Laboratorium 4

Wprowadzenie – klasy i metody abstrakcyjne – zadanie 1

Klasa abstrakcyjna to klasa, która nie posiada swoich reprezentantów pod postacią obiektów. Jest ona wykorzystywana wyłącznie w roli klasy bazowej dla innych klas. Klasa potomna względem klasy abstrakcyjnej musi implementować jej wszystkie abstrakcyjne metody i właściwości. Klasa abstrakcyjna może zawierać także pola oraz metody i właściwości, które nie są abstrakcyjne. Klasa, która zawiera abstrakcyjną właściwość lub metodę, również musi być abstrakcyjna. Klasa abstrakcyjna jest oznaczona modyfikatorem „abstract".

Metoda abstrakcyjna jest metodą oznaczoną modyfikatorem „abstract". Posiada ona jedynie deklarację w klasie abstrakcyjnej. Definicja metody znajduje się w klasach potomnych, dziedziczących po klasie abstrakcyjnej. Metody abstrakcyjne nie mogą być prywatne.

Utwórz nowy projekt.

Utwórz nową klasę `FiguraGeometryczna` (jako klasę publiczną, w osobnym pliku). Dodaj pole (zmienną) typu string o nazwie `nazwa` i przypisz wartość domyślną `null`. Utwórz gettery i settery do tego pola. Dodaj abstrakcyjną metodę `obliczPole()`, która zwracać będzie liczbę zmiennoprzecinkową typu `double`.

```
public abstract class FiguraGeometryczna {

    protected String nazwa = null;

    public abstract double obliczPole();

    public String getNazwa() {
        return nazwa;
    }

    public void setNazwa(String nazwa) {
        this.nazwa = nazwa;
    }

}
```

Następnie utwórz nową klasę publiczną (w osobnym pliku) o nazwie `Kolo`. Klasa ta będzie klasą potomną i będzie dziedziczyła po klasie `FiguraGeometryczna`. Klasa ta powinna przechowywać informację o wartości promienia, konstruktor, dzięki któremu przekazana zostanie wartość promienia oraz nadana nazwa figury „Kolo". Powinna również zaimplementować funkcję obliczPole() i zwracać obliczoną wartość pola koła.

```
public class Kolo extends FiguraGeometryczna {

    private double promien;

    public Kolo(double promien) {
        nazwa = "Kolo";
        this.promien = promien;
    }

    @Override
    public double obliczPole() {
        return 3.14 * promien * promien;
    }

}
```

Analogicznie do klasy `Kolo` przygotuj klasę `Prostokat`. Klasa `Prostokat` dziedziczy po klase `FiguraGeometryczna`, powinna zawierać dwa pola o nazwach `bok1` i `bok2` reprezentujące długości dwóch boków prostokąta, konstruktor, który przyjmuje wartości dwóch boków i określa nazwę „Prostokąt” oraz funkcję `obliczPole()`, która zwraca obliczone pole prostokąta.

```
public class Prostokat extends FiguraGeometryczna {
    private double bok1 = 0.0;
    private double bok2 = 0.0;

    public Prostokat(double bok1, double bok2) {
        nazwa = "Prostokat";
        this.bok1 = bok1;
        this.bok2 = bok2;
    }

    @Override
    public double obliczPole() {
        return bok1 * bok2;
    }
}
```

Przetestuj działanie obydwu klas uruchamiając przykładowy program.

```
public static void main(String[] args) {
      FiguraGeometryczna f1 = new Kolo(10);
     System.out.println( f1.getNazwa() + ", pole: " + f1.obliczPole() );

     FiguraGeometryczna f2 = new Prostokat(2, 3);
     System.out.println( f2.getNazwa() + ", pole: " + f2.obliczPole() );
}
```

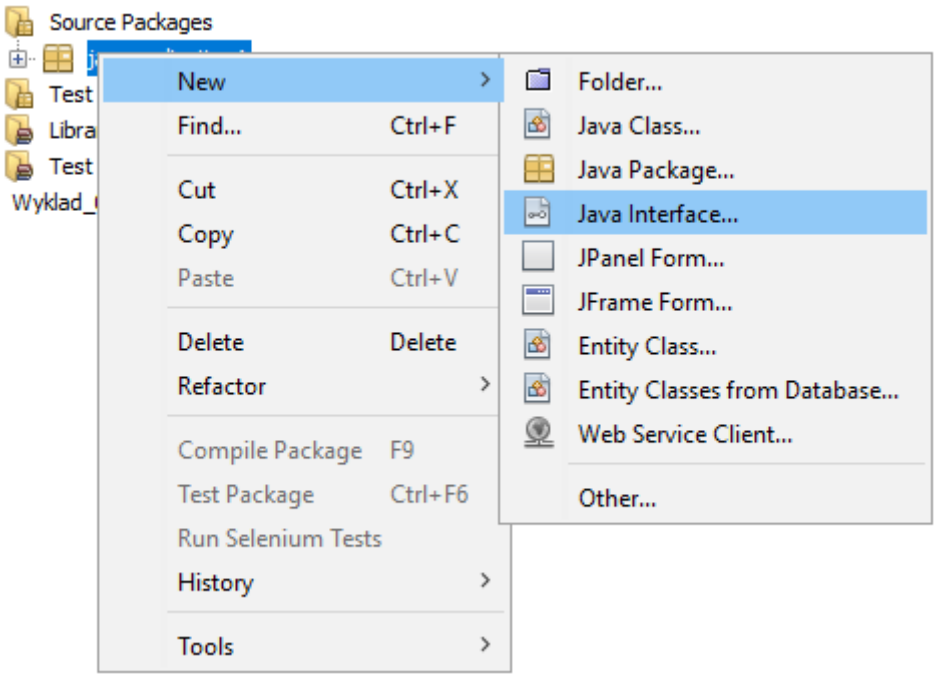
Analogicznie do powyższych przykładów przygotuj klasę reprezentującą `Trojkat`.

