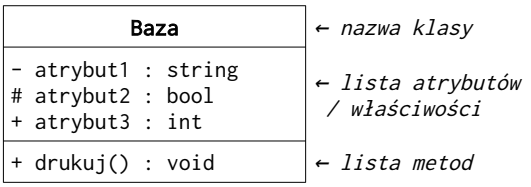
Definicja klasy w języku Java wygląda następująco (lewo), a w języku UML tak (prawo):

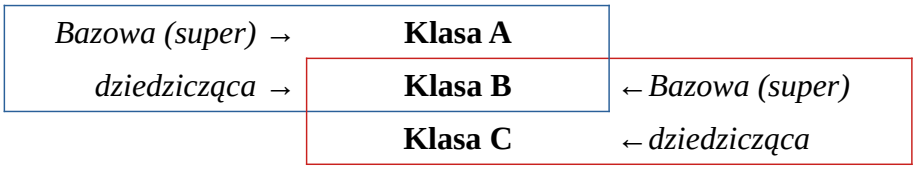
```
public class Baza {
        String atrybut1 = „Pierwsza klasa”;
        boolean atrybut2 = false;
        int atrybut3 = 1;
        public void drukuj(){
                String out = „attr1: ”+atrybut1;
                out+= ”\r\n”+„attr3: ”+atrybut3;
                System.out.println(out);
        }
}
```



W programowaniu obiektowym klasy współpracują ze sobą tworząc konkretne relacje. Możemy więc wyróżnić klasy:

- nadrzędne / bazowe / superklasy, czyli wzorce tworzenia kolejnych klas
- dziedziczące / pochodne / potomne, czyli tworzone na podstawie nadrzędnych, które mogą odziedziczyć po superklasach elementy takie jak: pola i metody.

Wybrana klasa może być jednocześnie klasą bazową dla klasy C i potomną dla klasy A.



Zadania teoretyczne:

1. Podaj składowe klasy Baza
2. Podaj składowe klasy Object
3. Umieść na powyższej tabelce klasę Baza oraz klasę Object

Zadania praktyczne oraz przykładowe kody są prezentowane w języku Java. Tam gdzie jest to wyraźnie zaznaczone zadania praktyczne należy jeszcze zaimplementować w języku PHP i/lub Python.

Zadania praktyczne:

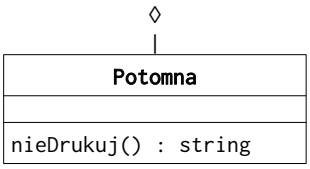
4. Zaimplementowaną w języku Java klasę Baza uzupełnij o modyfikatory dostępu.
5. Na podstawie diagramu UML klasy Baza zaimplementuj tę klasę w języku PHP.

Dziedziczenie

Jest mechanizmem ponownego użycia kodu poprzez tworzenie nowej klasy na podstawie wskazanej klasy wzorcowej. Klasa bazowa udostępnia klasie dziedziczącej własne pola, ich wartości oraz metody wymagając od klasy dziedziczącej zapisanie tylko unikalnych właściwości. Wymienione elementy zostaną udostępnione pod warunkiem wskazania odpowiedniego modyfikatora dostępu do składowych klasy bazowej.

Definicja klasy dziedziczącej wygląda identycznie oprócz sygnatury, która po nazwie nowej klasy zyskuje słowo kluczowe `extends` a po nim występuje nazwa super klasy, z której dziedziczymy.

```
public class Potomna extends Baza {
        public String nieDrukuj(){
                String out = „attr1: ”+atrybut1;
                out+= ”\r\n”+„attr3: ”+atrybut3;
                return out;
        }
}
```

