Code Smells.

Organizačné

- Budúci týždeň 6.máj - nebude prednáška
- Uvidíme sa 14. mája na oceňovaní projektov

OOP FIIT Awards!
14. Mája 9:00
Ocenenie najlepších a najzaujímavejších projektov

Aj vecné ceny :)

Code smells

Opis Code Smell

Ukážka v kóde

(Jedno z) Možných riešení

Bloaters

Long Method. Primitive obsession. Data Clumps. Large Class. Long Parameter List.

Long Method

Problém

- Metóda, ktorá obsahuje príliš veľa riadkov kódu.
- Všeobecne viac ako desať riadkov kódu v rámci metódy je hranica.

```java
private void playGame(int mode) {
    // Initialize game
    System.out.println("Welcome to the game!");
    Scanner scanner = new Scanner(System.in);
    int playerScore = 0;
    int computerScore = 0;
    int round = 1;
    int maxRounds = 5;
    boolean gameRunning = true;
    String[] choices = {"Rock", "Paper", "Scissors"};
    Random random = new Random();

    // Set difficulty based on mode
    int difficulty;
    if (mode == 1) {
        difficulty = 1;
        System.out.println("Easy mode selected");
        maxRounds = 3;
    } else if (mode == 2) {
        difficulty = 2;
        System.out.println("Medium mode selected");
        maxRounds = 5;
    } else if (mode == 3) {
        difficulty = 3;
        System.out.println("Hard mode selected");
        maxRounds = 7;
    } else {
        difficulty = 1;
        System.out.println("Default to easy mode");
        maxRounds = 3;
    }

    // Main game loop
    while (gameRunning && round <= maxRounds) {
        System.out.println("\n--- Round " + round + " ---");
        System.out.println("Choose: 1) Rock 2) Paper 3) Scissors");
        int playerChoice = scanner.nextInt() - 1;

        // Validate input
        if (playerChoice < 0 || playerChoice > 2) {
            System.out.println("Invalid choice! Try again.");
            continue;
        }

        // Computer makes choice
        int computerChoice;
        if (difficulty == 1) {
            computerChoice = random.nextInt(3);
        } else if (difficulty == 2) {
            computerChoice = random.nextInt(3);
            if (random.nextInt(100) < 30) {
                computerChoice = (playerChoice + 1) % 3;
            }
        } else {
            computerChoice = (playerChoice + 1) % 3;
            if (random.nextInt(100) < 20) {
                computerChoice = random.nextInt(3);
            }
        }

        System.out.println("You chose: " + choices[playerChoice]);
        System.out.println("Computer chose: " +
choices[computerChoice]);

        // Determine winner
        if (playerChoice == computerChoice) {
            System.out.println("It's a tie!");
        } else if ((playerChoice == 0 && computerChoice == 2) ||
                   (playerChoice == 1 && computerChoice == 0) ||
                   (playerChoice == 2 && computerChoice == 1)) {
            System.out.println("You win this round!");
            playerScore++;
        } else {
            System.out.println("Computer wins this round!");
            computerScore++;
        }

        // Display current score
        System.out.println("Score - You: " + playerScore + " |
Computer: " + computerScore);
        round++;
    }

    // Display final results
    System.out.println("\n=== GAME OVER ===");
    System.out.println("Final Score - You: " + playerScore + " |
Computer: " + computerScore);
    if (playerScore > computerScore) {
        System.out.println("Congratulations! You won the game!");
    } else if (computerScore > playerScore) {
        System.out.println("Computer won the game. Better luck next
time!");
    } else {
        System.out.println("The game ended in a tie!");
    }

    scanner.close();
}
```

```java
private void playGame(int mode) {
    // Initialize game
    System.out.println("Welcome to the game!");
    Scanner scanner = new Scanner(System.in);
    int playerScore = 0;
    int computerScore = 0;
    int round = 1;
    int maxRounds = 5;
    boolean gameRunning = true;
    String[] choices = {"Rock", "Paper", "Scissors"};
    Random random = new Random();

    // Set difficulty based on mode
    int difficulty;
    if (mode == 1) {
        difficulty = 1;
        System.out.println("Easy mode selected");
        maxRounds = 3;
    } else if (mode == 2) {
        difficulty = 2;
        System.out.println("Medium mode selected");
        maxRounds = 5;
    } else if (mode == 3) {
        difficulty = 3;
        System.out.println("Hard mode selected");
        maxRounds = 7;
    } else {
        difficulty = 1;
        System.out.println("Default to easy mode");
        maxRounds = 3;
    }

    // Main game loop
    while (gameRunning && round <= maxRounds) {
        System.out.println("\n--- Round " + round + " ---");
        System.out.println("Choose: 1) Rock 2) Paper 3) Scissors");
        int playerChoice = scanner.nextInt() - 1;

        // Validate input
        if (playerChoice < 0 || playerChoice > 2) {
            System.out.println("Invalid choice! Try again.");
            continue;
        }

        // Computer makes choice
        int computerChoice;
        if (difficulty == 1) {
            computerChoice = random.nextInt(3);
        } else if (difficulty == 2) {
            computerChoice = random.nextInt(3);
            if (random.nextInt(100) < 30) {
                computerChoice = (playerChoice + 1) % 3;
            }
        } else {
            computerChoice = (playerChoice + 1) % 3;
            if (random.nextInt(100) < 20) {
                computerChoice = random.nextInt(3);
            }
        }

        System.out.println("You chose: " + choices[playerChoice]);
        System.out.println("Computer chose: " +
choices[computerChoice]);

        // Determine winner
        if (playerChoice == computerChoice) {
            System.out.println("It's a tie!");
        } else if ((playerChoice == 0 && computerChoice == 2) ||
                   (playerChoice == 1 && computerChoice == 0) ||
                   (playerChoice == 2 && computerChoice == 1)) {
            System.out.println("You win this round!");
            playerScore++;
        } else {
            System.out.println("Computer wins this round!");
            computerScore++;
        }

        // Display current score
        System.out.println("Score - You: " + playerScore + " |
Computer: " + computerScore);
        round++;
    }

    // Display final results
    System.out.println("\n=== GAME OVER ===");
    System.out.println("Final Score - You: " + playerScore + " |
Computer: " + computerScore);
    if (playerScore > computerScore) {
        System.out.println("Congratulations! You won the game!");
    } else if (computerScore > playerScore) {
        System.out.println("Computer won the game. Better luck next
time!");
    } else {
        System.out.println("The game ended in a tie!");
    }

    scanner.close();
}
```

```java
private void playGame(int mode) {
    GameConfig config = configureGame(mode);
    GameState state = initializeGame();

    while (state.hasRoundsRemaining(config.getMaxRounds())) {
        playRound(state, config);
    }

    displayFinalResults(state);
}

private GameConfig configureGame(int mode) { /* ... */ }
private GameState initializeGame() { /* ... */ }
private void playRound(GameState state, GameConfig config) { /* ...
*/ }
private int getComputerChoice(int playerChoice, int difficulty) { /*
... */ }
private RoundResult determineWinner(int playerChoice, int
computerChoice) { /* ... */ }
private void displayFinalResults(GameState state) { /* ... */ }
```

