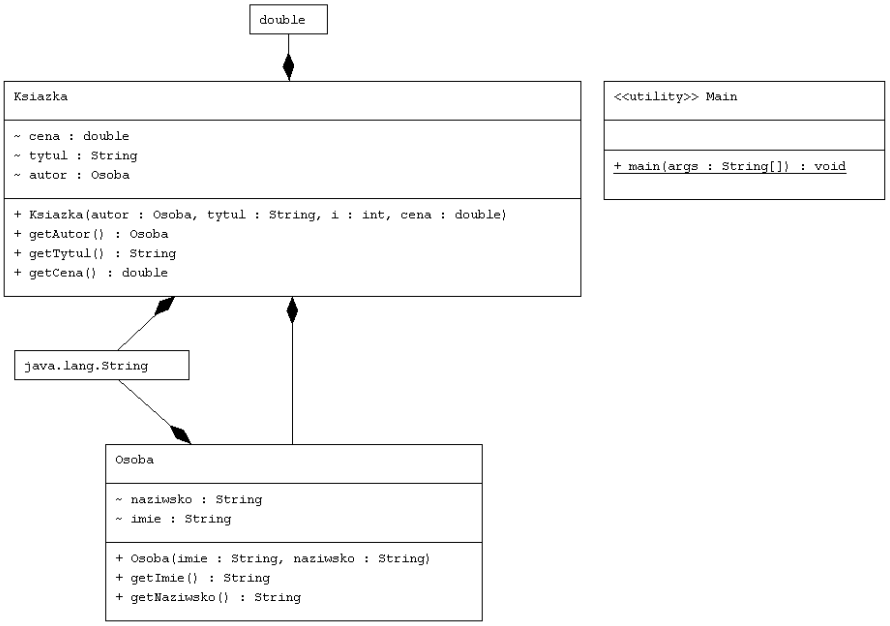
PROGRAMOWANIE OBIEKTOWE JAVA – LABORATORIUM

DZIEDZICZENIE

KOMPOZYCJA

Z koncepcyjnego punktu widzenia kompozycja oznacza, że "obiekt jest zawarty w innym obiekcie" . *Jest to relacja "całość – część" (B "zawiera" A)*. Np. obiekty typu Pojazd zawierają obiekty typu Rozmiar, Koła, Silnik itd.. Kompozycję uzyskujemy poprzez definiowanie w nowej klasie pól, które są obiektami istniejących klas.




```
package PO_UR.Lab06.Kompozycja;

public class Osoba {
    String imie, naziwsko;

    public Osoba(String imie, String naziwsko) {
        this.imie = imie;
        this.naziwsko = naziwsko;
    }

    public String getImie() {
        return imie;
    }

    public String getNaziwsko() {
        return naziwsko;
    }
}
```

```
package PO_UR.Lab06.Kompozycja;

public class Ksiazka {
    Osoba autor;
    String tytul;
    double cena;

    public Ksiazka(Osoba autor, String tytul, int i, double cena) {
        this.autor = autor;
```

```
        this.tytul = tytul;
        this.cena = cena;
    }

    public Osoba getAutor() {
        return autor;
    }

    public String getTytul() {
        return tytul;
    }

    public double getCena() {
        return cena;
    }
}
```

```
package PO_UR.Lab06.Kompozycja;

public class Main {
    public static void main(String[] args) {
        Ksiazka k1 = new Ksiazka(new
Osoba("Henryk","Sienkiewicz"),"Potop",32,50);
    }
}
```

DZIEDZICZENIE

Dziedziczenie jest jednym z podstawowych mechanizmów programowania obiektowego. Mechanizm ten umożliwia definiowanie nowych klas na bazie istniejących.

Dziedziczenie jest w języku Java mechanizmem wszechobecnym i niezwykle potężnym. Prawie każda klasa a mówiąc precyzyjniej każda klasa z wyjątkiem klasy java.lang.Object – dziedziczy z jakiejś innej klasy, każda bowiem klasa dziedziczy w sposób niejawny ze wspomnianej klasy Object.

W Javie **NIE MA** wielodziedziczenia - klasa może bezpośrednio odziedziczyć tylko jedną klasę.

Dziedziczenie - podobnie jak kompozycja (a nawet w większym stopniu) - pozwala na zmniejszanie nakładów na kodowanie (reusing). Jest to również odzwierciedlenie naturalnych sytuacji.

