COMP6106001 - Code Reengineering

Project Group 1

LB01

Anggota :

2602138214	-	Reynaldo Marchell Bagas Adji
2602114694	-	Ihsaan Hardyanto
2602056832	-	Gregorius Cahyadi Hariyanto
2602075812	-	Petra Michael
2602056845	-	Nikolas Goldy Sulihin

I. Pendahuluan

Judul Project: Personal Budget Tracker

Personal Budget Tracker merupakan sebuah program untuk mencatat pemasukan dan pengeluaran secara pribadi. Program ini memiliki beberapa fitur seperti berikut:

1. Set Budget - User harus memasukkan budget awal dan kurun waktu yang ditentukan

```
--- Personal Budget Tracker ---
1. Set Budget
2. Add Income
3. Add Expense
4. View Budget Summary
5. Create Account
6. Display Account Info
7. Exit
Choose an option: 1
Enter start date (yyyy-MM-dd): 2022-2-2
Enter end date (yyyy-MM-dd): 2022-2-10
Enter income: 1000000
Budget set successfully.
```

2. Add Income - User bisa memasukkan income

```
--- Personal Budget Tracker ---
1. Set Budget
2. Add Income
3. Add Expense
4. View Budget Summary
5. Create Account
6. Display Account Info
7. Exit
Choose an option: 2
Enter date (yyyy-MM-dd): 2022-2-2
Enter amount: 1000000
Enter description: gajian
Income added successfully.
```

3. Add Expense - User bisa memasukkan expense

```
--- Personal Budget Tracker ---
1. Set Budget
2. Add Income
3. Add Expense
4. View Budget Summary
5. Create Account
6. Display Account Info
7. Exit
Choose an option: 3
Enter date (yyyy-MM-dd): 2022-2-2
Enter amount: 30000
Enter description: makan
Expense added successfully.
```

4. View Budget Summary - User bisa melihat kembali income, total expense, dan balance yang mereka punya

```
--- Personal Budget Tracker ---
1. Set Budget
2. Add Income
3. Add Expense
4. View Budget Summary
5. Create Account
6. Display Account Info
7. Exit
Choose an option: 4
Income: 2000000.0, Total Expenses: 30000.0, Balance: 1970000.0
```

5. Create Account - User harus membuat account terlebih dahulu sebelum bisa menggunakan Add Income, Add Expense, dan Display Account Info

```
--- Personal Budget Tracker ---
1. Set Budget
2. Add Income
3. Add Expense
4. View Budget Summary
5. Create Account
6. Display Account Info
7. Exit
Choose an option: 2
Please set the budget and create an account first.

--- Personal Budget Tracker ---
1. Set Budget
2. Add Income
3. Add Expense
4. View Budget Summary
5. Create Account
6. Display Account Info
7. Exit
Choose an option: 3
Please set the budget and create an account first.
```

```
--- Personal Budget Tracker ---
1. Set Budget
2. Add Income
3. Add Expense
4. View Budget Summary
5. Create Account
6. Display Account Info
7. Exit
Choose an option: 5
1. Create Savings Account
2. Create Checking Account
Choose account type: 1
Enter account number: 123
Enter initial balance: 1000000
Enter interest rate: 5
```

6. Display Account Info - Menampilkan informasi account seperti Account Type (Savings / Checking), Account Number, Balance, dan Interest Rate / Overdraft Limit

```
--- Personal Budget Tracker ---
1. Set Budget
2. Add Income
3. Add Expense
4. View Budget Summary
5. Create Account
6. Display Account Info
7. Exit
Choose an option: 6
Savings Account - Account Number: 123, Balance: 1000000.0, Interest Rate: 5.0
```

7. Exit - Setelah selesai menggunakan, user dapat exit

```
--- Personal Budget Tracker ---
1. Set Budget
2. Add Income
3. Add Expense
4. View Budget Summary
5. Create Account
6. Display Account Info
7. Exit
Choose an option: 7
Exiting the program. Goodbye!
```

