LAPORAN PROYEK *OBJECT ORIENTED PROGRAMMING*
PRAKTIKUM PEMROGRAMAN DASAR TEKNIK KOMPUTER KELAS D

"SISTEM MANAJEMEN PENJUALAN SEDERHANA MENGGUNAKAN KONSEP OOP"

Disusun oleh:
Kelompok 3
Anggota Kelompok:

1. Faiz Arrafi	**255150307111066**
2. Prameswari Dian Larasati	**255150300111043**
3. Haya Azizah Ramadhani	**255150307111060**
4. Fathin Khiyar Nugraha	**255150300111035**

DEPARTEMEN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
2025

DAFTAR ISI

BAB I
LATAR BELAKANG

Object Oriented Programming (OOP) merupakan salah satu pendekatan yang banyak digunakan dalam pengembangan perangkat lunak modern. Konsep ini membantu programmer menyusun kode secara terstruktur sehingga lebih mudah dikelola, dikembangkan, dan dipahami. Salah satu pilar penting dalam OOP adalah *polymorphism*, yaitu kemampuan suatu *method* untuk menyesuaikan perilakunya berdasarkan objek yang memanggilnya. Fitur ini sangat bermanfaat ketika menangani objek yang berbeda tetapi berada dalam satu kategori yang sama.

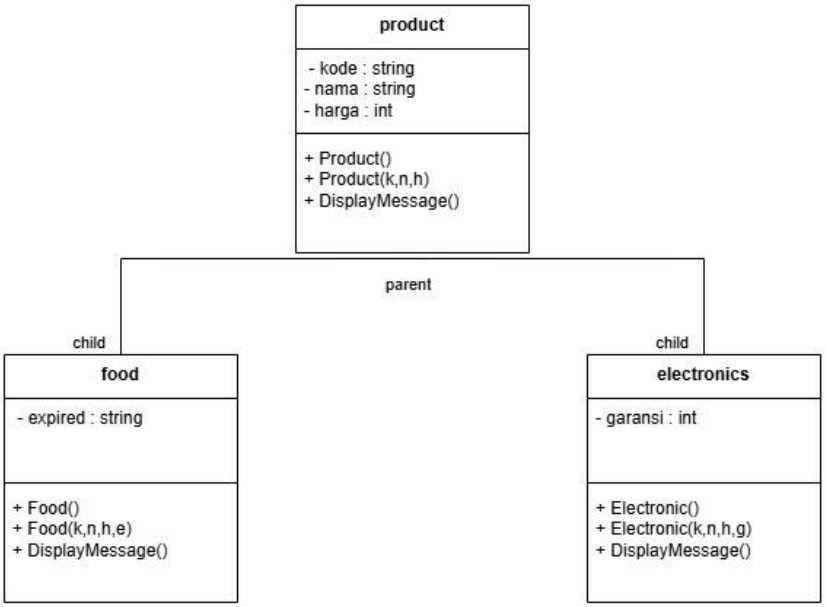
Dalam dunia nyata, sistem penjualan menjadi salah satu contoh yang membutuhkan pengelolaan data yang tertata dan fleksibel. Sebuah toko umumnya menjual berbagai jenis produk, mulai dari makanan hingga barang elektronik. Setiap kategori memiliki karakteristik masing-masing, tetapi seluruhnya tetap harus dikelola dalam satu sistem yang terintegrasi. Tanpa konsep OOP, proses ini bisa menjadi kompleks karena developer harus membuat banyak fungsi dan struktur data yang berbeda untuk tiap jenis produk.

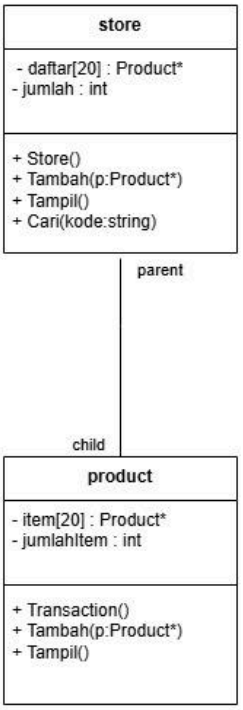
Karena itu, project ini dikembangkan untuk mensimulasikan Sistem Manajemen Penjualan Sederhana dengan memanfaatkan konsep *inheritance* dan *polymorphism*. Pada program ini, class Product bertindak sebagai dasar seluruh produk, sementara Food dan Electronic menjadi perluasannya dengan menambahkan atribut yang lebih spesifik, seperti tanggal kadaluarsa serta lama garansi. Dengan cara ini, program dapat menyimpan berbagai tipe produk dalam satu array pointer Product*, namun tetap menampilkan detail masing-masing produk melalui method DisplayMessage() yang di *override* oleh class turunan.

Project ini juga dirancang sebagai sarana pembelajaran untuk menunjukkan penerapan konsep OOP dalam kasus yang dekat dengan kehidupan sehari-hari. Pembagian class menjadi Store dan Transaction memperlihatkan bagaimana pemisahan tugas (*separation of concern*) membuat struktur program lebih rapi. Selain itu, pendekatan ini mempermudah pengembangan lebih lanjut jika suatu saat ingin menambah jenis produk baru, developer cukup membuat class turunan baru tanpa mengubah mekanisme utama program.

