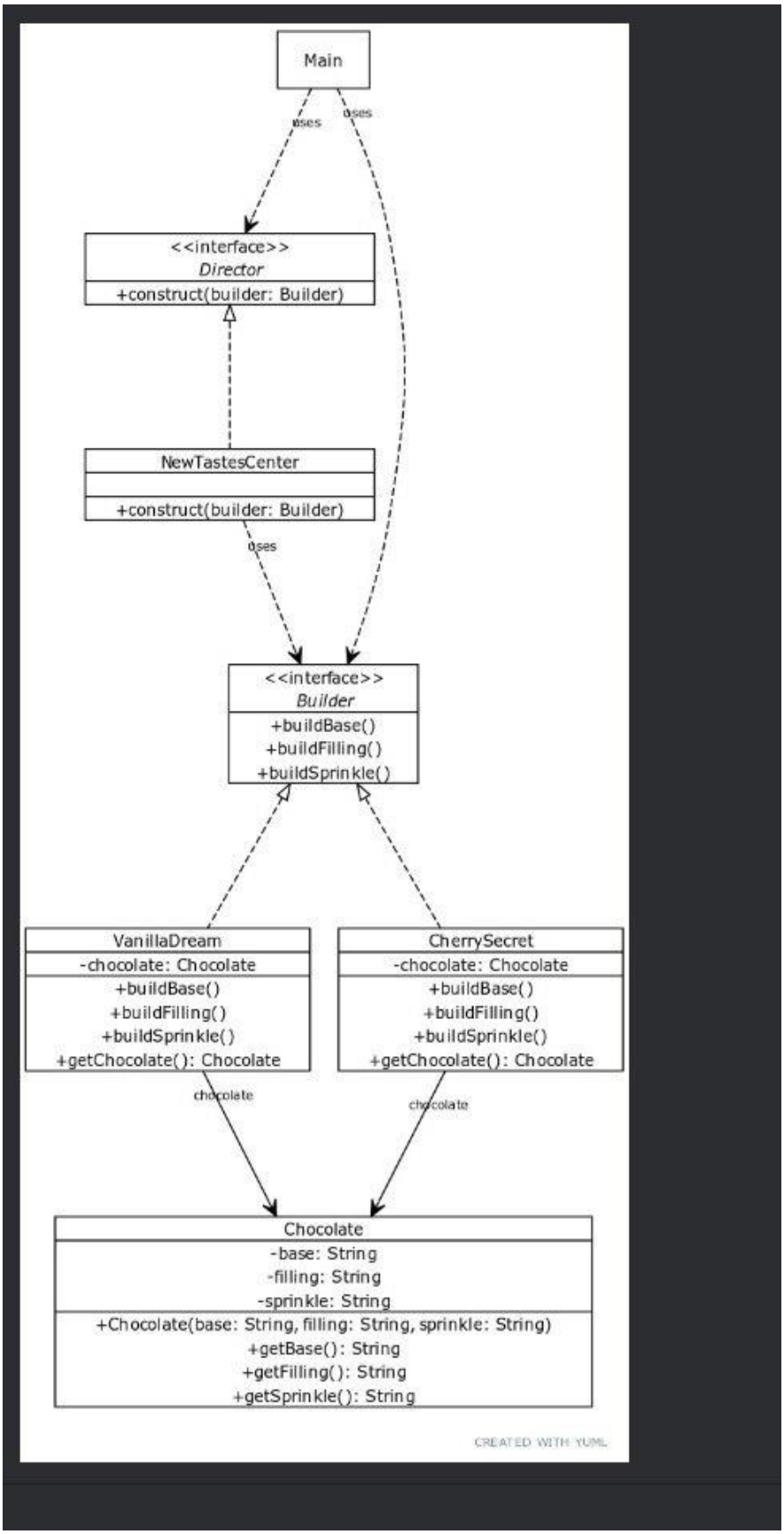
V priloženom Main súbore vytvorte metódu main() (vstupný bod programu), v ktorej použijete vašu implementáciu nasledovne:

Vytvorte dve čokolády pomocou skladacích tried VanillaDream a CherrySecret s využitím NewTastesCenter.

2. časť: Test

Vypracujte nasledovný test v systéme Moodle. Heslo do testu je skuska171

Pre prístup to kurzu OOP v systéme Moodle použite prihlasovací kľúč
OOP.skuska.2024


Main
uses
uses
<<interface>>
Director
+construct(builder: Builder)
NewTastesCenter
+construct(builder: Builder)
uses
<<interface>>
Builder
+buildBase()
+buildFilling()
+buildSprinkle()
VanillaDream
-chocolate: Chocolate
+buildBase()
+buildFilling()
+buildSprinkle()
+getChocolate(): Chocolate
CherrySecret
-chocolate: Chocolate
+buildBase()
+buildFilling()
+buildSprinkle()
+getChocolate(): Chocolate
chocolate
chocolate
Chocolate
-base: String
-filling: String
-sprinkle: String
+Chocolate(base: String, filling: String, sprinkle: String)
+getBase(): String
+getFilling(): String
+getSprinkle(): String
CREATED WITH YUML

```
Director director = new NewTastesCenter();
        VanillaDream vanillaBuilder = new VanillaDream();
        CherrySecret cherryBuilder = new CherrySecret();

        director.construct(vanillaBuilder);
        Chocolate vanillaChocolate = vanillaBuilder.getChocolate();

        director.construct(cherryBuilder);
        Chocolate cherryChocolate = cherryBuilder.getChocolate();



public interface Builder {
    void buildBase();
    void buildFilling();
    void buildSprinkle();
}

public Chocolate getChocolate() {
        if (chocolate == null) {
            chocolate = new Chocolate(base, filling, sprinkle);
        }
        return chocolate;
    }
```