OOP

#oop #object #instance #reference

Author: Piotr Niemczuk
Nickname: Pyoneru

Teoria

OOP - Object Oriented Programming (*pl.: Programowanie zorientowane obiektowo*)

Chcąc programować w modelu OOP (potocznie *obiektowo*), musimy zacząć patrzeć na otaczający nas świat nieco inaczej. Zaczniemy rozkładać otaczające nas rzeczy i zjawiska na poszczególne **paramtery**.

Zacznijmy zadawać pytania z jakich parametrów są skonstruowane otaczające nas obiekty?

source

Spróbujmy rozłożyć na parametry coś co każdy z nas zna - samochód.
Rozłożyć na parametry czyli odpowiedzieć sobie z czego każdy samochód może się składać.

- Każdy samochód - zresztą jak większość rzeczy fizycznych, będzie miał swój rozmiar. Szerokość, długość i wysokość.
- Każdy samochód ma swój kolor
- Każdy samochód ma również swój kształt
- Przy każdym samochodzie możemy zauważyć również znak producenta.
- Kolor świateł
- Ilość drzwi

Są to **parametry** widoczne, jakimi możemy opisać każde auto.
Ale auto również możemy opisać innymi **parametrami**

- Maksymalna prędkość
- Przyśpieszenie
- "Skrętność" - jak dobrze auto wchodzi w zakręty
- Pojemność paliwa

Są to **parametry** które nie widzimy, na pewno nie na pierwszy rzut oka.
Czy to wszystkie parametry jakimi możemy opisać samochód? Oczywiście że nie.

Możemy się także zastanowić czy samochód ma coś lub nie.

- Czy ma spoiler?
- Czy jest to automat czy manual?
- Czy ma bluetooth?
- Czy ma tempomat?

Możemy jeszcze określić tzw. "meta dane" samochodu, czyli informacje które w żaden sposób nie wpływają na sprawność lub wyposażenie pojazdu.

- Rok produkcji
- Historia pojazdu
- numer VIN

Trochę tego jest, ale zauważ proszę, że każdą z poszczególnych rzeczy możemy opisać w zmiennych.

```
// Parametry widoczne
int width;
int height;
int length;
String color;
String shape;
String manufacturerSymbol; // znak producenta
String lightColor;
int amountOfDoors;
// Parametry mniej widoczne
int maxVeclocity;
float acceleration;
float flexibility;
float capacityOfFluel;
// Czy coś ma?
```

```
boolean hasSpoiler;
boolean isAutomat;
boolean hasBluetooth;
boolean hasTempomat;
// Meta dane
int yearOdProduction;
String[] history;
String vin;
```

Trochę tego jest. Ale czy potrzebujemy te wszystkie informacje kiedy w naszym programie lub grze chcemy dodać coś co ma opisywać samochód? Nie koniecznie.

Dodajmy Kontekst (*ang. Context*)

Opisaliśmy wieloma parametrami samochód, ale po co? Po co nam to wszystko?
Przy opisywaniu dowolnego obiektu niezwykle istotny jest kontekst w jakim go opisujemy.

Chcemy stworzyć prostą grę 2D, wyścigową. Jakie parametry będą nam potrzebne?

Skoro to 2D, to wystarczy nam szerokość i wysokość abyśmy wiedzieli ile dany samochód zajmuje miejsca - jest to istotne gdy chcemy wykrywać kolizje, czyli czy dwa samochody się ze sobą zderzyły, lub z czymś innym.

Potrzebujemy widzieć jakiś obraz który prezentuje nasz samochód, przechowajmy go jako ścieżkę do pliku z obrazem.

Do sterowania naszym pojazdem przydadzą się na pewno parametry o jego możliwościach - maksymalna prędkość, przyśpieszenie, skrętność.

Dodajmy też parametry charakterystyczne dla gier - Aktualne położenie w świecie gry (X i Y), aktualna prędkość, rotacja - czyli jak obrócony jest nasz samochód.

Przedstawmy to za pomocą zmiennych

```
int width;
int height;
String pathToImage;
float maxVelocity;
float acceleration;
float flexibility;

int x;
int y;
```

```
float currentVelocity;
float rotation;
```

Teraz mamy zestaw parametrów opisujących samochód w naszej grze 2D. Teraz wiemy **po co** nam każdy parametr.

Opisaliśmy nasze auto w grze 2D, świetnie!
Jednakże... zauważmy że obiekty to nie tylko parametry, mogę one wykonywać jakieś zadania. Samochód się przemieszcza, pełni role transportową, pralka nam wypierze ubrania, zmywarka umyje naczynia, do lodówki możemy coś włożyć i wyjąć. Każdy obiekt pełni jakąś **funkcję**, a nawet kilka.

