Wyznaczanie klas dziedziczonych w Javie

Aleksander Nałęczyński

Klasy zagnieżdżone

- Klasa zagnieżdżona to klasa, która jest składową (member) innej klasy.
- Klasa A dziedzicząca z klasy C otrzymuje własną kopię pól i metod z C
- Klasy A i B zagnieżdżone w klasie C korzystają wspólnie z tych samych pól i metod z klasy C

Klasy zagnieżdżone

- Kiedy używamy klas zagnieżdżonych?

  Gdy klasa ma sens tylko w kontekście klasy otaczającej, np. klasa kursora w kontekście klasy określającej komponent tekstowy.

- W API Javovym często występują klasy zagnieżdżone

  Component.AccessibleAWTComponent.AccessibleAWTFocusHandler

Statyczne klasy zagnieżdżone

- Klasy zagnieżdżone mogą być oznaczone jako statyczne (nested static class)
- Możemy dowolnie tworzyć instancje statycznych klas zagnieżdżonych

Klasy wewnętrzne

- Niestatyczne klasy zagnieżdżone nazywamy klasami wewnętrznymi (inner class)
- Instancja klasy wewnętrznej jest ściśle związana z instancją klasy otaczającej. Klasy wewnętrzne nie mogą mieć własnych składowych typu static, ale mogą je odziedziczyć

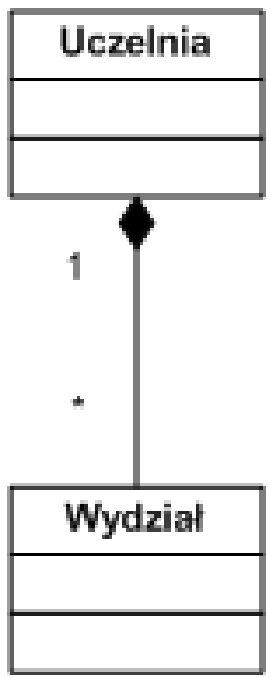
Klasy wewnętrzne

- Instancje klasy wewnętrznej można tworzyć tylko, jeśli istnieje instancja klasy otaczającej – jeśli chcemy tworzyć je poza kodem klasy otaczającej, używamy składni:

  <nazwa instancji>.new <konstruktor>;

Jak to wygląda w UMLu?


```
class A {
  class B {
    ...
  }
}
```



Klasy zagnieżdżone

- Można również deklarować klasy zagnieżdżone wewnątrz treści metod oraz jako klasy anonimowe

Przykład – klasa wewnętrzna

```
class Counter {
   int i;
   class Listener implements ActionListener {
       public void actionPerformed(ActionEvent e) {
             i++;
       }
   void listenTo(button B) {
       b.addActionListener(new Listener());
   }
}
```

Przykład – anonimowa klasa wewnętrzna

```
class Counter {
   int i;
   void listenTo(button B) {
       b.addActionListener(new ActionListener(){
       public void actionPerformed(ActionEvent e) {
              i++;
       });
   }
}
```

private, protected, public

- Klasy zagnieżdżone mają dostęp do wszystkich pól klasy otaczającej – również tych oznaczonych jako private
- UWAGA – w pierwszej wersji prezentacji znalazło się tutaj nieprawdziwe zdanie o podklasach klas zagnieżdżonych – przepraszam za błąd

Problem

```
class A {}
class B {
  class A {}
  class C extends A {}
  class D {
     class A {}
  }
  class E extends D.A {}
}
```

Po której klasie dziedziczy klasa C? A klasa E?

Rozwiązanie

- On correctness & completeness of an algorithm determining inherited classes and on uniqueness of solutions - Hans Langmaack, Andrzej Salwicki, Marek Warpechowski

Struktura klas

- Classes – zbiór klas zadeklarowanych w programie plus Root i Object
- Id – zbiór identyfikatorów klas
- Paths – zbiór ciągów identyfikatorów rozdzielonych kropkami

Oznaczenia

- decl: Classes – {Root} -> Classes

  decl(K) oznacza klasę zawierającą K
- name: Classes – {Root} -> Id

  name(K) oznacza nazwę klasy K, name(Root) jest niezdefiniowane, name(Object) = Object
- ext: Classes – {Root, Object} -> Paths

  ext(K) oznacza ścieżkę rozszerzenia wpisaną w tekście programu

Właściwości struktury klas

- decl(Object) = Root
- para <Classes, decl> jest drzewem, którego korzeniem jest Root
- Jeśli K ≠ M i decl(K) = decl(M),

  to name(K) ≠ name(M)

Definicja funkcji bind

- Funkcja bind określa, do której klasy się odwołujemy używając identyfikatora klasy w tekście programu
- bind(ε in K) = Object
- bind(C in K) = $(inh^i decl^j(K)).C$

  gdzie para (j,i) jest najmniejszą parą w porządku leksykograficznym taką, że klasa $(inh^i decl^j(K)).C$ jest zdefiniowana

