UML

Unified Modeling Language

Czym jest?

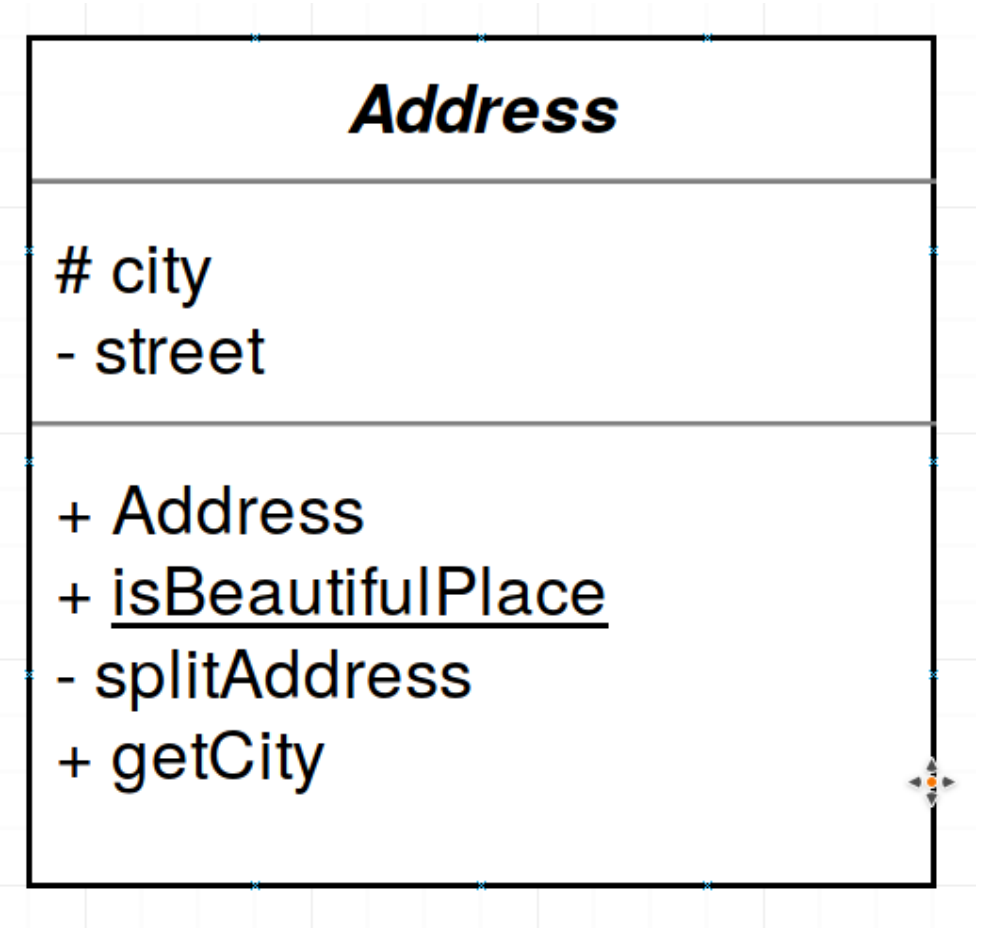
- Zunifikowany język modelowania pół-formalny wykorzystywany do graficznego modelowania aplikacji oraz systemów
- Meta-model oprogramowania
- Ułatwia zrozumienie różnych systemów komputerowych
- Diagramy pozwalają na ilustrację rozmaitych aspektów systemu

Diagram klas

- Statyczny diagram przedstawiający strukturę aplikacji bądź systemu w paradygmacie programowania obiektowego
- Opisuje strukturę całego systemu lub jego części
- Przedstawia poglądową strukturę programu wraz z metodami i polami danej klasy (nie ma wymogu wypisywania ich wszystkich)

Diagram klas

```java
public abstract class Address {

    protected String city;
    private String street;

    public Address(String address) {
        splitAddress(address);
    }

    static public boolean isBeautifulPlace() {
        return false;
    }

    private void splitAddress(String address) {
    }

    public String getCity() {
        return city;
    }

    public String getStreet() {
        return street;
    }
}
```

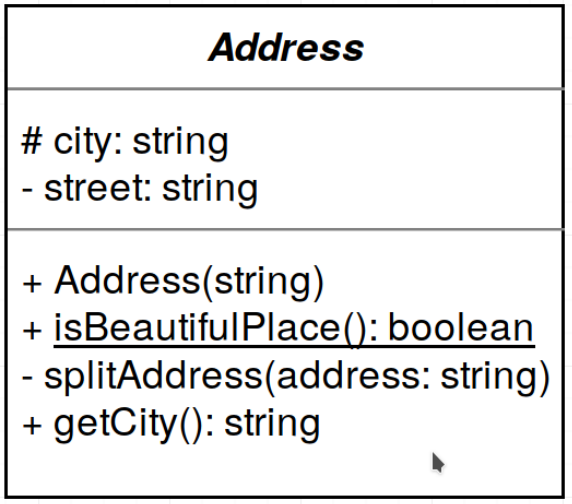


Notacja

Obszar górny
– nazwa klasy i opcjonalnie nazwa pakietu
<nazwa klasy>

Obszar środkowy
– lista atrybutów
<dostępność> <nazwa atrybutu>:<typ>=<wartość początkowa>

