Laboratorium 2a (IID-P)

Podstawy składni Javy.

Na zajęciach 13.03 były punkty 1-3, przy czym pkt 3 przed „przygotowanie projektu”- tj składnia klasycznej Javy-projekt „Java Application”. Z zadan: jedno 1-3

Poniżej najważniejsze elementy składni C#. W trakcie zajęć pojawią się kolejne mechanizmy np. operacje na kolekcjach, wyrażenia lambda.

1. Podstawowe typy danych:
`int, byte, short, double, float` -liczbowe
`char` – 1 znak (kod UTF-16)
`boolean` – prawda/fałsz (stosowany w instrukcjach warunkowych if, warunkach pętli, porównania < ,>,==,<=,>= zwracają typ boolean)

`String` – ciąg znaków (klasa), wyżej wymienione typy to typy podstawowe

Równolegle do powyższych typów są klasy opakowujące te wartości (np. Integer,...) i mają swoje metody (np. parseInt,valueOf). Po kropce środowisko (np. NetBeans/Eclipse...) je podpowie – pisząc typ -statyczne metody struktury, lub po zmiennej -metody obiektu)

2. Klasy
Dla potrzeb poniższego ćwiczenia utwórz projekt aplikacji Java (uwaga jeśli realizujesz pkt 3-własciwości -to od razu niech będzie to projekt JavaFX). Dodaj nową klasę.

```
package pl.edu.wsiz.lab3a;
/** @author lgajecki
 */
public class Pojazd {
}
```

Dyrektywa import dołącza pakiety (tu jeszcze nie występuje). Jednocześnie klasa Pojazd znajduje się w pakiecie `pl.edu.wsiz.lab3a`. Jest to ważne przy bardziej złożonych programach (np. tak samo może nazywać się klasa reprezentująca obiekt – rekord z bazy danych, klasa służąca np. przechowania tych danych do wyświetlenia w oknie, do zapisu do pliku,w przypadku aplikacji webowych zwrócenia zapytania z serwera itd., będą one musiały mieć inne przestrzenie nazw).
Klasa Pojazd będzie zawierała zmienne prywatne jak niżej (piszemy to w klasie)

```
    private String nr_rej;
    private String marka;
    private int liczba_kol;
```

Przykład konstruktora z parametrami:

```
    public Pojazd(String nr_rej,String marka, int liczba_kol) {
            this.nr_rej = nr_rej;
            this.marka = marka;
            this.liczba_kol = liczba_kol;
        }
```

Zauważ, że nr_rej to zmienna lokalna (parametr metody-konstruktora), więc przysłania ona zmienną klasy nr_rej. Do zmiennej (pola) klasy możemy odwołać sie przez this -wskaźnik na obiekt klasy.

Przykładowa metoda – tutaj wypisze ona wartości pól klasy (zmiennych klasy).

```
    public void wypisz() {
            System.out.println(nr_rej+","+marka+","+liczba_kol);
        }
```

Napisz metody publiczne - jedna ustawiająca wartość pola prywatnego klasy, druga – zwracająca wartość takiego pola. Nazywamy jest setterami i getterami.

```
    public  void setMarka(String marka){
        this.marka=marka;
    }

    public String getMarka(){
        return marka;
    }
```

Główna klasa aplikacji:

```
package pl.edu.wsiz.lab3a;
public class Lab3a {

    public static void main(String[] args) {
        System.out.println("Hello World!");
        Pojazd p = new Pojazd("RZ12345", "Fiat", 4);
        p.wypisz();
        p.setNr_Rej("KR10000");
        System.out.println("Nr rej: " + p.getNr_Rej());

    }
}
```

3. Właściwości (JavaFX)
W Javie nie ma określonej składni związanej z właściwościami. Interfejs właściwości określamy jako settery i gettery (przykład z poprzedniego punktu). Zakładamy nazwę metod getXXX setXXX , gdzie zmienna prywatna to xxx

```
    public  void setMarka(String marka){
        this.marka=marka;
    }

    public String getMarka(){
        return marka;
    }
```

Jest też możliwe użycie klas właściwości z pakietu javafx.beans.property (JavaFX)

Przygotowanie projektu.
W celu skorzystania z pakietów JavaFX utwórz projekt JavaFX. Zrób klasę aplikacji, zawierającą tylko metodę main

public static void main(String[] args)

Gdzie w tej metodzie jak wcześniej utworzysz obiekt klasy, wywołasz jego metody, wypiszesz na konsoli itd.