II. Object-oriented design principles

1. **Encapsulation:** Kelas User, Account, dan Budget menyimpan data sebagai variabel private dan menyediakan metode publik untuk mengakses dan memodifikasi data tersebut.

```
class User {
    private String username;    The value of the field User.username
    private String password;    The value of the field User.password
    private List<Account> accounts;

    public User(String username, String password) {
        this.username = username;
        this.password = password;
        this.accounts = new ArrayList<>();
    }

    public void addAccount(Account account) {
        accounts.add(account);
    }

    public void removeAccount(Account account) {
        accounts.remove(account);
    }

    public Account getAccount(String accountNumber) {
        for (Account account : accounts) {
            if (account.getAccountNumber().equals(accountNumber)) {
                return account;
            }
        }
        return null;
    }
}
```

2. **Inheritance:** Kelas SavingsAccount dan CheckingAccount merupakan subclass dari kelas Account.

```
abstract class Account {
    private String accountNumber;
    private String accountType;
    private double balance;

    public Account(String accountNumber, String accountType, double initialBalance) {
        this.accountNumber = accountNumber;
        this.accountType = accountType;
        this.balance = initialBalance;
    }
```

```java
class SavingsAccount extends Account {
    private double interestRate;

    public SavingsAccount(String accountNumber, double initialBalance, double interestRate) {
        super(accountNumber, accountType:"Savings", initialBalance);
        this.interestRate = interestRate;
    }

    @Override
    public void displayAccountInfo() {
        System.out.println("Savings Account - Account Number: " + getAccountNumber() + ", Balance: " + getBalance() +
        ", Interest Rate: " + interestRate);
    }
}

class CheckingAccount extends Account {
    private double overdraftLimit;

    public CheckingAccount(String accountNumber, double initialBalance, double overdraftLimit) {
        super(accountNumber, accountType:"Checking", initialBalance);
        this.overdraftLimit = overdraftLimit;
    }

    @Override
    public void displayAccountInfo() {
        System.out.println("Checking Account - Account Number: " + getAccountNumber() + ", Balance: " + getBalance() +
        ", Overdraft Limit: " + overdraftLimit);
    }
}
```

3. **Polymorphism:** Metode addTransaction pada kelas Budget menggunakan polymorphism untuk menangani berbagai jenis transaksi (baik IncomeTransaction maupun ExpenseTransaction).

```java
abstract class Transaction {
    private Date date;
    private double amount;
    private String description;

    public Transaction(Date date, double amount, String description) {
        this.date = date;
        this.amount = amount;
        this.description = description;
    }

    public double getAmount() {
        return amount;
    }

    public String getTransactionDetails() {
        return "Date: " + date + ", Amount: " + amount + ", Description: " + description;
    }

    public abstract void processTransaction(Account account);
}
```

```java
class IncomeTransaction extends Transaction {
    public IncomeTransaction(Date date, double amount, String description) {
        super(date, amount, description);
    }

    @Override
    public void processTransaction(Account account) {
        account.deposit(getAmount());
    }
}

class ExpenseTransaction extends Transaction {
    public ExpenseTransaction(Date date, double amount, String description) {
        super(date, amount, description);
    }

    @Override
    public void processTransaction(Account account) {
        account.withdraw(getAmount());
    }
}
```

```java
class Budget {
    private Date startDate;    The value of the field Budget.startDate is not used
    private Date endDate;    The value of the field Budget.endDate is not used
    private double income;
    private List<Transaction> transactions;

    public Budget(Date startDate, Date endDate, double income) {
        this.startDate = startDate;
        this.endDate = endDate;
        this.income = income;
        this.transactions = new ArrayList<>();
    }

    public double getIncome() {
        return income;
    }

    public void setIncome(double income) {
        this.income = income;
    }

    public void addTransaction(Transaction transaction, Account account) {
        transaction.processTransaction(account);
        transactions.add(transaction);
    }
```