Definicja funkcji bind

- bind(X.C in K) = (inh^{i}(bind(X in K))).C

  gdzie i jest najmniejszą liczbą taką, że klasa (inh^{i}(bind(X in K))).C jest zdefiniowana

Definicja relacji dep

- Definicja intuicyjna - $<K,C> \in$ dep wtw gdy do obliczenia inh(K) potrzebujemy obliczonego inh(C)
- dep = {<K, bind(ext(K)|^i in decl(K)))> :

  $0 < i \leq$ length(ext(K))}

  gdzie ext(K)|^i oznacza prefiks ext(K) długości i

Funkcja dziedziczenia

- Szukamy funkcji

  inh: Classes – {Root, Object} -> Classes
- Dla każdej klasy K $\notin$ {Root, Object}

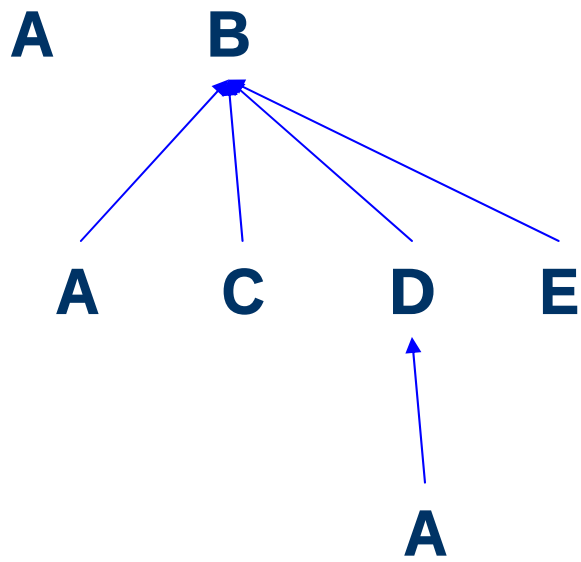
  inh(K) = bind(ext(K) in decl(K))

  to znaczy, że określamy klasę, po której dziedziczy K na podstawie tego, jak jest wiązana ścieżka ext(K) w klasie otaczającej.
- Indukowana relacja dep nie może mieć cykli

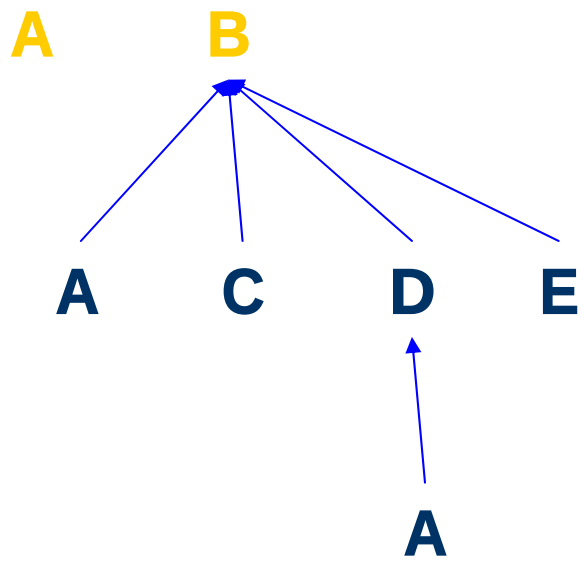
Algorytm

```
Visited := {Root; Object};
inh := Ø;
while Visited ≠ Classes
do
   Candidates := {K : decl(K) ∈ Visited ∧ K ∉ Visited}
   if (∃K ∈ Candidates) bind(ext(K) in decl(K)) ∈ Visited
   then
        let K be a Candidate found in the above test;
        M := bind(ext(K) in decl(K));
        inh := inh ∪ {< K,M >};
        Visited := Visited ∪ {K}
   else
        Error
   endif
endwhile
```

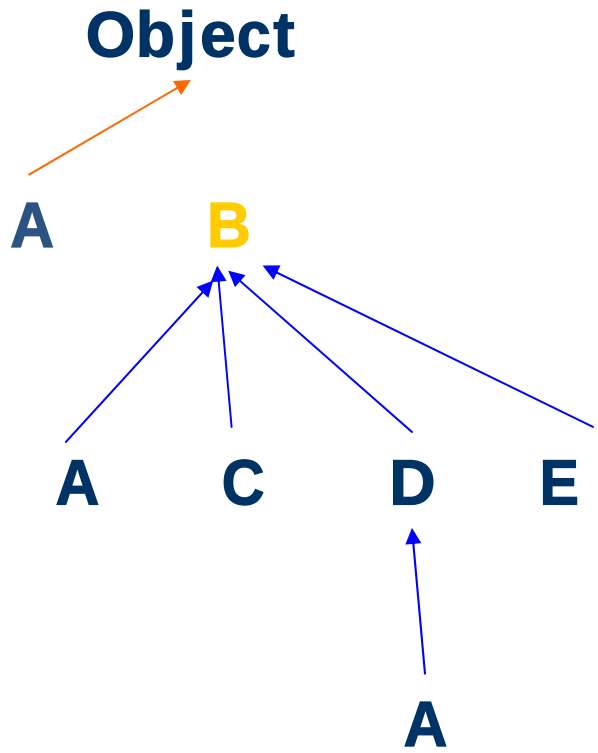
```
class A {}
class B {
        class A {}
        class C extends A {}
        class D {
                class A {}
        }
        class E extends D.A {}
}
```