Long Method

- Rozdeliť metódu do viacerých metód

Riešenie

```
void printOwing() {
  printBanner();

  // Print details.
  System.out.println("name: " + name);
  System.out.println("amount: " + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails(double outstanding) {
  System.out.println("name: " + name);
  System.out.println("amount: " + outstanding);
}
```

Primitive Obsession

- Používanie primitývných typov namiesto objektov.
- Procedurálny prístup namiesto objektového.
- Používanie rôzných prepínačov namiesto aktualizácie stavu objektov - ktorý si trieda aktualizuje vo svojej réžii.

```java
public class Customer {
    private String name;
    private String phoneCountryCode;  // "+1"
    private String phoneAreaCode;     // "415"
    private String phoneNumber;       // "5551234"

    private double balanceAmount;     // money as primitive
    private String balanceCurrency;   // "USD"

    private int customerType;         // 1=Regular, 2=Premium, 3=VIP

    public double applyDiscount(double price) {
        if (customerType == 1) return price;
        if (customerType == 2) return price * 0.9;
        if (customerType == 3) return price * 0.8;
        return price;
    }
}
```

```java
public class Money {
    private final double amount;
    private final String currency;

    public Money applyDiscount(double rate) {
        return new Money(amount * (1 - rate), currency);
    }
}

public enum CustomerType {
    REGULAR(0.0), PREMIUM(0.10), VIP(0.20);

    private final double discountRate;
    public double getDiscountRate() { return discountRate; }
}

public class Customer {
    private String name;
    private PhoneNumber phone;
    private Money balance;
    private CustomerType type;

    public Money applyDiscount(Money price) {
        return price.applyDiscount(type.getDiscountRate());
    }
}
```

```java
public class Customer {
    private String name;
    private String phoneCountryCode;  // "+1"
    private String phoneAreaCode;     // "415"
    private String phoneNumber;       // "5551234"

    private double balanceAmount;     // money as primitive
    private String balanceCurrency;   // "USD"

    private int customerType;         // 1=Regular, 2=Premium, 3=VIP

    public double applyDiscount(double price) {
        if (customerType == 1) return price;
        if (customerType == 2) return price * 0.9;
        if (customerType == 3) return price * 0.8;
        return price;
    }
}
```

```java
public class Money {
    private final double amount;
    private final String currency;

    public Money applyDiscount(double rate) {
        return new Money(amount * (1 - rate), currency);
    }
}

public enum CustomerType {
    REGULAR(0.0), PREMIUM(0.10), VIP(0.20);

    private final double discountRate;
    public double getDiscountRate() { return discountRate; }
}

public class Customer {
    private String name;
    private PhoneNumber phone;
    private Money balance;
    private CustomerType type;

    public Money applyDiscount(Money price) {
        return price.applyDiscount(type.getDiscountRate());
    }
}
```

Data Clumps

- V kóde sa nachádzajú skupiny premenných na viacerých miestách - údaje používateľa, polia adresy, pripojenie k databáze...
- Takéto skupiny, majú byť samostatnou triedou

```java
public class OrderService {

    public void createOrder(String street, String city,
                            String zipCode, String country) {
        validateAddress(street, city, zipCode, country);
        saveOrder(street, city, zipCode, country);
    }

    public void shipOrder(String street, String city,
                          String zipCode, String country) {
        System.out.println("Shipping to: " + street + ", " +
                           city + ", " + zipCode + ", " + country);
    }

    private void validateAddress(String street, String city,
                                 String zipCode, String country) {
        if (street == null || city == null ||
            zipCode == null || country == null) {
            throw new IllegalArgumentException("Invalid address");
        }
    }
}
```

```java
public class Address {
    private final String street;
    private final String city;
    private final String zipCode;
    private final String country;

    public Address(String street, String city, String zipCode, String country) {
        if (street == null || city == null ||
            zipCode == null || country == null) {
            throw new IllegalArgumentException("Invalid address");
        }
        this.street = street;
        this.city = city;
        this.zipCode = zipCode;
        this.country = country;
    }

    public String formatted() {
        return street + ", " + city + ", " + zipCode + ", " + country;
    }
}
```

Validácia Adresy


public class OrderService {
public void createOrder(Address address) {
saveOrder(address);
}
public void shipOrder(Address address) {
System.out.println("Shipping to: " + address.formatted());
}
}

```java
public class OrderService {

    public void createOrder(String street, String city,
                            String zipCode, String country) {
        validateAddress(street, city, zipCode, country);
        saveOrder(street, city, zipCode, country);
    }

    public void shipOrder(String street, String city,
                          String zipCode, String country) {
        System.out.println("Shipping to: " + street + ", " +
                           city + ", " + zipCode + ", " + country);
    }

    private void validateAddress(String street, String city,
                                 String zipCode, String country) {
        if (street == null || city == null ||
            zipCode == null || country == null) {
            throw new IllegalArgumentException("Invalid address");
        }
    }
}
```

```java
public class Address {
    private final String street;
    private final String city;
    private final String zipCode;
    private final String country;

    public Address(String street, String city, String zipCode, String country) {
        if (street == null || city == null ||
            zipCode == null || country == null) {
            throw new IllegalArgumentException("Invalid address");
        }
        this.street = street;
        this.city = city;
        this.zipCode = zipCode;
        this.country = country;
    }
```

```java
public class OrderService {

    public void createOrder(Address address) {
        saveOrder(address);
    }

    public void shipOrder(Address address) {
        System.out.println("Shipping to: " + address.formatted());
    }
}
```