4. **Abstraction:** Abstract class Account dan Transaction mendefinisikan kontrak untuk subclass mereka, sehingga hanya metode penting yang terlihat oleh pengguna.

```java
abstract class Account {
    private String accountNumber;
    private String accountType;
    private double balance;

    public Account(String accountNumber, String accountType, double initialBalance) {
        this.accountNumber = accountNumber;
        this.accountType = accountType;
        this.balance = initialBalance;
    }
```

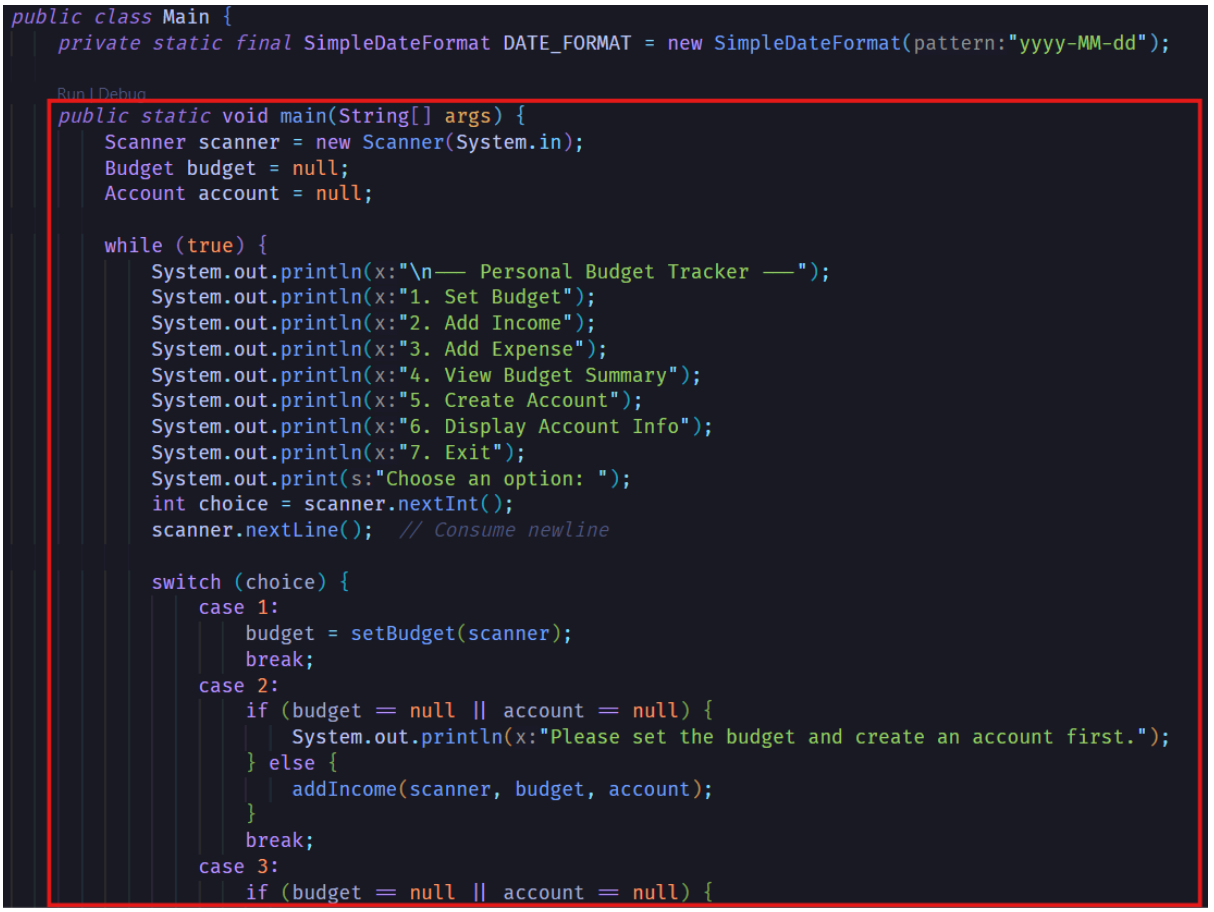
```java
abstract class Transaction {
    private Date date;
    private double amount;
    private String description;

    public Transaction(Date date, double amount, String description) {
        this.date = date;
        this.amount = amount;
        this.description = description;
    }

    public double getAmount() {
        return amount;
    }
```