Obszar dolny
– lista metod
<dostępność> <nazwa metody>(<lista argumentów>):<typ zwracanej wartości>



Notacja

Lista argumentów metody

<kierunek> <nazwa>:<typ>=<wartość domyślna>

Typy kierunków:

- in–parametr wejściowy
- out–parametr wyjściowy
- inout–parametr wejściowo/wyjściowy

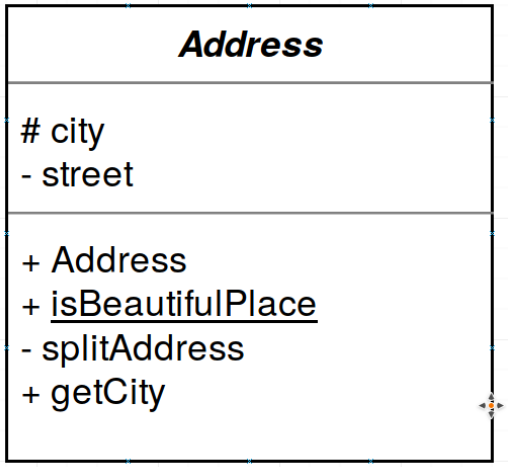
Poziomy dostępności

+ składnik publiczny (public)

składnik chroniony (protected)

– składnik prywatny (private)

~ składnik dostępny w obrębie projektu (package)



Składnik statyczny reprezentowany jest przez podkreślenie

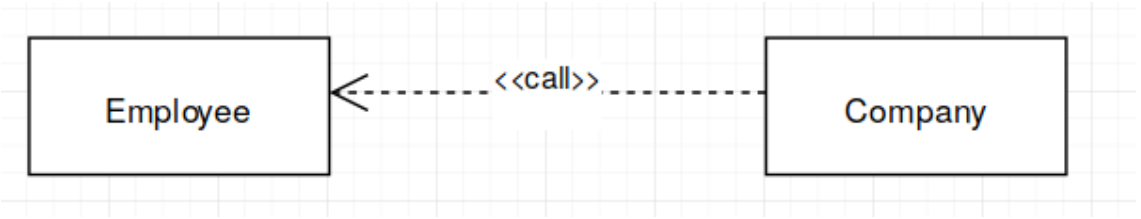
Składnik abstrakcyjny reprezentowany jest przez pochylenie

Interfejs oznaczamy podobnie jak nazwę klasy poprzedzając ją słowem kluczowym <<interface>>

Związki między klasami
Zależności

- Zależność jest najsłabszą relacją jaka może występować pomiędzy dwoma klasami
- Zależność występuje, gdy zmiana specyfikacji jednej klasy, może powodować konieczność wprowadzania zmiany w innej klasie

```
public class Company {

    private Employee employee;
    private Person person;



    public boolean update(Employee employee) {
        if (employee.getName().equals("")) {
            employee.update();
            return false;
        }
        return true;
    }
}
```



Związki między klasami
Zależności

- Zależność:
  - <<call>> - operacje w klasie A wywołują operacje w klasie B
  - <<create>> - klasa A tworzy instancje klasy B
  - <<use>> - do zaimplementowania klasy A wymagana jest B
- Zależności często opisuje się frazami: *„korzysta z”*, *„oddziałuje na”*, *„ma wpływ na”*, *„tworzy”*

Związki między klasami
Asocjacja

- Asocjacje są silniejszymi relacjami niż zależności
- Jeden obiekt jest związany z innym przez pewien czas, ale czas życia obu obiektów nie jest od siebie zależny
- Żaden obiekt nie jest właścicielem drugiego: nie tworzy go, nie zarządza nim, a moment usunięcia drugiego obiektu nie jest z nim związany

Związki między klasami
Agregacja częściowa

- Związek dwóch klas w formie relacji całość-część
- Szczególny rodzaj asocjacji
- Usunięcie klasy całość nie wpływa na istnienie klasy część

```
public class MyCompany {

    private ArrayList<Employee> list;

    public MyCompany() {
        this.list = new ArrayList<>();
    }


    public void addEmployee(Employee employee){
        this.list.add(employee);
    }
}
```

Firma zawiera pracownika

Związki między klasami
Agregacja całkowita (kompozycja)

- Agregacja całkowita jest najsilniejszą relacją
- Relacje całość-część, w których części są tworzone i zarządzane przez obiekt reprezentujący całość
- Oba obiekty nie mogą istnieć bez siebie, dlatego czasy ich istnienia są bardzo ściśle ze sobą związane i pokrywają się