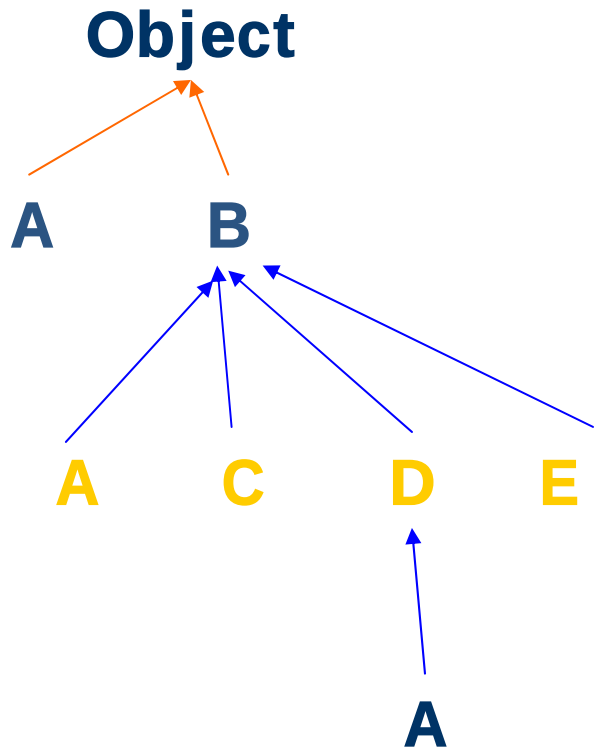
```
class A {}
class B {
        class A {}
        class C extends A {}
        class D {
                class A {}
        }
        class E extends D.A {}
}
```



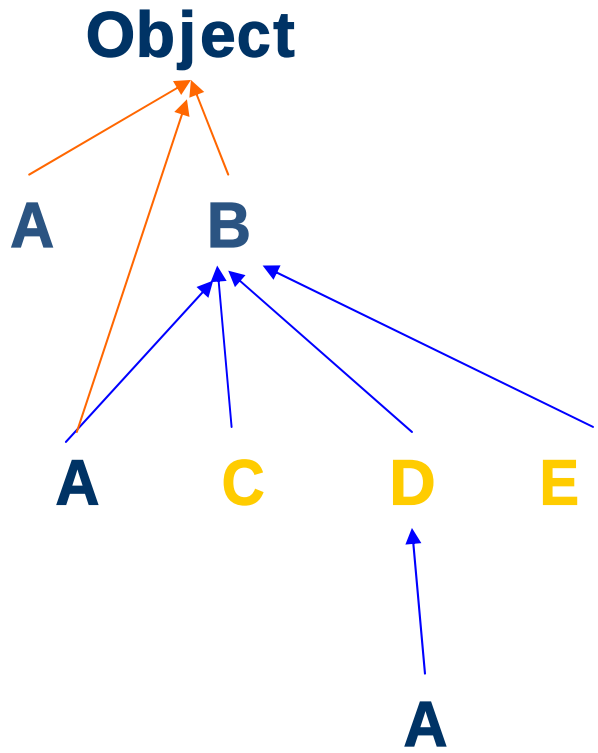
```
class A {}
class B {
        class A {}
        class C extends A {}
        class D {
                class A {}
        }
        class E extends D.A {}
}
```



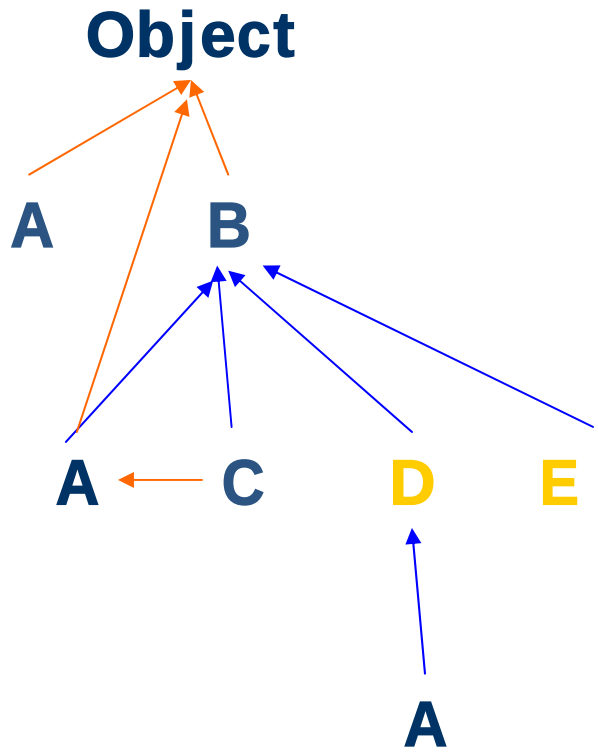
```
class A {}
class B {
        class A {}
        class C extends A {}
        class D {
                class A {}
        }
        class E extends D.A {}
}
```