III. Object Smell

1. **Bloaters - Long Method:** Metode main dalam kelas Main cukup panjang. Metode ini menangani banyak tanggung jawab, seperti menampilkan menu, menangani input pengguna, dan melakukan operasi budget serta account.



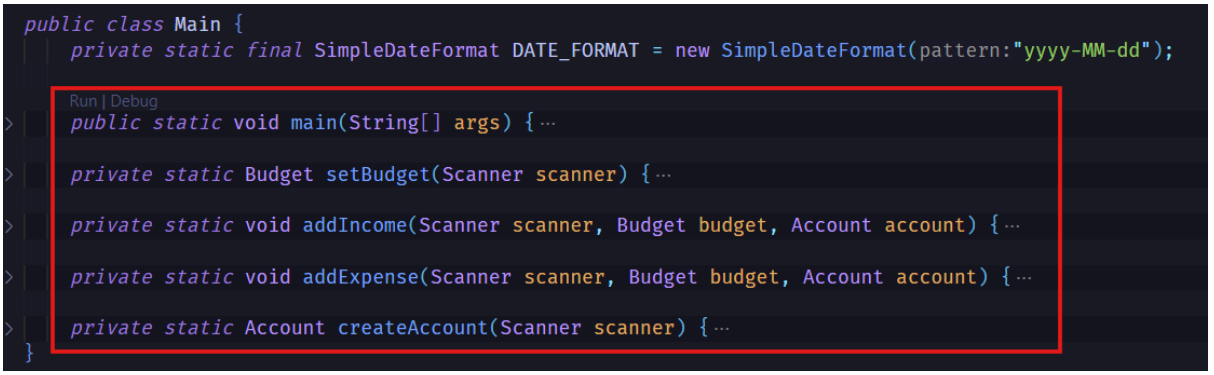
```
public class Main {
    private static final SimpleDateFormat DATE_FORMAT = new SimpleDateFormat(pattern:"yyyy-MM-dd");

    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Budget budget = null;
        Account account = null;

        while (true) {
            System.out.println(x:"\n—— Personal Budget Tracker ——");
            System.out.println(x:"1. Set Budget");
            System.out.println(x:"2. Add Income");
            System.out.println(x:"3. Add Expense");
            System.out.println(x:"4. View Budget Summary");
            System.out.println(x:"5. Create Account");
            System.out.println(x:"6. Display Account Info");
            System.out.println(x:"7. Exit");
            System.out.print(s:"Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();  // Consume newline

            switch (choice) {
                case 1:
                    budget = setBudget(scanner);
                    break;
                case 2:
                    if (budget == null || account == null) {
                        System.out.println(x:"Please set the budget and create an account first.");
                    } else {
                        addIncome(scanner, budget, account);
                    }
                    break;
                case 3:
                    if (budget == null || account == null) {
```

2. **Bloaters - Large Class:** Kelas Main mengandung banyak metode dan menangani terlalu banyak tanggung jawab, seperti operasi budget, pembuatan account, dan penanganan transaksi. Hal ini melanggar Single Responsibility Principle.



```
public class Main {
    private static final SimpleDateFormat DATE_FORMAT = new SimpleDateFormat(pattern:"yyyy-MM-dd");

    Run | Debug
    public static void main(String[] args) {···

    private static Budget setBudget(Scanner scanner) {···

    private static void addIncome(Scanner scanner, Budget budget, Account account) {···

    private static void addExpense(Scanner scanner, Budget budget, Account account) {···

    private static Account createAccount(Scanner scanner) {···
}
```

