Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2011/2012

6 febbraio 2012

1. Sia $\mathcal{L}$ il linguaggio generato dalla seguente grammatica a partire dal simbolo non terminale `Path`.

```
Path ::= Id | Id . Path
Id ::= Lt | Id Dg | Id Lt
Lt ::= a | ... | z | A | ... | Z // all upper and lower case English letters
Dg ::= 0 | ... | 9 // all digits
```

   (a) Definire una grammatica regolare destra che generi il linguaggio $\mathcal{L}$ *(una grammatica* (T, N, P) *è detta* regolare destra *se ogni produzione in* P *è della forma* S `::=` u *oppure* S `::=` u R *oppure* S `::=` ϵ*, dove* $S, R \in N$*,* $u \in T$*).*

   (b) Completare la seguente espressione Java in modo che venga creata un'istanza di `java.util.regex.Pattern` che riconosca il linguaggio $\mathcal{L}$.

```
java.util.regex.Pattern.compile("...");
```

2. (a) Definire, in modo diretto e senza parametro di accumulazione, la funzione

```
ordered : 'a list -> bool
```

   che controlla se una lista è ordinata in modo crescente e **senza ripetizioni**.

   (b) Definire, in modo diretto e senza parametro di accumulazione, la funzione

```
all : ('a -> 'a -> bool) -> 'a list -> bool
```

   tale che `all p l` si valuti in `true` se e solo se il predicato `p` è vero su tutti gli elementi contigui della lista `l` (quindi `all p [`e_1`;...;`e_n`] = p` e_1 e_2 `&& p` e_2 e_3 `&& ... && p` e_{n-1} e_n).

   (c) Utilizzando la funzione `all` specificata al punto 2

      i. Definire la funzione `ordered` specificata al punto 1.

      ii. Definire la funzione `interval : int list -> bool` che controlla che ogni elemento di una lista di interi sia il successore del numero che si trova alla posizione immediatamente precedente. Per esempio `interval [5;6;7;8] = true` e `interval [5;6;8] = false`.

3. Completare i metodi delle classi `SetUtil` e `Test`. Il metodo `sumAll` **deve essere definito** utilizzando la classe `Sum` e il metodo `SetUtil.iterate`.

```
package scritto2012_02_06;
// Functions of type X -> Y
public interface Function<X, Y> {
        // applies this function to x and returns the result of type Y
        Y apply(X x);
}
```

```
package scritto2012_02_06;
import java.util.Set;
public class SetUtil {

        // returns true iff predicate p is true for all elements of s
        public static <X> boolean all(Function<X, Boolean> p, Set<X> s) {/* completare */}

        // returns true iff predicate p is true for at least one element of s
        public static <X> boolean exists(Function<X, Boolean> p, Set<X> s) {/* completare */}

        // applies f to all elements of inSet and insert the results in outSet
        public static <X, Y> void map(Function<X, Y> f, Set<X> inSet, Set<Y> outSet) {/* completare */}

        // f is a curried function of type X -> Y -> X where the accumulator has type X
        // the method returns f (... (f (f initVal e_1) e_2) ...) e_n where s = {e_1, e_2, ..., e_n}
        public static <X, Y> X iterate(Function<X, Function<Y, X>> f, X initVal,
                        Set<Y> s) {/* completare */}

        // adds to outSet all elements of the union of inSet1 and inSet2
        // inSet1 and inSet2 are unmodified
        public static <X> void union(Set<X> inSet1, Set<X> inSet2, Set<X> outSet) {/* completare */}

        // adds to outSet all elements of the intersection of inSet1 and inSet2
        // inSet1 and inSet2 are unmodified
        public static <X> void intersect(Set<X> inSet1, Set<X> inSet2, Set<X> outSet) {/* completare */}
}
```

```
public class Test {

        // defines the function f such that f x y = x + y
        static final class Sum implements Function<Integer, Function<Integer, Integer>> {
                @Override
                public Function<Integer, Integer> apply(final Integer x) {
                        return new Function<Integer, Integer>() {
                                /* completare */
                        };
                }
        }

        // returns the sum of all elements of s
        // by using class Sum and method SetUtil.iterate
        static Integer sumAll(Set<Integer> s) {/* completare */}
}
```

4. Considerare le seguenti dichiarazioni di classi Java:

```
package a;
public interface A {String m(A a);}

package a;
public class AC implements A {
        public String m(A a) {return "AC m(A)";}
        String m(AC ac) {return "AC m(AC)";}
}

package a;
import b.*;
public class C extends BC implements A, B {
        public String m(A a) {return "C m(A)";}
        public String m(B b) {return "C m(B)";}
        public String m(AC ac) {return "C m(AC)";}
        public String m(BC bc) {return bc.m((B) bc) + " C m(BC)";}
}

package b;
public interface B {String m(B b);}

package b;
import a.AC;
public class BC extends AC implements B {
        public String m(B b) {return "BC m(B)";}
        String m(BC ac) {return "BC m(BC)";}
}

package c;
import a.*;
import b.*;
import static java.lang.System.out;
public class Main {
        public static void main(String[] args) {
                AC ac = new AC();
                BC bc = new BC();
                C c = new C();
                ...
        }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Main` l'espressione indicata.

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) `out.println(ac.m(ac));`

(b) `out.println(ac.m((A) ac));`

(c) `out.println(bc.m(ac));`

(d) `out.println(bc.m(bc));`

(e) `out.println(c.m(ac));`

(f) `out.println(c.m(bc));`