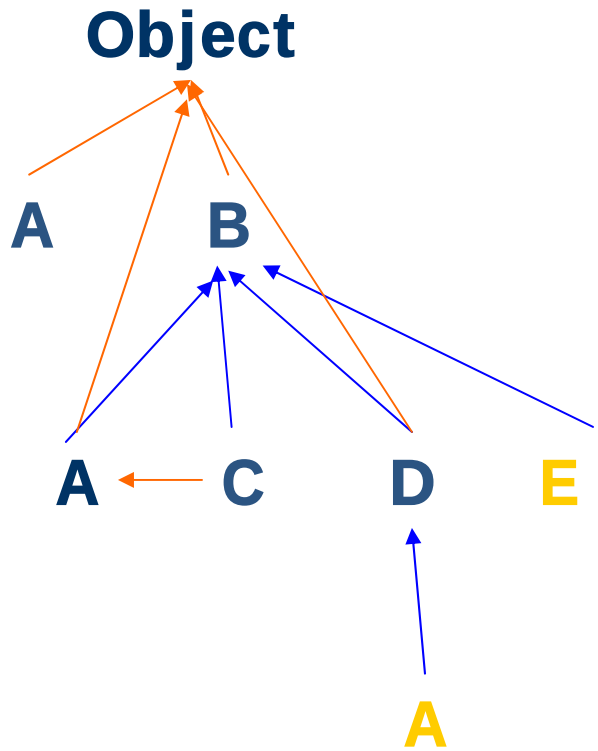
```
class A {}
class B {
        class A {}
        class C extends A {}
        class D {
                class A {}
        }
        class E extends D.A {}
}
```



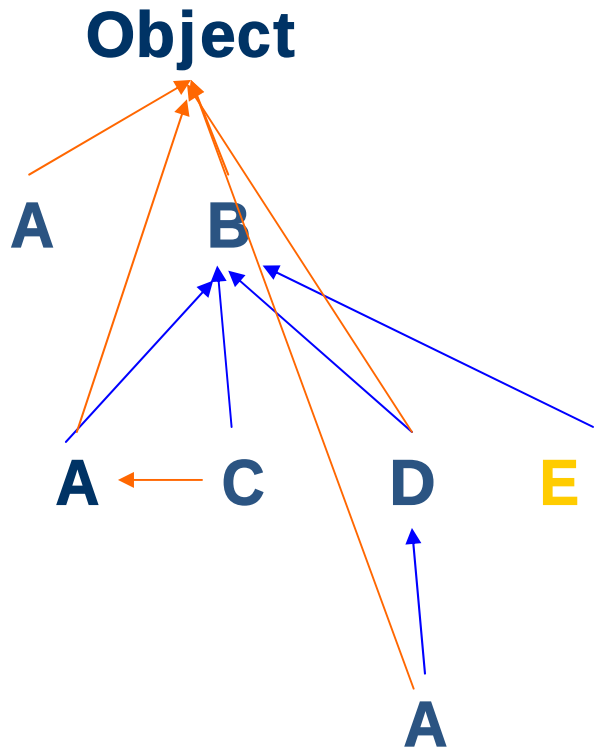
```
class A {}
class B {
        class A {}
        class C extends A {}
        class D {
                class A {}
        }
        class E extends D.A {}
}
```



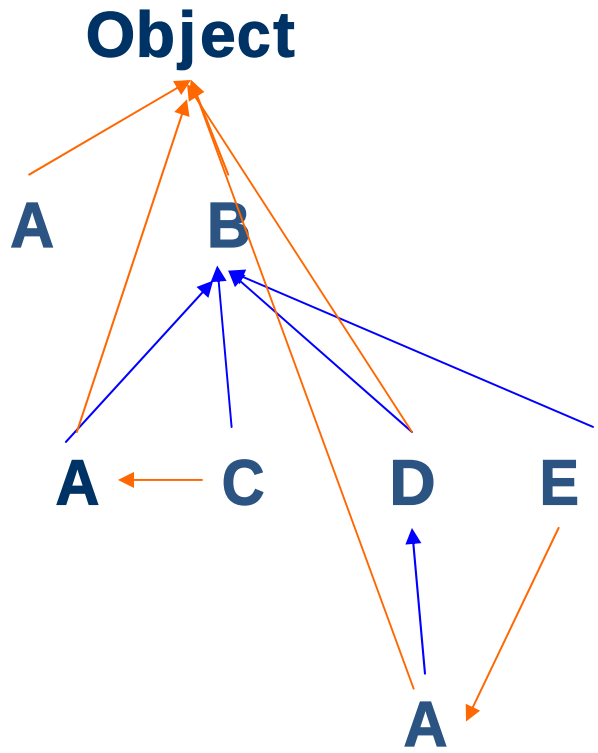
```
class A {}
class B {
        class A {}
        class C extends A {}
        class D {
                class A {}
        }
        class E extends D.A {}
}
```