Wprowadzenie – interfejsy – zadanie 2

Interfejs jest abstrakcyjną reprezentacją klasy, która deklaruje swoje metody (funkcje), ale ich nie implementuje. Tworzenie obiektów interfejsu nie jest możliwe. Klasy dziedziczące po interfejsie muszą implementować wszystkie jego składowe. Interfejsy nie mogą zawierać pól. Wszystkie składowe interfejsu muszą być publiczne. Klasa może dziedziczyć po kilku interfejsach jednocześnie. Interfejs tworzy się z wykorzystaniem słowa kluczowego „interface”. Implementację interfejsu oznaczamy przez słowo „implements”.

Utwórz nowy projekt.

Utwórz nowy publiczny interfejs w podobny sposób jak nową klasę, wybierając ją z menu kontekstowego:



Podaj nową nazwę interfejsu `IInstrument`, dodaj dwie funkcje: `graj()` oraz `getNazwa()`.

```
public interface IInstrument {

    public void graj();
    public String getNazwa();

}
```

Utwórz nową klasę Beben, która implementuje interfejs `IInstrument`. Zaimplementuj dwie metody z interfejsu.

```
public class Beben implements IInstrument{

    @Override
    public void graj() {
        System.out.println("Bum bum bum");
    }

    @Override
    public String getNazwa() {
        return "Bęben";
    }

}
```

Dodaj kolejną publiczną klasę `Traba`, która również implementuje interfejs `IInstrument` oraz dodaj dwie brakujące metody.

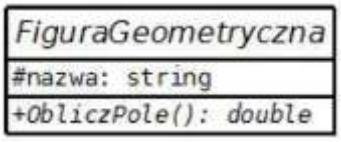
```
public class Traba implements IInstrument {

    @Override
    public void graj() {
        System.out.println("Tra ta ta");
    }

    @Override
    public String getNazwa() {
        return "Traba";
    }
}
```

Dodaj kolejne dwa instrumenty wzorując się na przykładzie. Przetestuj działanie interfejsów uruchamiając poniższy program. Dodaj do `orkiestra` kolejne dwa instrumenty.

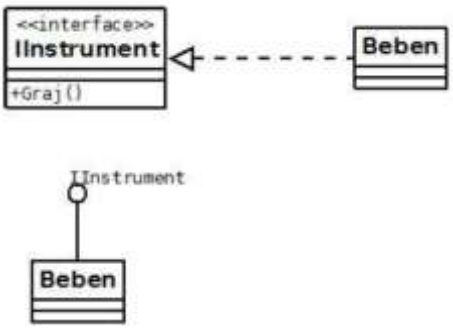
```
public static void main(String[] args) {
    List<IInstrument> orkiestra = new ArrayList<IInstrument>();
    orkiestra.add(b);
    orkiestra.add(t);

    System.out.print("Orkiestra w składzie: ");
    for (IInstrument instrument : orkiestra) {
        System.out.print( instrument.getNazwa() + ", " );
    }
    System.out.println();
    for (IInstrument instrument : orkiestra) {
        instrument.graj();
    }
}
```

Wprowadzenie – UML klasy abstrakcyjne i interfejsy

Klasy abstrakcyjne w języku UML przedstawia się pisząc ich nazwę oraz nazwę ich metod abstrakcyjnych kursywą.



Interfejs oraz dziedziczenie po interfejsie na diagramach klas można przedstawić w dwojaki sposób:

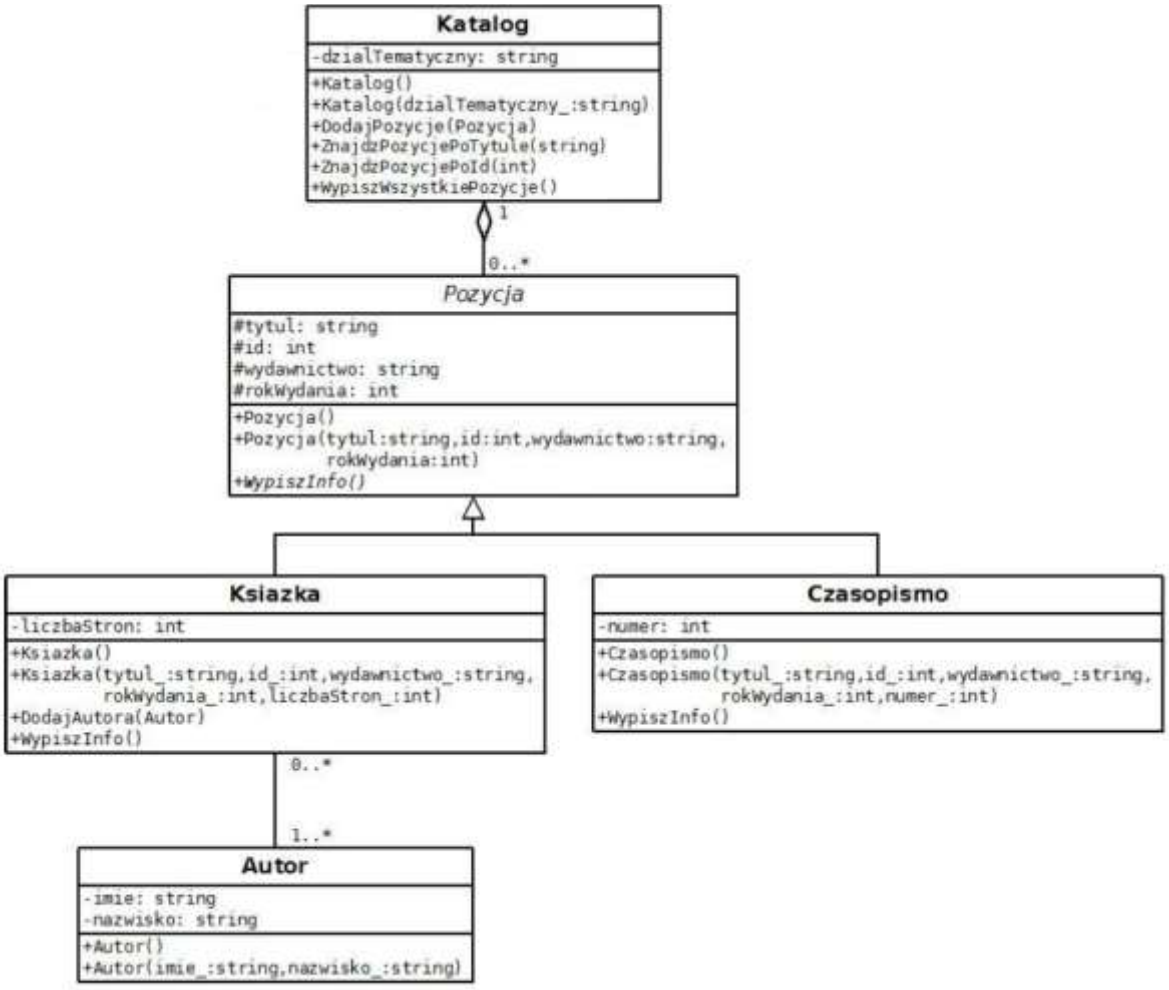


Zadanie 3

Zrealizuj aplikację obiektową, która powinna odznaczać się następującymi cechami:

- Aplikacja powinna zawierać klasy zaimplementowane zgodnie z zamieszczonym poniżej diagramem klas.

- Metody „WypiszInfo” powinny wypisywać na ekranie konsoli informacje na temat wartości wszystkich pól obiektów.
- Metody „ZnajdzPozycje…” powinny wyszukiwać w katalogu pozycję spełniającą dane kryteria i zwracać odpowiednią referencję. Jeśli żaden obiekt w katalogu nie spełnia danego kryterium, metoda powinna zwrócić wartość „null”.
- Metoda „WypiszWszystkiePozycje” powinna wypisywać informacje o wszystkich pozycjach w katalogu.
- Przechowywanie obiektów pozycji w klasie „Katalog” oraz przechowywanie obiektów autorów w klasie „Ksiazka” powinno być zrealizowane za pomocą kolekcji typu „List<T>”.
- Należy zwrócić uwagę na to, że klasa „Pozycja” jest klasą abstrakcyjną.
- Po wykonaniu zadania należy je przetestować za pomocą własnego kodu testowego.



Zadanie 4

Zrealizuj aplikację obiektową, która powinna odznaczać się następującymi cechami:

- Aplikacja stanowi rozszerzenie i modyfikację aplikacji z zadania nr 1.
- Aplikacja powinna zawierać klasy zaimplementowane zgodnie z zamieszczonym poniżej diagramem klas.
- Metody do zarządzania pozycjami zostały zdefiniowane w interfejsie „IZarzadzaniePozycjami”.

- Metody „ZnajdzPozycje…” i „WypiszWszystkiePozycje” z klasy „Biblioteka” mają przeszukiwać wszystkie katalogi zawarte w bibliotece.
- Metoda „DodajPozycje” z klasy biblioteka ma dodawać pozycję do katalogu o podanej nazwie działu tematycznego.
- Należy zauważyć, że metody do zarządzania pozycjami z klasy „Biblioteka” będą wykonywać operacje na obiektach typu „Katalog”, wykorzystując przy tym metody zdefiniowane w klasie „Katalog”.
- Po wykonaniu zadania należy je przetestować za pomocą własnego kodu testowego.