Secara keseluruhan, project ini tidak hanya menjadi latihan dalam memahami sintaks C++, tetapi juga memberikan gambaran nyata mengenai bagaimana konsep OOP khususnya *polymorphism* dapat digunakan untuk membangun sistem yang sederhana namun fleksibel dan mudah diperluas.

BAB II
UML


product
- kode : string
- nama : string
- harga : int
+ Product()
+ Product(k,n,h)
+ DisplayMessage()
parent
child
child
food
- expired : string
+ Food()
+ Food(k,n,h,e)
+ DisplayMessage()
electronics
- garansi : int
+ Electronic()
+ Electronic(k,n,h,g)
+ DisplayMessage()


store
- daftar[20] : Product*
- jumlah : int
+ Store()
+ Tambah(p:Product*)
+ Tampil()
+ Cari(kode:string)
parent
child
product
- item[20] : Product*
- jumlahItem : int
+ Transaction()
+ Tambah(p:Product*)
+ Tampil()

BAB III
PENJELASAN KODE PROGRAM

3.1 Class Product

3.1.1 Product.h

```
#ifndef PRODUCT_H
#define PRODUCT_H

#include <string>
using namespace std;

class Product {
public:
    string kode;
    string nama;
    int harga;

    Product();
    Product(string k, string n, int h);

    virtual void DisplayMessage();
};

#endif
```

Kode ini merupakan sebuah *header file* dalam bahasa C++ yang digunakan untuk mendefinisikan sebuah kelas bernama Product. File header ini biasanya di *include* pada file.cpp lain agar kelas yang didefinisikan dapat digunakan pada program utama.

1. **#ifndef, #define, dan #endif**

```
#ifndef STORE_H
#define STORE_H
```

```
#endif
```

Bagian ini disebut *header guard*. Fungsinya adalah untuk mencegah file header dibaca lebih dari satu kali oleh compiler. Jika file ini di *include* berkali-kali, program tidak akan error karena *header guard* memastikan isinya hanya diproses sekali.

2. **#include <string> dan using namespace std;**

```
#include <string>
using namespace std;
```

- #include <string> digunakan agar program dapat memakai tipe data string.

- using namespace std; memudahkan penulisan, sehingga kita cukup menulis string atau cout tanpa harus menulis std::string atau std::cout.

```
class Product {
public:
    string kode;
    string nama;
    int harga;

    Product();
    Product(string k, string n, int h);

    virtual void DisplayMessage();
};
```

3. Deklarasi Kelas Product

a. Akses public: Semua anggota (atribut dan fungsi) dibawah public dapat diakses dari luar kelas.

b. Atribut Kelas

```
string kode;
string nama;
int harga;
```

Kelas Produk memiliki tiga atribut:

- string kode; → kode produk (misalnya "P01")
- string nama; → nama produk (misalnya "Kopi")
- int harga; → harga produk dalam satuan integer

Atribut ini mewakili informasi dasar sebuah produk.

c. Constructor

```
Product();
Product(string k, string n, int h);
```

Kelas memiliki dua constructor:

1. Constructor default: Dipakai ketika objek dibuat tanpa nilai awal.
2. Constructor dengan parameter: Mengisi langsung kode, nama, dan harga saat objek dibuat.

Contoh penggunaan:

Product p1("A01", "Buku", 15000);

d. Fungsi virtual void DisplayMessage();

```
virtual void DisplayMessage();
```

Ini adalah fungsi yang bersifat virtual, yaitu dapat di *override* oleh kelas turunan (*inheritance*). Tujuannya agar kelas yang mewarisi Product dapat membuat versi fungsi sendiri untuk menampilkan pesan sesuai kebutuhan.

3.1.2 Product.cpp

```
#include "Product.h"
#include <iostream>
using namespace std;

Product::Product() {
    kode = "";
    nama = "";
    harga = 0;
}

Product::Product(string k, string n, int h) {
    kode = k;
    nama = n;
    harga = h;
}

void Product::DisplayMessage() {
    cout << "[" << kode << "] " << nama << " - Rp" << harga << endl;
}
```

Kode berikut merupakan file implementasi dari kelas Product yang sebelumnya dideklarasikan di Product.h. Di dalamnya terdapat definisi constructor dan fungsi anggota yang menentukan bagaimana objek Product berperilaku ketika digunakan dalam program.

1. **Penyertaan Header dan Library**

```
#include "Product.h"
#include <iostream>
using namespace std;
```

Bagian ini berfungsi untuk:

- Menghubungkan file implementasi dengan deklarasi kelas di Product.h.
- Menggunakan library <iostream> agar dapat memakai cout untuk menampilkan output.
- using namespace std; mempermudah penulisan agar tidak perlu menambahkan std::.

2. **Constructor Default**

```
Product::Product() {
    kode = "";
    nama = "";
    harga = 0;
}
```

Constructor ini dipanggil ketika objek Product dibuat tanpa nilai awal.
Nilai atribut di set sebagai berikut:

- kode → string kosong
- nama → string kosong
- harga → 0

Tujuannya supaya objek memiliki nilai awal yang aman dan terdefinisi.

3. Constructor dengan Parameter

```
Product::Product(string k, string n, int h) {
    kode = k;
    nama = n;
    harga = h;
}
```

Constructor ini digunakan ketika kita ingin langsung memberikan nilai awal pada atribut saat objek dibuat.