Large Class

- Trieda obsahuje priveľa atribútov, metód a riadkov kódu
- Zvyčajne komplexný proces, ktorý by mohol byť rozdelený medzi viacero tried je celý v rámci jednej triedy

```java
class UserManager {
    // User data
    private String name;
    private String email;

    // Authentication
    public boolean login(String password) {
        System.out.println("Logging in...");
        return true;
    }

    public void logout() {
        System.out.println("Logging out...");
    }

    // Email functionality
    public void sendEmail(String message) {
        System.out.println("Sending email: " + message);
    }

    // Reporting
    public void generateReport() {
        System.out.println("Generating report...");
    }

    // Persistence
    public void saveToDatabase() {
        System.out.println("Saving user...");
    }
}
```

```java
class ReportService {
    public void generateReport(User user) {
        System.out.println("Generating report...");
    }
}
```

```java
class EmailService {
    public void sendEmail(User user, String message) {
        System.out.println("Sending email: " + message);
    }
}
```

```java
class User {
    String name;
    String email;
}
```

```java
class AuthService {
    public boolean login(User user, String password) {
        System.out.println("Logging in...");
        return true;
    }

    public void logout(User user) {
        System.out.println("Logging out...");
    }
}
```

```java
class App {
  User user = new User();

  AuthService auth = new AuthService();
  auth.login(user, "password");

  EmailService email = new EmailService();
  email.sendEmail(user, "Hello!");

  UserRepository repo = new UserRepository();
  repo.save(user);
}
```

```java
class UserManager {
    // User data
    private String name;
    private String email;

    // Authentication
    public boolean login(String password) {
        System.out.println("Logging in...");
        return true;
    }

    public void logout() {
        System.out.println("Logging out...");
    }

    // Email functionality
    public void sendEmail(String message) {
        System.out.println("Sending email: " + message);
    }

    // Reporting
    public void generateReport() {
        System.out.println("Generating report...");
    }

    // Persistence
    public void saveToDatabase() {
        System.out.println("Saving user...");
    }
}
```

```java
class User {
```

```java
class AuthService {
```

```java
class EmailService {
```

```java
class ReportService {
```

```java
class App {
  User user = new User();

  AuthService auth = new AuthService();
  auth.login(user, "password");

  EmailService email = new EmailService();
  email.sendEmail(user, "Hello!");

  UserRepository repo = new UserRepository();
  repo.save(user);
}
```

Long Parameter List

```
class OrderService {
    public void createOrder(
        String customerName,
        String customerEmail,
        String customerAddress,
        String productName,
        int quantity,
        double price,
        String discountCode
    ) {
        System.out.println("Creating order for " + customerName);
    }
}
```

```java
class OrderService {
    public void createOrder(
        String customerName,
        String customerEmail,
        String customerAddress,
        String productName,
        int quantity,
        double price,
        String discountCode
    ) {
        System.out.println("Creating order for " + customerName);
    }
}
```

```java
class Customer {
    String name;
    String email;
    String address;
}

class Product {
    String name;
    int quantity;
    double price;
}
```

```java
class OrderService {
    public void createOrder(Customer customer, Product product, String discountCode) {
        System.out.println("Creating order for " + customer.name);
    }
}
```

Object-Orientation Abusers Alternatívne tiredy s rôznymi interface. Refused Bequest. Temporary Field. Switch Statements.

Alternative Classes with Different Interfaces

```
class PayPalPayment {
    public void makePayment(double amount) {
        System.out.println("Paid " + amount + " using PayPal");
    }
}
```

```
class CreditCardPayment {
    public void pay(double value) {
        System.out.println("Paid " + value + " using Credit Card");
    }
}
```

```java
class PayPalPayment {
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using PayPal");
    }
}

class CreditCardPayment {
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using Credit Card");
    }
}
```

```java
interface Payment {
    void pay(double amount);
}
```

```java
class PayPalPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using PayPal");
    }
}

class CreditCardPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using Credit Card");
    }
}
```

- Zjednotiť názvy metód, ktoré robia to isté - interface

```java
class PayPalPayment {
    public void makePayment(double amount) {
        System.out.println("Paid " + amount + " using PayPal");
    }
}
```

```java
class CreditCardPayment {
    public void pay(double value) {
        System.out.println("Paid " + value + " using Credit Card");
    }
}
```

```java
interface Payment {
    void pay(double amount);
}
```

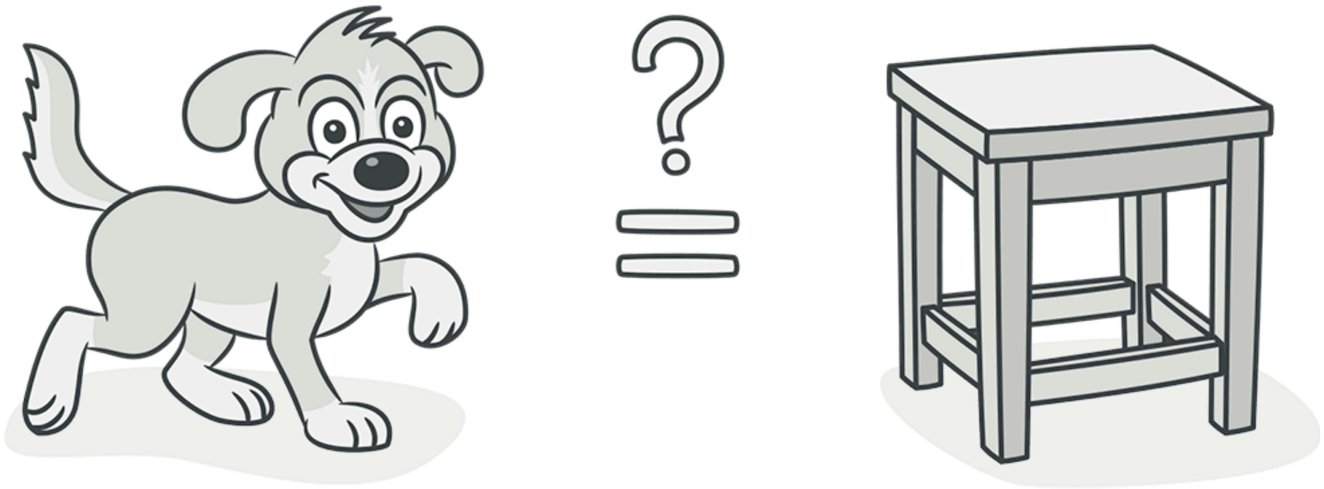
```java
class PayPalPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using PayPal");
    }
}

class CreditCardPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using Credit Card");
    }
}
```

Refused Bequest

```java
class Chair {
    public void sitOn() {
        System.out.println("Someone is sitting on the chair");
    }

    public int getNumberOfLegs() {
        return 4;
    }
}
```

