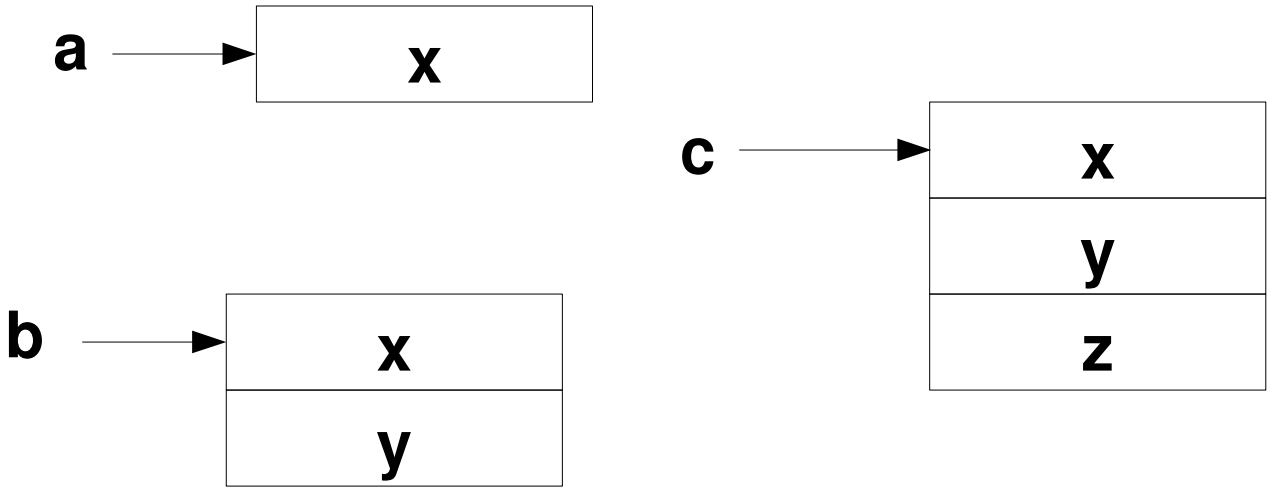
Kompilacja języków obiektowych

- Zagadnienia:
  - Hierarchia typów (dziedziczenie)
  - Polimorfizm rozumiany jako możliwość podstawienia do zmiennej wartości innego typu
  - Dynamiczne określanie implementacji wywoływanej metody (late/dynamic binding)
  - Prototypy (klasy) i ich instancje (obiekty)
  - Funkcje określania aktualnego typu oraz jego rozszerzania i zawężania
  - Zagnieżdżanie typów w odniesieniu do dziedziczenia
  - Dodatkowe narzuty czasu wykonania
  - Optymalizacja

Dziedziczenie pól

- Pojedyncze (to samo przesunięcie względem początku rekordu)

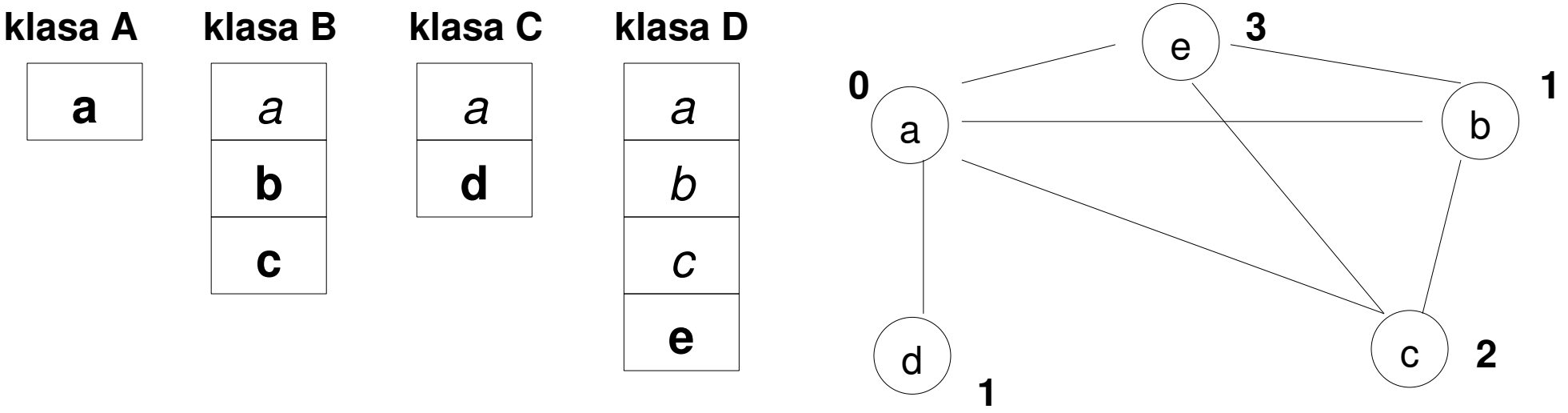
```
class A   { int x; }
class B:A { int y; }
class C:B { int z; }

a = new A();
b = new B();
c = new C();
```