Parameter:

- k untuk kode produk
- n untuk nama produk
- h untuk harga produk

Contoh penggunaan:
Product p("BRG01", "Roti", 10000);

4. Fungsi DisplayMessage()

```
void Product::DisplayMessage() {
    cout << "[" << kode << "] " << nama << " - Rp" << harga << endl;
}
```

Fungsi ini digunakan untuk menampilkan informasi produk ke layar dalam format:
[kode] nama - Rp(harga)
Fungsi ini didefinisikan sebagai virtual di file header, artinya fungsi ini dapat di override oleh kelas turunan jika diperlukan.

3.2 Class Food

3.2.1 Food.h

```
#ifndef FOOD_H
#define FOOD_H

#include "Product.h"

class Food : public Product {
public:
    string expired;

    Food();
    Food(string k, string n, int h, string e);

    void DisplayMessage() override;
};

#endif
```

File Food.h merupakan *header file* yang mendefinisikan kelas Food, yaitu kelas turunan dari kelas Product. Kelas ini digunakan untuk merepresentasikan produk makanan yang memiliki informasi tambahan berupa tanggal kedaluwarsa.

1. **Header Guard**

```
#ifndef FOOD_H
#define FOOD_H

#endif
```

Bagian ini digunakan untuk mencegah file terbaca lebih dari sekali saat proses kompilasi. Dengan header guard, compiler tidak akan mengulang membaca file yang sama sehingga menghindari error seperti *multiple definition*.

2. **Meng-include File Product.h**

```
#include "Product.h"
```

Karena kelas Food mewarisi kelas Product, maka file Product.h harus disertakan agar compiler mengetahui struktur kelas induknya.

3. **Deklarasi Kelas Food**

```
class Food : public Product {
public:
    string expired;

    Food();
    Food(string k, string n, int h, string e);

    void DisplayMessage() override;
};
```

a. Pewarisan (Inheritance)

```
class Food : public Product {
```

Kelas Food mewarisi atribut dan fungsi dari kelas Product.
Artinya, Food memiliki:

- kode
- nama
- harga
- fungsi DisplayMessage() (yang bisa dioverride)

Pewarisan dengan public membuat atribut dan fungsi publik dari Product tetap dapat diakses oleh objek Food.

b. Atribut Baru

```
public:
    string expired;
```

Atribut ini menambah informasi khusus yang hanya dimiliki produk makanan, yaitu tanggal kedaluwarsa.

c. Constructor

```
Food();
Food(string k, string n, int h, string e);
```

Kelas menyediakan dua constructor:

1. Constructor default : Digunakan jika objek dibuat tanpa nilai awal.
2. Constructor berparameter : Digunakan untuk langsung mengisi:
   - kode
   - nama
   - harga
   - expired

Saat objek dibuat, constructor ini juga akan otomatis memanggil constructor milik kelas Product.

d. Override Fungsi DisplayMessage()

```
void DisplayMessage() override;
```

Kata kunci override menunjukkan bahwa fungsi ini akan menggantikan implementasi DisplayMessage() dari kelas Product. Dengan *override*,

kelas Food bisa menampilkan informasi tambahan seperti tanggal kedaluwarsa.

3.2.2 Food.cpp

```
#include "Food.h"
#include <iostream>
using namespace std;

Food::Food() : Product() {
    expired = "";
}

Food::Food(string k, string n, int h, string e)
    : Product(k, n, h)
{
    expired = e;
}

void Food::DisplayMessage() {
    cout << "[" << kode << "] " << nama
         << " (Makanan) - Rp" << harga
         << " | Exp: " << expired << endl;
}
```

Kode ini merupakan file implementasi untuk kelas Food yang merupakan turunan dari kelas Product. File ini berisi constructor dan fungsi DisplayMessage() yang mengatur bagaimana objek Food menyimpan data dan menampilkan informasi.

1. **Penyertaan Header dan Library**

```
#include "Food.h"
#include <iostream>
using namespace std;
```

Bagian ini berfungsi untuk:

- Menghubungkan file implementasi dengan deklarasi kelas Food dalam Food.h.
- Menggunakan library <iostream> untuk output ke layar melalui cout.
- Menggunakan namespace std agar penulisan lebih ringkas.

2. **Constructor Default**

```
Food::Food() : Product() {
    expired = "";
}
```

Constructor ini dipanggil saat objek Food dibuat tanpa nilai awal.
Hal-hal yang dilakukan:

- Memanggil constructor default dari kelas Product dengan sintaks : Product().
- Menginisialisasi atribut tambahan yaitu:
  expired → string kosong

Ini memastikan objek memiliki nilai awal yang aman dan terdefinisi.

3. **Constructor dengan Parameter**

```
Food::Food(string k, string n, int h, string e)
    : Product(k, n, h)
{
    expired = e;
}
```

Constructor ini digunakan ketika ingin langsung memberikan nilai awal pada semua atribut, baik milik Product maupun tambahan milik Food.
Penjelasan:

- Product(k, n, h) digunakan untuk mengisi atribut kode, nama, dan harga melalui constructor induk.
- expired = e; mengisi tanggal kadaluarsa makanan.

```
void Food::DisplayMessage() {
    cout << "[" << kode << "] " << nama
         << " (Makanan) - Rp" << harga
         << " | Exp: " << expired << endl;
}
```

4. **Fungsi DisplayMessage()**
   Fungsi ini digunakan untuk menampilkan informasi lengkap khusus untuk objek makanan. Format informasi yang ditampilkan adalah:
   [kode] nama (Makanan) - Rp(harga) | Exp: (tanggal kedaluwarsa)
   Fungsi ini meng-override fungsi virtual dari kelas Product, sehingga memiliki tampilan yang lebih spesifik untuk jenis produk makanan.