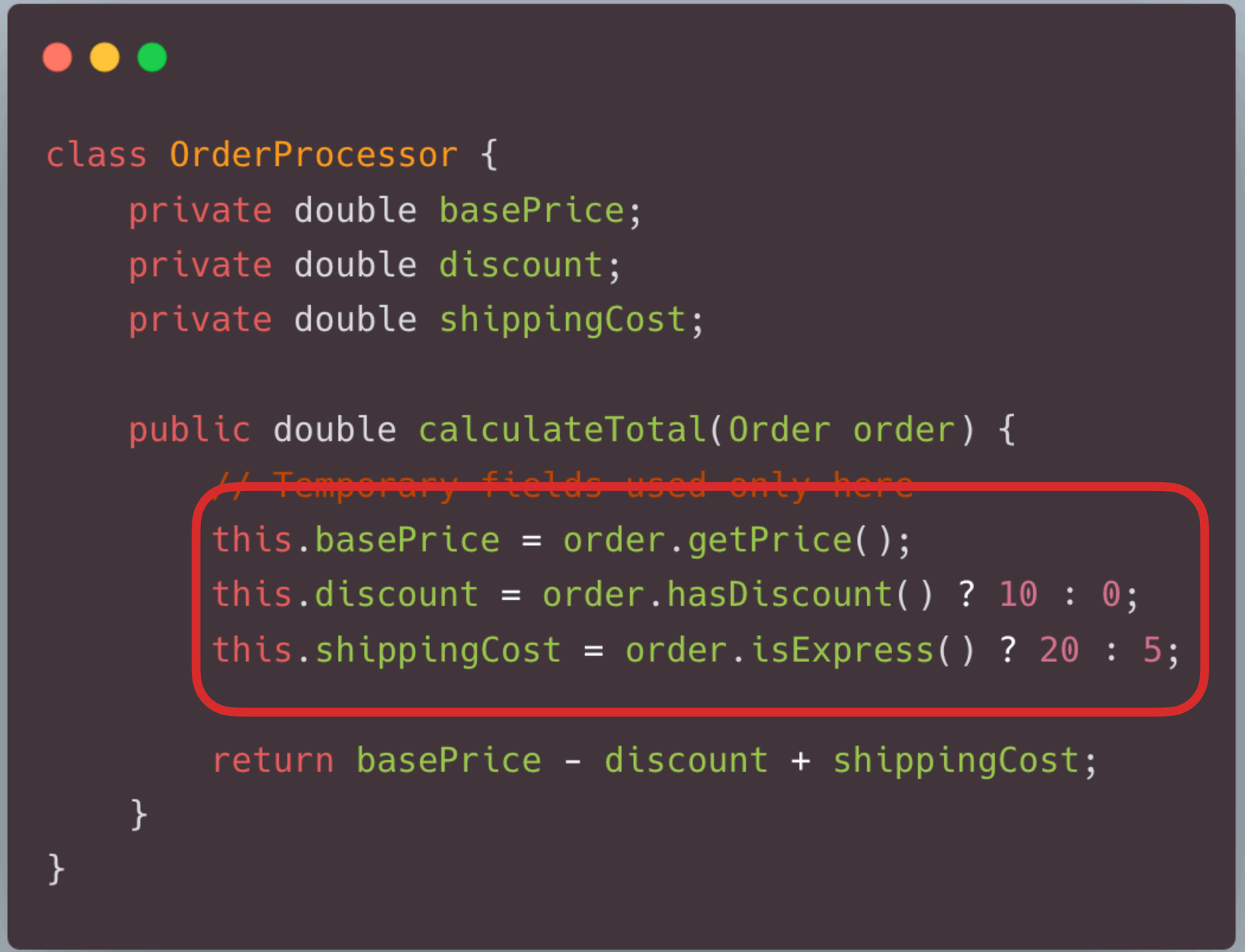
```java
class Dog extends Chair {
    public void bark() {
        System.out.println("Woof!");
    }

    @Override
    public void sitOn() {
        throw new UnsupportedOperationException("You shouldn't sit on a dog!");
    }
}
```

Refused Bequest



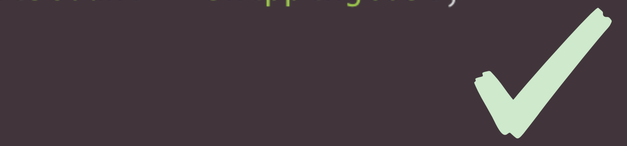
Temporary Field

- Dočasné premenné, ktoré sa používajú v rámci metódy - sú vytvorené ako atribúty v triede
- Namiesto použitia lokálných premenných
- **Tieto atribúty sa používajú len v rámci algoritmu a inak ostávajú prázdne**
- Ťakžko pochopiteľný kód


class OrderProcessor {
private double basePrice;
private double discount;
private double shippingCost;
public double calculateTotal(Order order) {
// Temporary fields used only here
this.basePrice = order.getPrice();
this.discount = order.hasDiscount() ? 10 : 0;
this.shippingCost = order.isExpress() ? 20 : 5;
return basePrice - discount + shippingCost;
}
}

```java
class OrderProcessor {
    private double basePrice;
    private double discount;
    private double shippingCost;

    public double calculateTotal(Order order) {
        // Temporary fields used only here
        this.basePrice = order.getPrice();
        this.discount = order.hasDiscount() ? 10 : 0;
        this.shippingCost = order.isExpress() ? 20 : 5;

        return basePrice - discount + shippingCost;
    }
}
```

```java
class OrderProcessor {
    public double calculateTotal(Order order) {
        double basePrice = order.getPrice();
        double discount = order.hasDiscount() ? 10 : 0;
        double shippingCost = order.isExpress() ? 20 : 5;

        return basePrice - discount + shippingCost;
    }
}
```



Switch Statements

- Používanie switch statement alebo if na definovanie rozdielneho správania

```
class SalaryCalculator {
    public double calculateSalary(String employeeType) {
        switch (employeeType) {
            case "FULL_TIME":
                return 5000;
            case "PART_TIME":
                return 3000;
            case "CONTRACTOR":
                return 4000;
            default:
                throw new IllegalArgumentException("Unknown type");
        }
    }
}
```

```
class SalaryCalculator {
    public double calculateSalary(String employeeType) {
        switch (employeeType) {
            case "FULL_TIME":
                return 5000;
            case "PART_TIME":
                return 3000;
            case "CONTRACTOR":
                return 4000;
            default:
                throw new IllegalArgumentException("Unknown type");
        }
    }
}
```

```
abstract class Employee {
    public abstract double calculateSalary();
}
```

```
class FullTimeEmployee extends Employee {
    public double calculateSalary() {
        return 5000;
    }
}

class PartTimeEmployee extends Employee {
    public double calculateSalary() {
        return 3000;
    }
}
```

Change Preventers

Divergent Change. Parallel Inheritance Hierarchies. Shotgun Surgery.