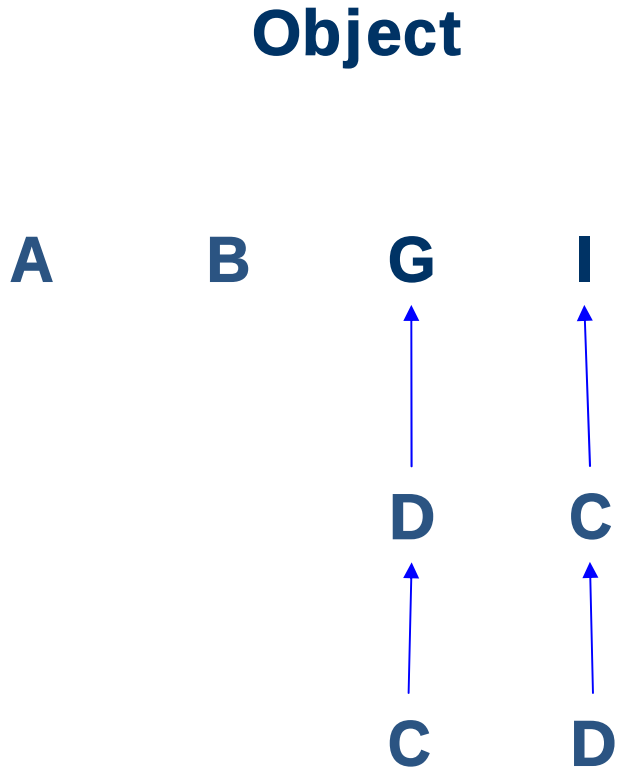
```
class A {}
class B {
        class A {}
        class C extends A {}
        class D {
                class A {}
        }
        class E extends D.A {}
}
```



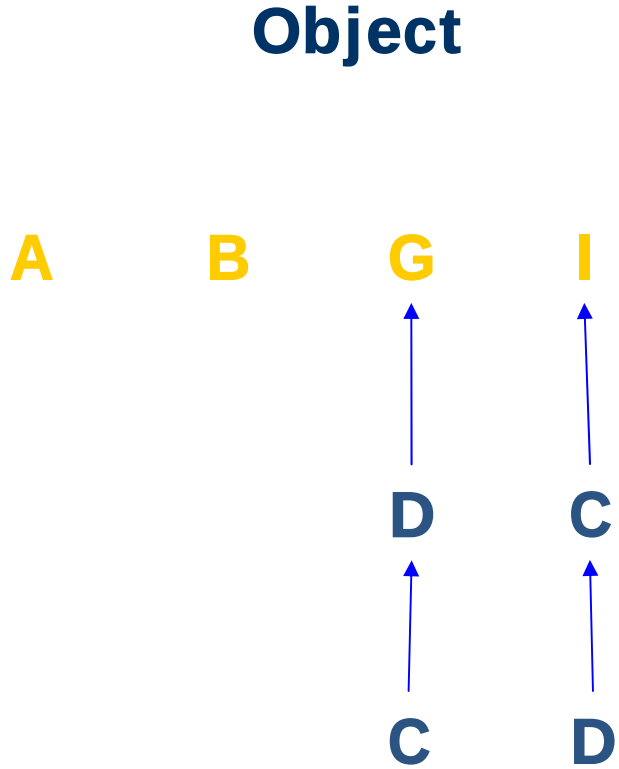
```
class A {}
class B {
        class A {}
        class C extends A {}
        class D {
                class A {}
        }
        class E extends D.A {}
}
```



