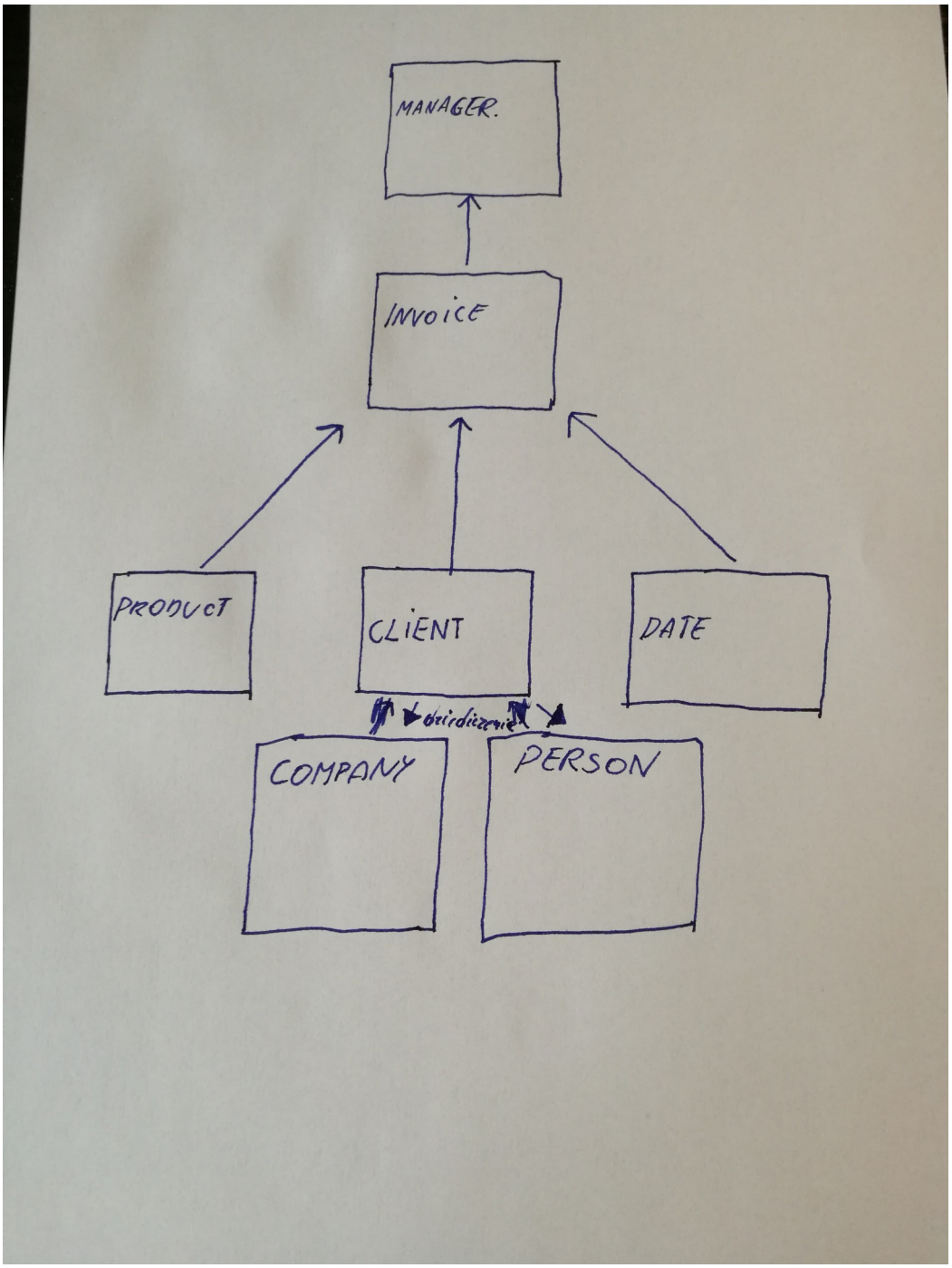
Zadanie 3.4. Logika programu wystawiającego faktury

1. Opis projektu

Dane, które mają znajdować się na fakturze podzieliłem na trzy obszary: dotyczące zakupionych produktów, informacje o kliencie oraz terminie wykonania transakcji. Dla każdego z tych działów stworzyłem osobną klasę, która zajmuje się ich przetwarzaniem. Dodatkowo klasa Client jest jedynie abstrakcyjną klasą bazową. Dziedziczą po niej klasy Company i Person. Klasą nadrzedną nad nimi jest Invoice. Za logikę programu odpowiada klasa Manager.



