Subtipagem

- O conjunto de valores de um tipo pode ser um subconjunto do conjunto de valores de outro tipo
- Podemos querer expressar isso no sistema de tipos através de uma *relação de subtipagem* ≤
- Em uma linguagem OO essa relação é declarada pelo programador; em Java ela é dada pelas cláusulas *extends* e *implements*
- A relação de subtipagem é *simétrica* (t ≤ t) e *transitiva* (r ≤ s e s ≤ t implica r ≤ t)
- Podemos usar a relação de subtipagem explicitamente nas regras, ou podemos introduzir uma *regra de subsunção*

Subtipagem explícita vs. subsunção

- Subtipagem explícita:

$$\frac{T(i2) = t_1 \quad T \vdash e : t_2 \quad t_2 \leq t_1}{T \vdash i2 := e}$$

- Subsunção:

$$\frac{T(i2) = t \quad T \vdash e : t}{T \vdash i2 := e}$$

subsunção

$$\frac{T \vdash e : t_1 \quad t_1 \leq t_2}{T \vdash e : t_2}$$

- Usar subsunção deixa o sistema mais sintético, usar a subtipagem explícita deixa ele mais fácil de implementar

Classes

- Para mostrar como funciona a subtipagem, podemos adicionar classes com herança simples a Tiny*:*

```
s : [classes ';'] [procs ';'] cmds
  ;

classes : classes ';' classe
        | classe
        ;

classe : CLASS ID [VAR decls] END
       | CLASS ID [VAR decls ';'] procs END
       | CLASS ID ':' ID [VAR decls] END
       | CLASS ID ':' ID [VAR decls ';'] procs END
       ;
```

```
cmd : <outros>
    | rexp '.' ID '(' [exps] ')'
    ;

exp : <outras>
    | rexp '.' ID '(' [exps] ')'
    | NEW ID '(' [exps] ')'
    | NIL
    ;

rexp : <outras>
     | rexp '.' ID '(' [exps] ')'
     ;
```

Escopos com classes

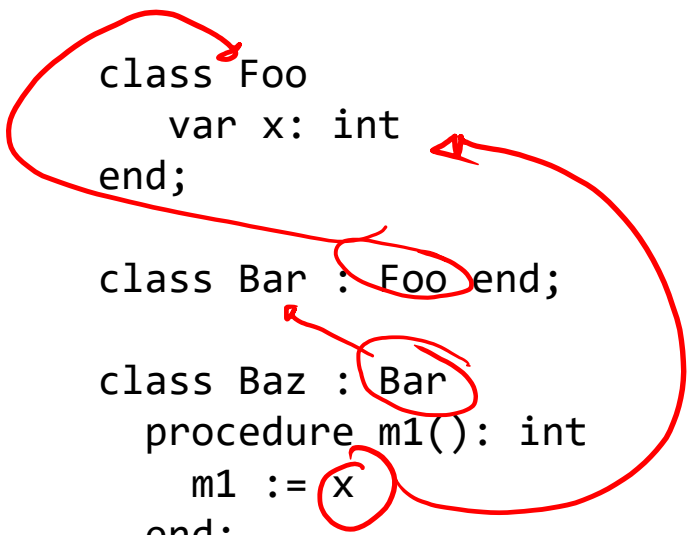
- Tiny com classes tem vários tipos de nomes:
  - Variáveis
  - Campos
  - Métodos e procedimentos
  - Classes
- Cada um desses tem suas regras de escopo; alguns compartilham espaços de nomes, outros têm espaços de nomes separados

Escopo de classes e campos

- O escopo das classes é *global*, e classes estão em seus próprio espaço de nomes
- Variáveis e campos compartilham o mesmo espaço de nomes, mas as regras de escopo são diferentes
- Um campo de uma classe é visível em todos os métodos daquela classe *e de todas as suas subclasses, diretas ou indiretas*
- Variáveis locais ocultam campos, mas campos não podem ser redefinidos nas subclasses

Exemplo – escopo de variáveis e campos

- O escopo do campo *x* inclui todas as subclasses de *Foo*



```
class Foo
   var x: int
end;

class Bar : Foo end;

class Baz : Bar
  procedure m1(): int
    m1 := x
  end;

  procedure m2(x: bool): bool
    m2 := x
  end
end
```

Métodos

- Como classes, métodos estão em seu próprio espaço de nomes
- Mas, como campos, o escopo de um método é a classe em que está definido e suas subclasses
- Um método não pode ser definido duas vezes em uma classe, mas pode ser redefinido em uma subclasse *contanto que a assinatura seja a mesma*
- A *assinatura* do método é o seu tipo de retorno, seu nome e os tipos dos seus parâmetros, na ordem na qual eles aparecem
- Como classes, métodos e campos podem ser referenciados antes de sua declaração, a verificação de escopo desses nomes também ocorre em duas passadas

Exemplo - métodos

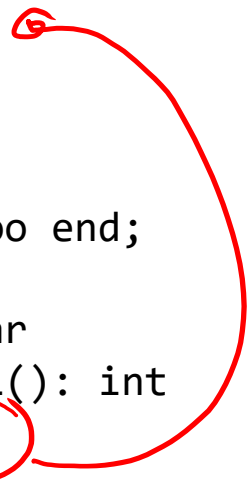
- O método *m2* é visível em Baz, que redefine *m1*

```
class Foo
  procedure m1(): int
    m1 := 0
  end;

  procedure m2(): int
    m2 := 1
  end
end;

class Bar : Foo end;

class Baz : Bar
  procedure m1(): int
    m1 := m2()
  end
end
```



Subtipagem de classes

- Determinar se uma classe é subtipo de outra é fácil com um algoritmo recursivo
- Uma classe é subtipo dela mesma
- Uma classe é subtipo da sua superclasse direta
- Se não for nenhum dos casos base acima, uma classe é subtipo de outra classe se a sua superclasse direta for subtipo dessa outra classe
- Ciclos na hierarquia de classes são um erro

Redefinição de métodos

- Quando redefinimos um método em uma subclasse, poderíamos permitir que os tipos dos parâmetros e o tipo de retorno mudem
- O tipo dos parâmetros na subclasse poderiam ser *supertipos* dos tipos na superclasse, e o tipo de retorno poderia ser um *subtipo*
- Por isso dizemos que métodos são *contravariantes* no tipo dos parâmetros e *covariantes* no tipo de retorno
- É fácil ver que parâmetros covariantes seriam inconsistentes, assim como um tipo de retorno contravariante