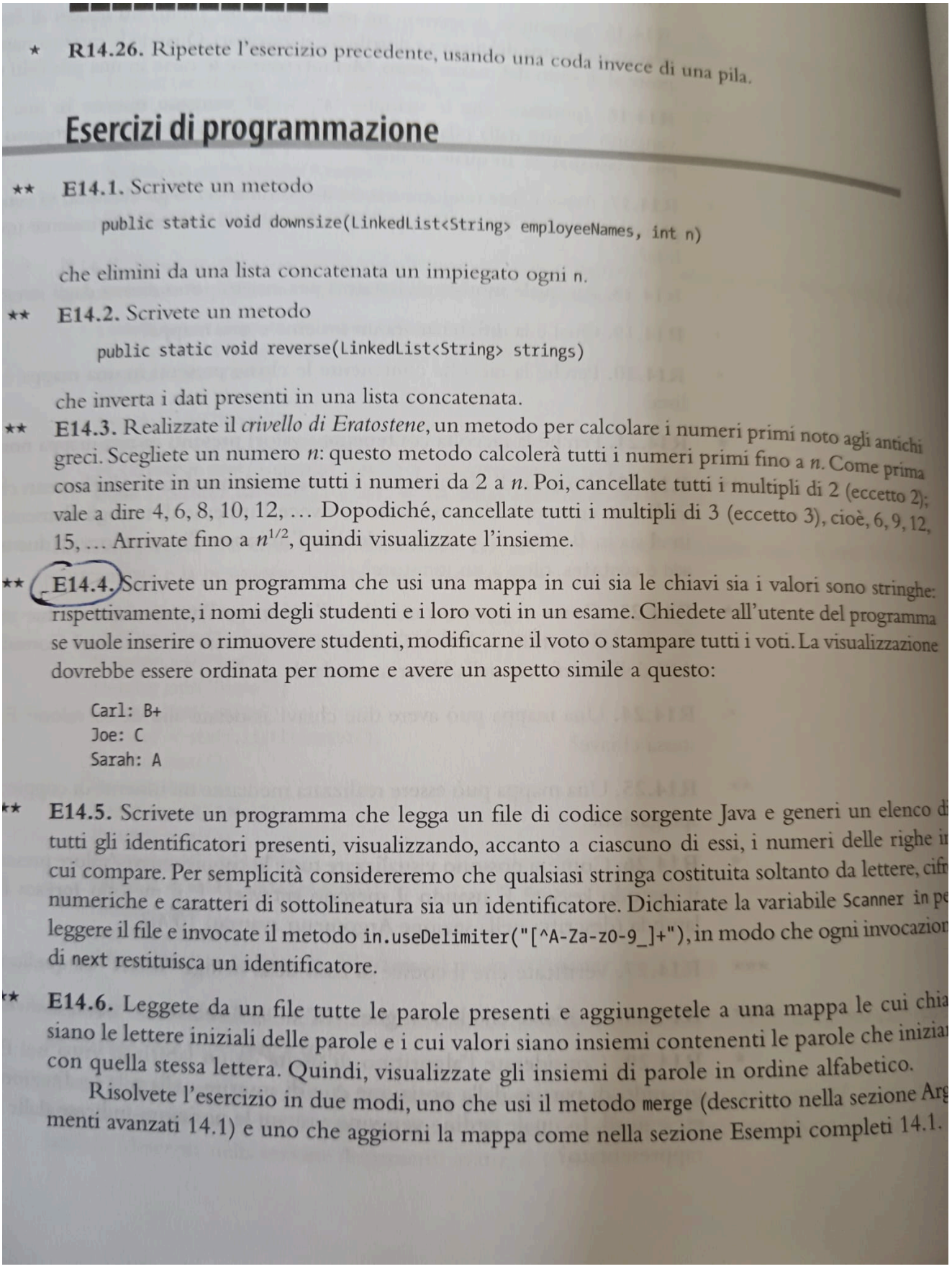
Es. 14.4



★ **R14.26.** Ripetete l'esercizio precedente, usando una coda invece di una pila.