Co może zrobić dla nas samochód w naszej grze 2D?

- Może przyśpieszyć, czyli jechać do przodu
- Może się cofać
- Może skręcać

Przedstawmy to za pomocą **przykładowych** funkcji.

```
// przyśpieszenie, czyli dodanie przyśpieszenia do aktulanej prędkości
void addAcceleration(){
        float newVelocity = currentVelocity + acceleration;
        if (newVelocity > maxVelocity){
                currentVelocity = maxVelocity; // ograniczamy prędkość do
maksymlanej
        }else{
                currentVelocity = newVelocity; // ustawiamy nową prędkość
        }
}

void turnLeft(){
        rotation -= flexibility;
}

void turnRight() {
        rotation += flexibility;
}
// Jechanie do tyłu niech będzie odwrtonością jechania do przodu, po prostu
zamieniamy znaki na -
void turnBack(){
```

```
        float newVelocity = currentVelocity - acceleration;
        if (newVelocity < -maxVelocity){
                currentVelocity = -maxVelocity; // ograniczamy prędkość do
maksymlanej
        }else{
                currentVelocity = newVelocity; // ustawiamy nową prędkość
        }
}
```

Tak mogłyby wyglądać przykładowo funkcje do poruszania się naszym samochodem.

> ⓘ **Można to jeszcze rozszerzyć**
>
> Zauważ że możemy dodać więcej parametrów do naszego auta, a następnie użyć ich w funkcjach dla lepszej kontroli.
> Możemy dodać parametry odpowiadające szybkości cofania i jak szybko możemy się cofać - w końcu samochody nie jeżdżą tak szybko do tyłu jak do przodu ;)
> Możemy dodać parametry symbolizujące poziom uszkodzeń, a następnie modyfikować za ich pomocą ruch.
> Możemy dodać "nitro" które pozwoli nam przekroczyć maksymalną prędkość, tutaj również doszłyby nam kilka parametrów.
> Można wymyśleć jeszcze więcej, ale należy pamiętać o **kontekście**, każdy parametr ma pełnić jakąś rolę.

Myślę że już trochę wiesz czym jest patrzenie na świat obiektowo ale zajmijmy się jeszcze jednym przykładem.
Opisaliśmy samochód, rzecz fizyczną którą łatwo można sobie wyobrazić i podyskutować jakimi parametrami można go jeszcze opisać.

W OOP, obiekty mogą być abstrakcyjne. Opiszmy sobie teraz **kolor**.
Jak opisać kolor? Jakimi parametrami?

W informatyce kolor możemy opisać za pomocą 3-4 wartości.
Jest to natężenie koloru czerwonego, zielonego i niebieskiego.
Łącząc te trzy kolory i ich natężenie możemy uzyskać dowolny kolor.

Jako czwartą wartość możemy dodać tzw. kanał alpha. Czyli jak bardzo kolor jest przezroczysty, prześwitujący.

Przedstawmy zatem kolor za pomocą zmiennych

```
int red;
int green;
int blue;
int alpha;
```

> **Ciekawostka**
>
> Gdy będziesz pracował z kolorami w swoich programach to zazwyczaj się spotkasz że wartość koloru czerwonego, zielonego lub niebieskiego będzie z zakresu 0-255 przy liczbach całkowitych lub 0-1 przy liczbach zmienno-przecinkowych.

Obiekty mogą być jeszcze bardziej abstrakcyjne ale to zostawimy sobie na później.

Praktyka

Teraz trochę praktyki.

W programowaniu obiektowym, tworzymy swoje własne typy danych, na podstawie typów prostych. Takie typy nazywamy **klasami**.

```
Scanner in = new Scanner(System.in);
in.nextLine();

Random rand = new Random();
rand.nextInt();
```

