CODE SMELL PART 1 CODE SNIPPET 1

Conditional Complexity + Feature Envy

Code Smells:

- **Conditional Complexity: Nested if-else blocks make it hard to read.**
- **Feature Envy: The method relies heavily on Order and Customer data instead of encapsulating behavior.**

```
public class DiscountCalculator {
  public double calculateDiscount(Order order) {
    if (order.getCustomer().getType().equals("Regular")) {
      if (order.getTotalAmount() > 1000) {
        return order.getCustomer().getLoyaltyPoints() * 0.01;
      } else {
        return 5;
      }
    } else if (order.getCustomer().getType().equals("Premium")) {
      if (order.getTotalAmount() > 1000) {
        return order.getCustomer().getLoyaltyPoints() * 0.02;
      } else {
        return 10;
      }
    }
    return 0;
    }

}
```

CODE SMELL PART 1 CODE SNIPPET 2

Long Methods + Large Classes

⍰ Long Methods: generate_report does too much.

Large Classes: Multiple unrelated responsibilities (cleaning, formatting, sending) in one class.

```
class ReportManager:
    def generate_report(self, data):
        print("Validating data...")
        if not data:
            print("No data provided.")
            return


        print("Cleaning data...")
        cleaned_data = self.clean_data(data)


        print("Formatting report...")
        formatted_report = self.format_report(cleaned_data)
              print("Sending report...")
        self.send_report(formatted_report)


    def clean_data(self, data):
        return [d.strip() for d in data if d]


    def format_report(self, data):
        return "\n".join(data)


    def send_report(self, report):
        print("Sending report via email...")
        print(report)
```

CODE SMELL PART 1 CODE SNIPPET 3

Duplicate Behavior + Data Clumps

▯ Duplicate Behavior: Similar logic in createInvoice and updateInvoice.

▯ Data Clumps: Repeated parameter groups (customerName, address, etc.).

```
public class InvoiceService {
    public void createInvoice(String customerName, String address, String productName, int quantity) {
        System.out.println("Creating invoice for " + customerName);
        // More logic...
    }
    public void updateInvoice(String customerName, String address, String productName, int quantity) {
        System.out.println("Updating invoice for " + customerName);
        // More logic...
    }
}
```

CODE SMELL PART 1 CODE SNIPPET 4

Primitive Obsession + Feature Envy

- **Primitive Obsession: Overuse of primitives for first_name, last_name, and birth_year.**
- **Feature Envy: Greeting accesses User's data excessively.**

```
class User:
    def __init__(self, first_name, last_name, birth_year):
        self.first_name = first_name
        self.last_name = last_name
        self.birth_year = birth_year

class Greeting:
    def get_greeting(user):
        age = 2025 - user.birth_year
        return f"Hello {user.first_name} {user.last_name}, you are {age} years old!"
```

CODE SMELL PART 1 CODE SNIPPET 5

Divergent Change + Large Classes

Code Smells:

- **Divergent Change: Any change to email, SMS, or logging logic affects this class.**
- **Large Classes: Handles multiple unrelated responsibilities.**

```
public class NotificationService {
  public void sendEmail(String email, String message) {
    System.out.println("Sending email to " + email);
  }
  public void sendSMS(String phone, String message) {
    System.out.println("Sending SMS to " + phone);
  }
  public void logNotification(String message) {
    System.out.println("Logging notification: " + message);
  }
}
```

CODE SMELL PART 1 CODE SNIPPET 6

Shotgun Surgery + Conditional Complexity

� Shotgun Surgery: A small change (e.g., logging logic) requires updating multiple parts.

� Conditional Complexity: Repeated if checks for different keys.

```
def update_user_profile(user, updates):
  if "name" in updates:
    user["name"] = updates["name"]
    log_change("name updated")
  if "email" in updates:
    user["email"] = updates["email"]
```

```
        log_change("email updated")
    if "address" in updates:
        user["address"] = updates["address"]
        log_change("address updated")
```

CODE SMELL PART 1 CODE SNIPPET 7

Long Parameter Lists + Primitive Obsession

Code Smells:

- **Long Parameter Lists: Too many arguments make the method hard to use.**
- **Primitive Obsession: No domain-specific objects for user information.**

```
public void createUser(String firstName, String lastName, String email, int age, String phone, String address) {
    // Logic to create a user
}
```

CODE SMELL PART 1 CODE SNIPPET 8

Duplicate Behavior + Primitive Obsession

Code Smells:

- **Duplicate Behavior: Both methods deal with the same parameters in similar ways.**
-

```
def calculate_rectangle_area(length, width):
    return length * width


def calculate_rectangle_perimeter(length, width):
    return 2 * (length + width)
```

CODE SMELL PART 1 CODE SNIPPET 9

Feature Envy + Shotgun Surgery

⍰ Feature Envy: Overly reliant on Order and its Customer details.

⍰ Shotgun Surgery: Any change in how order or customer details are retrieved impacts this method.

```
public class InvoicePrinter {
    public void printInvoice(Order order) {
        System.out.println("Order ID: " + order.getId());
        System.out.println("Customer: " + order.getCustomer().getName());
        System.out.println("Total: " + order.calculateTotal());
    }
}
```

CODE SMELL PART 1 CODE SNIPPET 10

Data Clumps + Long Methods

⍰ Data Clumps: Repeatedly passing customer and product information.

⍰ Long Methods: Handles multiple unrelated tasks.

```
def generate_invoice(customer_name, customer_email, product_name, quantity, price):
    print(f"Invoice for {customer_name}")
    total = quantity * price
    print(f"Product: {product_name}, Quantity: {quantity}, Total: {total}")
    send_email(customer_email, total)
```

CODE SMELL PART 1 CODE SNIPPET 11

Large Classes + Conditional Complexity

⍰ Large Classes: Handles multiple unrelated payment methods.

⍰ Conditional Complexity: Extending to new payment types increases complexity.

```
public class PaymentProcessor {

    public void processCreditCardPayment(double amount) {
        System.out.println("Processing credit card payment...");
    }

    public void processPayPalPayment(double amount) {
        System.out.println("Processing PayPal payment...");
    }
    public void processCryptoPayment(double amount) {
        System.out.println("Processing cryptocurrency payment...");
    }
}
```

CODE SMELL PART 1 CODE SNIPPET 12

Duplicate Behavior

```
def create_pdf_report(data):
    print("Creating PDF report...")
    print("Adding data to PDF...")
    print("Saving PDF...")

def create_csv_report(data):
    print("Creating CSV report...")
    print("Adding data to CSV...")
    print("Saving CSV...")
```

CODE SMELL PART 1 CODE SNIPPET 13

Shotgun Surgery + Divergent Change

```
public class UserDetails {
    private String name;
    private int birthYear;
    public int calculateAge() {
        return 2025 - birthYear;
    }
}
```

CODE SMELL PART 1 CODE SNIPPET 14

Conditional Complexity

```
class EmailService:
    def send_email(self, recipient, subject, message):
        print(f"Email to {recipient}: {subject}\n{message}")

class LoggingService:
    def log_message(self, message):
        print(f"Log: {message}")
```

CODE SMELL PART 1 CODE SNIPPET 15

```
public double calculateTax(String type, double amount) {
    if (type.equals("Food")) {
        return amount * 0.05;
    } else if (type.equals("Electronics")) {
        return amount * 0.15;
    } else {
        return amount * 0.10;
    }
}
```