Właściwości

```
    public Pojazd(String nr_rej,String marka, int liczba_kol) {
        this.nr_rej = new SimpleStringProperty(nr_rej);
        //...
    }

    public  void setNr_rej(String nr_rej){
        this.nr_rej.set(nr_rej);
    }

    public String getNr_rej(){
        return nr_rej.get();
    }

    public SimpleStringProperty getNr_rejProperty(){
        return nr_rej;
    }
```

4. Dziedziczenie

W klasie Pojazd zmień kwalifikatory dostępu do zmiennych na protected (chroniony), co oznacza, że na zewnątrz klasy będą niewidoczne, ale widoczne w klasie pochodnej.

```
public class Pojazd {
    protected SimpleStringProperty nr_rej;
//można też: private SimpleStringProperty nr_rej = new SimpleStringProperty();
    protected String marka;
    protected int liczba_kol;
}
```

W osobnym pliku napisz klasę Samochod, która dziedziczy z klasy Pojazd oraz deklaruje zmienną prywatną liczba_miejsc typu int.

```
public class Samochod extends Pojazd {
    private int liczba_miejsc;
}
```

Możemy też nadać wartości wszystkich zmiennych w zwykły sposób, pamiętając, że jednak wywołujemy jakiś konstruktor klasy podstawowej (co można sprawdzić debugując program).

Napisz konstruktor z parametrami. Konstruktor wywołuje konstruktor klasy bazowej przez słowo super. Jeśli bezpośrednio nie wywoła konstruktora klasy bazowej, to znaczy, że wywołuje konstruktor bezparametrowy super()- tutaj byłby to Pojazd() - ale ponieważ zdefiniowaliśmy już inny konstruktor w tej klasie, to takie nie mamy. Wywołanie konstruktora klasy bazowej może być użyte do nadania wartości zmiennym klasy bazowej. Wówczas w zwykły sposób (np. przypisaniami) nadajemy wartości nowym zmiennym bieżącej klasy.

```
public class Samochod extends Pojazd {
    private int liczba_miejsc;
    public Samochod(String nr_rej, String marka, int liczba_kol, int liczba_miejsc){
            super(nr_rej,marka,liczba_kol);
            this.liczba_miejsc = liczba_miejsc;
        }
```

Możemy też nadać wartości wszystkich zmiennych w zwykły sposób, pamiętając, że jednak wywołujemy jakiś konstruktor klasy podstawowej (co można sprawdzić debugując program).

Przykładowe utworzenie obiektu klasy Samochod (np. w metodzie Main klasy App):

```
        Samochod s = new Samochod("RZ10101", "Hundai", 4, 5);
        s.wypisz();
        System.out.println("Po zmianie NrRej: ");
        s.setNr_rej("RZ10102");
        s.wypisz();
```

Metoda wypisz() klasy Pojazd będzie polimorficzna. Wszystkie metody w Javie są wirtualne.

```
    public void wypisz() { //w klasie bazowej
}
```

Natomiast w klasie pochodnej użyjemy słowa override lub je pominiemy, ale będzie ten sam zestaw parametrów przy tak samo nazywającej się metodzie klasy bazowej.

```
    @Override
    public void wypisz() {
            System.out.println(getNr_rej()+","+marka+","+liczba_kol+","+liczba_miejsc);
        }
```

Wywołanie metody polimorficznej (np. w metodzie Main klasy Program):

```
        Pojazd p1 = new Samochod("RZ20000", "BMW", 4, 3);
        p1.wypisz(); ///metoda wypisz klasy Samochod, bo jest to obiekt Samochod

        Pojazd p2 = p1; //przypisanie -to tylko skopiowanie wskaźnika, np. zmieniając
                        //jedną zmienną/właściwość w obiekcie p2 zmienimy też w p1
        p2.wypisz();
```

Aby mechanizm polimorfizmu zadziałał obiekt klasy Samochod mógłby też zostać przekazany do metody mającej parametr klasy bazowej np.

```
    public static void fun(Pojazd p) {
               p.wypisz();
     }
```

Nadpisanie metody ToString zdefiniowanej w klasie object:

```
    @Override
    public String toString(){
        return String.format("%s %s %d %d", this.getNr_rej(),marka,liczba_kol,liczba_miejsc);
    }
```

Można ciąg znaków tworzyć, przez łączenie ciągów, np:

```
this.getNr_rej()+","+marka
```

Użyliśmy tutaj metody String.format pozwalajacej na kontrolę nad formatowaniem ciągów znaków. Pierwszy parametr -ciąg formatujący, kolejne parametry – przekazywane zmienne. Najważniejsze zasady ciągu formatującego: %Indeks_argumentu$konwersja
konwersja: %d- liczba całkowita dziesiętna,%f zmienno przecinkowa, %x szesnastkowa, %s ciąg znaków)
Indeks_argumentu – gdy się pominie -to będzie kolejny argument (przekazywana zmienna), można podając indeks wyświetlać zmienne w innej kolejności, lub powtórzyć zmienną. Numerowanie argumentów od 1. np.
`"%1$s"` -argument 1, formatowany jako ciąg znaków.