3.3 Class Electronic

3.3.1 Electronic.h

```
#include "Electronic.h"
#include <iostream>
using namespace std;

Electronic::Electronic() : Product() {
    garansi = 0;
}

Electronic::Electronic(string k, string n, int h, int g)
    : Product(k, n, h)
{
    garansi = g;
}

void Electronic::DisplayMessage() {
    cout << "[" << kode << "] " << nama
         << " (Elektronik) - Rp" << harga
         << " | Garansi: " << garansi << " bln" << endl;
}
```

File Electronic.h adalah *header file* yang berisi deklarasi kelas Electronic, yaitu kelas turunan (*derived class*) dari class Product. Kelas ini digunakan untuk merepresentasikan produk elektronik yang memiliki atribut tambahan berupa masa garansi.

1. **Bagian Header Guard**

```
#ifndef ELECTRONIC_H
#define ELECTRONIC_H

#endif
```

Header guard berfungsi untuk mencegah file terbaca dua kali oleh compiler. Hal ini penting untuk menghindari error duplikasi deklarasi.

2. **Penyertaan Header Product.h**

```
#include "Product.h"
```

Bagian ini diperlukan karena kelas Electronic mewarisi (public Product) semua atribut dan fungsi dari kelas Product. Oleh karena itu, deklarasinya harus dimuat terlebih dahulu.

3. Deklarasi Kelas Electronic

```
class Electronic : public Product {
public:
    int garansi;

    Electronic();
    Electronic(string k, string n, int h, int g);

    void DisplayMessage();
};
```

a. Pewarisan public Product
   Ini berarti kelas Electronic adalah versi khusus dari Product yang memiliki fitur tambahan. Semua atribut kode, nama, dan harga dapat digunakan dalam Electronic.

b. Atribut Tambahan

```
public:
    int garansi;
```

   Atribut ini menyimpan lama garansi barang elektronik (misalnya dalam jumlah bulan).

c. Constructor

```
Electronic();
Electronic(string k, string n, int h, int g);
```

   Kelas memiliki dua constructor:
   1. Constructor default: Untuk membuat objek tanpa nilai awal.
   2. Constructor dengan parameter: Untuk mengisi kode, nama, harga, dan garansi sekaligus saat objek dibuat.

d. Fungsi DisplayMessage()

```
void DisplayMessage();
```

   Fungsi ini akan digunakan untuk menampilkan informasi produk elektronik. Karena fungsi ini tidak diberi kata kunci virtual di *header*, tetapi ada pada Product, maka fungsi ini otomatis *override* meski tidak menggunakan kata kunci *override*.

2.3.2 Electronic.cpp

```cpp
#include "Electronic.h"
#include <iostream>
using namespace std;

Electronic::Electronic() : Product() {
    garansi = 0;
}

Electronic::Electronic(string k, string n, int h, int g)
    : Product(k, n, h)
{
    garansi = g;
}

void Electronic::DisplayMessage() {
    cout << "[" << kode << "] " << nama
         << " (Elektronik) - Rp" << harga
         << " | Garansi: " << garansi << " bln" << endl;
}
```

Kode berikut merupakan file implementasi dari kelas Electronic yang sebelumnya telah dideklarasikan di file header Electronic.h. Di dalam file ini terdapat definisi constructor serta fungsi DisplayMessage() yang mengatur bagaimana objek Electronic menyimpan data dan menampilkan informasi ke layar.

1. **Penyertaan Header dan Library**

```cpp
#include "Electronic.h"
#include <iostream>
using namespace std;
```

Bagian ini berfungsi untuk:

- Menghubungkan file implementasi dengan deklarasi kelas di Electronic.h.
- Menggunakan library <iostream> agar program dapat menampilkan output menggunakan cout.
- using namespace std; digunakan untuk mempermudah penulisan tipe dan fungsi dari namespace std.

2. **Constructor Default**

```cpp
Electronic::Electronic() : Product() {
    garansi = 0;
}
```

Constructor ini dipanggil ketika objek dibuat tanpa nilai awal. Selain memanggil constructor default milik Product melalui: Product(), constructor ini juga menginisialisasi:
garansi → 0 bulan
Dengan demikian, objek memiliki nilai awal yang aman serta konsisten.

3. **Constructor dengan Parameter**

```
Electronic::Electronic(string k, string n, int h, int g)
    : Product(k, n, h)
{
    garansi = g;
}
```

Constructor ini digunakan ketika objek ingin langsung diberikan nilai saat dibuat. Constructor milik Product dipanggil terlebih dahulu untuk mengisi kode, nama, dan harga, sedangkan atribut baru milik kelas turunan diisi sebagai berikut:
g mengisi garansi

4. **Fungsi DisplayMessage()**

```
void Electronic::DisplayMessage() {
    cout << "[" << kode << "] " << nama
         << " (Elektronik) - Rp" << harga
         << " | Garansi: " << garansi << " bln" << endl;
}
```

Fungsi ini digunakan untuk menampilkan informasi lengkap produk elektronik, termasuk:

- kode produk
- nama produk
- harga
- lama garansi (dalam bulan)

Format tampilannya:
[Kode] Nama (Elektronik) - RpHarga | Garansi: X bln
Fungsi ini mengoverride fungsi virtual milik kelas Product, sehingga tampilannya dapat disesuaikan khusus untuk jenis produk elektronik.