- Wielokrotne

```
class A { int a; }
class B : A { int b; int c; }
class C : A { int d; }
class D : A,B { int e; }
```

Dziedziczenie pól c.d.



- Quiz
  - Co się stanie, jeżeli w powyższym przykładzie klasa **B** nie będzie dziedziczyła z **A** ?
  - Jak będzie wyglądała reprezentacja poniższej hierarchii klas ?

```
class A { int a; }  class B extends A { double a; }
```

Dziedziczenie (i uruchamianie) metod

- Realizacja wymaga opracowania deskryptora typu zawierającego wskazania na faktyczne implementacje metod i jest to bezpośrednio zwązane z dynamicznym określaniem implementacji (oraz popularnym „polimorfizmem OO”)

```
class A { int x;
          void printA();
        }
class B:A { int y;
          void printB();
        }
class C:B { int z;
          void printA();
          void printC();
        }
```

```
int cX() {...}
int cY() {...}
int count()
       { return cX() + cY();}
```

```
int cX(){ ... }
int cY(){ ... }
```

```
int cX(){ ... }
int cY(){ ... }
```

```
C zmienna = new C();
zmienna.count();
```

Dziedziczenie (i uruchamianie) metod c.d.

- Przykładowa struktura deskryptora typu
- Metody statyczne klas
- Wielodziedziczenie
  - nazwy metod w globalnym grafie
  - konflikty nazw w klasach nadrzędnych (dotyczy także pól)

Polimorfizm

- Założenia językowe dotyczące instrukcji podstawienia
- Sprawdzanie typów podczas kompilacji

```
class A {}   class B:A {}   class C:A {}
```

```
A x = new B(); C y = x;
```

Sprawdzanie typu w czasie wykonywania programu

- deskryptor klasy posiada wskazanie na deskryptor klasy nadrzędnej - można przejrzeć ten łańcuch (kosztowne)
- inaczej (Paul F.Dietz)
  - w trakcie kompilacji:
    - budujemy drzewo dziedziczenia
    - przeglądamy w porządku *preorder* i numerujemy węzły
    - przeglądamy w porządku *postorder* i numerujemy węzły
    - zapisujemy numery w deskryptorze typu
  - w trakcie wykonania:
    - Niech **O** oznacza deskryptor typu sprawdzanego obiektu a **T** deskryptor typu jaki sprawdzamy. Nasz obiekt jest klasy **T** jeżeli

      T.preorder <= O.preorder AND T.postorder >= O.postorder

Sprawdzanie typu c.d.

```
class A { }  class B:A {}   class C:A {}
class D:C {} class E:B {}
class F:C {} class G:C {}
```

? G instanceof A ? *? G instanceof B ?*

- Zawężanie typu (narrow)
- Rozszerzanie typu (qua?)

Różne

- Dostępność pól i metod (statyczna i dynamiczna)
- Optymalizacja języków obiektowych