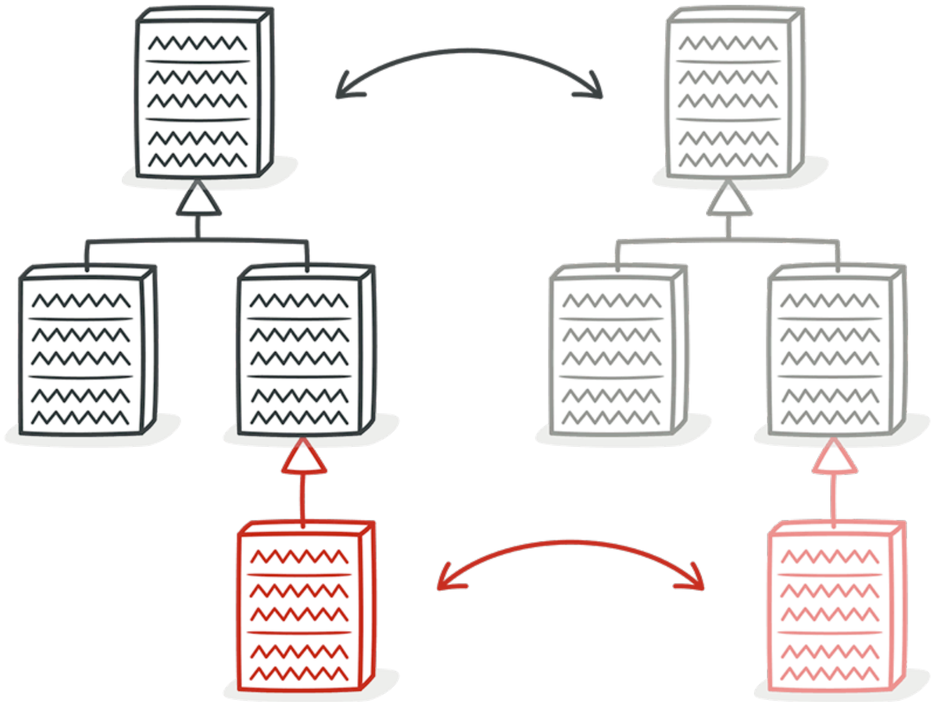
Divergent Change

- Pri zmenách v triede zistíte, že je potrebné upraviť aj ďalšie metódy
- Vôbec však spolu nesúvisia
- Opačný extrém - shotgun surgery

Shotgun Surgery

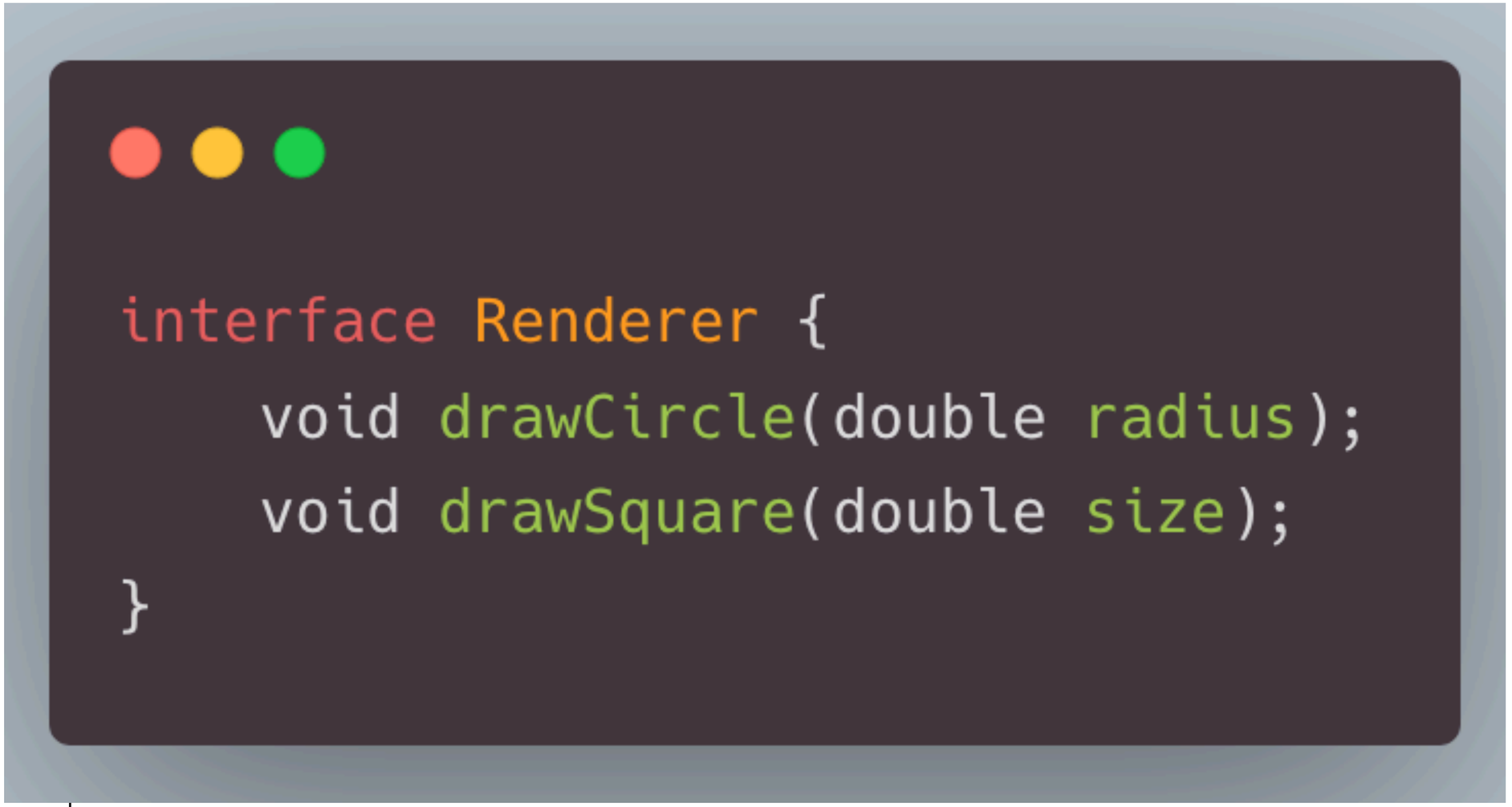
- Jedna úloha bola porozdelovaná medzi väčšie množstvo tried.
- Úprava tejto úlohy vyžaduje zásad do mnohých týchto tried