Rys.1. Hierarchia klas

2. Opis klas

2.1. Product

Obiekt klasy produkt zawiera informacje o pojedynczym rodzaju produktu będącym przedmiotem transakcji. Jej składnikami są pola przechowujące informacje nt. liczby zakupionych produktów, ich nazwy, ceny oraz odpowiadającej stawce podatku. Metody tej klasy odpowiadają za nadawanie polom wartości oraz zwracanie ich. Wyjątkiem są metody GetNetto() zwracająca łączną wartość transakcji netto, GetTara() zwracająca wartość zapłaconego podatku oraz GetTotal() zwracającą informacje o całościowej wartości zakupu.

2.2. Date

Obiekt tej klasy przechowuje dzień, miesiąc i rok wystawienia faktury. Żeby uniknąć możliwości fałszowania jedynym sposobem na wprowadzenie danych do obiektu tej klasy jest wprowadzenie ich jako parametry podczas deklaracji obiektu. Klasa zawiera metody zwracające wartości pól.

2.3 Client

Metody tej klasy pozwalają jedynie na ustalanie jej pól oraz zwracanie ich wartości. Są to podstawowe informacje jak imię, nazwisko, czy adres zamieszkania. Klasa zawiera również funkcje wirtualne Get i SetUnique. Funkcje te odpowiadają za obsługę pól znajdujących się w klasach pochodnych.

2.4. Company i Person

Klasy pochodne od klasy Client. Wprowadzenie tych klas było spowodowane potrzebą rozróżnienia osób prywatnych od prawnych. Klasa Company zawiera pole nip, a person surname.

2.5. Invoice

Klasy Product, Date i Client nie mają między sobą żadnych powiązań. Ich obiekty są za to polami klasy Invoice. Obiekt tej klasy przechowuje wektor obiektów klasy Product oraz po jednym obiekcie klas Date oraz wskaźnik na Client. Jej polami są również zmienne zawierające podstawowe informacje o fakturze jak liczba rodzajów zakupionych produktów, czy też numer faktury. Metody tej funkcji z wyjątkiem SetNumber() nie przyjmują, w przeciwieństwie do metod innych klas, danych w formie liczb, czy znaków, lecz w formie obiektów. W celu przypisania wartości jednemu z pól należy przekazać mu obiekt odpowiedniej klasy. Jeśli chce się np. wprowadzić do obiektu dane kupującego, to należy przekazać metodzie SetBuyer() klasy Invoice obiekt typu Client. Analogicznie prezentuje się sytuacja ze zwracaniem wartości pól. Funkcja zwraca obiekty, a nie ich poszczególne wartości. Byłoby to niepotrzebne rozbudowywanie kodu.

2.6. InvoiceManager

Funkcja nadrzędna nad Invoice, zawiera wektor obiektów klasy Invoice. Metody publiczne tej klasy pozwalają na tworzenie nowych faktur – CreateInvoice(), dodawanie produktów do wybranej faktury – Add() lub też wyszukiwanie faktury o danym numerze – Search(). Umożliwiają również zmianę danych na fakturze. Faktury w wektorze są automatycznie sortowane rosnąco wg numeru. Utrzymanie kolejności jest odporne na zmiany numerów, czy dodawanie nowych faktur, dane zawsze sortowane są na bieżąco.

3. Rozbudowa programu

Rozbudowa programu o nowe funkcjonalności jest zgodnie ze standardami obiektowego paradygmatu programowania łatwa i intuicyjna. Byłaby niezbędna w celu stworzenia użytecznego programu. Funkcjonalności, o które program powinien zostać rozbudowany to:

3.1. Interfejs graficzny

Komponent ten odpowiadałby za wprowadzanie danych do faktur oraz wyświetlanie ich. Pierwsza czynność wykonywana byłaby poprzez przyjmowanie danych od użytkownika, tworzenie obiektów klas Client, Date i Product oraz przekazywanie ich do obiektu klasy Manager, który tworzyłby nowe faktury. Wyświetlanie faktur odbywałoby się poprzez wyświetlanie wartości zwracanych przez metody klas.

3.2. Przekazywanie daty

Przydatna byłaby również funkcjonalność pobierająca z systemu informacje o dacie i przekazująca je do faktury. Odbywałoby się to poprzez inicjalizację obiektu typu Date i przekazanie go do odpowiedniej faktury za pomocą metody SetDate().

3.3. System zapisu i wczytywania

Zadaniem tego komponentu byłoby odtwarzanie faktur z plików tekstowych i zapisywanie ich do tychże. Zasada działanie byłaby podobna do interfejsu graficznego, z tym że analiza danych do faktur odbywałaby się całkowicie w środowisku tekstowym.

3.4. Nawigacja między wczytanymi fakturami

Jedno z podzadań interfejsu graficznego. Ten komponent wyświetlałby faktury znajdujące się w pamięci w odpowiedni sposób, nie byłoby problematyczne, jako że faktury w klasie Manager są zawsze posortowane. Umożliwiałby również wyszukiwanie faktur za pomocą metody Search() klasy Manager.