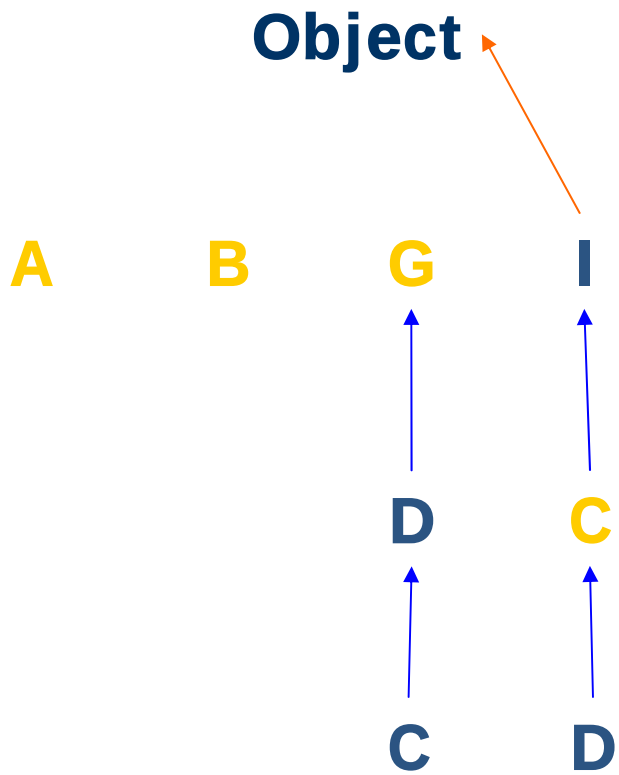
```
class A extends B.C {}
class B extends A.D {}
class G {
  class D {
    class C extends G {}
    }
  }
class I {
  class C {
    class D extends I{}
  }
}
```



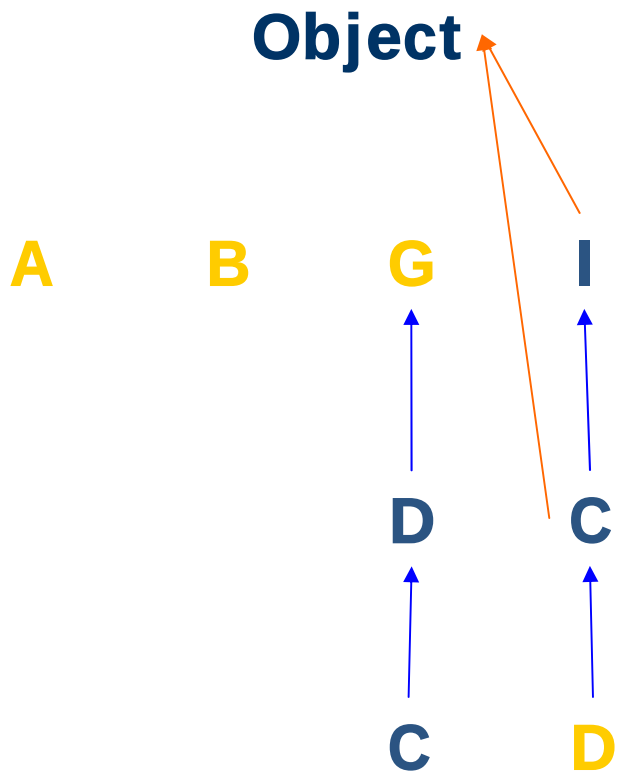
```
class A extends B.C {}
class B extends A.D {}
class G {
  class D {
    class C extends G {}
    }
  }
class I {
  class C {
    class D extends I{}
  }
}
```



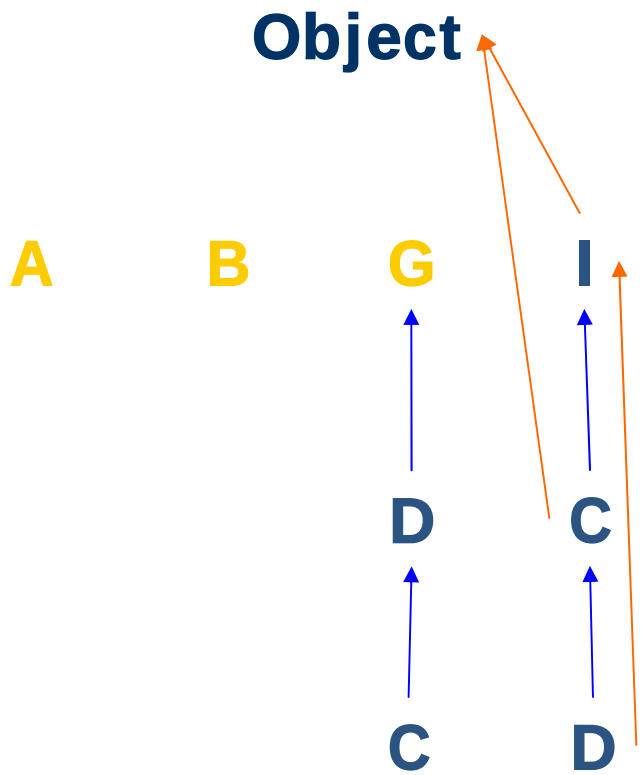
```
class A extends B.C {}
class B extends A.D {}
class G {
  class D {
    class C extends G {}
    }
  }
class I {
  class C {
    class D extends I{}
  }
}
```



