Errata Corrige di “Java 6”

Titolo

pag. XV – Prefazione – penultima riga

```
in varie importanti
```

in realtà è:

```
in importanti
```

pag. 15 - par. 1.5 – sestultima 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. 17 - par. 1.5 - 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. 26 – par. 2.2 – quartultima riga

```
 “modello” “definizione”
```

in realtà è:

```
 “modello”, “definizione”
```

pag. 56 – par. 3.2.1 – Terzo dei cinque riquadri che riportano linee di codice

```
byte b = 128; //il massimo per short è 127
```

in realtà è:

```
byte b = 128; //il massimo per byte è 127
```

pag. 62 – par. 3.2.3 – Terza riga

```
Esempi di literals sono ovviamente true e false, ma anche ,
```

in realtà è:

```
Esempi di literals sono ovviamente true e false, ma anche 3.14F,
```

```
pag. 63 - quarta riga
      \n che equivale ad andare a capo (tasto return)
      in realtà è:
      \n che equivale ad andare a capo (new line)
```

pag. 85 – par. 4.1.4 – tabella della verità, quarta colonna

```
Op1 ORO Op2
```

in realtà è:

```
Op1 OR Op2
```

pag. 89 – par. 4.1.4 – riquadro in grigio, terzultima riga

```
Se se solo se
```

in realtà è:

```
se e solo se
```

pag. 91 - 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. 111 – par. 5.2.1 – terza riga

```
o Astrazione funzionaleAstrazione dei datiAstrazione del sistema
```

in realtà è:

```
o Astrazione funzionale

o Astrazione dei dati

o Astrazione del sistema
```

pag. 126 – par. 5.5 – seconda riga

```
È è
```

in realtà è:

```
È
```

pag. 127 – par. 5.5.1 – penultimo rigo del paragrafo

```
In particolare diremo che LibroSuJava è sottoclasse di Libro, e Libro è
sottoclasse di LibroSuJava.
```

in realtà è:

```
In particolare diremo che LibroSuJava è sottoclasse di Libro, e Libro è
superclasse di LibroSuJava.
```

pag. 139 – par. 6.2.2 – Terzo dei quattro riquadri che riportano linee di codice

```
0.0
6.0
3.0
74.0
```

in realtà è:

```
6.0
0.0
3.0
74.0
```

pag. 141 – par. 6.2.3 – Esempio codice classi Punto e PuntoTridimenzionale (mancano tutti i parametri dei metodi setter

in realtà è:

```
public class Punto {
    private int x, y;

    public void setX(int x) {

        this.x = x;

    }

    public int getX() {

        return x;

    }

    public void setY(int y) {

        this.y = y;

    }

    public int getY() {

        return y;

    }

    public double distanzaDallOrigine() {

        int tmp = (x*x) + (y*y);

        return Math.sqrt(tmp);

    }

}
```

```
public class PuntoTridimensionale extends Punto {

    private int z;

    public void setZ(int z) {

        this.z = z;

    }

    public int getZ() {

        return z;

    }

    public double distanzaDallOrigine() {

        int tmp = (getX()*getX()) + (getY()*getY())

        + (z*z); // N.B. : x ed y non sono ereditate

        return Math.sqrt(tmp);

    }

}
```

pag. 145 – par. 6.2.3 – Primo riquadro che riporta linee di codice, terza riga

```
Punto that = (Punto obj);
```

in realtà è:

```
Punto that = (Punto) obj;
```

pag. 145 - par. 6.2.4 - Inizio pagina

```
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. 172 – par. 7.3.5 – riquadro che riporta linee di codice, ventunesima riga

```
int p1Z = Integer.parseInt(args[3]);
```

```
int p2X = Integer.parseInt(args[4]);
int p2Y = Integer.parseInt(args[5]);
int p2Z = Integer.parseInt(args[6]);
```

in realtà è:

```
int p1Z = Integer.parseInt(args[2]);
int p2X = Integer.parseInt(args[3]);
int p2Y = Integer.parseInt(args[4]);
int p2Z = Integer.parseInt(args[5]);
```

pag. 185 - par. 8.3.1 - Righe tra il blocco di codice ed il blocco evidenziato

```
La chiamata al costruttore della superclasse mediante super deve essere la
prima istruzione di un costruttore e, ovviamente, non potrà essere inserita
all'interno di un metodo che non sia un costruttore.
```

in realtà non deve esserci, dato che è già riportata, tale e quale, nella pagina precedente poco prima del riquadro evidenziato, in basso.

pag. 188 - par. 8.4.1 - Penultima riga del paragrafo

```
a tali variabili d'istanza,, che in molti casi
```

in realtà è:

```
a tali variabili d'istanza, che in molti casi
```

pag. 189 - par. 8.4.2 - Quinta riga dopo il codice

```
se esterne Outer.
```

in realtà è:

```
se esterna Outer
```

pag. 192 - par. 8.4.3 - Quarta riga

```
}   0
```

in realtà è:

```
}
```

pag. 201 – par. 9.3.1 – Prima riga

1. Per raggiungere l'obiettivo ci sono due soluzioni:Impostare la variabile d'ambiente CLASSHPATH direttamente da sistema operativo.
2. Impostare CLASSPATH solo per la nostra applicazione.

in realtà è:

```
Per raggiungere l'obiettivo ci sono due soluzioni:
```

1. Impostare la variabile d'ambiente CLASSHPATH direttamente da sistema operativo.
2. Impostare CLASSPATH solo per la nostra applicazione.

pag. 203 – par. 9.3.3 – prima riga del secondo box grigio

```
java -cp .; C:\cartellaConFileJAR\'*' miopackage.MiaClasseConMainSi
```

in realtà è:

```
java -cp .; C:\cartellaConFileJAR\'*' miopackage.MiaClasseConMain

Si
```

pag. 212 – par. 9.5.4 – ultimo riquadro che riporta linee di codice

```
double d = Math.sqrt(4);
```

in realtà è:

```
double d = sqrt(4);
```

pag. 214 - par. 9.6.2 - Ultima riga del secondo riquadro

```
può esserlo.
```

in realtà è:

```
deve esserlo.
```

pag. 223 – par. 9.8.3 – rige 3,6 e 9 primo riquadro che riporta linee di codice

```
case AZIONE.AVANTI:
...
case AZIONE.INDIETRO:
...
case AZIONE.FERMO:
```

in realtà è:

```
case Azione.AVANTI:
...
case Azione.INDIETRO:
...
case Azione.FERMO:
```

pag. 238 – par. 10.4 – ultima riga

```
Infatti, per un particolare programma.
```

in realtà è:

```
Infatti, per un particolare programma,
```

pag. 246 - par. 10.5 - Inizio quint'ultima riga

```
te dallo sviluppatore fa sulla propria applicazione.
```

in realtà è:

```
te dallo sviluppatore sulla propria applicazione.
```

pag. 258 - par. 10.6 - Inizio terza riga del paragrafo 10.6

```
sulla classi principali.
```

in realtà è:

```
sulle classi principali.
```

pag. 260 – par. 11.1 – terz'ultima riga paragrafo 11.1

```
u semplice applet
```

in realtà è:

```
una semplice applet
```

pag. 273 – par. 11.3.1 – i numeri di riga non sono corretti dovrebbero essere corretti sottraendo un'unità

Per esempio

```
Alla riga 27 viene assegnata...
```

in realtà è:

```
Alla riga 26 viene assegnata...
```

pag. 279 – par. 11.4.1 – sotto “output senza sincronizzazione”

```
[Hello[Synchronized[World]
```

in realtà è:

```
[Hello[Synchronized[World]

]

]
```

pag. 279 – par. 11.4.1 – quint'ultima riga

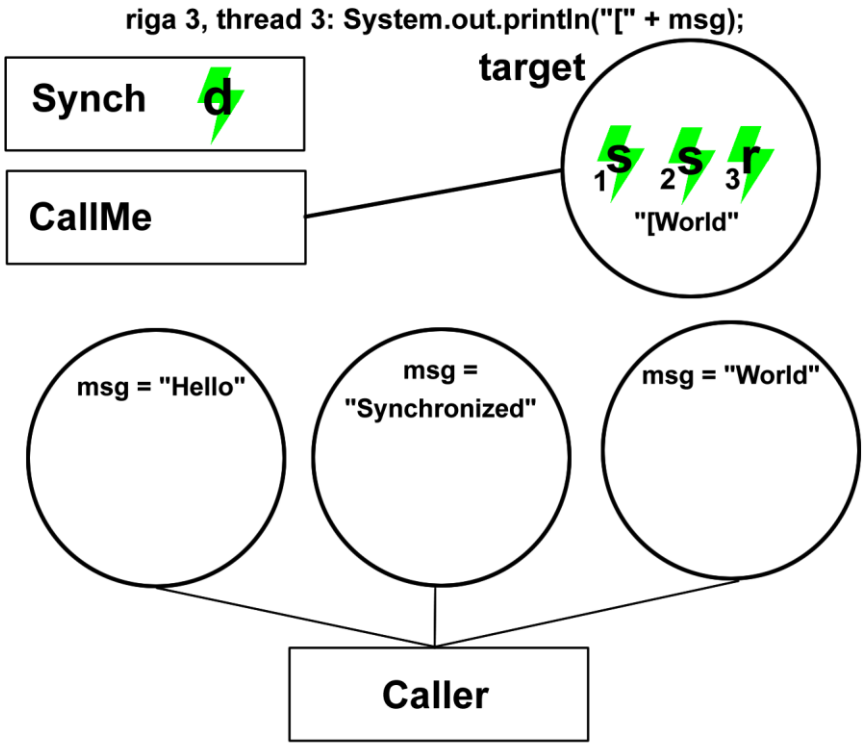
```
stringa world
```

in realtà è:

```
stringa World
```

pag. 282 – par. 11.4.1 – Figura 10.20 errata

in realtà è:



pag. 298 – par. 12.1.2 – quinta riga del riquadro che riporta linee di codice

```
for(int i = 0; i <=table.size(); i++) {
```

in realtà è:

```
for(int i = 1; i <=table.size(); i++) {
```

pag. 301 - par. 12.1.3 - Inizio seconda riga dopo il primo riquadro

```
le anche ottenere un istanza di Iterator
```

in realtà è:

```
le anche ottenere un'istanza di Iterator
```

pag. 301 – par. 12.1.3 – seconda riga sotto il riquadro che riporta linee di codice

```
c
b
```

in realtà è:

```
c
b
a
```

pag. 305 – par. 12.1.5 – terza riga sotto il box grigio

```
remove() ritorna null, mentre poll() lancia un'eccezione
```

in realtà è:

```
poll() ritorna null, mentre remove() lancia un'eccezione
```

pag. 305 – par. 12.1.5 – terza riga sotto il box grigio

```
element() ritorna null, mentre peek() lancia un'eccezione
```

in realtà è:

```
peek() ritorna null, mentre element() lancia un'eccezione
```

pag. 305 - 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 l'ultimo 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 primo 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. 321 – par. 12.1.13 – ultimo riquadro che riporta linee di codice

```
StringTokenizer st = new StringTokenizer(“questo è un test”,”t”,true);
```

in realtà è:

```
StringTokenizer st = new StringTokenizer(“questo è un test”,”t”,false);
```

pag. 326 - par. 12.2.1 - 4° 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. 338 - par. 13.2.1 - Fine ultima riga della pagina

```
nel nostro esempio 2 elevato alla quarta potenza,
```
in realtà è:
```
nel nostro esempio 2 elevato alla quarta potenza.
```

pag. 343 – par. 13.3 – quarta riga del paragrafo 13.3

```
tipo di scrittura output
```
in realtà è:
```
tipo di scrittura (output)
```

pag. 343 – par. 13.3 – quinta riga del paragrafo 13.3

```
scrittura sono molt
```
in realtà è:
```
scrittura sono molte
```

pag. 347 – par. 13.3.3 – ultima riga

```
caratteri ed array di caratteri characters
```
in realtà è:
```
caratteri ed array di caratteri
```

pag. 354 . par. 13.4.2 - Inizio 1a riga

```
due oggetti: File: inputFile, che rappresenta il file originale,
```
in realtà è:
```
due oggetti File: inputFile, che rappresenta il file originale,
```

pag. 356 - par. 13.4.2 - Inizio terza riga della pagina

```
Anch'esso metodo restituisce il valore 0L se il nome del file non coincide
```
in realtà è:
```
Anche questo metodo restituisce il valore 0L se il nome del file non
coincide
```

pag. 362 – par. 13.5– nona riga sotto il riquadro che riporta linee di codice

```
il metodo writeLine()
```
in realtà è:
```
il metodo write()
```

pag. 364 - par. 13.6 - Terz'ultima riga

```
Tutto ciò è stato preceduto da un velocissima introduzione
```
in realtà è:
```
Tutto ciò è stato preceduto da una velocissima introduzione
```

pag. 372 – par. 14.3.1– ottava riga del riquadro che riporta linee di codice

```
try {
    Properties p = new Properties();
    p.load(new FileInputStream("config.properties"));
```

in realtà è:

```
try {
    Properties p = new Properties();
            try {
                p.load(new FileInputStream("config.properties"));
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
```

pag. 376 – par. 14.3.5 – seconda riga dopo la prima tabella

```
associatia i metodi
```

in realtà è:

```
associati ai metodi
```

pag. 376 - par. 14.3.5 - Inizio seconda riga della seconda tabella

```
gsetASCIIStream
```

in realtà è:

```
setASCIIStream
```

pag 381 – 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 6. 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. 394 - par. 14.4.4 - 4a riga del primo listato

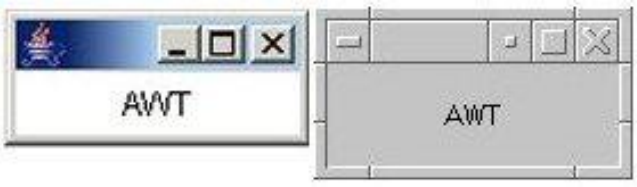
Document doc = factory.newDocument();

in realtà è:

Document doc = factory.newDocumentBuilder().newDocument();

pag. 403 – par. 15.2– figura 15.1 errata

in realtà è:



pag. 422 – par. 15.4.2– decima riga del riquadro che riporta linee di codice

```
b.addActionListener(new InnerButtonHandler(l));
```
in realtà è:
```
b.addActionListener(new InnerButtonHandler());
```

pag. 422 – par. 15.4.2– prima riga dopo il riquadro che riporta linee di codice

```
Si può notare come la proprietà delle classi anonime di vedere
```
in realtà è:
```
Si può notare come la proprietà delle classi innestate di vedere
```

pag. 428 – par. 15.5– penultima riga del box grigio

```
In queste pagine si parlerà delle appleta "al maschile"
```
in realtà è:
```
In queste pagine si parlerà delle appleta "al femminile"
```

pag. 435 – par. 15.6.1– seconda riga di codice

```
f.getContetPane().add(p)
```
in realtà è:
```
f.getContentPane().add(p)
```

pag. 455 - par. 14.4.5 - 4a riga del primo listato

```
Document doc = factory.newDocument();
```
in realtà è:
```
Document doc = factory.newDocumentBuilder().newDocument();
```

pag. 458 - par. 16.3.7 - 2a riga del 3° riquadro di codice

```
for (Iterator<A> i = list.iterator(); i.hasNext(); ) {
```
in realtà è:
```
for (Iterator<N> i = list.iterator(); i.hasNext(); ) {
```

pag. 479 - par. 17.2.2 - 1° riquadro del paragrafo 17.2.2 - Prima riga

```
Le enumerazioni sono trasformate in classi dal compilatore
```
in realtà è:
```
Le enumerazioni sono trasformate in classi dal compilatore.
```

pag. 479 - par. 17.2.2 - 1° riquadro del paragrafo 17.2.2 - Ultima riga

```
ma può implementare interfacce
```
in realtà è:
```
ma può implementare interfacce.
```

pag. 486 - par. 17.2.4 - 3a riga del testo tra i 2 riquadri di codice

```
con il nome dell'enumerazione in seguente modo:
```

in realtà è:

```
con il nome dell'enumerazione nel seguente modo:
```

pag. 496 - par. 18.1.2 - Quart'ultima riga della pagina

```
Infine, il numero "4" specificato dopo il "." iindica che il
```

in realtà è:

```
Infine, il numero "4" specificato dopo il "." indica che il
```

pag. 501 - par. 18.2.1 - 2a riga dell'ultimo riquadro della pagina

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

in realtà è:

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

pag. 511 – par. 19.2– prima riga dopo il riquadro contenente codice

```
La classe AnnotionsPublisher
```

in realtà è:

```
La classe AnnotationsPublisher
```

pag. 516 – par. 19.2.2– ultima riga del riquadro contenente codice

```
alfabeto value()
```

in realtà è:

```
Alfabeto value()
```

pag. 517 – par. 19.2.2– prima riga del riquadro contenente codice

```
enum alfabeto
```

in realtà è:

```
enum Alfabeto
```

pag. 521 – par. 19.3.3– terza riga del primo riquadro contenente codice

```
@Retention(RetentionPolicy.RUNTIME);
```

in realtà è:

```
@Retention(RetentionPolicy.RUNTIME)
```

Errata corrige di “88-203-3658-8_Esercizi e Soluzioni.pdf”

pag. 7 - Esercizio 2.c) - Domanda 1

```
public void metodo ()
{
return 5;
```

```
}
```
in realtà è:
```
public void metodo ()
{
return;
}
```

pag. 10 – Soluzion1 esercizio 2.c) - Risposta 3

La risposta 3 è errata e da non considerarsi, le risposte successive vanno decrementate di un'unità.

pag. 17 - Testo esercizio 4.d) - Domanda 1 - Terza riga

```
cint i = 5;
```
in realtà è:
```
int i = 5;
```

pag. 76 - Esercizio 18.a.10

```
Nel caso in cui si passi un array come varargs al metodo printf() di
java.io.PrintStream, questo verrò trattato non come oggetto singo-
lo, ma come se fossero stati passati ad uno ad uno, ogni suo elemento
```
in realtà è:
```
Nel caso in cui si passi un array come varargs al metodo printf() di
java.io.PrintStream, questo verrà trattato non come oggetto singo-
lo, ma come se fossero stati passati ad uno ad uno, ogni suo elemento
```