Zauważ,że (np. w metodzie Main klasy Program) możemy wywołać metodę `toString()` bezpośrednio (np. by wypisać na konsoli):

```
        System.out.println(s.toString());
//s jest wcześniej utworzonym obiektem klasy Samochod
```

Możemy też przekazać zmienną s bezpośrednio do metody WriteLine, albo połaczyć ze stringiem. Wówczas automatycznie zostanie wywołana metoda ToString():

```
       System.out.println(s);
```

Rzutowanie typu. Wyrażenie instanceof -sprawdza czy zmienna jest określonego typu, np.:

```
        if (s instanceof Samochod)
            System.out.println("s to obiekt klasy Samochod");
```

Wyrażenie `(typ)zmienna` dokonuje konwersji typu na typ klasy pochodnej. Jeśli obiekt nie jest tej klasy albo klasy pochodnej od podanej jest zwracany null.
Aby to sprawdzić wypiszmy właściwość z klasy Samochod. Patrz przykłady wyżej: zmienna p1 jest zadeklarowana jako Pojazd (klasa bazowa), ale wpisaliśmy do niej obiekt klasy Samochod (klasa potomna) Aby otrzymać tą własciwość Liczba_miejsc, dla zmiennej p1 wykonamy rzutowanie:

```
        int m=((Samochod)p1).getLiczba_miejsc();
        System.out.println("p1 jest klasy Samochod,  p1.getLiczba_miejsc():" + m);
```

5. Interfejsy mogą zawierać metody. Są one wirtualne i publiczne, nie używamy jednak public czy virtual. Do tego nie zawierają implementacji metod, a jedynie ich sygnatury. Mogą zawierać właściwości (bo właściwości są metodami).

```
public interface Writable {
    void wypisz();
}
```

Aby klasa Pojazd dziedziczyła z tego interfejsu należy zmienić jej deklarację:

```
public class Pojazd implements Writable {
}
```

oraz dodać override w metodzie wypisz.

```
    @Override
    public void wypisz() {
}
```

Klasa dziedzicząca z interfejsu musi implementować wszystkie jego metody -tu write(). Jeśli jest klasą abstracyjną (abstract) nie musi implementować wszystkich metod, ale nie można tworzyć jej obiektów. Tak samo nie można tworzyć obiektów interfejsu, vo jego metody nie są zaimplementowane. Można za to użyć zmiennych typu interfejsu.
Przykład wywołania:

```
        Writable wr = p1;
         System.out.println("Metoda interfejsu:");
         wr.wypisz();
```

6. Dokumentowanie klasy. Szablon dokumentowania jest dostepny po napisaniu /** przed metodą klasy klasą, zmienną klasy, właściwościami, (w przyszłości zdarzeniami, delegatami,…). Szablon ten jest rozpoznawany przez narzędzia Javadoc czy Doxygen – do generowania dokumentacji. Doxygen ma jeszcze inne typy dokumentacji, jak np. w tej samej linijce co deklaracja zmiennej.

```
/**
```

```
     *
     * @param nr_rej numer rejestracyjny
     * @param marka marka samochodu
     * @param liczba_kol liczba kół
     * @param liczba_miejsc liczba miejsc
     */
    public Samochod(String nr_rej, String marka, int liczba_kol, int liczba_miejsc){ ///
}
```

Zadania

Na zajęciach 13.03 było z zadań: przynajmniej jedno 1-3

1. Napisz klasę Prostokat , posiadającą:
zmienne a,b (boki prostokąta), typu np. double
właściwości a,b (- tj. metody setA,getA, itd. pozwalające na zapis/odczyt zmiennych odpowiednio a,b)
metody:
void info() - wypisującą wartości zmiennych a, b
double pole() zwracającą wartość pola prostokąta
konstruktor (do wyboru: bezparametrowy, lub pozwalający na podanie parametrów)

2. Utwórz obiekt klasy Prostokat , przetestuj
działanie metod, właściwości
działanie konstruktora, w tym jednocześnie przypisując wartości do właściwości

3. Napisz klasę Figura z metodami wirtualnymi void info() oraz double pole(), wykonaj by klasa Prostokat dziedziczyła z klasy Figura, odpowiedni konstruktor.

4. Przetestuj działanie polimorfizmu dla klas Figura, Prostokat, na podobnej zasadzie możesz utworzyć klasę Kolo ze zmienną promien

5. Utwórz kolekcję zawierającą obiekty klasy Prostokąt, Kolo

6. Wyszukaj obiekty spełniające określone warunki, np. pole()<4