Esercizi di programmazione

★★ **E14.1.** Scrivete un metodo

```
public static void downsize(LinkedList<String> employeeNames, int n)
```

che elimini da una lista concatenata un impiegato ogni `n`.

★★ **E14.2.** Scrivete un metodo

```
public static void reverse(LinkedList<String> strings)
```

che inverta i dati presenti in una lista concatenata.

★★ **E14.3.** Realizzate il *crivello di Eratostene*, un metodo per calcolare i numeri primi noto agli antichi greci. Scegliete un numero n: questo metodo calcolerà tutti i numeri primi fino a n. Come prima cosa inserite in un insieme tutti i numeri da 2 a n. Poi, cancellate tutti i multipli di 2 (eccetto 2); vale a dire 4, 6, 8, 10, 12, … Dopodiché, cancellate tutti i multipli di 3 (eccetto 3), cioè, 6, 9, 12, 15, … Arrivate fino a $n^{1/2}$, quindi visualizzate l'insieme.

★★ **E14.4.** Scrivete un programma che usi una mappa in cui sia le chiavi sia i valori sono stringhe: rispettivamente, i nomi degli studenti e i loro voti in un esame. Chiedete all'utente del programma se vuole inserire o rimuovere studenti, modificarne il voto o stampare tutti i voti. La visualizzazione dovrebbe essere ordinata per nome e avere un aspetto simile a questo:

```
Carl: B+
Joe: C
Sarah: A
```

★★ **E14.5.** Scrivete un programma che legga un file di codice sorgente Java e generi un elenco di tutti gli identificatori presenti, visualizzando, accanto a ciascuno di essi, i numeri delle righe in cui compare. Per semplicità considereremo che qualsiasi stringa costituita soltanto da lettere, cifre numeriche e caratteri di sottolineatura sia un identificatore. Dichiarate la variabile `Scanner` in per leggere il file e invocate il metodo `in.useDelimiter("[^A-Za-z0-9_]+")`, in modo che ogni invocazione di `next` restituisca un identificatore.

★★ **E14.6.** Leggete da un file tutte le parole presenti e aggiungetele a una mappa le cui chiavi siano le lettere iniziali delle parole e i cui valori siano insiemi contenenti le parole che iniziano con quella stessa lettera. Quindi, visualizzate gli insiemi di parole in ordine alfabetico.