3. **Object Orientation Abusers - Switch Statements:** Penggunaan pernyataan switch dalam metode main untuk menangani input pengguna dapat menyebabkan kurangnya ekstensi. Menambahkan fitur baru akan memerlukan modifikasi pada switch statement.

```
public class Main {
    public static void main(String[] args) {
        switch (choice) {
            case 1:
                budget = setBudget(scanner);
                break;
            case 2:
                if (budget == null || account == null) {
                    System.out.println(x:"Please set the budget and create an account first.");
                } else {
                    addIncome(scanner, budget, account);
                }
                break;
            case 3:
                if (budget == null || account == null) {
                    System.out.println(x:"Please set the budget and create an account first.");
                } else {
                    addExpense(scanner, budget, account);
                    if (budget.isBudgetExceeded()) {
                        System.out.println(x:"Warning: You have exceeded your budget!");
                    }
                }
                break;
            case 4:
                if (budget == null) {
                    System.out.println(x:"Please set the budget first.");
                } else {
                    System.out.println(budget.getBudgetSummary());
                }
                break;
            case 5:
                account = createAccount(scanner);
                break;
            case 6:
                if (account ≠ null) {
                    account.displayAccountInfo();
                } else {
```

4. **Dispensables - Comments:** Beberapa komentar digunakan untuk menjelaskan apa yang dilakukan kode, yang dapat dihindari dengan menggunakan nama variabel dan metode yang menjelaskan diri sendiri.

```java
public class Main {
    private static void addExpense(Scanner scanner, Budget budget, Account account) {
    }

    private static Account createAccount(Scanner scanner) {
        System.out.println(x:"1. Create Savings Account");
        System.out.println(x:"2. Create Checking Account");
        System.out.print(s:"Choose account type: ");
        int accountType = scanner.nextInt();
        scanner.nextLine();  // Consume newline

        System.out.print(s:"Enter account number: ");
        String accountNumber = scanner.nextLine();

        System.out.print(s:"Enter initial balance: ");
        double initialBalance = scanner.nextDouble();
        scanner.nextLine();  // Consume newline

        if (accountType == 1) {
            System.out.print(s:"Enter interest rate: ");
            double interestRate = scanner.nextDouble();
            scanner.nextLine();  // Consume newline
            return new SavingsAccount(accountNumber, initialBalance, interestRate);
        } else if (accountType == 2) {
            System.out.print(s:"Enter overdraft limit: ");
            double overdraftLimit = scanner.nextDouble();
            scanner.nextLine();  // Consume newline
            return new CheckingAccount(accountNumber, initialBalance, overdraftLimit);
        } else {
            System.out.println(x:"Invalid account type. Please try again.");
            return null;
        }
    }
}
```

5. **Dispensables - Duplicate Code:** Pola kode serupa untuk menangani input pengguna dan parsing tanggal diulang di berbagai metode.

```java
private static void addIncome(Scanner scanner, Budget budget, Account account) {
    try {
        System.out.print(s:"Enter date (yyyy-MM-dd): ");
        Date date = DATE_FORMAT.parse(scanner.nextLine());

        System.out.print(s:"Enter amount: ");
        double amount = scanner.nextDouble();
        scanner.nextLine();  // Consume newline

        System.out.print(s:"Enter description: ");
        String description = scanner.nextLine();

        Transaction income = new IncomeTransaction(date, amount, description);
        income.processTransaction(account);  // Ensure account balance is updated
        budget.setIncome(budget.getIncome() + amount);  // Ensure income is updated in the budget

        System.out.println(x:"Income added successfully.");
    } catch (ParseException e) {
        System.out.println(x:"Invalid date format. Please try again.");
    }
}

private static void addExpense(Scanner scanner, Budget budget, Account account) {
    try {
        System.out.print(s:"Enter date (yyyy-MM-dd): ");
        Date date = DATE_FORMAT.parse(scanner.nextLine());

        System.out.print(s:"Enter amount: ");
        double amount = scanner.nextDouble();
        scanner.nextLine();  // Consume newline

        System.out.print(s:"Enter description: ");
        String description = scanner.nextLine();

        Transaction expense = new ExpenseTransaction(date, amount, description);
```

