

Nama : Dimas Gary Irawan
NIM : 21120122140164
Mata Kuliah : Rekayasa Perangkat Lunak Berbasis Komponen / B
Teknik Komputer / 2022

SINGLE RESPONSIBILITY PRINCIPLE

Prinsip Single Responsibility (SRP) merupakan prinsip pertama dari SOLID. Prinsip ini menyatakan bahwa: “Sebuah kelas seharusnya hanya memiliki satu tanggung jawab, tugas, atau fokus pada satu tujuan tertentu dan hanya memiliki satu alasan untuk berubah”

SRP sangat penting dalam pengembangan perangkat lunak karena dapat membantu mengurangi kompleksitas, meningkatkan keterbacaan, dan mempermudah perawatan kode program. Dengan menerapkan prinsip SRP, setiap bagian dari program memiliki satu tujuan yang jelas dan terpisah dari tugas lain, sehingga perubahan atau penambahan fitur pada satu bagian tidak akan memengaruhi bagian lain.

1. Penerapan SRP yang salah dalam contoh kasus sistem transaksi saham:

Source Code

```
class StockTransaction:
    def __init__(self, stock_symbol, quantity, price):
        self.stock_symbol = stock_symbol
        self.quantity = quantity
        self.price = price

    def validate_transaction(self):
        print(f"Validating transaction for {self.quantity} shares
of {self.stock_symbol}.")
        return True

    def execute_transaction(self):
        print(f"Executing transaction for {self.quantity} shares
of {self.stock_symbol} at ${self.price} each.")

    def generate_report(self):
```

```

        print(f"Transaction report: {self.quantity} shares of
        {self.stock_symbol} bought at ${self.price} each.")

transaction = StockTransaction("AAPL", 10, 150)

if transaction.validate_transaction():
    transaction.execute_transaction()
    transaction.generate_report()

```

Hasil *running*:

```

class StockTransaction:
    def __init__(self, stock_symbol, quantity, price):
        self.stock_symbol = stock_symbol
        self.quantity = quantity
        self.price = price

    def validate_transaction(self):
        # Misalnya validasi apakah jumlah saham cukup
        print(f"Validating transaction for {self.quantity} shares of {self.stock_symbol}.")
        return True

    def execute_transaction(self):
        # Misalnya eksekusi transaksi
        print(f"Executing transaction for {self.quantity} shares of {self.stock_symbol} at ${self.price} each.")

    def generate_report(self):
        # Misalnya pembuatan Laporan
        print(f"Transaction report: {self.quantity} shares of {self.stock_symbol} bought at ${self.price} each.")

# Contoh penggunaan yang salah
transaction = StockTransaction("AAPL", 10, 150)
if transaction.validate_transaction():

```

Terminal Output:

```

gHts reserved.
c:\Users\gary\python>c:\Users\gary\AppData\Local\Programs\python\python311\python.exe c:\Users\gary\python\RPLBK\pelanggaranSRP.py
Validating transaction for 10 shares of AAPL.
Executing transaction for 10 shares of AAPL at $150 each.
Transaction report: 10 shares of AAPL bought at $150 each.
c:\Users\gary\python>

```

Gambar 1 Hasil *running* kode yang melanggar SRP

Kelas 'StockTransaction' melanggar prinsip *Single Responsibility Principle* (SRP) karena bertanggung jawab atas beberapa hal sekaligus, seperti validasi transaksi, pelaksanaan transaksi, dan pembuatan laporan. Ini menunjukkan bahwa kelas tersebut mencampuradukkan logika yang seharusnya terpisah, sehingga menjadi lebih rentan terhadap perubahan dan kesalahan. Sebagai contoh, jika kita perlu mengubah proses validasi transaksi, maka kita harus melakukan perubahan dalam kelas ini. Namun, perubahan tersebut bisa saja mempengaruhi pelaksanaan transaksi atau pembuatan laporan, meskipun keduanya tidak ada kaitannya dengan validasi. Situasi ini tidak hanya meningkatkan potensi bug yang tak terduga, tetapi juga membuat pengembangan dan pemeliharaan kode menjadi lebih rumit. Dengan tidak mematuhi SRP, kelas StockTransaction menjadi lebih rumit dan sulit diatur, yang pada akhirnya bisa menurunkan kualitas dan fleksibilitas perangkat lunak. Oleh karena itu, sangat penting

untuk memisahkan tanggung jawab ini ke dalam kelas-kelas yang lebih spesifik, sehingga setiap kelas hanya memiliki satu alasan untuk berubah.

