Linguaggi e Programmazione Orientata agli Oggetti

Soluzioni della prova scritta del 3 settembre

a.a. 2012/2013

1. (a) Data la seguente linea di codice Java

```
Pattern p = Pattern.compile("0_*[0-7]([0-7_]*[0-7])?[lL]?");
```

Indicare quali delle seguenti asserzioni falliscono, motivando la risposta.

L'espressione definisce un sottoinsieme dei literal di tipo **`double`** per il linguaggio Java.

i. **`assert`** `p.matcher("0").matches();` fallisce poiché dopo la cifra `0` iniziale deve seguire almeno un'altra cifra ottale

ii. **`assert`** `p.matcher("042__").matches();` fallisce poiché dopo una sequenza di delimitatori _ deve seguire una cifra ottale

iii. **`assert`** `p.matcher("04_2L_").matches();` fallisce poiché il carattere `L` è ammesso solo in fondo alla stringa

iv. **`assert`** `p.matcher("04_2l").matches();` ha successo

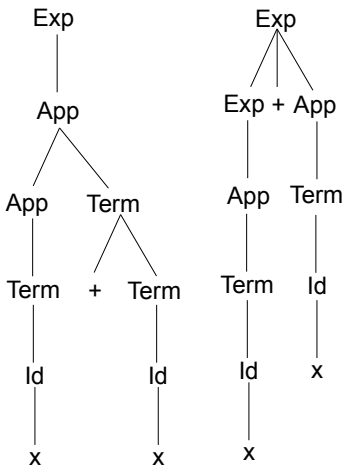
v. **`assert`** `p.matcher("04_2L").matches();` ha successo

vi. **`assert`** `p.matcher("0__4__2").matches();` ha successo

(b) Mostrare che la seguente grammatica è ambigua.

```
Exp  ::= App | Exp + App
App  ::= Term | App Term
Term ::= Id | + Term
Id ::= x | y | z
```

Esistono due diversi alberi di derivazione per la stringa `x+x`



(c) Modificare la grammatica definita al punto precedente in modo che **non sia ambigua** e che il linguaggio generato a partire dal non terminale `Exp` **rimanga lo stesso**.

```
Exp  ::= App | Exp + App
App  ::= Term | App Id
Term ::= Id | + Term
Id ::= x | y | z
```

2. Vedere il file `soluzione.ml`

3.
```
public class P {
        public String m(int i) {
              return "P.m(int)";
        }
```

```
      public String m(long l) {
            return "P.m(long)";
      }

      public String m(double d) {
            return "P.m(double)";
      }

}
public class H extends P {

      public String m(Integer i) {
            return "H.m(Integer) " + super.m(i);
      }

      public String m(Long l) {
            return "H.m(Long) " + super.m(l);
      }

      public String m(Double d) {
            return "H.m(Double) " + super.m(d);
      }

}

public class Test {
        public static void main(String[] args) {
                H h = new H();
                P p = h;
                System.out.println(...);
        }

}
```

I metodi sono tutti **public**, quindi accessibili in `Test`.

(a) `p.m(1)`: `p` e `1` hanno rispettivamente tipo statico `P` e **int**. Tutti i metodi in `P` sono applicabili, ma `m(int)` è il più specifico (**int** $\leq$ **long**, **int** $\leq$ **double**) ed è anche appropriato.

A run-time `p` contiene un oggetto della classe `H`, che eredita il metodo `m(int)` da `P`, quindi viene stampata la stringa

```
P.m(int)
```

(b) `p.m((Long) (1L))`: il cast è staticamente e dinamicamente corretto per boxing; ricevitore e argomento hanno rispettivamente tipo statico `P` e `Long`. Nessun metodo è applicabile per sottotipo, mentre entrambi `m(long)` e `m(double)` sono applicabili per unboxing (ed eventuale reference widening); viene selezionato il metodo `m(long)` che è il più specifico (**long** $\leq$ **double**) (ed è anche appropriato).

A run-time `p` contiene un oggetto della classe `H`, che eredita il metodo `m(long)` da `P`, quindi viene stampata la stringa

```
P.m(long)
```

(c) `h.m((Long) (1L))`: il cast è corretto per lo stesso motivo del punto precedente; ricevitore e argomento hanno rispettivamente tipo statico `H` e `Long`.

Solamente il metodo `m(Long)` è applicabile per sottotipo (ed è anche appropriato).

A run-time `h` contiene un oggetto della classe `H`, quindi il metodo invocato è quello in `H`; nel body del metodo l'invocazione **super**`.m(l)` è staticamente corretta e la segnatura selezionata è `m(long)`, analogamente al punto precedente.

Viene stampata la stringa

```
H.m(Long) P.m(long)
```

(d) `h.m(4.2)`: ricevitore e argomento hanno rispettivamente tipo statico `H` e **double**. Il metodo `m(double)` è l'unico applicabile per sottotipo (ed è anche appropriato).

A run-time `h` contiene un oggetto della classe `H`, che eredita il metodo `m(double)` da `P`, quindi viene stampata la stringa

```
P.m(double)
```

(e) `h.m(4.2F)`: ricevitore e argomento hanno rispettivamente tipo statico `H` e **float**. Il metodo `m(double)` è l'unico applicabile per sottotipo (ed è anche appropriato).

A run-time `h` contiene un oggetto della classe `H`, che eredita il metodo `m(double)` da `P`, quindi viene stampata la stringa

```
P.m(double)
```

(f) `h.m((Double) 4.2)`: il cast è staticamente e dinamicamente corretto per boxing; ricevitore e argomento hanno rispettivamente tipo statico `H` e `Double`.

Solamente il metodo `m(Double)` è applicabile per sottotipo (ed è anche appropriato).

A run-time `h` contiene un oggetto della classe `H`, quindi il metodo invocato è quello in `H`; nel body del metodo l'invocazione **`super`**`.m(d)` è staticamente corretta poiché nessun metodo è applicabile per sottotipo e l'unico metodo applicabile per unboxing è `m(`**`double`**`)` (che è anche appropriato).

Viene stampata la stringa

```
H.m(Double) P.m(double)
```

4. Vedere le soluzioni nel file `soluzione.jar`.