6. **Couplers - Feature Envy:** Kelas Budget mungkin memiliki feature envy karena mengakses detail akun dan memproses transaksi, yang seharusnya ditangani dalam kelas Account itu sendiri.

```
class Budget {
    }

    public void addTransaction(Transaction transaction, Account account) {
        transaction.processTransaction(account);
        transactions.add(transaction);
    }
```

7. **Couplers - Message Chains:** Beberapa metode bergantung pada pemanggilan beberapa metode atau mengakses objek bersarang, yang menyebabkan nested chaining.

```
public class Main {
    private static Budget setBudget(Scanner scanner) {
        try {
            System.out.print(s:"Enter start date (yyyy-MM-dd): ");
            Date startDate = DATE_FORMAT.parse(scanner.nextLine());

            System.out.print(s:"Enter end date (yyyy-MM-dd): ");
            Date endDate = DATE_FORMAT.parse(scanner.nextLine());

            System.out.print(s:"Enter income: ");
            double income = scanner.nextDouble();
            scanner.nextLine();  // Consume newline

            Budget budget = new Budget(startDate, endDate, income);
            System.out.println(x:"Budget set successfully.");
            return budget;
        } catch (ParseException e) {
            System.out.println(x:"Invalid date format. Please try again.");
            return null;
        }
    }
```

IV. Refactoring Proses

1. Bloaters - Long Method

- **Extract Methods:** Buat metode printMenu, setBudget, addIncome, addExpense, getBudgetSummary, createAccount, displayAccountInfo, dan exitProgram untuk mengurangi panjang metode main dan membuatnya lebih terstruktur.
- **Separation of Concerns:** Pisahkan logika yang berbeda ke dalam metode-metode yang relevan.

```java
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Budget budget = null;
    Account account = null;

    while (true) {
        printMenu();
        int choice = scanner.nextInt();
        scanner.nextLine();  // Consume newline

        switch (choice) {
            case 1:
                budget = setBudget(scanner);
                break;
            case 2:
                if (budget == null || account == null) {
                    System.out.println(x:"Please set the budget and create an account first.");
                } else {
                    addIncome(scanner, budget, account);
                }
                break;
            case 3:
                if (budget == null || account == null) {
                    System.out.println(x:"Please set the budget and create an account first.");
                } else {
                    addExpense(scanner, budget, account);
                    if (budget.isBudgetExceeded()) {
                        System.out.println(x:"Warning: You have exceeded your budget!");
                    }
                }
                break;
            case 4:
                if (budget == null) {
                    System.out.println(x:"Please set the budget first.");
                } else {
                    System.out.println(budget.getBudgetSummary());
                }
                break;
            case 5:
                account = createAccount(scanner);
                break;
            case 6:
                displayAccountInfo(account);
                break;
            case 7:
                exitProgram(scanner);
                return;
            default:
                System.out.println(x:"Invalid option. Please try again.");
        }
    }
}
```

2. **Bloaters - Large Class**

- **Extract Class:** Pisahkan tanggung jawab Main class ke dalam PersonalBudgetTracker class.

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        PersonalBudgetTracker personalBudgetTracker = new PersonalBudgetTracker();
        personalBudgetTracker.start();
    }
}

class PersonalBudgetTracker {
    private static final SimpleDateFormat DATE_FORMAT = new SimpleDateFormat(pattern:"yyyy-MM-dd");
    private Budget budget;
    private Account account;
    private Scanner scanner;

    public PersonalBudgetTracker() {
        this.budget = null;
        this.account = null;
        this.scanner = new Scanner(System.in);
    }

    public void start() {
        while (true) {
            printMenu();
            int choice = scanner.nextInt();
            scanner.nextLine();  // Consume newline

            switch (choice) {
                case 1:
                    setBudget();
                    break;
```

3. **Object Orientation Abusers - Switch Statements:**

- Untuk mengatasi "Switch Statements," kita bisa menggunakan Polymorphism dengan mengubah pendekatan berbasis switch menjadi pendekatan berbasis command class.

