Progettazione e Sviluppo di un Sistema Software Complesso

Lezione 10

Scopo della Lezione

- Approfondire la conoscenza con la costruzione e l'uso delle classi in Java;
- Imparare a progettare un sistema complesso;
- Conoscere ed imparare nuove classi della libreria standard di Java;
- Approfondire l'uso dei File.

Descrizione del Problema

Costruire un sistema per la gestione di una biblioteca.

La biblioteca ha punti di prestito e di gestione:

- i bibliotecari nei punti di prestito interagiscono con gli utenti per il prestito dei libri;
- i bibliotecari nei punti di gestione si occupano di gestire i libri (inserimento nuovi arrivi);

La biblioteca può possedere più copie per ogni libro:

- i libri si identificano tramite titolo ed autore (si considera un solo autore per volume);
- gli utenti possono avere al più un libro in prestito;

Analisi degli Oggetti Coinvolti

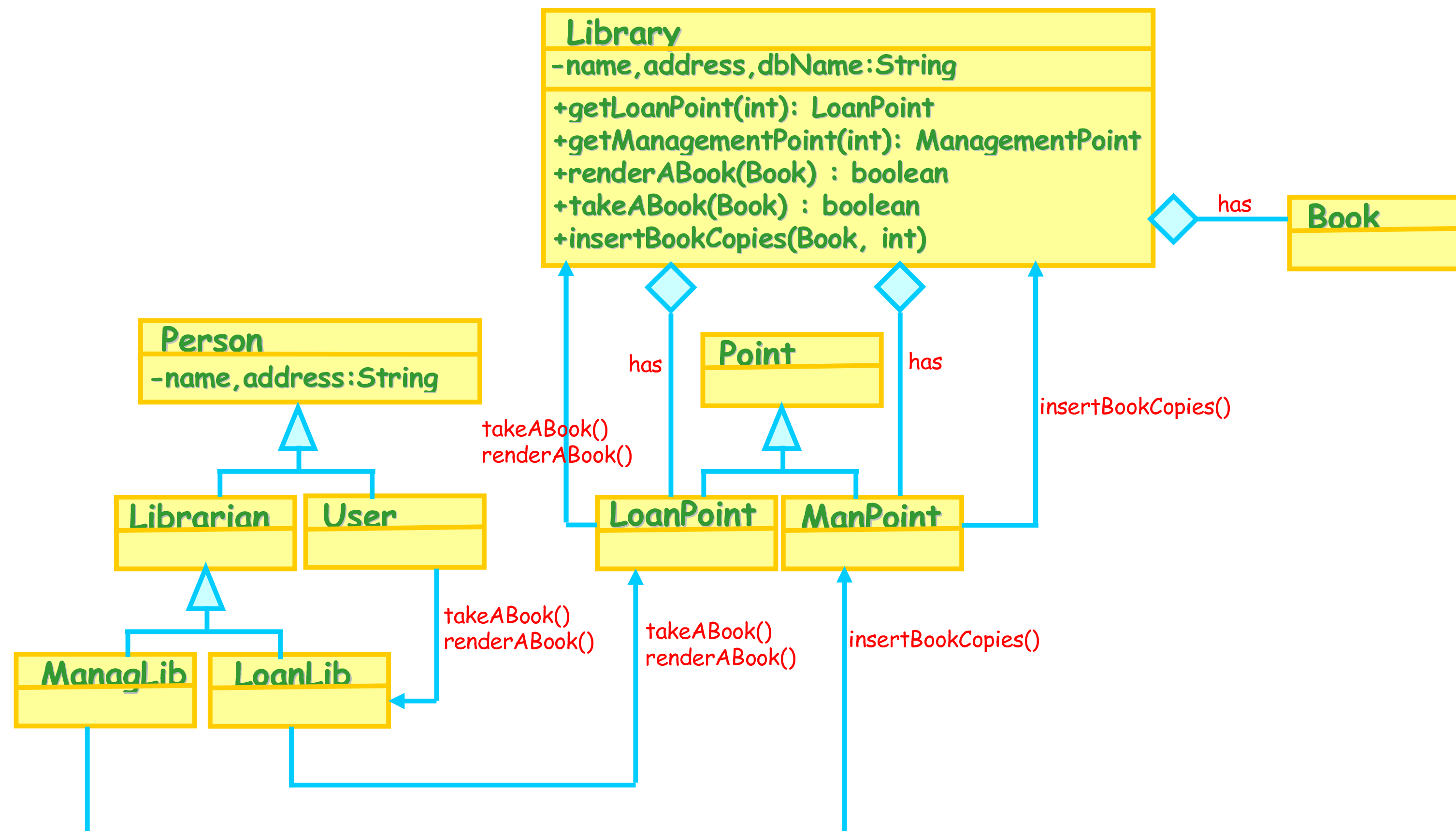
La biblioteca (Library) ha:
- punti di gestione (ManagementPoint); e
- punti di prestito (LoanPoint)