3.4 Class Store

3.4.1 Store.h

```
#ifndef STORE_H
#define STORE_H

#include "Product.h"

class Store {
public:
    Product* daftar[20];
    int jumlah;

    Store();
    void Tambah(Product* p);
    void Tampil();
    Product* Cari(string k);
};

#endif
```

Kode ini merupakan *header file* yang berisi **deklarasi kelas Store**, yaitu sebuah kelas yang berfungsi untuk menyimpan dan mengelola objek–objek Product. File ini dideklarasikan agar dapat digunakan oleh file implementasi .cpp maupun file lain dalam program.

1. **Header Guard**

```
#ifndef STORE_H
#define STORE_H
```

```
#endif
```

Bagian ini berfungsi sebagai **header guard** untuk mencegah file Store.h dibaca lebih dari satu kali oleh compiler, sehingga tidak terjadi duplikasi deklarasi.

2. **#include "Product.h"**

```
#include "Product.h"
```

Kelas Store menggunakan pointer ke Product, sehingga file ini harus menyertakan deklarasi kelas Product terlebih dahulu.

3. **Deklarasi Kelas Store**

```
class Store {
public:
    Product* daftar[20];
    int jumlah;

    Store();
    void Tambah(Product* p);
    void Tampil();
    Product* Cari(string k);
};
```

a. Atribut Kelas

```
public:
    Product* daftar[20];
    int jumlah;
```

- Product* daftar[20];
  Array berukuran 20 yang digunakan untuk menyimpan pointer ke objek–objek Product. Artinya, Store dapat menampung maksimal 20 produk.
- int jumlah;
  Mencatat jumlah produk yang saat ini sudah tersimpan dalam daftar.

b. Constructor

```
Store();
```

Constructor ini akan menginisialisasi nilai awal objek Store ketika dibuat. Definisi lengkapnya berada di file .cpp.

c. Fungsi Anggota

```
void Tambah(Product* p);
void Tampil();
Product* Cari(string k);
```

1. void Tambah(Product* p);
   Digunakan untuk menambahkan objek Product ke dalam daftar produk.
2. void Tampil();
   Menampilkan seluruh produk yang tersimpan dalam Store.
3. Product* Cari(string k);
   Mencari produk berdasarkan kode (string). Jika ditemukan, fungsi mengembalikan pointer ke objek produk tersebut.

3.4.2 Store.cpp

```
#include <iostream>
using namespace std;

Store::Store() {
    jumlah = 0;
}

void Store::Tambah(Product* p) {
    daftar[jumlah] = p;
    jumlah++;
}

void Store::Tampil() {
    for (int i = 0; i < jumlah; i++) {
        daftar[i]->DisplayMessage();
    }
}
```

Kode ini merupakan file implementasi untuk kelas Store yang sebelumnya telah dideklarasikan di Store.h. Di dalam file ini terdapat constructor serta beberapa fungsi anggota yang mengatur bagaimana objek Store menyimpan, menambah, menampilkan, dan mencari produk dalam daftar.

1. Penyertaan Header dan Library

   ```
   #include <iostream>
   ```

   Bagian ini digunakan untuk:
   - Menghubungkan file implementasi dengan deklarasi kelas di Store.h.
   - Menggunakan <iostream> agar dapat menampilkan output melalui cout.

2. Constructor Default

   ```
   Store::Store() {
       jumlah = 0;
   }
   ```

   Constructor ini akan dipanggil ketika objek Store dibuat. Nilai 'jumlah' diinisialisasi menjadi **0**, menunjukkan bahwa belum ada produk yang tersimpan di dalam daftar.

3. Fungsi Tambah()

   ```
   void Store::Tambah(Product* p) {
       daftar[jumlah] = p;
       jumlah++;
   }
   ```

Fungsi ini digunakan untuk menambahkan sebuah objek Product ke dalam array daftar.
Langkah kerjanya:

1. Menempatkan pointer produk pada indeks ke-jumlah.
2. Menambah nilai jumlah sebagai penanda bahwa daftar kini berisi lebih banyak produk.

4. Fungsi Tampil()

```
void Store::Tampil() {
    for (int i = 0; i < jumlah; i++) {
        daftar[i]->DisplayMessage();
    }
}
```

Fungsi ini menampilkan seluruh produk yang tersimpan dalam Store. Setiap produk dipanggil fungsi DisplayMessage()-nya, sehingga informasi ditampilkan sesuai jenis produk (karena memakai konsep *polymorphism*).

5. Fungsi Cari()

```
Product* Store::Cari(string k) {
    for (int i = 0; i < jumlah; i++) {
        if (daftar[i]->kode == k)
            return daftar[i];
    }
    return NULL;
}
```

Fungsi ini digunakan untuk mencari produk berdasarkan kode.
Alur kerja:

1. Melakukan pencarian linear pada array daftar.
2. Jika ditemukan produk dengan kode yang sesuai, fungsi mengembalikan pointer produk tersebut.
3. Jika tidak ditemukan, fungsi mengembalikan NULL.