```
class PersonalBudgetTracker {
    private static final SimpleDateFormat DATE_FORMAT = new SimpleDateFormat(pattern:"yyyy-MM-dd");
    private Budget budget;
    private Account account;
    private Scanner scanner;
    private Map<Integer, Runnable> menuOptions;

    public PersonalBudgetTracker() {
        this.budget = null;
        this.account = null;
        this.scanner = new Scanner(System.in);
        this.menuOptions = new HashMap<>();
        initializeMenuOptions();
    }

    private void initializeMenuOptions() {
        menuOptions.put(key:1, this::setBudget);
        menuOptions.put(key:2, this::addIncome);
        menuOptions.put(key:3, this::addExpense);
        menuOptions.put(key:4, this::viewBudgetSummary);
        menuOptions.put(key:5, this::createAccount);
        menuOptions.put(key:6, this::displayAccountInfo);
        menuOptions.put(key:7, this::exitProgram);
    }

    public void start() {
        while (true) {
            printMenu();
            int choice = scanner.nextInt();
            scanner.nextLine();  // Consume newline
```

4. **Dispensables - Comments**

- Extract Method: Membuat metode getInputString dan getInputDouble untuk menangani input dari pengguna dan menghilangkan komentar.

```
private static String getInputString(Scanner scanner, String prompt) {
    System.out.print(prompt);
    return scanner.nextLine();
}

private static double getInputDouble(Scanner scanner, String prompt) {
    System.out.print(prompt);
    double value = scanner.nextDouble();
    scanner.nextLine(); // Consume newline
    return value;
}
```

- Rename Variable: Gunakan nama variabel yang deskriptif getInputString dan getInputDouble untuk mengurangi penggunaan comments.

```java
private static Account createAccount(Scanner scanner) {
    System.out.println(x:"1. Create Savings Account");
    System.out.println(x:"2. Create Checking Account");
    System.out.print(s:"Choose account type: ");
    int accountType = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    String accountNumber = getInputString(scanner, prompt:"Enter account number: ");
    double initialBalance = getInputDouble(scanner, prompt:"Enter initial balance: ");

    if (accountType == 1) {
        double interestRate = getInputDouble(scanner, prompt:"Enter interest rate: ");
        return new SavingsAccount(accountNumber, initialBalance, interestRate);
    } else if (accountType == 2) {
        double overdraftLimit = getInputDouble(scanner, prompt:"Enter overdraft limit: ");
        return new CheckingAccount(accountNumber, initialBalance, overdraftLimit);
    } else {
        System.out.println(x:"Invalid account type. Please try again.");
        return null;
    }
}
```

5. **Dispensables - Duplicate Code**

   - Extract Method: Pindah kode yang berulang ke dalam metode baru. Dalam hal ini, metode getInputTransactionDetails digunakan untuk menangani input pengguna dan parsing tanggal.

   Class TransactionDetails: Class ini digunakan untuk mengelompokkan hasil input pengguna menjadi satu objek.

```java
private static TransactionDetails getInputTransactionDetails(Scanner scanner) throws ParseException {
    System.out.print(s:"Enter date (yyyy-MM-dd): ");
    Date date = DATE_FORMAT.parse(scanner.nextLine());

    System.out.print(s:"Enter amount: ");
    double amount = scanner.nextDouble();
    scanner.nextLine(); // Consume newline

    System.out.print(s:"Enter description: ");
    String description = scanner.nextLine();

    return new TransactionDetails(date, amount, description);
}

private static class TransactionDetails {
    private Date date;
    private double amount;
    private String description;

    public TransactionDetails(Date date, double amount, String description) {
        this.date = date;
        this.amount = amount;
        this.description = description;
    }

    public Date getDate() {
        return date;
    }

    public double getAmount() {
        return amount;
    }

    public String getDescription() {
        return description;
    }
}
```

- **Update** : addIncome & addExpense methods