I bibliotecari (Librarian) gestiscono le operazioni dei vari punti (Point), le mansioni dipendono dal punto:
- bibliotecari dei punti di gestione (ManagementLibrarian);
- bibliotecari dei punti di prestito (LoanLibrarian).

Ovviamente ci sono i libri (Book) e gli utenti della biblioteca (User).

Queste sono anche le classi che dovremo andare ad implementare.

Relazioni tra le Classi



La Biblioteca: Main()


```
import java.io.*;

class LibraryManagementSystem {
  public static void main(String[] args) throws IOException {
    BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
    String title, author; ← variabili locali temporanee
    int choice, copies;        ↙ u1 è un utente della biblioteca
    User u1 = new User("Walter", "Piazza la Bomba e Scappa, 7");
    Library l = new Library( "The Library", "Alexandria of Egypt", ← l è la biblioteca
                new LoanPoint[]{
                     new LoanPoint("A1", "Via Dodecaneso 35, GE"),
                     new LoanPoint("A2", "Via Comelico 39/41, MI")},
                new ManagementPoint[]{new ManagementPoint("B1", "Via Bicocca degli
    Arcimboldi 7, MI")},
            "db.txt"); ← file su cui verrà memorizzata la biblioteca
    LoanLibrarian llp = new LoanLibrarian("Cazzola", "", l.getLoanPoint(0), l);
    ManagementLibrarian lmp = new ManagementLibrarian("Malchiodi", "",
    l.getManagementPoint(0), l);
                                              ↖ lmp e llp sono i bibliotecari
    System.out.println("Library Management System:");
    System.out.println("  1. Insert a new book.\n  2. Borrow a book.");
    System.out.println("  3. Render a book.\n  0. Close the Library.");
```

La Biblioteca: Main()


```
    do {
      do {
        System.out.print("Enter Your Choice: ");
        choice = Integer.parseInt(input.readLine());
      } while ((choice <0) || (choice>3));
      switch (choice) {
        case 1: System.out.print("Title: "); title = input.readLine();
                System.out.print("Author: "); author = input.readLine();
                System.out.print("Copies: "); copies = Integer.parseInt(input.readLine());
                lmp.newBook(title, author, copies); break;
        case 2: System.out.print("Title: "); title = input.readLine();
                System.out.print("Author: "); author = input.readLine();
                llp.takeABook(title, author, u1); break;
        case 3: System.out.print("Title: "); title = input.readLine();
                System.out.print("Author: "); author = input.readLine();
                llp.renderABook(title, author, u1); break;
        case 0: l.close();
      }
    } while(choice != 0);
  }
}
```

La Biblioteca: Es. Esecuzione


```
[21:25]cazzola@ulik:esercizi/>java LibraryManagementSystem
Library Management System:
  1. Insert a new book.
  2. Borrow a book.
  3. Render a book.
  0. Close the Library.
Enter Your Choice: 1
Title: Modern Operating Systems
Author: Tanembaum, Andrews
Copies: 3
Enter Your Choice: 1
Title: Understanding Linux
Author: Torvals, Linus
Copies: 16
Enter Your Choice: 2
Title: TeX
Author: Lamport, Leslie
I'm sorry there is no book with such title and author
Enter Your Choice: 2
Title: LaTeX
Author: Lamport, Leslie
This one is the book you have requested
Enter Your Choice: 2
Title: TeX
Author: Knuth, Donald
I'm sorry you already have taken a book
Enter Your Choice: 3
Title: TeX
Author: Knuth, Donald
We have lended you a different book!
Enter Your Choice: 3
Title: LaTeX
Author: Lamport, Leslie
Thank you
Enter Your Choice: 0
```

