Errata Corrige di “Java 7”

pag. 17 - par. 1.6 - testo nel riquadro dalla 4a riga

Non è specificato che i 2 punti sono validi solo in ambiente Windows. Manca quindi il seguente punto:

```
- In ambienti Unix-Linux, tenere premuto SHIFT - ALT GR, e poi il tasto con
il simbolo della parentesi quadra aperta "[".
```

pag. 19 - par. 1.6 - testo nel primo riquadro dalla 4a riga

Non è specificato che i 2 punti sono validi solo in ambiente Windows. Manca quindi il seguente punto:

```
- In ambienti Unix-Linux, tenere premuto SHIFT - ALT GR, e poi il tasto con
il simbolo della parentesi quadra aperta "]".
```

pag. 18 - par. 1.6 - prima riga della pagina

```
A questo punto possiamo iniziare ad aprire un prompt di Dos e spostarci
all'interno
```

dato che tutti i comandi funzionano sia in Windows che in ambienti Unix-Linux, in realtà è:

```
A questo punto possiamo iniziare ad aprire un prompt di Dos in Windows
(oppure una shell in ambiente Unix-Linux) e spostarci all'interno
```

pag. 75 – par. 3.2.5 – Quinta riga del paragrafo

```
con esso siamo in gradi di rappresentare
```

in realtà è:

```
con esso siamo in grado di rappresentare
```

pag. 75 - par. 3.2.5 – terzultima riga

```
\n che equivale ad andare a capo (tasto return)
```

in realtà è:

```
\n che equivale ad andare a capo (new line)
```

pag. 101 – par 4.1.4 – Ultima riga della prima tabella

```
Shift a destra senza segno di assegnazione >>=
```

in realtà è

```
Shift a destra senza segno di assegnazione >>>=
```

pag. 106 - par. 4.1.8 - Seconda riga tabella

```
da sx a dx                  ++  --  +  -  ~  !  (tipi di dati)
```

in realtà è:

```
da sx a dx                  ++  --  ~  !  (tipi di dati)
```

pag. 171 - par. 6.2.4 – Secondo riquadro che riporta linee di codice

```
public boolean equals(Object obj) {
    if (obj instanceof Punto) return false;
        Punto that = (Punto obj);
        return this.x == that.x && this.y == that.y;
    }
}
```

in realtà è:

```
public boolean equals(Object obj) {
    if (obj instanceof Punto) {
        Punto that = (Punto) obj;
        return this.x == that.x && this.y == that.y;
    } else return false;
}
```

pag. 300 – Par. 10.8 – Esercizio 10.a) numero 1

```
Ogni eccezione che estende in qualche modo una ArithmeticException è una
checked exception
```

in realtà è:

```
Ogni eccezione che estende in qualche modo una ArithmeticException è una
unchecked exception
```

pag. 353 - par. 12.1.5 - Dalla fine della settima riga dopo tabella e riquadro

```
Esistono code FIFO (che sta per "First In First Out") che definiscono come
testa della coda il primo elemento inserito. Un'implementazione di coda FIFO
l'abbiamo già vista: la classe LinkedList, che mette a disposizione i metodi
addLast(), getLast() e removeLast(). In realtà LinkedList implementa anche
la classe Deque e quindi può essere utilizzata come coda LIFO (che sta per
"Last In First Out") dove testa della coda è il l'ultimo elemento inserito.
Infatti mette a disposizione anche i metodi addFirst(), getFirst() e
removeFirst().
```

in realtà è:

```
Esistono code LIFO (che sta per "Last In First Out") che definiscono come
testa della coda l'ultimo elemento inserito. Un'implementazione di coda LIFO
l'abbiamo già vista: la classe LinkedList, che mette a disposizione i metodi
addLast(), getLast() e removeLast(). In realtà LinkedList implementa anche
l'interfaccia Deque e quindi può essere utilizzata come coda FIFO (che sta
per "First In First Out") dove testa della coda è il primo elemento
inserito. Infatti mette a disposizione anche i metodi addFirst(), getFirst()
e removeFirst().
```

pag. 373 - par. 12.2.1 - 8° punto (checkbox)

```
String substring(int startIndex, int number) restituisce una sottostringa
della stringa corrente, composta dal numero number di caratteri che partono
dall'indice startIndex
```

in realtà è:

```
String substring(int startIndex, int endIndex) restituisce una sottostringa
```

```
della stringa corrente, composta dai caratteri che partono  dall'indice
startIndex fino all'indice endIndex
```

pag. 420 – par. 13.6 – sette righe prima della fine della pagina

```
piu' complessi ed efficienti si semplificato
```

in realtà è:

```
piu' complessi ed efficienti si è semplificato
```

pag. 430, 431, 434, 435 – par. 14.2.2 e 14.3.1 – esercizio JDBCApp

Gli esempi non compilano in quanto:

1) le variabili res, con, cmd devono essere definiti prima della try principale come segue:

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try { . . .
```

2) la parte finally vuole la gestione via try() catch() delle close() come segue:

```
finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            rs = null;
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            stmt = null;
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
                conn = null;
            }
        } . . .
```

pag 442 – par. 14.3.10 – intero paragrafo

Le novità introdotte dalla versione 4.0 in poi in cui si parla anche nelle pagine seguenti, in realtà non sono state introdotte nella versione definitiva di Java 7. Il libro è stato scritto precedentemente all'uscita della versione stabile quando sembrava che tutte le interfacce definite (Select, Update, BaseQuery, DataSet etc.) sembrassero dover rientrare nei piani di Oracle.

pag. 448 – par. 14.3.10 – penultima riga

```
ResultSet rs = stmt.executeQuery("SELECT * FROM PERSONA")) {.....
```

in realtà è:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM PERSONA")); {.....
```

pag 487 – par. 15.4.2 - penultima riga prima della nota

```
Una buona programmazione ad oggetti richiederebbe che ogni evento abbia il
suo "gestore personale"
```

in realtà è:

```
Una buona programmazione ad oggetti richiede che ogni evento abbia il suo
"gestore personale"
```

pag. 577 - par. 18.2.1 - seconda riga del primo riquadro della pagina

```
import applicazione.utility.db.*;
```

in realtà è:

```
import applicazione.db.utility.*;
```