Hierarchia dziedziczenia w Javie

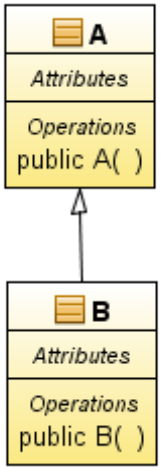
1. Wszystko jest obiektem. Każda klasa w Javie dziedziczy po klasie Object.
2. Hierarchia dziedziczenia jest drzewem z korzeniem będącym klasą Object. Kazda klasa moze dziedziczyc tylko po jednej klasie nadrzędnej.

Modyfikatory dostępu

1. **public** - dostęp do pola lub metody oznaczonej w ten sposób mają wszyscy.
2. **protected** - dostęp do pola lub metody oznaczonej w ten sposób mają tylko klasy bezpośrednio i pośrednio dziedziczące po klasie zawierającej te metody/pola. Modyfikator *protected* daje jednocześnie dostęp pakietowy (*package access*).
3. **private** - dostęp do pola lub metody oznaczonej w ten sposób ma tylko klasa je zawierajaca
4. ***package access*** - W Javie, jesli nie podamy żadnego modyfikatora dostępu przed metodą/polem/klasą hest ona widoczna dla wszystkich kals w obrębie jednego pakietu.
5. Jeśli metoda jest oznaczona jako *protected* albo *public* można ją nadpisać w klasie pochodnej.
6. Metod oznacoznych jako *private* nie da się nadpisać;
7. Mozna nadpisać metodę w klasie pochodnej zmieniając jej modyfikator zasięgu z mocniejszego na słabszy (np. z protected na public, ale nie odwrotnie).

Słowo kluczowe "extends"

Służy do opisania relacji dziedziczenia pomiędzy dwiema klasami. Klasa B rozszerza klasę A, co oznacza iż klasa B dziedziczy po klasie A.




```
class A{
  public A(){

     ...

  }
}

class B extends A{
  public B(){

     ...

  }
}
```

Konstruktory i dziedziczenie

Konstruktorów się nie dziedziczy. Wywoływanie konstruktora klasy bazowej musi być pierwszą operacją wewnątrz konstruktora klasy pochodnej. Konstruktor klasy bazowej wywołuje się za pomocą słowa kluczowego **super**:

```
public class A{
  public A(int n, double nn){
    ...
  }
}

...

public class B extends A{
  public B(int Bn, double Bnn, float Bnnn){
    super(Bn,Bnn);
        ...
  }
}
...
```

Rzutowanie

1. W Javie każde rzutowanie w dół (*downcasting*) należy wykonywać jawnie:

```
...
Object obj = new String("Ala ma kota");
String str = (String)obj;
   ...
```

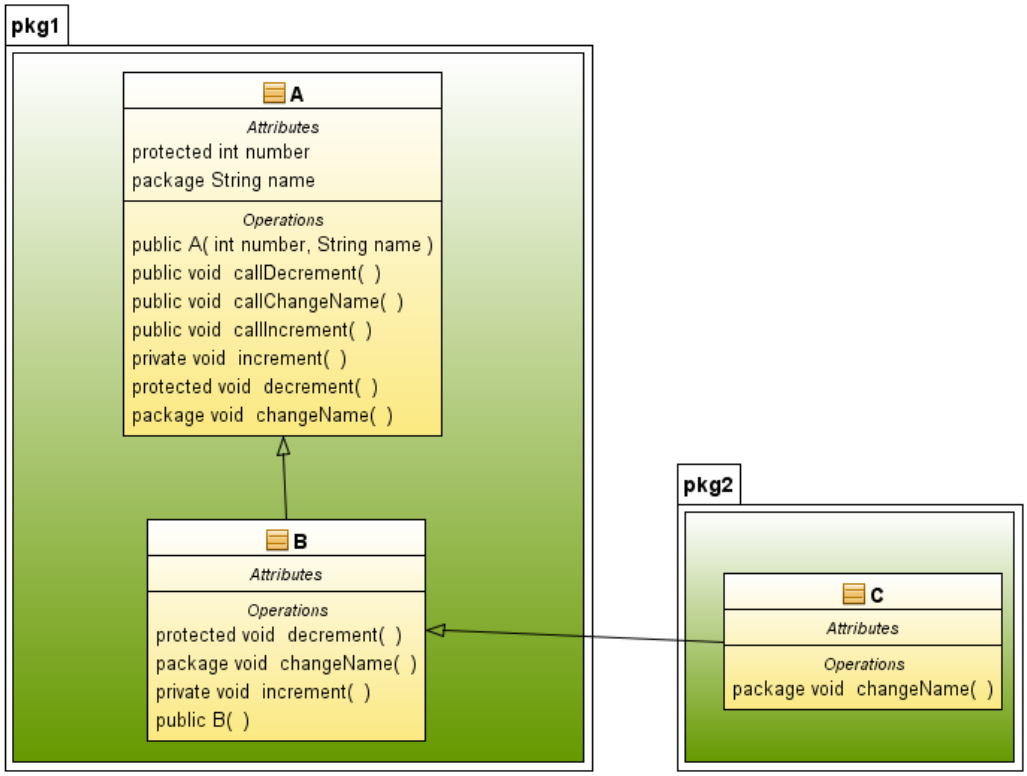
5. Rzutowanie w górę (*upcasting*) może być wykonywane niejawnie:

```
...
String nazwa = new String("Ala ma kota");
Object obj = nazwa;
   ...
```

9. Do sprawdzania typów służy słowo kluczowe **instanceof**:

```
 ...
 Object o = new String("Ala ma kota");
 if(o instanceof String)
   String s = (String)o;
...
```

1. Zaimplementuj następującą relację pomiędzy klasami i przetestuj metody *call** dla wszystkich 3 obiektów. Metody w klasie B powinny różnić się do tych z klasy A (np. metoda increment() z klasy A zwiększa zmienną number o 1, natomiast ta sama metoda w klasie B o 2) [1.0]:



2. Napisz klasę **Kwadrat**, zawierająca pole typu **double** określające długość boku *a*. Klasa powinna mieć jedynie konstruktor parametrowy przyjmujący wartość *a*. **Klasa nie może mieć konstruktora bezparametrowego**.
    - Metody klasy do zaimplementowania:
        - **getA, setA**;
        - **area** - zwracająca pole powierzchni tego kwadratu (wartość double)
        - **isBigger** - pobierająca jako parametr **Kwadrat** i zwracająca wartość typu bool. True jeśli figura z parametru ma większą powierzchnię, false w przeciwnym wypadku
3. Napisz klasę **Prostokąt**, dziedziczącą po **Kwadrat** i przechowującą dodatkowo pole długości boku *b*. Klasa powinna posiadać konstruktor parametrowy podobnie jak klasa **Kwadrat**
    - Metody klasy do zaimplementowania:
        - **setB, getB**
        - **area** zwracająca pole powierzchni zadanego prostokąta (wartość double).
        - **isBigger** - pobierająca jako parametr **Prostokąt** i zwracająca wartość typu bool. True jeśli figura z parametru ma większą powierzchnię, false w przeciwnym wypadku
4. [2] Napisz klasę **Test**, zawierającą metodę **main**, oraz pole **LinkedList<Prostokąt> figury**. Po uruchomieniu programu powinno pojawić się proste tekstowe menu umożliwiające wykonywanie wymienionych w nim operacji:
    - Wczytaj prostokąt
    - Wyświetl wszystkie prostokąty
    - Oblicz sumę pól wszystkich prostokątów
    - Zakończ