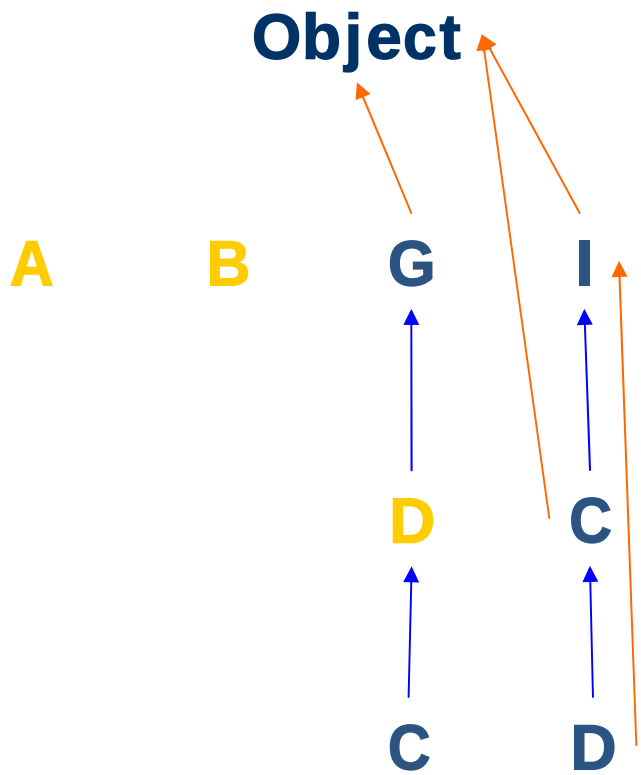
```
class A extends B.C {}
class B extends A.D {}
class G {
  class D {
    class C extends G {}
    }
  }
class I {
  class C {
    class D extends I{}
  }
}
```



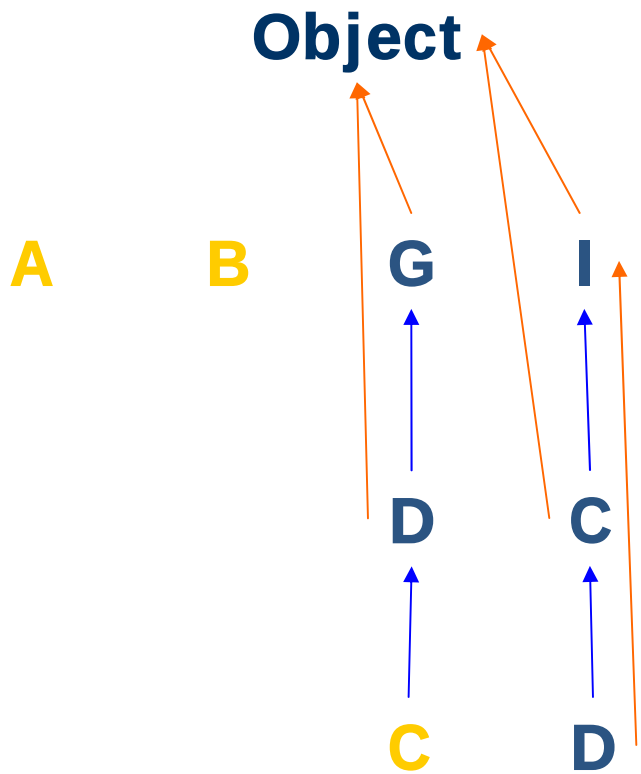
```
class A extends B.C {}
class B extends A.D {}
class G {
  class D {
    class C extends G {}
    }
  }
class I {
  class C {
    class D extends I{}
  }
}
```



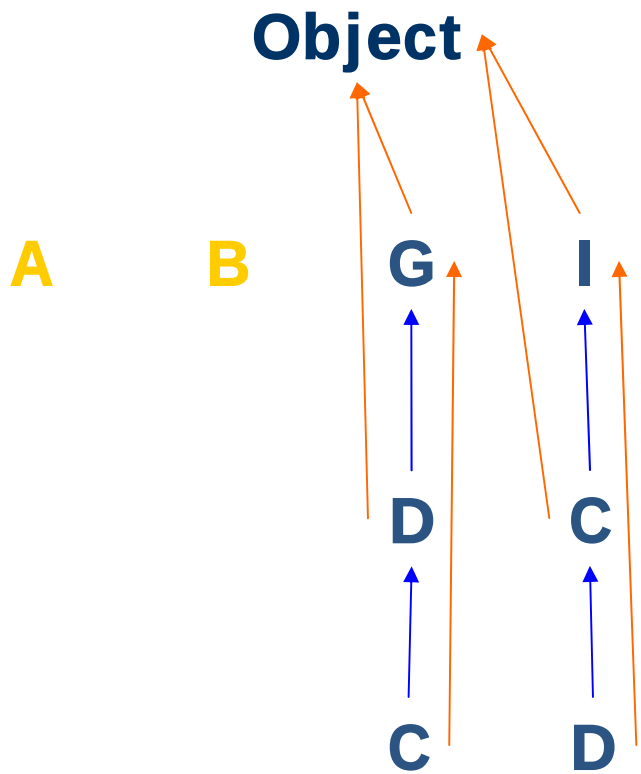
```
class A extends B.C {}
class B extends A.D {}
class G {
  class D {
    class C extends G {}
    }
  }
class I {
  class C {
    class D extends I{}
  }
}
```