Parallel Inheritance Hierarchies



```
class Shape {}
class Circle extends Shape {}
class Square extends Shape {}
```

```
class Renderer {}
class CircleRenderer extends Renderer {}
class SquareRenderer extends Renderer {}
```


interface Renderer {
void drawCircle(double radius);
void drawSquare(double size);
}

```java
interface Renderer {
    void drawCircle(double radius);
    void drawSquare(double size);
}
```

```java
class VectorRenderer implements Renderer {
    public void drawCircle(double radius) {
        System.out.println("Drawing a circle with radius " + radius + " using vectors");
        // real implementation would draw using graphics API
    }

    public void drawSquare(double size) {
        System.out.println("Drawing a square with size " + size + " using vectors");
    }
}
```

```java
interface Renderer {
    void drawCircle(double radius);
    void drawSquare(double size);
}
```

```java
class VectorRenderer implements Renderer {
    public void drawCircle(double radius) {
        System.out.println("Drawing a circle with radius " + radius + " using vectors");
        // real implementation would draw using graphics API
    }

    public void drawSquare(double size) {
        System.out.println("Drawing a square with size " + size + " using vectors");
    }
}
```

```java
interface Renderer {
    void drawCircle(double radius);
    void drawSquare(double size);
}
```

```java
class VectorRenderer implements Renderer {
    public void drawCircle(double radius) {
        System.out.println("Drawing a circle with radius " + radius + " using vectors");
        // real implementation would draw using graphics API
    }

    public void drawSquare(double size) {
        System.out.println("Drawing a square with size " + size + " using vectors");
    }
}
```

```java
abstract class Shape {
    protected Renderer renderer;

    public Shape(Renderer renderer) {
        this.renderer = renderer;
    }

    public abstract void draw();
}
```

```java
interface Renderer {
    void drawCircle(double radius);
    void drawSquare(double size);
}
```

```java
class VectorRenderer implements Renderer {
    public void drawCircle(double radius) {
        System.out.println("Drawing a circle with radius " + radius + " using vectors");
        // real implementation would draw using graphics API
    }

    public void drawSquare(double size) {
        System.out.println("Drawing a square with size " + size + " using vectors");
    }
}
```

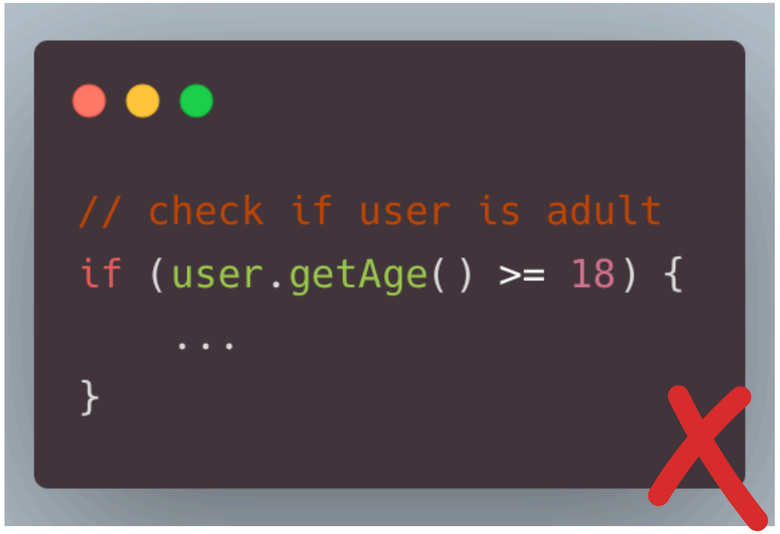
```java
abstract class Shape {
    protected Renderer renderer;

    public Shape(Renderer renderer) {
        this.renderer = renderer;
    }

    public abstract void draw();
}
```

```java
class Circle extends Shape {
    private double radius;

    public Circle(Renderer renderer, double radius) {
        super(renderer);
        this.radius = radius;
    }

    public void draw() {
        renderer.drawCircle(radius);
    }
}
```

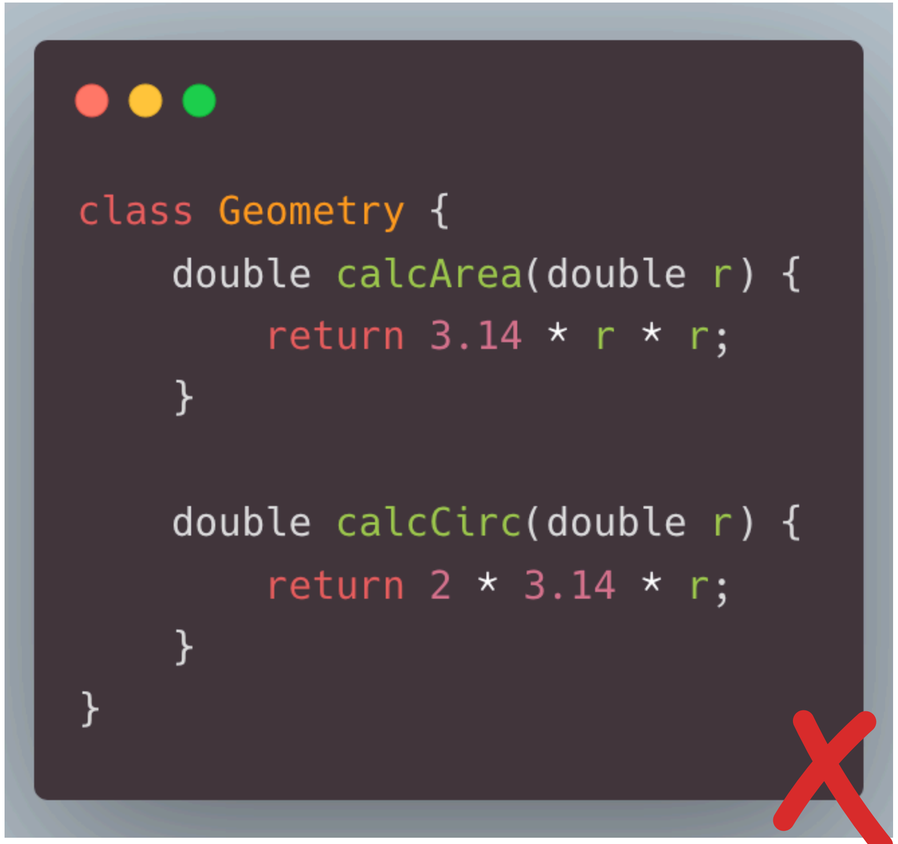
Dispensables

Comments. Data Class. Dead Code. Duplicate Code. Lazy Class. Speculative Generality.