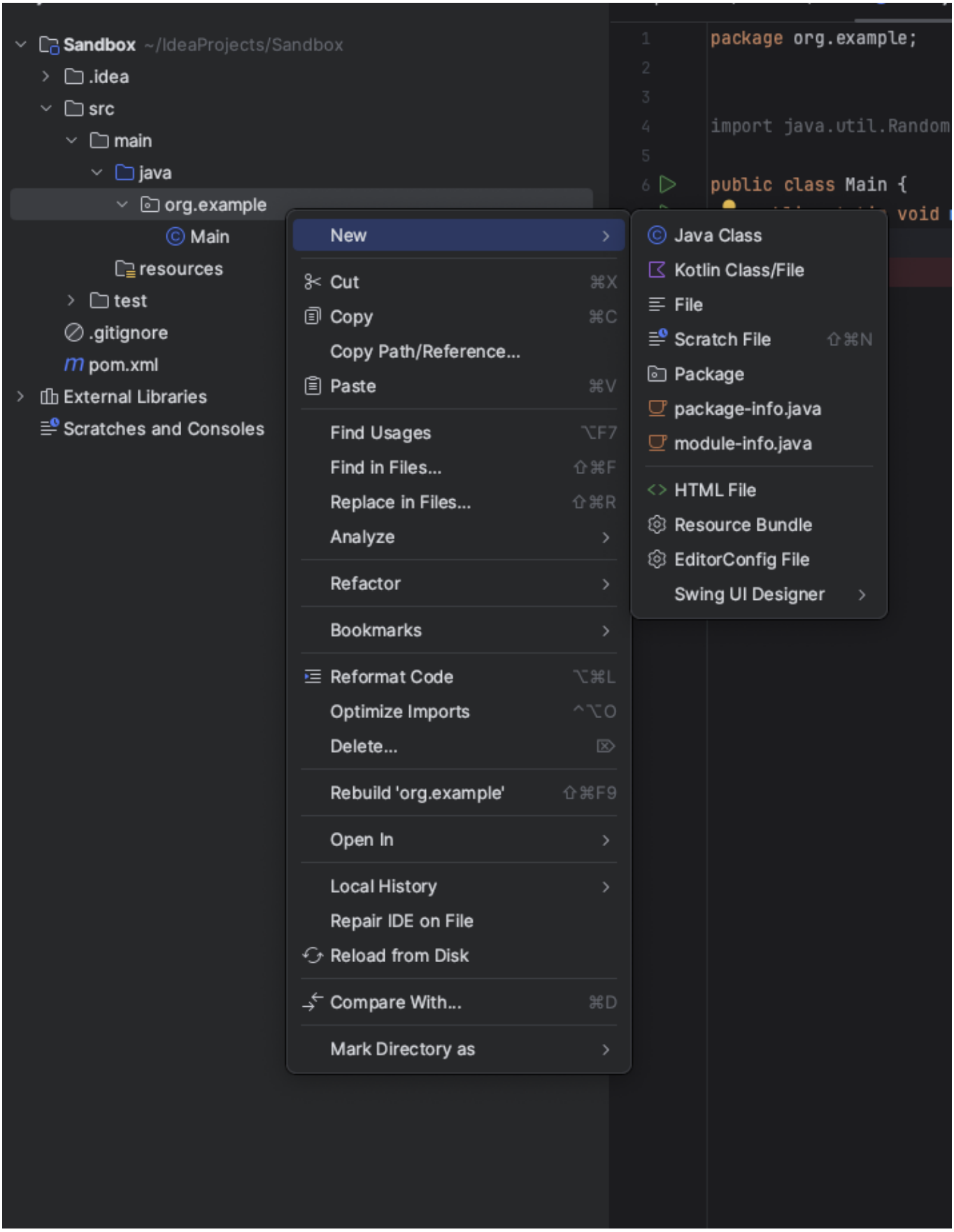
Scanner i **Random** są przykładem **klas** wbudowanych w język.

Klasa

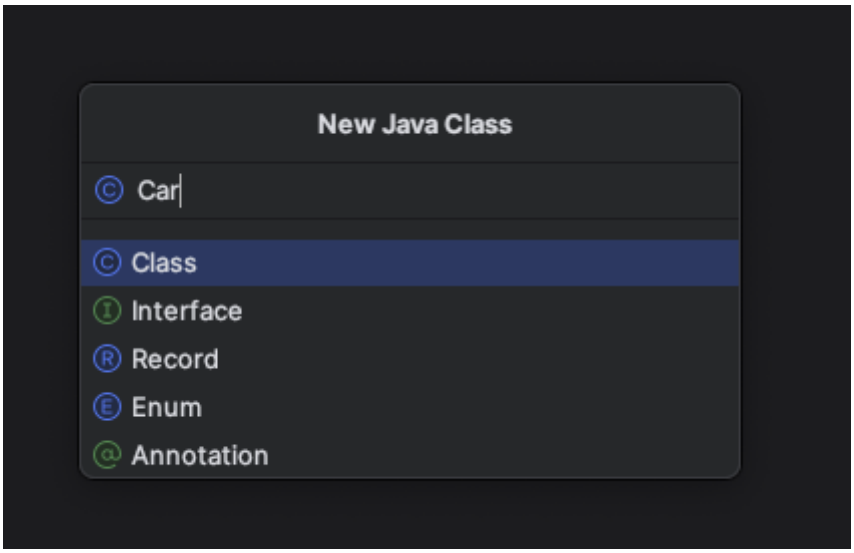
Często się mówi że klasa to taki wzór, przepis i jest to prawda, zdefiniowanie klasy nie utworzy nam obiektu, tam samo jak zaprojektowanie domu nie sprawi że ten dom się pojawi.