La Biblioteca: il Database


```
[21:27]cazzola@ulik:esercizi>ll db.txt
-rw-r--r--    1 cazzola  collab        128 Dec  7 21:29 db.txt
[21:27]cazzola@ulik:esercizi>cat db.txt
Understanding Linux|Torvals, Linus|16
LaTeX|Lamport, Leslie|2
Modern Operating Systems|Tanembaum, Andrews|3
TeX|Knuth, Donald|7
```

La Classe Library


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

class Library {
  private String name, address, dbName; ← attributi: nome e indirizzo della biblioteca, e database
  private LoanPoint[] LPs; ← punti di prestito della biblioteca
  private ManagementPoint[] MPs; ← punti di gestione della biblioteca
  private Hashtable books; ← collezione dei libri disponibili in biblioteca

  public Library(String n, String a, LoanPoint[] lps, ManagementPoint[] mps, String db) throws
     FileNotFoundException, IOException { ← costruttore
                         …
  }
  private Point getPoint(Point[] p,int i) { return p[i]; } ↙ selettori per i punti di gestione e prestito
  public LoanPoint getLoanPoint(int i) { return (LoanPoint)getPoint(LPs, i); }
  public ManagementPoint getManagementPoint(int i) { return (ManagementPoint)getPoint(MPs, i); }
  public boolean renderABook(Book b) { … } ← gestione restituzione libri in prestito
  public boolean takeABook(Book b) { … } ← gestione prestito libri
  public void insertBookCopies(Book b, int n) { … } ← gestione inserimento nuovi libri in catalogo
  public void close() throws FileNotFoundException, IOException { … } ←salva il contenuto della biblioteca
}
```

La Classe Hashtable

La collezione di libri della biblioteca è implementata da un'Hashtable<K,V>.

Le Hashtable sono strutture dati che, analogamente agli Array, permettono di:

- memorizzare elementi (di tipo V) identificati da una chiave (di tipo K);
- ritrovarli grazie a quella chiave;

Negli array la chiave è un intero qui un oggetto che implementa equals() e hashCode().

L'accesso agli elementi non avviene tramite [] ma tramite i metodi put(K, V) e get(Object) per accedere agli elementi;

La Classe Library: Persistenza

La biblioteca è persistente, cioè i libri a catalogo saranno in biblioteca ogni volta che faremo partire il programma: sono memorizzati su file.

Sarà il costruttore che leggerà il catalogo.

```
public Library(String n, String a, LoanPoint[] lps, ManagementPoint[] mps, String db)
  throws FileNotFoundException, IOException {
   name = n; address = a; ← inizializzazione dello stato della libreria
   LPs = lps; MPs = mps;
   dbName = db;
   books = new Hashtable(); ← creazione della Hashtable      ↙ apro il file in lettura
   BufferedReader in = new BufferedReader(new FileReader(dbName));
   String line = in.readLine(); ← leggo il file per righe
   StringTokenizer s;
   while (line != null) { ← finché ci sono righe …
     s = new StringTokenizer(line, "|"); ← tokenizzo l'input su |
     books.put(new Book(s.nextToken(), s.nextToken()), new Integer(s.nextToken()));
     line = in.readLine();                      ↖ inserisco il nuovo libro nella biblioteca
   }
}
```

La Class StringTokenizer

La classe **StringTokenizer** permette di spezzare una stringa nelle sue sottocomponenti (dette token).

Il costruttore si occupa della tokenizzazione in base al carattere fornito (nel nostro caso il carattere '|').

I token sono navigabili tramite un interatore ed i relativi metodi (**nextToken()**,**hasMoreTokens()**, etc.)

Es.

```
StringTokenizer st = new StringTokenizer("this is a test");
while (st.hasMoreTokens())
    System.out.println(st.nextToken());
```

Risultato

```
this
is
a
test
```

Class Library: Persistenza 2

Quando la biblioteca chiude, la situazione corrente viene salvata su un file.