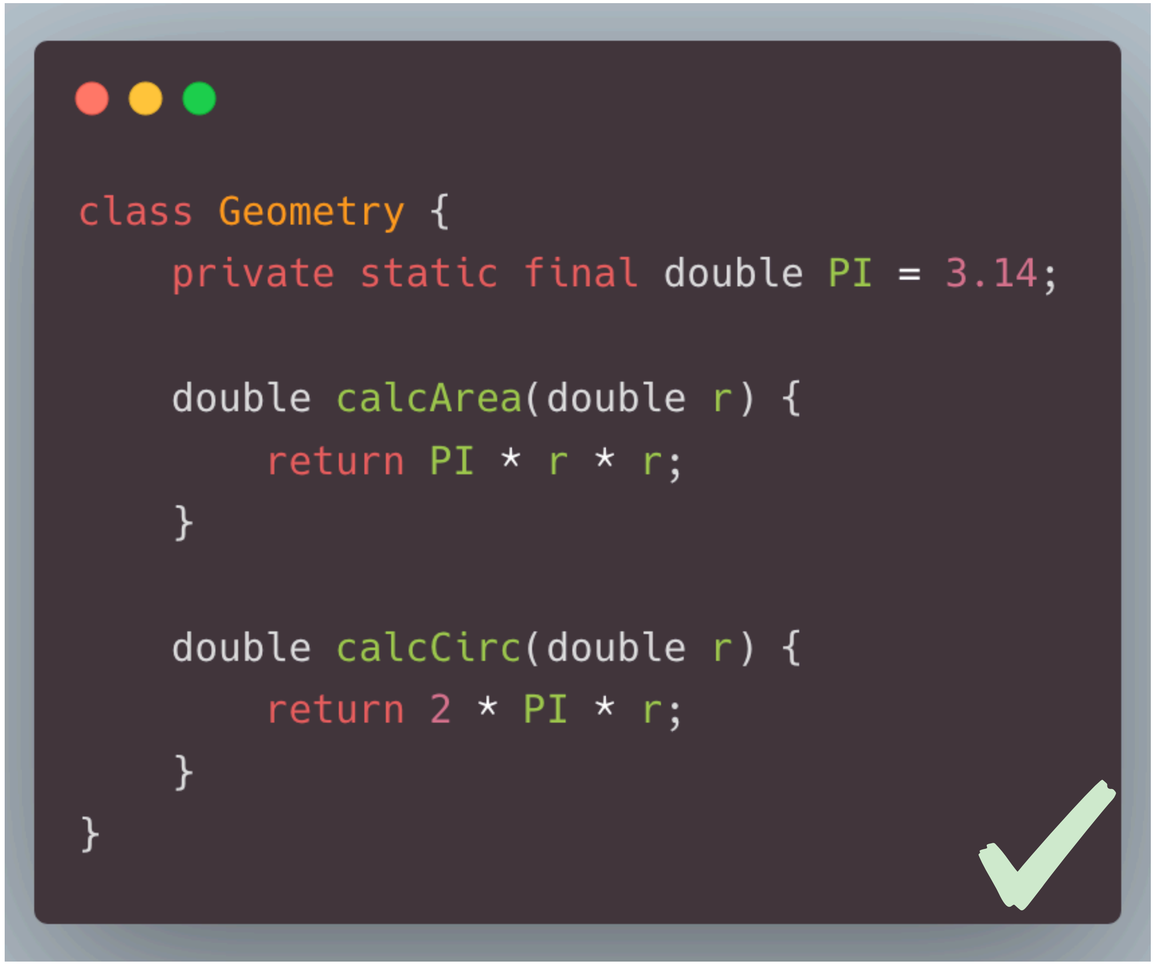
Comments



Duplicate Code



```java
class Geometry {
    double calcArea(double r) {
        return 3.14 * r * r;
    }

    double calcCirc(double r) {
        return 2 * 3.14 * r;
    }
}
```



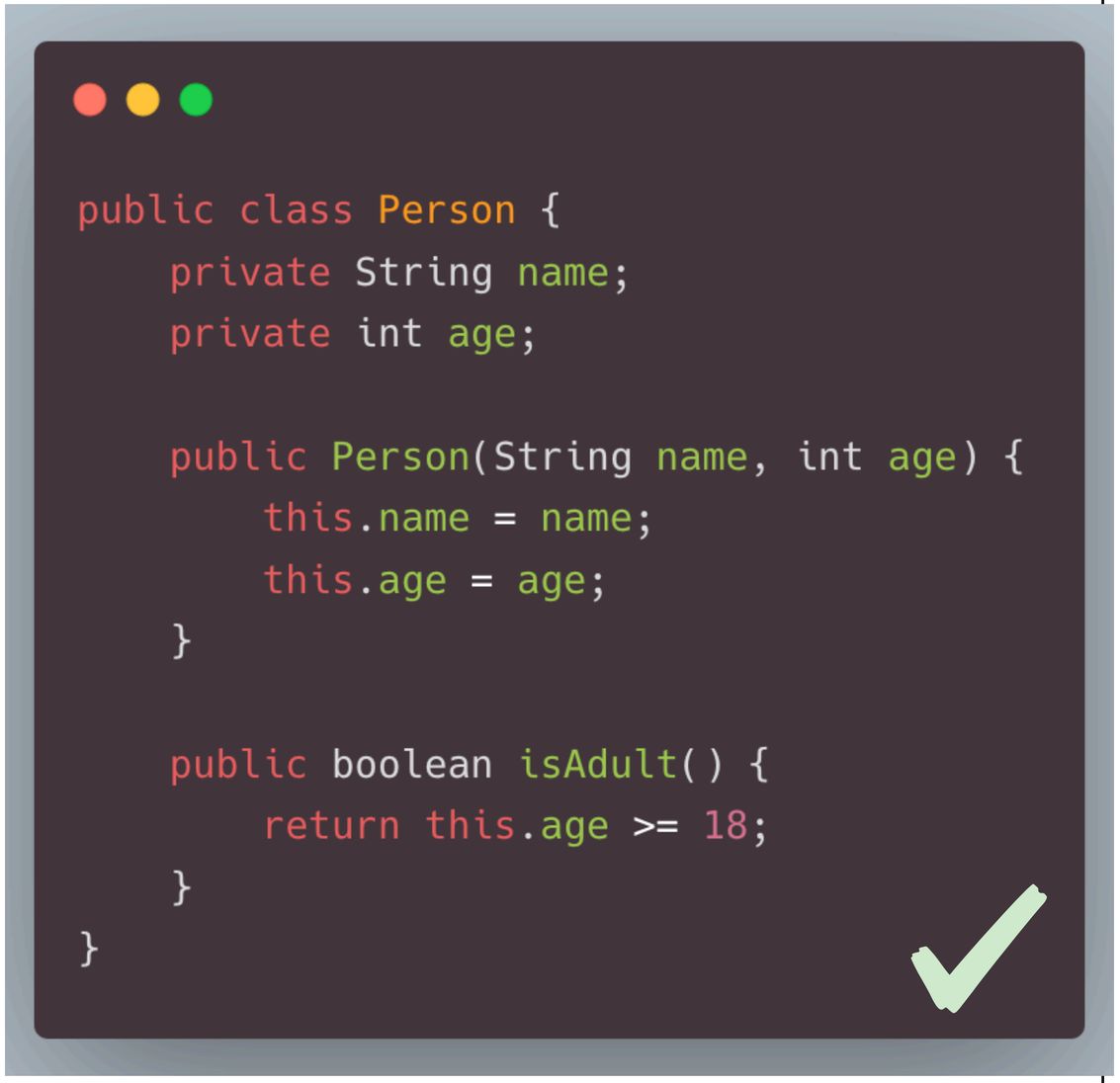
```java
class Geometry {
    private static final double PI = 3.14;

    double calcArea(double r) {
        return PI * r * r;
    }

    double calcCirc(double r) {
        return 2 * PI * r;
    }
}
```