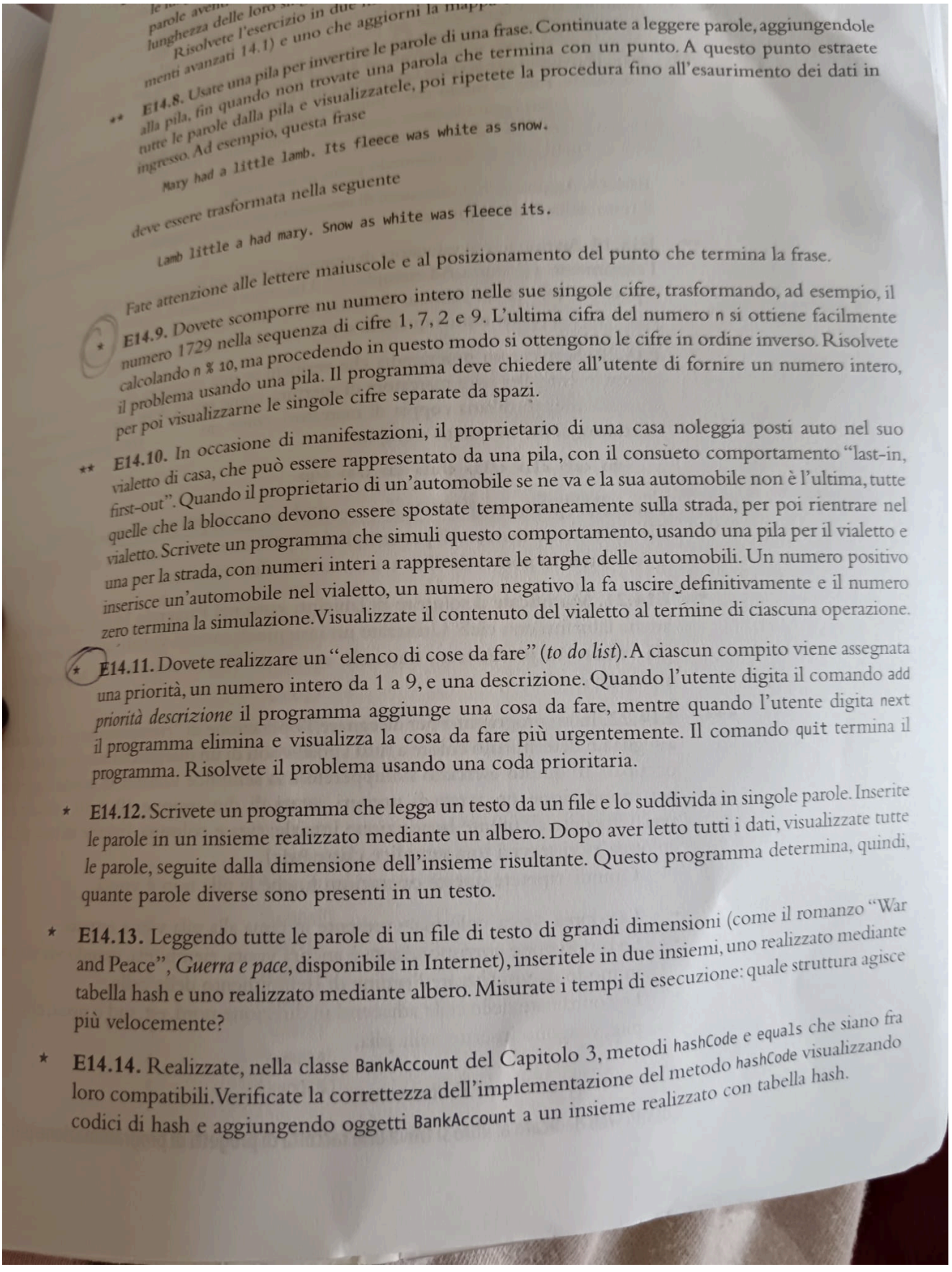
Risolvete l'esercizio in due modi, uno che usi il metodo `merge` (descritto nella sezione Argomenti avanzati 14.1) e uno che aggiorni la mappa come nella sezione Esempi completi 14.1.

```
import java.util.*;
```

```
public class GestioneVotiStudenti {
    private Map<String, String> studentiVoti = new TreeMap<>();
    private Scanner input = new Scanner(System.in);

    public void eseguiProgramma() {
        boolean continua = true;
        while (continua) {
            System.out.println("\nMenu:");
            System.out.println("1. Inserisci studente");
            System.out.println("2. Rimuovi studente");
            System.out.println("3. Modifica voto");
            System.out.println("4. Visualizza tutti i voti");
            System.out.println("5. Esci");
            System.out.print("Scelta: ");

            int scelta = input.nextInt();
            input.nextLine(); // Consumare il newline

            switch (scelta) {
                case 1:
                    inserisciStudente();
                    break;
                case 2:
                    rimuoviStudente();
                    break;
                case 3:
                    modificaVoto();
                    break;
                case 4:
                    visualizzaVoti();
                    break;
                case 5:
                    continua = false;
                    break;
                default:
                    System.out.println("Opzione non valida!");
            }
        }
    }

    private void inserisciStudente() {
        System.out.print("Nome dello studente: ");
        String nome = input.nextLine();
        System.out.print("Voto (A, B+, B, C+, C, D, F): ");
        String voto = input.nextLine();
        studentiVoti.put(nome, voto);
        System.out.println("Studente aggiunto con successo!");
    }

    private void rimuoviStudente() {
```

```
        System.out.print("Nome dello studente da rimuovere: ");
        String nome = input.nextLine();
        if (studentiVoti.containsKey(nome)) {
            studentiVoti.remove(nome);
            System.out.println("Studente rimosso con successo!");
        } else {
            System.out.println("Studente non trovato!");
        }
    }

    private void modificaVoto() {
        System.out.print("Nome dello studente da modificare: ");
        String nome = input.nextLine();
        if (studentiVoti.containsKey(nome)) {
            System.out.print("Nuovo voto: ");
            String nuovoVoto = input.nextLine();
            studentiVoti.put(nome, nuovoVoto);
            System.out.println("Voto modificato con successo!");
        } else {
            System.out.println("Studente non trovato!");
        }
    }

    private void visualizzaVoti() {
        if (studentiVoti.isEmpty()) {
            System.out.println("Nessuno studente presente!");
            return;
        }

        System.out.println("\nElenco voti:");
        for (Map.Entry<String, String> entry : studentiVoti.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }

    public static void main(String[] args) {
        GestioneVotiStudenti programma = new GestioneVotiStudenti();
        programma.eseguiProgramma();
    }
}
```

