190600213201 5
Programlama Dilleri Temelleri

BAİBÜ Bilgisayar Müh.

Ders 5

Dr. Araş. Gör. İsmail Hakkı Parlak

Kaynaklar: Watt, David A., Programming Language Design Concepts, Wiley

Bazı şekiller kaynak kitaptan kopyalanmıştır.

5. Tip Sistemleri (Type Systems)

- 5.1. **Katma polimorfizmi** (Inclusion Polymorphism): bir türün, bu türden işlemleri devralan alt türlere sahip olabileceği bir tür sistemidir. Katma polimorfizmi, bir sınıfın o sınıftan yöntemleri miras alan alt sınıflara sahip olabileceği, oop dillerdeki anahtar bir kavramdır.
- 5.1.1 Tipler ve Alt tipler (Types & Subtypes): T tipinin bazı işlemlerle donatılmış bir değerler kümesi olduğunu hatırlayın. T'nin bir alt türü, T ile aynı işlemlerle donatılmış, T değerlerinin bir alt kümesidir. Alt türün her değeri aynı zamanda T türünde bir değerdir ve bu nedenle, T türünde bir değerin beklendiği bir bağlamda kullanılabilir.

5.1.1 Tipler ve Alt tipler

- C dilinde:
  - long -> alt tipler: int, char
  - double -> alt tipler: float
- Programlama dili alt türleri destekliyorsa, bir değerin (value) alt türünü benzersiz olarak belirtemeyiz.
  - Örn: 3.15 hem float hem double'dır.
- S'in T'nin bir alt tipi olabilmesi için, S'nin bütün değerlerinin T'nin de bir değeri olması gereklidir. Yani $S \subseteq T$. T tipinde bir değerin geçtiği ifadede S tipinde bir değer güvenle kullanılabilir.

5.1.1 Tipler ve Alt tipler

- S ⊆ T ise S, T'de tanımlı bütün operasyonları miras alır (inherit).

```
// C dili
long f(long a, long b) {
  return a + b;
}

int main(void) {
  int x = 3;
  char y = 5;
  printf("%ld", f(x, y)); // 8
  return 0;
}
```

```
-- Haskell
f :: Num a => a -> a -> a

f 3 5
f 5.7 3.8
```

5.1.2 Sınıflar (Classes) ve Alt sınıflar (Subclasses)

- Bir C sınıfı ve bir S alt sınıfı düşünün. C sınıfının her nesnesi bir veya daha fazla değişken bileşene (variable components) sahiptir ve değişken bileşenlere erişen yöntemlerle (methods) donatılmıştır. S sınıfının her nesnesi, C sınıfı nesnelerinin tüm değişken bileşenlerini miras alır (inherits) ve ek değişken bileşenlere sahip olabilir.
- S sınıfının her nesnesi, potansiyel olarak C sınıfı nesnelerinin tüm yöntemlerini devralır ve ek değişken bileşenlerine erişen ek yöntemlerle donatılabilir.
- Alt sınıflar, alt türlere tam olarak benzemez. S alt sınıfının nesneleri, C sınıfının nesnelerinin beklendiği her yerde kullanılabilir, S sınıfının ek bileşenleri olabilir, bu nedenle S alt sınıfının nesneleri kümesi, C nesnelerinin bir alt kümesi değildir.

5.1.2 Sınıflar (Classes) ve Alt sınıflar (Subclasses)

```
// Java
class Point {
  protected int x, y;

  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public String toString() {
    return "Point: " + x + ", " + y;
  }
}
```

```
class Circle extends Point {
  private int r;

  public Circle(int x, int y, int r) {
    super(x, y);
    this.r = r;
  }

  public String toString() {
    return super.toString() +
    " radius: " + r;
  }
}
```

5.2 Parametrik Çok biçimlilik (Polymorphisim)

- Bir listenin uzunluğunu hesaplamak için bir fonksiyon prosedürü düşünün. Sıradan bir prosedür char listelerinde çalışabilir, ancak diğer türlerin listelerinde de çalışamaz. Öte yandan, polimorfik bir prosedür, işlevi uygulamak için gereken algoritmanın (liste öğelerini saymak) liste öğelerinin türüne bağlı olmadığı gerçeğinden yararlanarak herhangi bir tür liste üzerinde çalışabilir.
- Tek biçimli (monomorphic) bir prosedür yalnızca sabit tipteki argümanlar üzerinde çalışabilir. Bir çok biçimli (polymorphic) bir prosedür, bütün bir tür ailesinin argümanları üzerinde aynı şekilde çalışabilir.
- Parametrik polimorfizm, polimorfik prosedürler yazabildiğimiz bir tip sistemdir (type system).