Lazy Class

- Trieda, ktorá nedokáže obhájiť svoju existenciu.
- Spojiť do inej triedy. Zmazať ak je skutočne nevyužiteľná

Data Class

```java
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```



```java
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public boolean isAdult() {
        return this.age >= 18;
    }
}
```

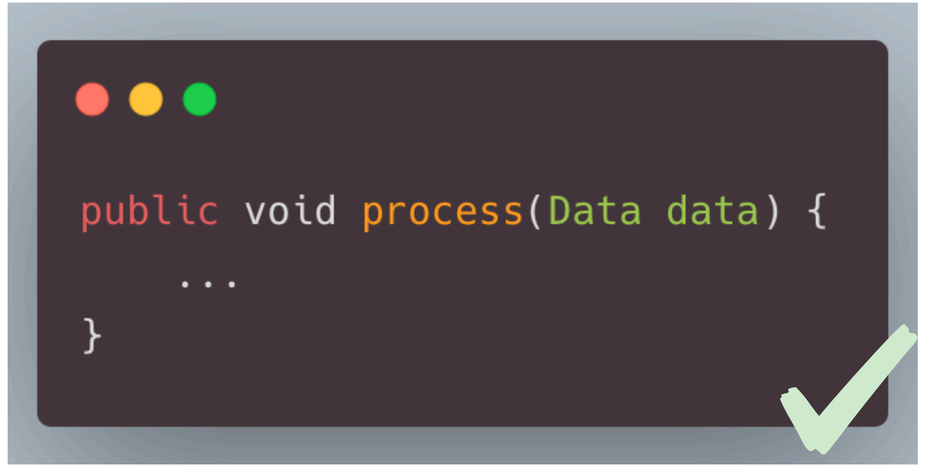
Dead Code

- Metóda, ktorá sa nikde nevolá
- If statement, ktorý nikdy nemôže byť vykonaný
- Nedosiahnuteľný kód

Speculative Generality

- Vyhnúť sa pridavaniu komplexity kým to nie je nutné...

```
public void process(Data data, boolean useCache) {
    if (useCache) {
        ...
    }
}
```



Couplers

Feature Envy. Incomplete Library Class. Middle Man. Inappropriate Intimacy. Message Chains.

Feature Envy

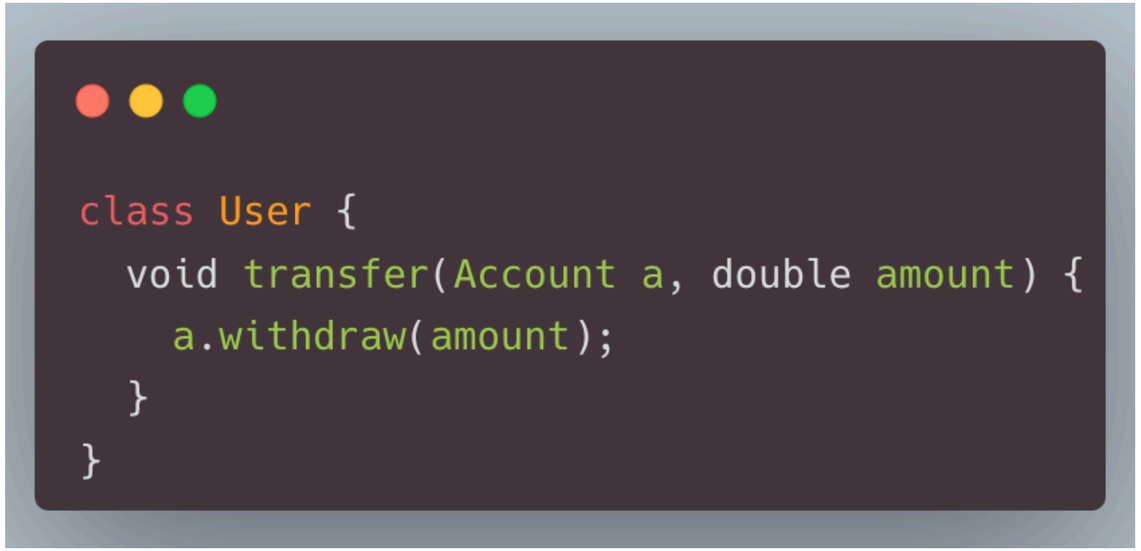
- Metóda volá getter iného objektu viackrát

```
class Report {
  calculateAge(person) {
    return new Date().getFullYear() - person.getBirthYear();
  }

  isAdult(person) {
    return this.calculateAge(person) >= 18;
  }
}
```

```
class Person {
  calculateAge() {
    return new Date().getFullYear() - this.birthYear;
  }

  isAdult() {
    return this.calculateAge() >= 18;
  }
}
```

Inappropriate Intimacy

```
class User {
  void transfer(Account a, double amount) {
    a.balance -= amount;
    a.transactions.add("debit");
  }
}
```



```
class User {
  void transfer(Account a, double amount) {
    a.withdraw(amount);
  }
}
```

Message Chains

- Dlhá postupnosť volaní. Každá návratová hodnota je vstupom pre ďalšie volanie.
- Riešenie - Facade

Middle Man

- Triedy ktorá nerobí nič iné ako len deleguje úlohy na inú triedu.

OOP FIIT Awards!
14. Mája 9:00
Ocenenie najlepších a najzaujímavejších projektov

Aj vecné ceny :)