Es. 14.9 e 14.11



parole avent
lunghezza delle loro
Risolvete l'esercizio in due
menti avanzati 14.1) e uno che aggiorni la mapp

** **E14.8.** Usate una pila per invertire le parole di una frase. Continuate a leggere parole, aggiungendole alla pila, fin quando non trovate una parola che termina con un punto. A questo punto estraete tutte le parole dalla pila e visualizzatele, poi ripetete la procedura fino all'esaurimento dei dati in ingresso. Ad esempio, questa frase

```
Mary had a little lamb. Its fleece was white as snow.
```

deve essere trasformata nella seguente

```
Lamb little a had mary. Snow as white was fleece its.
```

Fate attenzione alle lettere maiuscole e al posizionamento del punto che termina la frase.

* **E14.9.** Dovete scomporre nu numero intero nelle sue singole cifre, trasformando, ad esempio, il numero 1729 nella sequenza di cifre 1, 7, 2 e 9. L'ultima cifra del numero `n` si ottiene facilmente calcolando `n % 10`, ma procedendo in questo modo si ottengono le cifre in ordine inverso. Risolvete il problema usando una pila. Il programma deve chiedere all'utente di fornire un numero intero, per poi visualizzarne le singole cifre separate da spazi.

** **E14.10.** In occasione di manifestazioni, il proprietario di una casa noleggia posti auto nel suo vialetto di casa, che può essere rappresentato da una pila, con il consueto comportamento "last-in, first-out". Quando il proprietario di un'automobile se ne va e la sua automobile non è l'ultima, tutte quelle che la bloccano devono essere spostate temporaneamente sulla strada, per poi rientrare nel vialetto. Scrivete un programma che simuli questo comportamento, usando una pila per il vialetto e una per la strada, con numeri interi a rappresentare le targhe delle automobili. Un numero positivo inserisce un'automobile nel vialetto, un numero negativo la fa uscire definitivamente e il numero zero termina la simulazione. Visualizzate il contenuto del vialetto al termine di ciascuna operazione.

* **E14.11.** Dovete realizzare un "elenco di cose da fare" (*to do list*). A ciascun compito viene assegnata una priorità, un numero intero da 1 a 9, e una descrizione. Quando l'utente digita il comando `add` *priorità descrizione* il programma aggiunge una cosa da fare, mentre quando l'utente digita `next` il programma elimina e visualizza la cosa da fare più urgentemente. Il comando `quit` termina il programma. Risolvete il problema usando una coda prioritaria.

* **E14.12.** Scrivete un programma che legga un testo da un file e lo suddivida in singole parole. Inserite le parole in un insieme realizzato mediante un albero. Dopo aver letto tutti i dati, visualizzate tutte le parole, seguite dalla dimensione dell'insieme risultante. Questo programma determina, quindi, quante parole diverse sono presenti in un testo.

* **E14.13.** Leggendo tutte le parole di un file di testo di grandi dimensioni (come il romanzo "War and Peace", *Guerra e pace*, disponibile in Internet), inseritele in due insiemi, uno realizzato mediante tabella hash e uno realizzato mediante albero. Misurate i tempi di esecuzione: quale struttura agisce più velocemente?