5.2.1 Polimorfik Prosedürler

```
ikincisi :: (Int, Int) -> Int -- monomorfik
ikincisi (x, y) = y
a = ikincisi ('c', 7) -- hata

birincisi :: (a, b) -> a -- polimorfik
birincisi (x, y) = x
b = birincisi (7, "hello") -- hata yok
```

5.2.2 Parametrize (Parametrized) Tipler

- Parametrize tip, diğer tipleri parametre olarak alan bir tiptir.
- Örneğin, C'deki dizi türlerini düşünün. τ [ ]'yi parametrize bir tip olarak düşünebiliriz. τ yerine gerçek bir tip koyarak τ [ ] ifadesini sıradan bir türe (char[ ] veya float[ ] veya float[ ][ ] gibi) dönüştürebiliriz.

```
type Ikili t = (t, t)
x :: Ikili Int
x = (3, 5)
```

5.2.3 Tip Çıkarımı

- Statik tipli programlama dillerinin çoğu, programlarımızda bildirilen her varlığın türünü açıkça beyan etmemizde ısrar eder.
- Tür çıkarımı, açıkça belirtilmediği durumlarda, bildirilen bir varlığın türünün çıkarıldığı bir süreçtir.

```
cift n = (n `mod` 2) == 0
```

çıkarım: `cift :: Integral a => a -> Bool`

5.3 Aşırı Yükleme (Overloading)

- Bir tanımlayıcı (identifier), aynı kapsamda (scope) iki veya daha fazla farklı prosedürü belirtiyorsa, aşırı yüklenmiş olduğu söylenir.
- Bu tür bir aşırı yükleme, yalnızca her prosedür çağrısının net olması durumunda kabul edilebilir, yani, derleyici çağrılacak prosedürü yalnızca tür bilgisini kullanarak benzersiz bir şekilde tanımlayabilir.
- C dilindeki "-" operatörü:
  - int üzerinde işareti ters çevirme (int i = 2; int j = - i;)
  - float üzerinde işareti ters çevirme
  - integer çıkarma işlemi (int a = 3, b = 5; a - b;)
  - float çıkarma işlemi

5.3 Aşırı Yükleme (Overloading)

- Aşırı yükleme (overloading) ve parametrik polimorfizm kavramlarını karıştırmamaya dikkat etmeliyiz.
- Aşırı yükleme, ayrı olarak tanımlanmış prosedürlerin aynı tanımlayıcıya sahip olması anlamına gelir.
- Polimorfizm, geniş bir ilgili türler ailesinin argümanlarını kabul eden tek bir prosedürün özelliğidir; parametrik prosedür bir kez tanımlanır ve türleri ne olursa olsun argümanları üzerinde tek tip olarak çalışır.
- Parametrik polimorfizm, dilin ifade gücünü gerçekten arttırır, çünkü polimorfik bir prosedür sınırsız çeşitlilikte argüman alabilir.

5.3 Aşırı Yükleme (Overloading)

```
// C# overloading
public class Area {
    public double area(double s) {
        double area = s * s;
        return area;
    }

    public double area(double l, double b) {
        double area = l * b;
        return area;
    }
}


-- Haskell polymorphism
birincisi :: (a, b) -> a
birincisi (x, y) = x
```

5.4 Tip Dönüşümleri (Type Conversions)

- Tip dönüştürme, bir tipin değerlerinden, farklı bir tipin karşılık gelen değerlerine eşlemedir (mapping).
- Örn: Tam sayılardan reel sayılara eşleme

  {..., −1 → −1.0, 0 → 0.0, 1 → 1.0, 2 → 2.0, ...}

- Örn: char'ları uzunluğu 1 olan stringlere eşleme

  {..., 'a' → "a", 'b' →"b", 'c' → "c", ...}

- Örn: Reel sayılardan tam sayılara eşleme

  {..., −1.7 → −1, 0.3 → 0, 1.1 → 1, 2.9 → 2, ...}

5.4 Tip Dönüşümleri (Type Conversions)

- Bazı tür dönüşümleri toplam eşlemelerdir (örneğin, karakterlerden karakter kodlarına eşleme), ancak diğerleri kısmi eşlemelerdir (örneğin, karakter kodlarından karakterlere eşleme). Bir tür dönüştürmenin kısmi bir eşleme olduğu durumlarda program başarısız olabilir.
- Cast etme, açık bir tür dönüşümüdür. C, C++ ve JAVA'da bir cast etme “(T)E” biçimindedir. (int) d
- Zorlama (coersion), örtük bir tür dönüştürmedir ve sözdizimsel bağlamın gerektirdiği her yerde otomatik olarak gerçekleştirilir. C dili tamsayılardan gerçek sayılara, dar aralıktan geniş aralıklı tam sayılara, düşük kesinlikten (precision) yüksek kesinliğe kadar zorlamalara izin verir. Java: subclass -> superclass