3.5 Class Transaction

3.5.1 Transaction.h

```
#ifndef TRANSACTION_H
#define TRANSACTION_H

#include "Product.h"

class Transaction {
public:
    Product* item[20];
    int jumlahItem;

    Transaction();
    void Tambah(Product* p);
    void Tampil();
};

#endif
```

Kode berikut merupakan *header file* yang berfungsi untuk mendeklarasikan kelas Transaction, yaitu kelas yang digunakan untuk mencatat daftar produk dalam sebuah transaksi. File ini hanya berisi deklarasi struktur kelas dan fungsi-fungsi yang nantinya akan diimplementasikan pada file .cpp.

1. **Penyertaan Header**

```
#include "Product.h"
```

Baris ini digunakan agar kelas Transaction dapat menggunakan pointer ke objek Product, sehingga setiap transaksi dapat menyimpan kumpulan produk yang dibeli.

2. **Deklarasi Kelas Transaction**

```
class Transaction {
public:
    Product* item[20];
    int jumlahItem;

    Transaction();
    void Tambah(Product* p);
    void Tampil();
};
```

a. Atribut

```
Product* item[20];
int jumlahItem;
```

1. Product* item[20]
   Array berisi pointer ke objek Product. Artinya, satu transaksi dapat menyimpan hingga 20 produk.

2. int jumlahItem
   Menyimpan jumlah item yang saat ini ada dalam transaksi. Atribut ini akan bertambah setiap kali produk ditambahkan.

b. Constructor

```
Transaction();
```

Constructor ini berfungsi untuk menginisialisasi nilai awal objek transaksi, seperti mengatur jumlah item menjadi nol. Implementasinya ada di file .cpp.

c. Fungsi-fungsi Anggota

```
void Tambah(Product* p);
void Tampil();
```

1. Void Tambah(Product* p);
   Digunakan untuk menambahkan objek Product ke dalam array item.
2. void Tampil();
   Menampilkan daftar produk dalam transaksi dengan memanggil fungsi DisplayMessage() milik setiap produk.

3.5.2 Transaction.cpp

```
#include "Transaction.h"
#include <iostream>
using namespace std;

Transaction::Transaction() {
    jumlahItem = 0;
}

void Transaction::Tambah(Product* p) {
    item[jumlahItem] = p;
    jumlahItem++;
}

void Transaction::Tampil() {
    int total = 0;

    cout << "\n======= DETAIL TRANSAKSI =======\n";
    for (int i = 0; i < jumlahItem; i++) {
        item[i]->DisplayMessage();
        total += item[i]->harga;
    }
    cout << "Total Belanja: Rp" << total << endl;
}
```

Kode berikut merupakan file implementasi dari kelas Transaction yang sebelumnya telah dideklarasikan di file header Transaction.h. Di dalam file ini terdapat constructor dan fungsi anggota yang mengatur bagaimana objek Transaction menyimpan dan menampilkan data transaksi.

1. Penyertaan Header dan Library

```
#include "Transaction.h"
#include <iostream>
using namespace std;
```

Bagian ini berfungsi untuk:

- Menghubungkan file implementasi dengan deklarasi kelas Transaction di Transaction.h.
- Menggunakan library <iostream> untuk menampilkan output ke layar.

2. Constructor Default

```
Transaction::Transaction() {
    jumlahItem = 0;
}
```

Constructor ini dipanggil ketika objek Transaction dibuat. Atribut jumlahItem diinisialisasi dengan nilai **0** sebagai tanda bahwa transaksi belum memiliki item.

3. Fungsi Tambah()

```
void Transaction::Tambah(Product* p) {
    item[jumlahItem] = p;
    jumlahItem++;
}
```

Fungsi ini digunakan untuk menambahkan sebuah produk ke dalam transaksi.
Cara kerjanya:

- Produk yang diterima sebagai pointer (Product*) disimpan ke array item.
- jumlahItem dinaikkan satu setiap kali produk ditambahkan.

4. Fungsi Tampil()

```
void Transaction::Tampil() {
    int total = 0;
```

Fungsi ini bertanggung jawab untuk menampilkan detail transaksi.
Langkah-langkah yang dilakukan:

1. Menginisialisasi variabel total untuk menghitung total harga.
2. Menampilkan header laporan transaksi.
3. Melakukan loop melalui seluruh item dalam transaksi.

4. Memanggil DisplayMessage() untuk menampilkan detail masing-masing produk.
5. Menjumlahkan harga setiap produk untuk mendapatkan total biaya.
6. Menampilkan total belanja akhir.

