Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2016/2017

15 febbraio 2017

1. (a) Indicare quali delle asserzioni contenute nel seguente codice Java hanno successo e quali falliscono, motivando la risposta.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class MatcherTest {
   public static void main(String[] args) {
      Pattern regEx = Pattern.compile("([a-zA-Z][a-zA-Z0-9]*)|([0-9]+([eE][0-9]+)?)|(\\s+)");
      Matcher m = regEx.matcher("x1E0 42e04");
      m.lookingAt();
      assert m.group(1).equals("x");
      assert m.group(0).equals("x1E0");
      m.region(m.end(), m.regionEnd());
      m.lookingAt();
      assert m.group(4) != null;
      m.region(m.end(), m.regionEnd());
      m.lookingAt();
      assert m.group(2).equals("42");
      assert m.group(3).equals("04");
      assert m.group(0).equals("42e04");
   }
}
```

   (b) Mostrare che la seguente grammatica è ambigua.

```
Exp  ::= ! Exp | Exp [ Exp ] | ( Exp ) | Id
Id   ::= x | y | z
```

   (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` **resti invariato**.

2. Considerare la funzione `replace : 'a -> 'a -> 'a list -> 'a list` tale che `replace` x y l sostituisce nella lista l tutte le occorrenze di x con y, lasciando gli altri elementi invariati. Esempio:

```
# replace 'L' 'l' ['H';'e';'L';'L';'o'];;
- : char list = ['H'; 'e'; 'l'; 'l'; 'o']
```

   (a) Definire la funzione `replace` senza uso di parametri di accumulazione.

   (b) Definire la funzione `replace` usando un parametro di accumulazione affinché la ricorsione sia di coda.

   (c) Definire la funzione `replace` come specializzazione della funzione `it_list` così definita:

```
let rec it_list f a = function x::l -> it_list f (f a x) l | _ -> a;;
val it_list : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
```

3. Considerare la seguente implementazione di successioni finite di interi tale che **new** `IntSeq(`*min*`, `*max*`, `*step*`)` rappresenta l'insieme di interi $\{min + k{\cdot}step \mid k \geq 0$ e $min + k \cdot step \leq max\}$.

   Per esempio, **new** `IntSeq(1, 10, 4)` corrisponde all'insieme $\{1, 5, 9\}$.

```
public class IntSeq implements Iterable<Integer> {
    /* implements the set { min+k*step | k >= 0 and min+k*step <= max } */
    private final int min;
    private final int max;
    private final int step; // invariant: step > 0

    public IntSeq(int min, int max) { // default step is 1
        ...
    }
    public IntSeq(int min, int max, int step) {
        ...
    }
    public int getMin() {
        ...
    }
    public int getMax() {
        ...
    }
    public int getStep() {
        ...
    }
    public Iterator<Integer> iterator() {
        ...
    }
}
class IntSeqIterator implements Iterator<Integer> {
    private int min;
    private final int max;
    private final int step;

    public IntSeqIterator(IntSeq seq) {
        ...
    }
    public boolean hasNext() {
        ...
    }
    public Integer next() {
        ...
    }
}
```

   (a) Completare le definizioni dei costruttori e dei metodi *getter* della classe `IntSeq`.

   (b) Completare la definizione del metodo `iterator()` della classe `IntSeq`.

   (c) Completare la definizione del costruttore della classe `IntSeqIterator`.

   (d) Completare le definizioni dei metodi `hasNext()` e `next()` della classe `IntSeqIterator`.

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(long l) {
        return "P.m(long)";
    }
    String m(double d) {
        return "P.m(double)";
    }
    String m(Object... os) {
        return "P.m(Object...)";
    }
}
public class H extends P {
    String m(long l) {
        return super.m((double) l) + " H.m(long)";
    }
    String m(Double d) {
```

```
        return super.m(d, d + 1) + " H.m(Double)";
    }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

(a) `p.m(42L)`

(b) `p2.m(42L)`

(c) `h.m(42L)`

(d) `p.m(42.0)`

(e) `p2.m(42.0)`

(f) `h.m(Double.valueOf(42.0))`