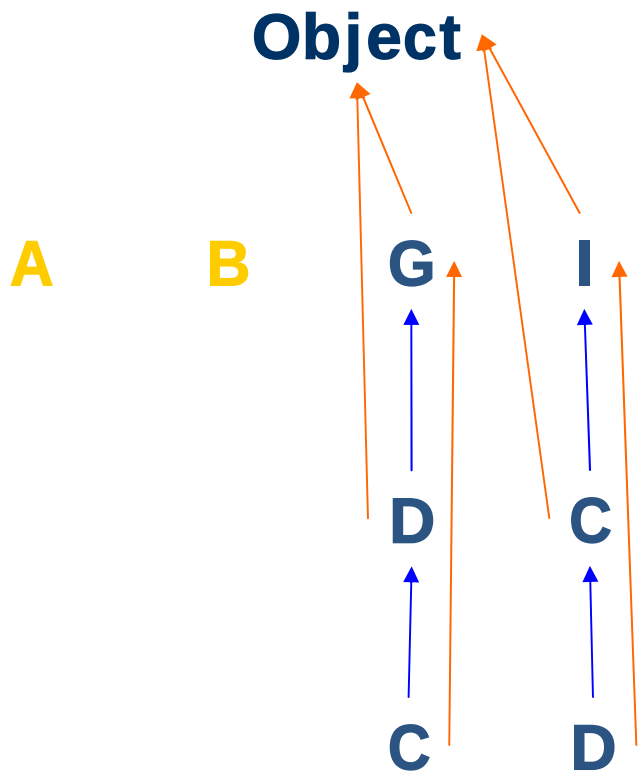
```
class A extends B.C {}
class B extends A.D {}
class G {
  class D {
    class C extends G {}
    }
  }
class I {
  class C {
    class D extends I{}
  }
}
```



```
class A extends B.C {}
class B extends A.D {}
class G {
  class D {
    class C extends G {}
    }
  }
class I {
  class C {
    class D extends I{}
  }
}
```

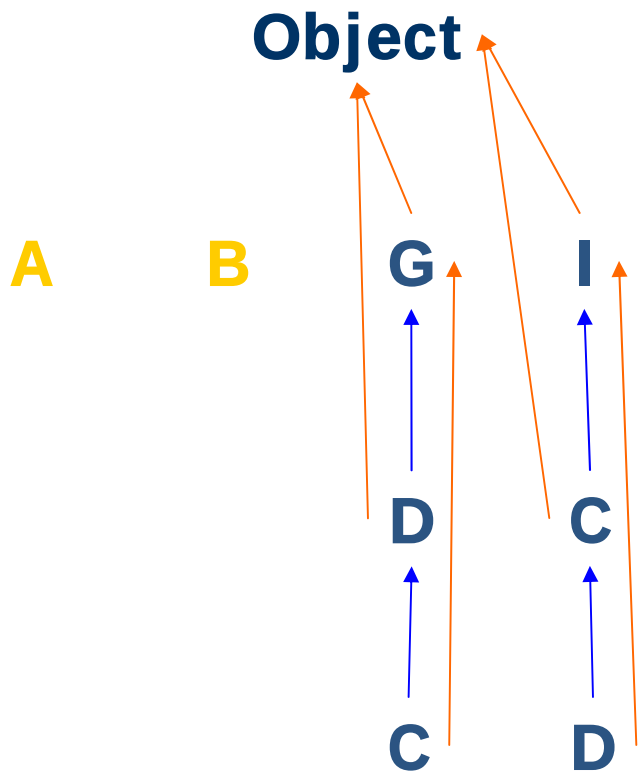


```
class A extends B.C {}
class B extends A.D {}
class G {
  class D {
    class C extends G {}
    }
  }
class I {
  class C {
    class D extends I{}
  }
}
```



Error!

```
class A extends B.C {}
class B extends A.D {}
class G {
  class D {
    class C extends G {}
    }
  }
class I {
  class C {
    class D extends I{}
  }
}
```



W relacji dep jest cykl:
<A,B> ∈ dep i <B,A> ∈ dep
Istnieją dwa możliwe rozwiązania:
1. inh(A) = GDC, inh(B) = G$D
2. inh(A) = I$C, inh(B) = I$C$D

Error!