* **E14.14.** Realizzate, nella classe `BankAccount` del Capitolo 3, metodi `hashCode` e `equals` che siano fra loro compatibili. Verificate la correttezza dell'implementazione del metodo `hashCode` visualizzando codici di hash e aggiungendo oggetti `BankAccount` a un insieme realizzato con tabella hash.

Primo

```java
import java.util.*;

public class ScomposizioneNumero {
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
        System.out.print("Inserisci un numero intero: ");
        int numero = input.nextInt();

        // Uso di una pila per memorizzare le cifre in ordine inverso
        Stack<Integer> pila = new Stack<>();

        // Caso speciale per lo zero
        if (numero == 0) {
            pila.push(0);
        } else {
            // Gestione numeri negativi
            boolean negativo = numero < 0;
            if (negativo) numero = -numero;

            // Scomposizione del numero nelle sue cifre
            while (numero > 0) {
                int cifra = numero % 10;
                pila.push(cifra);
                numero /= 10;
            }

            // Se il numero era negativo, aggiungiamo il segno
            if (negativo) {
                System.out.print("-");
            }
        }

        // Visualizzazione delle cifre separate da spazi
        System.out.print("Cifre del numero: ");
        while (!pila.isEmpty()) {
            System.out.print(pila.pop() + " ");
        }

        input.close();
    }
}
```

