

IV Appello - 2 luglio 2009

Cognome

Nome

Matricola

1 (4 punti)

Scrivete il programma Java, **Esercizio**, che riceve gli argomenti da riga di comando, li stampa a caratteri maiuscoli, li conta (i token sono separati da spazi bianchi) e stampa il numero ottenuto. Ad esempio:

```
> java Esercizio prova di esame!
```

Produce in output:

```
PROVA DI ESAME!
```

```
3
```

```
public class Esercizio {  
  
    public static void main(String[] args) {  
        ConsoleOutputManager out = new ConsoleOutputManager();  
        String ris = new String();  
        for (int i=0; i<args.length; i++)  
            ris += args[i].toUpperCase() + " ";  
        out.println(ris);  
        out.println(args.length);  
    }  
  
}
```

2 (4 punti)

Supponete di aver istanziato l'oggetto **tokenizer** di classe **StringTokenizer** con la stringa **frase** di input:

```
StringTokenizer tokenizer = new StringTokenizer(frase);
```

Ora vogliamo manipolare con una data regola i singoli token per produrre una traduzione della frase, ad esempio con il ciclo:

```
for (int i=0; i < tokenizer.countTokens(); i++)  
    traduzione += traduci(tokenizer.nextToken());
```

Cosa succede con la stringa di input frase "prova di esame di luglio"?

Vengono trattati solo i primi 3 token. Immaginando che la traduzione sia in verità una semplice manipolazione, come portare la stringa in maiuscolo, avremmo come output:
PROVA DI ESAME

Se c'è un problema spiegate e correggete. Ricordo che il metodo **countTokens** calcola il numero di volte in cui può essere chiamato il metodo **nextToken** su questo tokenizer prima di generare un'eccezione.

Il problema è creato dalla chiamata ad ogni passo del ciclo del metodo **countTokens()** che dinamicamente riporta il numero di tokens rimasti dopo che il metodo **nextToken()** ha già estratto il prossimo token da manipolare. Una soluzione è salvare in una variabile il dato all'inizio del ciclo:

```
int limite = tokenizer.countTokens();
```

Oppure invocare nel ciclo il metodo **hasMoreTokens()**.

3 (3 punti)

Commentate lo spezzone di codice seguente, spiegando concisamente il suo effetto:

```
String[] archivio = new String[MAX];  
for (inti=0; i<MAX; i++)  
    out.println(archivio[i].length());
```

Solleva l'eccezione **nullPointerException** poiché si invoca il metodo **length()** su un oggetto non istanziato (**archivio[i]** che è inizializzato a **null** infatti)

4 (3 punti)

Fornite una concisa spiegazione del perchè **nullPointerException** NON è un'eccezione controllata.

Solo a run time si può controllare se un oggetto è stato creato. Staticamente il compilatore può solo controllare che il metodo chiamato sia tra quelli disponibili per un dato oggetto, ma non può sapere se l'oggetto è stato creato.

5 (3 punti)

Stabilire qual'è l'output del seguente codice Java, supponendo che sia stata dichiarata la variabile `int i`:

<pre>for (i = 1; i < 10; i++) { if (i%3 == 0) break; System.out.println (i); } System.out.println (i);</pre>	<pre>1 2 3 // i all'uscita del ciclo</pre>	
<pre>for (i = 1; i < 10; i++) { if (i%3 == 0) continue; System.out.println (i); } System.out.println (i);</pre>	<pre>1 2 4 5 7 8 10 // i all'uscita del ciclo</pre>	
<pre>for (i = 1; i < 10; i++) { if (i%3 == 0) System.exit(0); System.out.println (i); } System.out.println (i);</pre>	<pre>1 2</pre>	

6 (3 punti)

Supponete sia definita una classe `BarcaVela` derivata dalla classe `Barca` e supponete che quest'ultima classe abbia definito i metodi pubblici `velocitàNodi()` e `accendiMotori()`.