3.6 Program Utama

```
#include "Store.h"
#include "Transaction.h"

using namespace std;

int main() {

    Store toko;

    toko.Tambah(new Food("F001", "Indomie Goreng", 3000, "12-12-2025"));
    toko.Tambah(new Food("F002", "Mie Sedap", 2800, "10-10-2025"));

    toko.Tambah(new Electronic("E001", "Kipas Angin", 150000, 12));
    toko.Tambah(new Electronic("E002", "Rice Cooker", 250000, 6));

    Transaction trx;

    int ulang = 1;
    string kode;

    while (ulang == 1) {

        cout << "\n======= DAFTAR PRODUK TOKO =======\n";
        toko.Tampil();

        cout << "\nMasukkan kode produk yang mau dibeli: ";
        cin >> kode;

        Product* p = toko.Cari(kode);

        if (p != NULL) {
            trx.Tambah(p);
            cout << "Produk ditambahkan ke transaksi!\n";
        } else {
            cout << "Kode tidak ditemukan!\n";
        }

        cout << "Beli lagi? (1 = ya, 0 = tidak): ";
        cin >> ulang;
    }

    trx.Tampil();

    return 0;
}
```

Kode ini merupakan program utama yang menggabungkan seluruh kelas yang telah dibuat sebelumnya, yaitu Product, Food, Electronic, Store, dan Transaction. Pada file ini, seluruh objek dan fungsi digunakan untuk mensimulasikan proses belanja di sebuah toko sederhana: menampilkan produk, memilih produk, serta menghitung total pembayaran.

1. **Penyertaan Header dan Namespace**

```
#include <iostream>
#include "Food.h"
#include "Electronic.h"
#include "Store.h"
#include "Transaction.h"

using namespace std;
```

Bagian ini memuat library dan file header yang dibutuhkan untuk menjalankan program.

- <iostream> dipakai untuk input-output (cin, cout).
- File header Food.h, Electronic.h, Store.h, dan Transaction.h berisi deklarasi kelas yang digunakan dalam program ini.
- using namespace std; agar penulisan lebih sederhana.

2. **Membuat Objek Store**

```
int main() {

    Store toko;
```

Objek toko digunakan untuk menyimpan daftar produk yang tersedia di toko. Objek ini memakai array pointer Product* sehingga dapat menyimpan objek dari kelas turunan apapun (Food/Electronic).

3. **Menambahkan Produk ke Dalam Toko**

```
toko.Tambah(new Food("F001", "Indomie Goreng", 3000, "12-12-2025"));
toko.Tambah(new Food("F002", "Mie Sedap", 2800, "10-10-2025"));

toko.Tambah(new Electronic("E001", "Kipas Angin", 150000, 12));
toko.Tambah(new Electronic("E002", "Rice Cooker", 250000, 6));
```

Kode di atas menambahkan 4 produk ke dalam toko:

- 2 produk makanan (Food) dengan kode, nama, harga, dan tanggal kedaluwarsa.
- 2 produk elektronik (Electronic) dengan kode, nama, harga, dan garansi.

Objek dibuat dengan ‘new’ karena disimpan sebagai pointer dalam array.

4. **Membuat Objek Transaksi**

```
Transaction trx;
```

Objek ‘trx’ digunakan untuk menyimpan daftar produk yang dibeli pelanggan. Produk yang ditambahkan berasal dari hasil pencarian di dalam Store.

5. **Proses Input Pembelian**

Variabel untuk pengulangan:

```
int ulang = 1;
string kode;
```

- Loop Pembelian

```
while (ulang == 1) {

    cout << "\n======= DAFTAR PRODUK TOKO =======\n";
    toko.Tampil();

    cout << "\nMasukkan kode produk yang mau dibeli: ";
    cin >> kode;

    Product* p = toko.Cari(kode);

    if (p != NULL) {
        trx.Tambah(p);
        cout << "Produk ditambahkan ke transaksi!\n";
    } else {
        cout << "Kode tidak ditemukan!\n";
    }

    cout << "Beli lagi? (1 = ya, 0 = tidak): ";
    cin >> ulang;
}
```

Selama pengguna memilih “1”, proses pemilihan produk akan terus berulang.

- Menampilkan Daftar Produk

```
cout << "\n======= DAFTAR PRODUK TOKO =======\n";
toko.Tampil();
```

Memanggil fungsi DisplayMessage() dari setiap objek di dalam toko.

- Input Kode Produk

```
cout << "\nMasukkan kode produk yang mau dibeli: ";
cin >> kode;
```

- Mencari Produk

```
Product* p = toko.Cari(kode);
```

➢ Jika ditemukan → pointer produk dikembalikan.
➢ Jika tidak → mengembalikan NULL.

- Menambahkan Produk ke Transaksi

```
if (p != NULL) {
    trx.Tambah(p);
    cout << "Produk ditambahkan ke transaksi!\n";
} else {
    cout << "Kode tidak ditemukan!\n";
}
```

Jika produk ditemukan, maka ditambahkan ke daftar belanja.

- Opsi Melanjutkan Pembelian

```
cout << "Beli lagi? (1 = ya, 0 = tidak): ";
cin >> ulang;
```

6. **Menampilkan Detail Transaksi**

```
trx.Tampil();
```

Menampilkan semua produk yang dibeli dan menghitung total harga belanjaan.
Output berisi:

- Daftar produk yang dipilih
- Total belanja dalam format rupiah

BAB IV
SOURCE CODE DAN LINK GITHUB

4.1 *Source Code*

```
1   // SISTEM MANAJEMEN PENJUALAN SEDERHANA MENGGUNAKAN KONSEP OOP
2
3   #include <iostream>
4   #include <string>
5   using namespace std;
6
7   // ======================================
8   // CLASS PRODUCT (INDUK)
9   // ======================================
10  class Product {
11  public:
12      string kode;
13      string nama;
14      int harga;
15
16      Product();
17      Product(string k, string n, int h);
18
19      virtual void DisplayMessage(); //  POLYMORPHISM  (VIRTUAL)
20  };
21
22  Product::Product() {
23      kode = "";
24      nama = "";
25      harga = 0;
26  }
27
28  Product::Product(string k, string n, int h) {
29      kode = k;
30      nama = n;
31      harga = h;
32  }
33
34  void Product::DisplayMessage() {
35      cout << "[" << kode << "] " << nama << " - Rp" << harga << endl;
36  }
37
38
39  // ======================================
40  // CLASS FOOD (CHILD) - OVERRIDE (POLYMORPHISM)
41  // ======================================
42  class Food : public Product {
43  public:
44      string expired;
45
46      Food();
47      Food(string k, string n, int h, string e);
48
```

```
49      void DisplayMessage(); // <-- override dari virtual
50  };
51
52  Food::Food() : Product() {
53      expired = "";
54  }
55
56  Food::Food(string k, string n, int h, string e)
57      : Product(k, n, h)
58  {
59      expired = e;
60  }
61
62  void Food::DisplayMessage() {
63      cout << "[" << kode << "] " << nama
64           << " (Makanan) - Rp" << harga
65           << " | Exp: " << expired << endl;
66  }
67
68
69  // ======================================
70  // CLASS ELECTRONIC (CHILD) - OVERRIDE
71  // ======================================
72  class Electronic : public Product {
73  public:
74      int garansi;
75
76      Electronic();
77      Electronic(string k, string n, int h, int g);
78
79      void DisplayMessage(); // <-- override polymorphism
80  };
81
82  Electronic::Electronic() : Product() {
83      garansi = 0;
84  }
85
86  Electronic::Electronic(string k, string n, int h, int g)
87      : Product(k, n, h)
88  {
89      garansi = g;
90  }
91
92  void Electronic::DisplayMessage() {
93      cout << "[" << kode << "] " << nama
94           << " (Elektronik) - Rp" << harga
95           << " | Garansi: " << garansi << " bln" << endl;
96  }
97
98
99  // ======================================
100 // CLASS STORE
101 // ======================================
102 class Store {
103 public:
104     Product* daftar[20];
```

```
105     int jumlah;
106
107     Store();
108     void Tambah(Product* p);
109     void Tampil();
110     Product* Cari(string k);
111 };
112
113 Store::Store() {
114     jumlah = 0;
115 }
116
117 void Store::Tambah(Product* p) {
118     daftar[jumlah] = p;
119     jumlah++;
120 }
121
122 void Store::Tampil() {
123     int i;
124     for (i = 0; i < jumlah; i++)
125         daftar[i]->DisplayMessage(); // <-- POLYMORPHISM AKTIF DI SINI
126 }
127
128 Product* Store::Cari(string k) {
129     int i;
130     for (i = 0; i < jumlah; i++) {
131         if (daftar[i]->kode == k)
132             return daftar[i];
133     }
134     return NULL;
135 }
136
137
138 // ======================================
139 // CLASS TRANSACTION
140 // ======================================
141 class Transaction {
142 public:
143     Product* item[20];
144     int jumlahItem;
145
146     Transaction();
147     void Tambah(Product* p);
148     void Tampil();
149 };
150
151 Transaction::Transaction() {
152     jumlahItem = 0;
153 }
154
155 void Transaction::Tambah(Product* p) {
156     item[jumlahItem] = p;
157     jumlahItem++;
158 }
159
160 void Transaction::Tampil() {
```

```
161     int total = 0;
162     int i;
163 
164     cout << "\n======= DETAIL TRANSAKSI =======\n";
165     for (i = 0; i < jumlahItem; i++) {
166         item[i]->DisplayMessage(); // <-- POLYMORPHISM AKTIF LAGI
167         total += item[i]->harga;
168     }
169     cout << "Total Belanja: Rp" << total << endl;
170 }
171 
172 
173 // ======================================
174 // MAIN PROGRAM
175 // ======================================
176 int main() {
177 
178     Store toko;
179 
180     toko.Tambah(new Food("F001", "Indomie Goreng", 3000, "12-12-2025"));
181     toko.Tambah(new Food("F002", "Mie Sedap", 2800, "10-10-2025"));
182 
183     toko.Tambah(new Electronic("E001", "Kipas Angin", 150000, 12));
184     toko.Tambah(new Electronic("E002", "Rice Cooker", 250000, 6));
185 
186     Transaction trx;
187 
188     int ulang = 1;
189     string kode;
190 
191     while (ulang == 1) {
192 
193         cout << "\n======= DAFTAR PRODUK TOKO =======\n";
194         toko.Tampil();
195 
196         cout << "\nMasukkan kode produk yang mau dibeli: ";
197         cin >> kode;
198 
199         Product* p = toko.Cari(kode);
200 
201         if (p != NULL) {
202             trx.Tambah(p);
203             cout << "Produk ditambahkan ke transaksi!\n";
204         } else {
205             cout << "Kode tidak ditemukan!\n";
206         }
207 
208         cout << "Beli lagi? (1 = ya, 0 = tidak): ";
209         cin >> ulang;
210     }
211 
212     trx.Tampil();
213 
214     return 0;
215 }
```

4.2 Link GitHub

https://github.com/faizarrafi5/PROJECT-LIVE-CODING-KELOMPOK-3/blob/main/README.md