2. Penerapan SRP yang benar dalam contoh kasus sistem transaksi saham:

```
class StockTransaction:
    def __init__(self, stock_symbol, quantity, price):
        self.stock_symbol = stock_symbol
        self.quantity = quantity
        self.price = price

class TransactionValidator:
    def validate(self, transaction):
        # Misalnya validasi apakah jumlah saham cukup
        print(f"Validating {transaction.quantity} shares of {transaction.stock_symbol}.")
        return True

class TransactionExecutor:
    def execute(self, transaction):
        # Misalnya eksekusi transaksi
        print(f"Executing transaction for {transaction.quantity} shares of {transaction.stock_symbol} at ${transaction.price} each.")

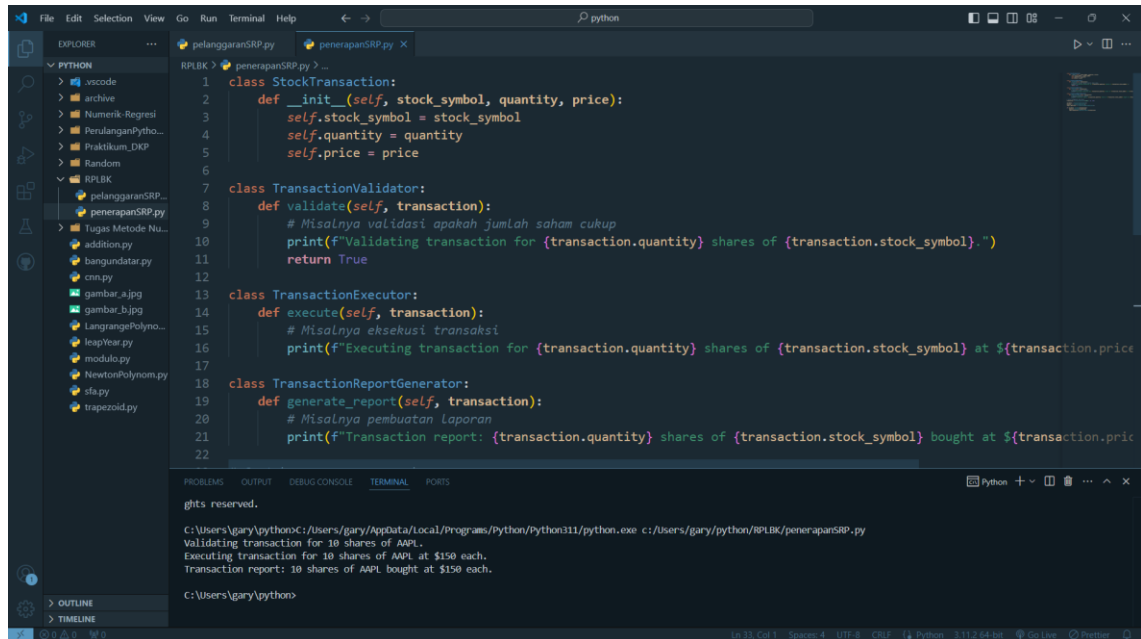
class TransactionReportGenerator:
    def generate_report(self, transaction):
        # Misalnya pembuatan laporan
        print(f"Transaction report: {transaction.quantity} shares of {transaction.stock_symbol} bought at ${transaction.price} each.")

# Contoh penggunaan yang benar
transaction = StockTransaction("AAPL", 10, 150)

validator = TransactionValidator()
executor = TransactionExecutor()
report_generator = TransactionReportGenerator()
```

```
if validator.validate(transaction):  
    executor.execute(transaction)  
    report_generator.generate_report(transaction)
```

Hasil *running*:



```
1 class StockTransaction:  
2     def __init__(self, stock_symbol, quantity, price):  
3         self.stock_symbol = stock_symbol  
4         self.quantity = quantity  
5         self.price = price  
6  
7 class TransactionValidator:  
8     def validate(self, transaction):  
9         # Misalnya validasi apakah jumlah saham cukup  
10        print(f"Validating transaction for {transaction.quantity} shares of {transaction.stock_symbol}.")  
11        return True  
12  
13 class TransactionExecutor:  
14     def execute(self, transaction):  
15         # Misalnya eksekusi transaksi  
16        print(f"Executing transaction for {transaction.quantity} shares of {transaction.stock_symbol} at ${transaction.price}")  
17  
18 class TransactionReportGenerator:  
19     def generate_report(self, transaction):  
20         # Misalnya pembuatan laporan  
21        print(f"Transaction report: {transaction.quantity} shares of {transaction.stock_symbol} bought at ${transaction.price} each.")  
22
```

gits reserved.
C:\Users\gary\python>C:\Users\gary\AppData\Local\Programs\python\python311\python.exe c:/Users/gary/python/RPLBK/penerapanSRP.py
Validating transaction for 10 shares of AAPL.
Executing transaction for 10 shares of AAPL at \$150 each.
Transaction report: 10 shares of AAPL bought at \$150 each.
C:\Users\gary\python>

Gambar 2 Hasil *running* kode yang menerapkan SRP

Source code di atas telah menerapkan SRP dengan benar, kita bisa memisahkan setiap tanggung jawab ke dalam kelas-kelas yang berbeda, yaitu:

1. `'StockTransaction'`: Bertugas untuk menyimpan informasi dasar tentang transaksi saham, seperti simbol saham, jumlah saham, dan harga.
2. `'TransactionValidator'`: Bertanggung jawab untuk melakukan validasi terhadap transaksi saham, memastikan bahwa transaksi tersebut sah dan dapat dilanjutkan.
3. `'TransactionExecutor'`: Mengurus pelaksanaan transaksi, seperti memperbarui jumlah saham yang dimiliki setelah transaksi terjadi.
4. `'TransactionReportGenerator'`: Berfungsi untuk membuat laporan yang merinci detail transaksi saham tersebut.

Dengan membagi setiap fungsi ke dalam kelas-kelas yang terpisah:

- a. Setiap kelas hanya memiliki satu alasan untuk mengalami perubahan. Sebagai contoh, jika aturan validasi berubah, hanya `'TransactionValidator'` yang perlu disesuaikan.

- b. Kode menjadi lebih mudah untuk dikelola dan dikembangkan. Jika kita ingin menambahkan fitur seperti logging atau audit transaksi, kita bisa melakukannya tanpa mengganggu kelas-kelas lainnya.