Dziedziczenie polega na przejęciu właściwości i funkcjonalności obiektów innej klasy i ewentualnej modyfikacji tych właściwości i funkcjonalności w taki sposób, by były one bardziej wyspecjalizowane. Jest to relacja, nazywana *generalizacją-specjalizacją*: *B "jest typu" A*, *"B jest A"*, a jednocześnie *B specjalizuje A*. *A jest generalizacją B*.

```
package PO_UR.Lab06.Dziedziczenie;

public class Pracownik {
    String imie, nazwisko;
    int wyplata;

    public Pracownik() {
        imie = " ";
        nazwisko = " ";
        wyplata = 0;
    }

    public Pracownik(String imie, String nazwisko, int wyplata) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.wyplata = wyplata;
```

```
    }

}
```

```
package PO_UR.Lab06.Dziedziczenie;

public class Szef extends Pracownik{
    int premia;

    public Szef(String imie, String nazwisko, int wyplata, int premia) {
        super(imie, nazwisko, wyplata);
        this.premia = premia;
    }
}
```

```
package PO_UR.Lab06.Dziedziczenie;

public class Firma {
    public static void main(String args[]){
        Pracownik prac = new Pracownik("Wlodek", "Zięba", 3000);
        System.out.println("Imię: "+prac.imie);
        System.out.println("Nazwisko: "+prac.nazwisko);
        System.out.println("Wypłata: "+prac.wyplata+"\n");

        //najpierw stwórzmy obiekt klasy Szef korzystając z domyślnego
konstruktora
        Szef szef = new Szef("jan","kowalki",12345,1);

        //zobaczmy jak wyglądają odpowiednie pola
        System.out.println("Imię: "+szef.imie);
        System.out.println("Nazwisko: "+szef.nazwisko);
        System.out.println("Wypłata: "+szef.wyplata);
        System.out.println("Premia: "+szef.premia+"\n");

        //teraz ustawiamy dane szefa
        szef.imie = "Tadeusz";
        szef.nazwisko = "Kowalski";
        szef.wyplata = 10000;
        szef.premia = 2000;
        System.out.println("Imię: "+szef.imie);
        System.out.println("Nazwisko: "+szef.nazwisko);
        System.out.println("Wypłata: "+szef.wyplata);
        System.out.println("Premia: "+szef.premia);
    }
}
```

RZUTOWANIE OBIEKTÓW, STWIERDZENIE TYPU

Rzutowanie (ang. cast) jest to operacja polegająca na zmianie zmiennej referencyjnej jednego typu na zmienna referencyjną innego typu. W Javie wyróżniamy dwa rodzaj rzutowania obiektów:

- rzutowanie w górę (ang. upcasting) – bezpieczne,
- rzutowanie w dół (ang. downcasting) – wymaga testowania (stwierdzenia typu instancji obiektu).

Stwierdzeniu jakiego typu jest referencja służy operator instanceof

```
ref  instanceof  T
```

Przy tym:

- wyrażenie null instanceof dowolny_typ zawsze ma wartość false,
- wyrażenie x instanceof T, będzie błędne składniowo (wystąpi błąd w kompilacji), jeśli typ referencji x i typ T nie są związane stosunkiem dziedziczenia,
- wyrażenie x instanceof T będzie miało wartość false, jeśli faktyczny typ referencji x jest nadtypem typu T.

Przykład

```
//rzutowanie w górę
Szef kierownik = new Szef();
Pracownik prac = (Szef) kierownik; // rzutowanie w goróę Szef -> Pracownik

//rzutowanie w dół
Pracownik pra1 = (Pracownik) new Szef();
if(pra1 instanceof Szef){   //spr przed rzutowaniem
    Szef szef = (Szef) pra1;    //rzutowanie w dol pracownik na szef
}
```

PRZESŁONIĘCIE METOD W DZIEDZICZENIU

Łatwo sobie wyobrazić sytuację, w której metoda o tej samej sygnaturze występuje zarówno w klasie bazowej jak i klasie pochodnej. W tej sytuacji mówimy o tym, że klasa pochodna przesłania metodę z klasy bazowej (ang. override). Proszę spójrz na przykład poniżej.

W klasie pracownik mamy metodę:

```
public void Wypisz(){
    System.out.println("jestem pracownikiem, moje dane: \tImie: "+imie+"
\tNazwisko: "+nazwisko+" \tWyplata: "+wyplata);
}
```

W klasie szef

```
public void Wypisz(){
    System.out.println("jestem szefem, moje dane: \tImie: "+imie+"
\tNazwisko: "+nazwisko+" \tWyplata: "+wyplata);
    System.out.println("Ponadto mam premie: "+premia);
}
```

wywołanie:

```
prac.Wypisz();
szef.Wypisz();
```

rezultat:

```
jestem pracownikiem, moje dane:     Imie: Wlodek    Nazwisko: Zięba
Wyplata: 3000
jestem szefem, moje dane:   Imie: jan   Nazwisko: kowalki   Wyplata: 12345
Ponadto mam premie: 1
```