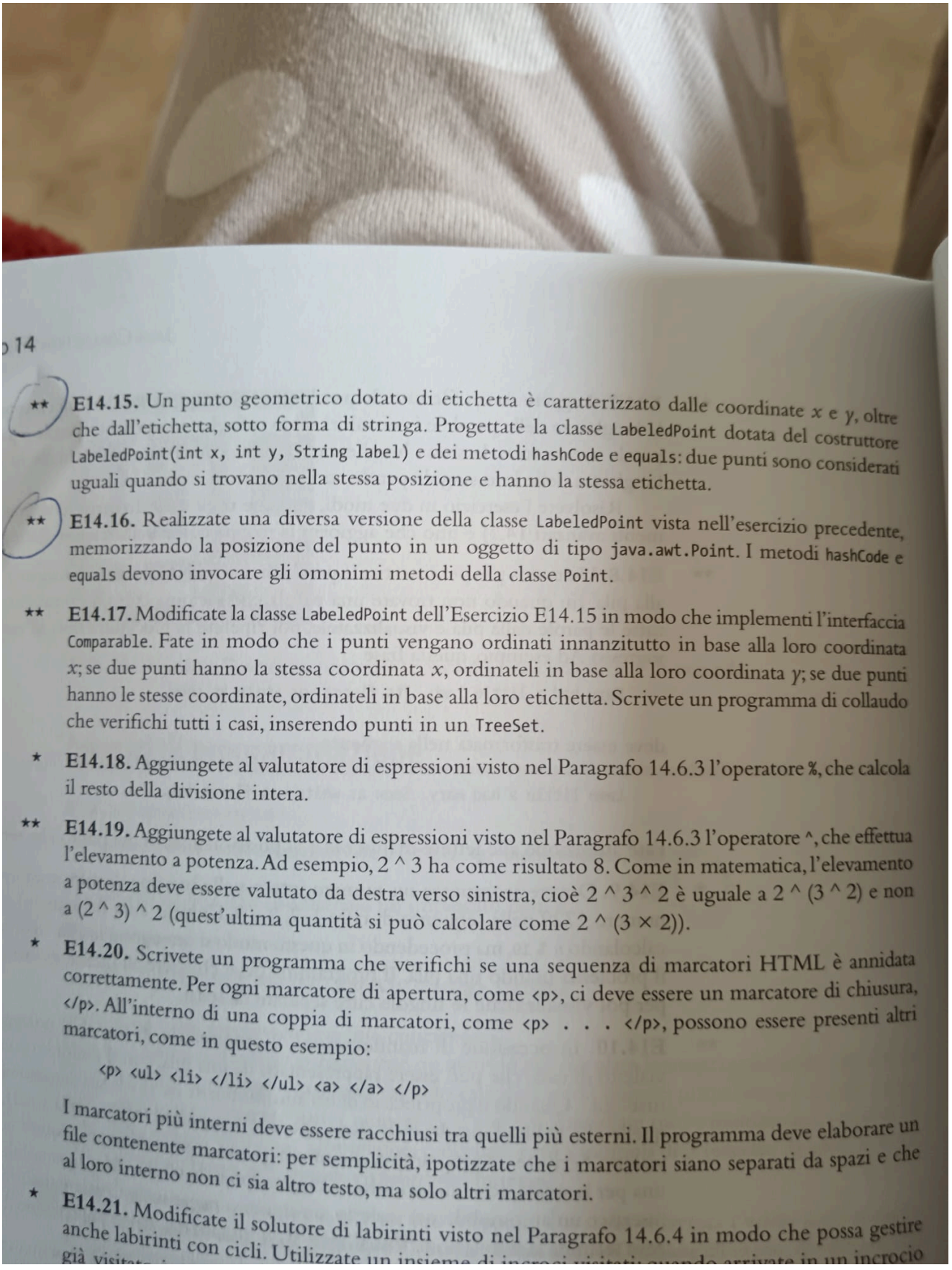
Secondo

```
import java.util.*;

public class TodoList {
    private PriorityQueue<Task> tasks;
    private Scanner input;

    class Task implements Comparable<Task> {
        private int priority;
```

```
        private String description;

        public Task(int priority, String description) {
            this.priority = priority;
            this.description = description;
        }

        @Override
        public int compareTo(Task other) {
            // Ordine crescente per priorità (1 è la più alta)
            return Integer.compare(this.priority, other.priority);
        }

        @Override
        public String toString() {
            return "[Priorità: " + priority + "] " + description;
        }
    }

    public TodoList() {
        tasks = new PriorityQueue<>();
        input = new Scanner(System.in);
    }

    public void run() {
        boolean running = true;

        while (running) {
            System.out.println("\n=== To-Do List ===");
            System.out.println("1. Aggiungi attività (add)");
            System.out.println("2. Prossima attività (next)");
            System.out.println("3. Esci (quit)");
            System.out.print("> ");

            String command = input.nextLine().trim().toLowerCase();

            switch (command) {
                case "1":
                case "add":
                    addTask();
                    break;
                case "2":
                case "next":
                    processNextTask();
                    break;
                case "3":
                case "quit":
                    running = false;
                    System.out.println("Arrivederci!");
                    break;
```

```
                default:
                    System.out.println("Comando non riconosciuto.
Riprova.");
            }
        }
    }

    private void addTask() {
        System.out.print("Priorità (1-9, 1 = più urgente): ");
        int priority;
        try {
            priority = Integer.parseInt(input.nextLine().trim());
            if (priority < 1 || priority > 9) {
                System.out.println("La priorità deve essere un numero tra 1
e 9.");
                return;
            }
        } catch (NumberFormatException e) {
            System.out.println("Inserisci un numero valido.");
            return;
        }

        System.out.print("Descrizione: ");
        String description = input.nextLine().trim();

        if (description.isEmpty()) {
            System.out.println("La descrizione non può essere vuota.");
            return;
        }

        tasks.add(new Task(priority, description));
        System.out.println("Attività aggiunta con successo!");
    }

    private void processNextTask() {
        if (tasks.isEmpty()) {
            System.out.println("Non ci sono attività da fare!");
            return;
        }

        Task nextTask = tasks.poll();
        System.out.println("Attività completata: " + nextTask);
    }

    public static void main(String[] args) {
        TodoList todoList = new TodoList();
        todoList.run();
    }
}
```

Es. 14.15 e 14.16



14

** **E14.15.** Un punto geometrico dotato di etichetta è caratterizzato dalle coordinate x e y, oltre che dall'etichetta, sotto forma di stringa. Progettate la classe `LabeledPoint` dotata del costruttore `LabeledPoint(int x, int y, String label)` e dei metodi `hashCode` e `equals`: due punti sono considerati uguali quando si trovano nella stessa posizione e hanno la stessa etichetta.