Każdą klasę tworzymy w osobnym pliku. Nazwa klasy **musi** nazywać się **dokładnie** tak samo jak nazwa pliku.

Stwórzmy nową klasę w tym samym miejscu co **Main**



Kliknij **PPM** (Prawy przycisk myszki) na katalog, wybierz *new* > *Java Class*



Podaj nazwę swojej klasy i naciśnij **Enter**
Wygeneruje nam to pustą klasę 'Car'

```
package org.example;

public class Car {
}
```

> ⓘ **Interface, Record, Enum, Annotation**
>
> Na ten moment, nie zaprzątaj sobie głowy znaczeniem tych opcji. Interfejsy i enumy poznasz w przyszłych materiałach. Adnotacjami (*ang. annotations*) nie będziemy się zajmować w tym kursie, a Record'y to nowość w Javie.

> ⓘ **Nazewnictwo**
>
> Nazwę klasy zaczynamy dużą literą, jeżeli ma więcej niż jedno słowo w nazwie tak analogicznie jak przy zmiennych, stosujemy *camel case*.
>
> - Car
> - CarFactory

Properties

Aby dodać właściwości klasy, powiedzieć z czego ona się składa to należy między nawiasy `{}` dodać zmienne.

```
public class Car {
        public int width;
        public int height;
        public float maxSpeed;
        public float acceleration = 0.5;
}
```

> **public**
>
> Public to jeden z akcesorów dostępu, tym zagadnieniem zajmiemy się w następnych materiałach. Na razie korzystaj z public.

Zmiennej *acceleration* ustawiliśmy wartość domyślną `0.5`

Metody

Pamiętasz że możemy wywołać metody na jakieś klasie?

```
Scanner in = new Scanner(System.in);
in.nextLine(); // metoda

Random rand = new Random();
rand.nextInt(); // metoda
```

Tak na prawdę już wiesz jak napisać taką metodę, jest to zwykła funkcja wewnątrz klasy.

```
public class Car {
        public int width;
        public int height;
        public float maxSpeed;
        public float acceleration;

        public void printAcceleration(){
                System.out.prinln("Przyśpieszenie wynosi: " + acceleration +
"m/s");
        }

        public int area(){
                return width * height;
        }
}
```

Zauważ że teraz wewnątrz funkcji możemy korzystać z zmiennych klasy.

> ⓘ **static**
>
> Zwróć również uwagę że tutaj nie używamy słowa kluczowego **static**. W następnych materiałach wyjaśnię jego znaczenie.

Instancja

Stworzyliśmy naszą klasę, czyli przepis, wzór na nasz obiekt. Jednakże, wciąż projekt domu nie sprawi że pojawi nam się dom. Jednak na podstawie projektu, możemy zbudować dom.

Spójrz jak to robiliśmy z klasą `Random`.

```
Random rand = new Random();
```

W taki sposób stworzymy **instancję naszej klasy**.

```
Car myCar = new Car();
```

> ⚠ **Instancja**
>
> Jeżeli tworzymy nowy obiekt, czyli gdy używamy słowa kluczowego **new** i nazwy klasy to mówimy że tworzymy **instancje danej klasy**.
> Czyli jak usłyszysz że masz stworzyć **instancje klasy car** to znaczy że masz użyć słowa kluczowego new.

Teraz możemy używać naszej **instancji** klasy car.

```
myCar.width = 5;
myCar.height = 10;
myCar.maxSpeed = 100;
myCar.acceleration = 1.5f;
myCar.printAcceleration();
System.out.println(myCar.area());
```

Instancja to stworzenie obiektu na bazie jego schematu. Schematem jest klasa.
Każda instancja jest podobna, ale unikatowa.
Oznacza to że możemy stworzyć wiele instancji danej klasy, jednak ich właściwości są oddzielne.
Tak jak producent TV tworzy wiele instancji "tych samych" modeli TV, to wcale nie znaczy że jeżeli my przełączymy kanał, to przełączy się on na każdym TV z tego modelu.

```
Car car1 = new Car();
car1.acceleration = 2.5f;
Car car2 = new Car();
car2.acceleration = 1.5f;

car1.printAcceleration();
car2.printAcceleration();
```

Output

```
Przyśpieszenie wynosi: 2.5m/s
Przyśpieszenie wynosi: 1.5m/s
```

Tworzymy dwa różne obiekt, ale na bazie tej samej klasy.

Podsumowanie

Poznaliśmy dziś podstawy programowania obiektowego, przed nami kilka dość złożonych zagadnień dlatego ważne jest abyś przyswoił/a to co dziś poznaliśmy.