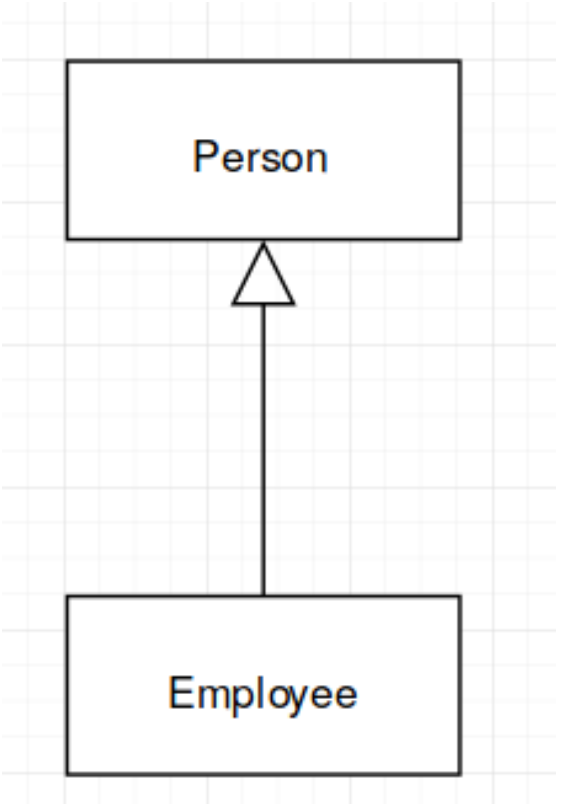
```
public class Company {

    private Employee employee;
    private Person person;

    public Company() {
        this.employee = new Employee();
    }
```

Związki między klasami
Dziedziczenie

- Dziedziczenie umożliwia wyodrębnienie cech wspólnych dla kilku klas i zamknięciu ich w klasie bardziej ogólnej – o wyższym poziomie abstrakcji

```
public class Person {

    private String name;

    public Person(String name)
    {
        this.name = name;
    }
```

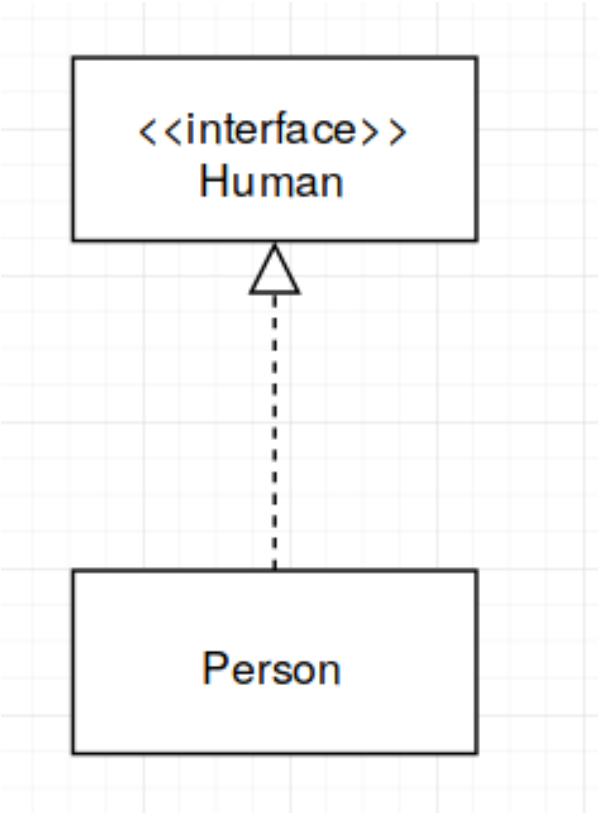
```
public class Employee extends Person{

    private String name;

    public Employee(String name) {
        super(name);
    }
```



Klasa Employee dziedziczy po klasie Person

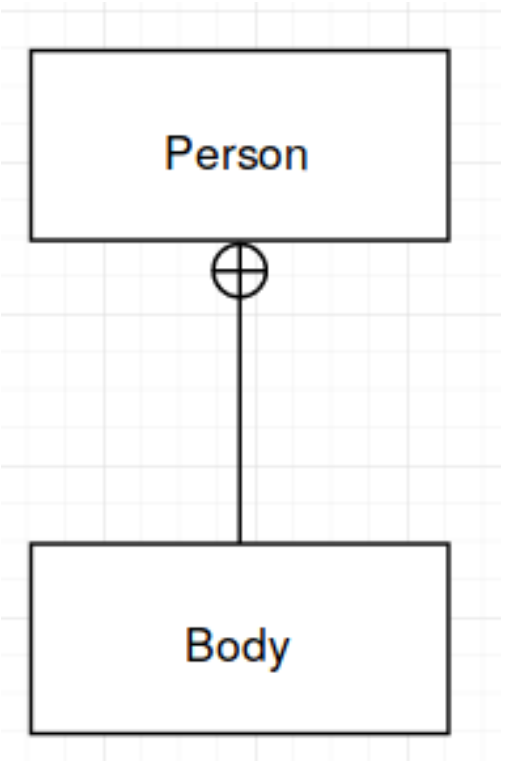
Związki między klasami
Interfejsy

```
public class Person implements Human{

    private String name;

    public Person(String name)
    {
        this.name = name;
    }

    public class Body{

    }
```



Związki między klasami
Klasy zagnieżdżone

```java
public class Person {

    private String name;

    public Person(String name)
    {
        this.name = name;
    }

    public class Body{

    }
```



Klasa Body jest zagnieżdżona w klasie Person