** **E14.16.** Realizzate una diversa versione della classe `LabeledPoint` vista nell'esercizio precedente, memorizzando la posizione del punto in un oggetto di tipo `java.awt.Point`. I metodi `hashCode` e `equals` devono invocare gli omonimi metodi della classe `Point`.

** **E14.17.** Modificate la classe `LabeledPoint` dell'Esercizio E14.15 in modo che implementi l'interfaccia `Comparable`. Fate in modo che i punti vengano ordinati innanzitutto in base alla loro coordinata x; se due punti hanno la stessa coordinata x, ordinateli in base alla loro coordinata y; se due punti hanno le stesse coordinate, ordinateli in base alla loro etichetta. Scrivete un programma di collaudo che verifichi tutti i casi, inserendo punti in un `TreeSet`.

* **E14.18.** Aggiungete al valutatore di espressioni visto nel Paragrafo 14.6.3 l'operatore `%`, che calcola il resto della divisione intera.

** **E14.19.** Aggiungete al valutatore di espressioni visto nel Paragrafo 14.6.3 l'operatore `^`, che effettua l'elevamento a potenza. Ad esempio, 2 ^ 3 ha come risultato 8. Come in matematica, l'elevamento a potenza deve essere valutato da destra verso sinistra, cioè 2 ^ 3 ^ 2 è uguale a 2 ^ (3 ^ 2) e non a (2 ^ 3) ^ 2 (quest'ultima quantità si può calcolare come 2 ^ (3 × 2)).

* **E14.20.** Scrivete un programma che verifichi se una sequenza di marcatori HTML è annidata correttamente. Per ogni marcatore di apertura, come `<p>`, ci deve essere un marcatore di chiusura, `</p>`. All'interno di una coppia di marcatori, come `<p> . . . </p>`, possono essere presenti altri marcatori, come in questo esempio:

```
<p> <ul> <li> </li> </ul> <a> </a> </p>
```

I marcatori più interni deve essere racchiusi tra quelli più esterni. Il programma deve elaborare un file contenente marcatori: per semplicità, ipotizzate che i marcatori siano separati da spazi e che al loro interno non ci sia altro testo, ma solo altri marcatori.

* **E14.21.** Modificate il solutore di labirinti visto nel Paragrafo 14.6.4 in modo che possa gestire anche labirinti con cicli. Utilizzate un insieme di incroci visitati; quando arrivate in un incrocio già visitat

Primo

```
/**
 * Rappresenta un punto geometrico dotato di etichetta.
```

```java
 * Due punti sono considerati uguali quando hanno stesse coordinate e stessa etichetta.
 */
public class LabeledPoint {
    private int x;
    private int y;
    private String label;

    /**
     * Costruttore per creare un punto con coordinate ed etichetta.
     */
    public LabeledPoint(int x, int y, String label) {
        this.x = x;
        this.y = y;
        this.label = label;
    }

    /**
     * Restituisce la coordinata x del punto.
     */
    public int getX() {
        return x;
    }

    /**
     * Restituisce la coordinata y del punto.
     */
    public int getY() {
        return y;
    }

    /**
     * Restituisce l'etichetta del punto.
     */
    public String getLabel() {
        return label;
    }

    /**
     * Calcola l'hashcode del punto basato su coordinate ed etichetta.
     */
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + x;
        result = prime * result + y;
        result = prime * result + ((label == null) ? 0 : label.hashCode());
        return result;
    }
```

```java
    /**
     * Verifica se due punti sono uguali.
     * Due punti sono uguali se hanno stesse coordinate e stessa etichetta.
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;

        LabeledPoint other = (LabeledPoint) obj;

        // Verifica delle coordinate
        if (x != other.x) return false;
        if (y != other.y) return false;

        // Verifica dell'etichetta
        if (label == null) {
            if (other.label != null) return false;
        } else if (!label.equals(other.label)) {
            return false;
        }

        return true;
    }

    /**
     * Restituisce una rappresentazione testuale del punto.
     */
    @Override
    public String toString() {
        return "LabeledPoint [x=" + x + ", y=" + y + ", label=" + label +
"]";
    }

    // Test della classe
    public static void main(String[] args) {
        LabeledPoint p1 = new LabeledPoint(5, 10, "Punto A");
        LabeledPoint p2 = new LabeledPoint(5, 10, "Punto A");
        LabeledPoint p3 = new LabeledPoint(5, 10, "Punto B");
        LabeledPoint p4 = new LabeledPoint(3, 7, "Punto A");

        System.out.println("p1 equals p2: " + p1.equals(p2)); // true
        System.out.println("p1 equals p3: " + p1.equals(p3)); // false
(etichetta diversa)
        System.out.println("p1 equals p4: " + p1.equals(p4)); // false
(coordinate diverse)

        System.out.println("hashCode p1: " + p1.hashCode());
```

```
        System.out.println("hashCode p2: " + p2.hashCode());
        System.out.println("hashCode p3: " + p3.hashCode());
    }
}
```