```java
private static void addIncome(Scanner scanner, Budget budget, Account account) {
    try {
        TransactionDetails details = getInputTransactionDetails(scanner);
        Transaction income = new IncomeTransaction(details.getDate(), details.getAmount(),
                details.getDescription());
        income.processTransaction(account);
        budget.setIncome(budget.getIncome() + details.getAmount());

        System.out.println(x:"Income added successfully.");
    } catch (ParseException e) {
        System.out.println(x:"Invalid date format. Please try again.");
    }
}

private static void addExpense(Scanner scanner, Budget budget, Account account) {
    try {
        TransactionDetails details = getInputTransactionDetails(scanner);
        Transaction expense = new ExpenseTransaction(details.getDate(), details.getAmount(),
                details.getDescription());
        expense.processTransaction(account);
        budget.addTransaction(expense, account);

        System.out.println(x:"Expense added successfully.");
    } catch (ParseException e) {
        System.out.println(x:"Invalid date format. Please try again.");
    }
}
```

6. **Couplers - Feature Envy**

   - Move Method : Pindahkan metode processTransaction dari kelas Transaction ke kelas Account, untuk menangani logika pemrosesan transaksi.

```
abstract class Account {
    public void processTransaction(Transaction transaction) {
        if (transaction instanceof IncomeTransaction) {
            deposit(transaction.getAmount());
        } else if (transaction instanceof ExpenseTransaction) {
            withdraw(transaction.getAmount());
        }
    }
}
```

```
public void addTransaction(Transaction transaction, Account account) {
    account.processTransaction(transaction);
    transactions.add(transaction);
}
```

7. **Couplers - Message Chains**

   - Extract Method : Pisah method yang panjang menjadi beberapa metode yang lebih kecil dan lebih fokus. Dimana Method getInputDate dan getInputIncome di-extract untuk mengambil input tanggal dan input income dari pengguna.

```
private static Budget setBudget(Scanner scanner) {
    try {
        Date startDate = getInputDate(scanner, message:"Enter start date (yyyy-MM-dd): ");
        Date endDate = getInputDate(scanner, message:"Enter end date (yyyy-MM-dd): ");
        double income = getInputIncome(scanner);

        Budget budget = new Budget(startDate, endDate, income);
        System.out.println(x:"Budget set successfully.");
        return budget;
    } catch (ParseException e) {
        System.out.println(x:"Invalid date format. Please try again.");
        return null;
    }
}

private static Date getInputDate(Scanner scanner, String message) throws ParseException {
    System.out.print(message);
    return DATE_FORMAT.parse(scanner.nextLine());
}

private static double getInputIncome(Scanner scanner) {
    System.out.print(s:"Enter income: ");
    double income = scanner.nextDouble();
    scanner.nextLine(); // Consume newline
    return income;
}
```

V. Kesimpulan

Proyek Code Reengineering untuk Personal Budget Tracker telah berhasil meningkatkan kualitas desain dan implementasi program. Melalui penerapan prinsip-prinsip desain berorientasi objek seperti encapsulation, inheritance , polymorphism , dan abstract, kami telah menciptakan struktur kode yang lebih modular dan mudah untuk diperluas. Selain itu, melalui identifikasi dan refactoring dari object smells seperti long methods, large classes, switch statements, excessive comments, duplicate code, dan message chains, kami berhasil memperbaiki berbagai masalah yang menghambat pemeliharaan dan pengembangan lebih lanjut.

Langkah-langkah refactoring yang dilakukan tidak hanya meningkatkan keterbacaan dan pemahaman kode tetapi juga meningkatkan performa dan kehandalan program. Dengan memecah metode besar menjadi metode yang lebih kecil dan spesifik, mengurangi ketergantungan antar kelas, dan menghilangkan duplikasi kode, program ini menjadi lebih mudah untuk diuji dan di-debug.