Modyfikacja metody w klasie Szef z użyciem metody super

```
public void Wypisz(){
    super.Wypisz();
    System.out.println("Ponadto mam premie: "+premia);
}
```

Zadania do samodzielnego rozwiązania:

Zadanie 1

1. Utwórz projekt zawierający funkcje Main.
2. Utwórz klasę Punkt, która zawiera publiczne pola (współrzędne): x oraz y.
3. Dla klasy Punkt zdefiniuj konstruktor bezparametrowy, który zainicjuje początkowe wartości atrybutów.
4. Zdefiniuj drugi konstruktor, który przekazane parametry zapamięta, jako wartości atrybutów: x i y.
5. Zdefiniuj następujące metody dla klasy punkt:
   - gettery i settery
   - void zeruj();
   - void opis();
   - void przesun(int x, int y);
6. W metodzie main() utwórz trzy obiekty typu Punkt.
7. Pokaż, w jaki sposób można wykorzystać każdą ze zdefiniowanych metod.
8. Do pakietu pliki Figury, Prostokąt i Trojkat (udostępnione przez prowadzącego).
9. Utworzyć obiekty typu Figura, Prostokat i Trojkat. Sprawdzić działanie wybranych metod dla utworzonych obiektów.
10. Zaprojektować klasę Okrag zawierającą pola:
    - środek klasy Punkt – środek okręgu,
    - promień typu double

oraz metody:

- getPowierzchnia() zwracająca pole powierzchni,
- getSrednica() zwracająca średnice
- setPromien(double p) ustawiająca nowy promień
- getPromien() zwracająca promień
- wSrodku(Punkt) sprawdzająca czy dany punkt znajduje się wewnątrz okręgu. Skorzystaj ze wzoru: $(x-a)^2+(y-b)^2 \le r^2$, $S=(a,b)$

Konstruktory:

- Pusty – inicjujący pola wartościami domyślnymi punkt (0,0), promień 0,
- Określający punkt oraz promień

11. Zmodyfikować klasę Prostokat i Trojkat, tak, aby dziedziczyły z klasy Figura.
12. Dodać konstruktor przeciążony dla klasy Prostokat postaci: Prostokat(float wys,float szer, String kolor). Wewnątrz konstruktora powinien być wywołany konstruktor z klasy bazowej.
13. Dodać konstruktor przeciążony dla klasy Trojkat postaci Trojkat(float wys,float podst,String kolor).

14. Dodać metodę przesuwającą prostokąt o dane współrzędne void przesun(float x, float y).
15. Dla obiektu typu Prostokat wywołać metodę przesun(3,5).
16. Zdefiniować nową klasę Kwadrat dziedziczącą z klasy Prostokat. Umieścić w niej gettery i setery.
17. Utworzyć dowolny obiekt typu Kwadrat i nadać mu dowolną wartość początkową.
18. Przysłonić metody opis() we wszystkich klasach.
19. Dla utworzonych obiektów wywołać metodę opis(), w taki sposób, aby opis o danym obiekcie został wyświetlony na konsolę.
20. Zmodyfikować klasę Okrag zawierającą, tak, aby dziedziczyła po klasie Figura.
21. Sprawdzić działanie zdefiniowanych metod dla przykładowych obiektów.

Zadanie 2

Napisz program, w którym będą dwie klasy: Samochod i SamochodOsobowy. W klasach tych powinny znajdować się następujące pola:
Samochod: Marka, Model, Nadwozie, Kolor, Rok produkcji, Przebieg (nie może być ujemny)
SamochodOsobowy: Waga (powinna być z przedziału 2 t – 4,5 t), Pojemność silnika (powinna być z przedziału 0,8-3,0), Ilość osób
Klasa SamochodOsobowy dziedziczy po klasie Samochod. W obydwu klasach utwórz konstruktor, który pobierze dane od użytkownika. Dodatkowo w klasie Samochod przeciąż konstruktor w taki sposób, by wartości pól były parametrami metody. W klasie Samochod utwórz także metodę, która wyświetli informacje o samochodzie. Przesłoń ją w klasie SamochodOsobowy. W metodzie Main() utwórz obiekt klasy SamochodOsobowy oraz dwa obiekty klasy Samochod (wykorzystując różne konstruktory). Wyświetl informacje o samochodach.

Zadanie 3.

Należy zaimplementować aplikację wykorzystującą: kompozycję i dziedziczenie, rzutowanie oraz operator this i metodę super dla następujących klas Księgarnia, Podręcznik, Powieść, Klient, Książka