Un oggetto di classe `BarcaVela` risponde ai metodi `velocitàNodi()` e `accendiMotori()`? **SI NO**

Questi metodi dovranno eseguire le stesse azioni sia per gli oggetti di classe `BarcaVela` sia per gli oggetti di classe `Barca`, sempre e comunque? **SI NO**

Come si chiama in Java il concetto di specializzazione dei metodi?
Sovrascrittura

Se per la classe `BarcaVela` è stato definito il metodo `alzareRanda()`, questo può essere invocato su un oggetto di classe `Barca`? **SI NO**

7 (5 punti)

Data la classe seguente con i costruttori elencati:

<pre>class Dipendente { private String nome; }</pre>	<pre>Dipendente(){ nome = new String(); } Dipendente(String unNome) { nome = unNome; } }</pre>
--	--

E' possibile al di fuori della classe cambiare il valore del membro `nome` di un oggetto già istanziato? **SI NO**

Come? (scrivere le istruzioni necessarie)

```
public void modificaNome (String nuovoNome) {
    nome = nuovoNome;
}
```

Per estendere la classe con una specializzazione, introduciamo la sottoclasse `Impiegato`, completare quindi il codice seguente opportunamente:

<pre>class Impiegato extends Dipendente { private int settore; // 1, 2, 3 ad esempio Impiegato() { super(); settore = 1; // valore di default } }</pre>	<pre>Impiegato (String unNome, int unSettore) { super (unNome); settore = unSettore; } }</pre>
---	---

Supponendo di aver eseguito l'istruzione: `Impiegato imp = new Impiegato();`

E' possibile accedere al campo `nome` della superclasse? **indirettamente SI NO**

Per accedere al campo `nome` è corretta l'espressione: `imp.nome` ? **SI NO**

Esistono altri modi? **SI NO**

Definire il codice di possibili alternative, per accedere al campo: **nella classe Dipendente** definiamo un metodo pubblico **riportaNome()** che verrà ereditato dalle sottoclassi

```
public String riportaNome() {  
    return nome;  
}
```

quindi si potrà invocare **imp.riportaNome()**

In generale, in una sottoclasse è possibile accedere direttamente ad un campo privato di un'istanza della superclasse? **SI NO**

In una sottoclasse è possibile usare un metodo privato della superclasse? **SI NO**

8 (5 punti)

Studiate la seguente funzione ricorsiva di una data classe:

```
public static int f(int m, int n) {  
    if (m < 5)  
        return n; //indirizzo A  
    else if (m > n)  
        return 1 + f(m-1, n+1); //indirizzo B  
    else  
        return 1 - f(m-2, n+2); //indirizzo C  
}
```

calcolate il valore riportato dalla funzione e date una traccia del processo ricorsivo indicando il numero delle chiamate successive alla prima e mettendo in evidenza l'indirizzo in cui avviene la chiamata ricorsiva

$f(8, 2) = -3$

numero delle chiamate (oltre alla prima): **4**

traccia dell'esecuzione:

```
f(8, 2) →  
    1 + f(7, 3)           indirizzo B  
        1 + f(6, 4)       indirizzo B  
            1 + f(5, 5)     indirizzo B  
                1 - f(3, 7) indirizzo C  
                    7         indirizzo A
```

$f(5, 3) = 5$

numero delle chiamate (oltre alla prima): **1**

traccia dell'esecuzione:

```
f(5, 3) →  
    1 + f(4, 4)           indirizzo B  
        4                 indirizzo A
```

$f(9, 1) = -2$

numero delle chiamate (oltre alla prima): **5**
traccia dell'esecuzione:

```
f(9, 1) →  
  1 + f(8, 2)           indirizzo B  
    1 + f(7, 3)         indirizzo B  
      1 + f(6, 4)       indirizzo B  
        1 + f(5, 5)     indirizzo B  
          1 - f(3, 7)   indirizzo C  
            7           indirizzo A
```