Secondo

```
import java.awt.Point;

/**
 * Versione alternativa della classe LabeledPoint che utilizza
java.awt.Point
 * per memorizzare la posizione del punto.
 */
public class LabeledPoint {
    private Point position; // Utilizziamo java.awt.Point per la posizione
    private String label;

    /**
     * Costruttore per creare un punto con coordinate ed etichetta.
     */
    public LabeledPoint(int x, int y, String label) {
        this.position = new Point(x, y);
        this.label = label;
    }

    /**
     * Restituisce la coordinata x del punto.
     */
    public int getX() {
        return position.x;
    }

    /**
     * Restituisce la coordinata y del punto.
     */
    public int getY() {
        return position.y;
    }

    /**
     * Restituisce l'etichetta del punto.
     */
    public String getLabel() {
        return label;
    }

    /**
```

```java
     * Restituisce l'oggetto Point che rappresenta la posizione.
     */
    public Point getPosition() {
        return new Point(position); // Restituiamo una copia per
incapsulamento
    }

    /**
     * Calcola l'hashcode del punto basato su posizione ed etichetta.
     * Utilizza il metodo hashCode della classe Point.
     */
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + position.hashCode(); // Usa hashCode di
Point
        result = prime * result + ((label == null) ? 0 : label.hashCode());
        return result;
    }

    /**
     * Verifica se due punti sono uguali.
     * Due punti sono uguali se hanno stessa posizione e stessa etichetta.
     * Utilizza il metodo equals della classe Point.
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;

        LabeledPoint other = (LabeledPoint) obj;

        // Verifica della posizione usando equals di Point
        if (!position.equals(other.position)) return false;

        // Verifica dell'etichetta
        if (label == null) {
            if (other.label != null) return false;
        } else if (!label.equals(other.label)) {
            return false;
        }

        return true;
    }

    /**
     * Restituisce una rappresentazione testuale del punto.
     */
```

```
    @Override
    public String toString() {
        return "LabeledPoint [position=" + position + ", label=" + label +
"]";
    }

    // Test della classe
    public static void main(String[] args) {
        LabeledPoint p1 = new LabeledPoint(5, 10, "Punto A");
        LabeledPoint p2 = new LabeledPoint(5, 10, "Punto A");
        LabeledPoint p3 = new LabeledPoint(5, 10, "Punto B");
        LabeledPoint p4 = new LabeledPoint(3, 7, "Punto A");

        System.out.println("p1 equals p2: " + p1.equals(p2)); // true
        System.out.println("p1 equals p3: " + p1.equals(p3)); // false
(etichetta diversa)
        System.out.println("p1 equals p4: " + p1.equals(p4)); // false
(coordinate diverse)

        System.out.println("hashCode p1: " + p1.hashCode());
        System.out.println("hashCode p2: " + p2.hashCode());
        System.out.println("hashCode p3: " + p3.hashCode());

        // Verifica che le coordinate siano accessibili
        System.out.println("Coordinate p1: (" + p1.getX() + ", " + p1.getY()
+ ")");
    }
}
```