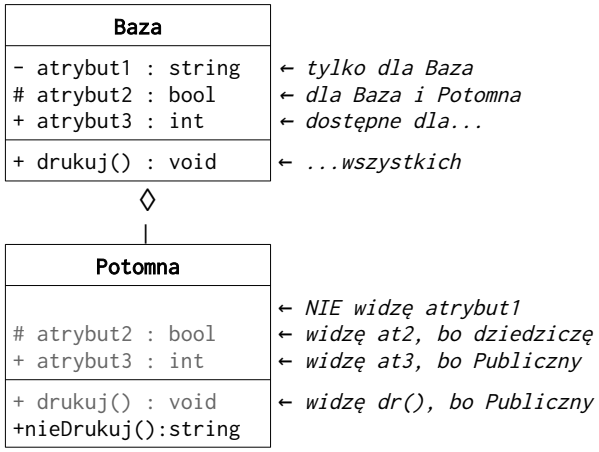


Informacje nt. dziedziczenia:

- klasa może być jednocześnie Bazową i Dziedziczącą
- klasa może dziedziczyć bezpośrednio tylko po jednej klasie naraz
- żeby zablokować dziedziczenie w klasie należy dodać do niej słowo `final`
- klasa może dziedziczyć po kilku klasach, ale „łańcuchowo”
- dziedzicząca klasa zyskuje właściwości i metody klasy bazowej w zależności od jej modyfikatorów dostępu

Modyfikator dostępu, który wprowadzono do mechanizmu dziedziczenia to `Protected` (chroniony), który na diagramie UML oznacza się znakiem # (płotek/hasztag). Jego działanie sprowadza się do modyfikatora `Private` (prywatny) oznaczający dostęp tylko dla klasy własnej, ale skoro klasa dziedziczy po nadrzędnej klasie, to tak jakby były to składowe własne.

```
public class Baza {
        String atrybut1 = „Pierwsza klasa”;
        boolean atrybut2 = false;
        int atrybut3 = 1;
        public void drukuj(){
                String out = „attr1: ”+atrybut1;
                out+= ”\r\n”+„attr3: ”+atrybut3;
                System.out.println(out);
        }
}
public class Potomna extends Baza {
        public String nieDrukuj(){
                String out = „attr1: ”+atrybut1;
                out+= ”\r\n”+„attr3: ”+atrybut3;
                return out;
        }
}
```



Zadania teoretyczne:

6. Podaj składowe klasy `Potomna` z diagramu UML
7. Podaj składowe klasy `Potomna` dziedziczącej z klasy `Baza`
8. Podaj składowe klasy `Baza` dziedziczącej z klasy `Object`

Inne mechanizmy przy stosowaniu dziedziczenia:

- konstruktory w Superklasach i implementowanie konstruktorów w klasach Pochodnych
- nadpisywanie metod Superklasy w klasie Podrzędnej
- dostęp do właściwości i metod Superklasy poprzez użycie słowa `super.*`

Polimorfizm

Jest efektem otrzymywanym dzięki mechanizmom dziedziczenia oraz interfejsom. Sam polimorfizm oznacza wielopostaciowość, czyli możliwość traktowania w taki sam sposób obiektów różnego podtypu otrzymanych z klasy bazowej określonego typu. Dane mogą być jednego, szczegółowego typu, natomiast zmienna referencji do obiektu danych może być typu ogólnego.

```
public class Demo {
	public static void main(){
		String s = new String(„To właśnie polimorfizm”);
		Object o = new String(„To właśnie polimorfizm”);
		System.out.println(s);  // jest równoznaczne z wywołaniem s.toString();
		System.out.println(o);  // jest --||-- o.toString();
	}
}
```

Takie rozwiązania można stosować w przypadku wielu wbudowanych klas, np. `ArrayList`, czy `LinkedList` oraz `List`, `HashMap`, czy `LinkedMap` oraz `Map`, swoich własnych klas, czy klas odpowiedzialnych za UI w Javie, czyli wbudowanych klas widgetów Swing, z których można dziedziczyć tworząc własne modyfikacje paneli, czy kontrolek.

Zadania praktyczne:

9. Zaimplementuj tablicę wraz z pętlą, których referencje będą typów ogólnych, a dane typów szczegółowych.