```
public void close() throws FileNotFoundException, IOException {
  FileWriter out = new FileWriter(dbName); ← apro il file in scrittura
  Iterator<Map.Entry<Book, Integer>> i =
            books.entrySet().iterator(); ← costruisco un iteratore sui libri
  Book b;
  Map.Entry<Book, Integer> m; ← gli elementi dell'iteratore sono di tipo Map.Entry
  while(i.hasNext()) { ← finché ci sono libri in biblioteca …
    m = i.next(); ← prendo il successivo
    b = m.getKey(); ← estraggo i dati del libro …
    out.write(b.getTitle()+'|'+b.getAuthor()+'|'+m.getValue()+'\n');
  }                                   ↖ … e li scrivo sul file!
  out.close(); ← chiudo il file
}
```

Map.Entry<K, V> descrive le coppie (chiave, valore) inserite in una **Hashtable**

Class Library: Gestione

Compito della biblioteca centrale consiste nel mettere a disposizione dei punti di gestione e di prestito l'elenco aggiornato dei libri disponibili.

Quindi si ha una gestione centralizzata del catalogo che verrà interpellato dai vari punti di gestione e prestito.

I metodi:

- insertBookCopies(), permette di aggiungere copie o nuovi libri;
- takeABook() e renderABook() gestiscono il prestito e la restituzione di un libro.

```
public void insertBookCopies(Book b, int n) {
  Integer copies = books.get(b);
  books.put(b, new Integer((copies != null)?(copies.intValue()+n):n));
}
```

Class Library: Prestiti e Restituzioni


```
public boolean renderABook(Book b) {
  Integer copies = books.get(b);
  if (copies == null) { ← controllo se si sta restituendo un libro della biblioteca
    System.out.println("I'm sorry we don't have books with such title and author!\n You cannot render it.");
    return false;
  } ← se si, rimetto al prestito la copia resa
  int n = copies.intValue();
  books.put(b, new Integer(++n));
  return true;
}
public boolean takeABook(Book b) {
  Integer copies = books.get(b);
  if (copies == null) { ← controllo se la biblioteca possiede il libro richiesto
    System.out.println("I'm sorry there is no book with such title and author");
    return false;
  }
  int n = copies.intValue();
  if (n==0) { ← controllo se c'è ancora una copia del libro richiesto
    System.out.println("I'm sorry all books are already been lended");
    return false;
  } ← se si, ne tolgo una copia dal prestito
  books.put(b, new Integer(--n));
  return true;
}
```

Class Book: hashCode() e equals()

Un libro (**Book**) viene descritto in base a titolo ed autore (suoi attributi) però devo anche rendere la classe adatta ad essere usata come chiave di ricerca nelle **Hashtable** …

```
class Book {
  private String title, author;
  public Book(String t, String a) {title=t; author=a;}
  public String getAuthor() {return author;}
  public String getTitle() {return title;}
```

… implementando il metodo **equals()** in modo che confronti gli attributi dei libri e non il contenitore e …

```
 public boolean equals(Object o) {
    Book b = (Book)o;
    return (((b.title).compareTo(title) == 0) && ((b.author).compareTo(author) == 0));
  }
```

… implementando il metodo **hashCode()**: metodo ereditato da **Object**, associa un numero (univoco) ad ogni istanza della classe (noi sfruttiamo l'implementazione offerta da **String**).

```
 public int hashCode() {return (title+author).hashCode();}
}
```

Restituzione: Delegazione delle Competenze


```
public void renderABook(String t, String a, User u) {
    if (((LoanPoint)point).renderABook(new Book(t, a), u))
        System.out.println("Thank you");
}
```

LoanLibrarian

```
public boolean renderABook(Book b, User u) {
    Book b1 = borrows.get(u);
    if (b1 == null) {
        System.out.println("You cannot … our books!");
        return false;
    }
    if (!b1.equals(b)) {
        System.out.println("We have … book
        return false;
    }
    if (getLibrary().renderABook(b)) {
        borrows.remove(u);
        return true;
    } else return false;
}
```

LoanPoint

```
public boolean renderABook(Book b) {
    Integer copies = books.get(b);
    if (copies == null) {
        System.out.println("I'm sorry … render it.");
        return false;
    }
    int n = copies.intValue();
    books.put(b, new Integer(++n));
    return true;
}
```

Library

Comportamento analogo per takeABook() e insertBookCopies()

Tipologie di Bibliotecari: Separazione dei Compiti


```
class Librarian extends Person {
  protected Point point;
  public Librarian(String n,String a,Point p,Library l) {super(n,a); p.setLibrary(l); point=p;}
}

class LoanLibrarian extends Librarian { ← bibliotecario che lavora nel punto di prestito
  public LoanLibrarian(String n, String a, Point p, Library l) {super(n,a,p,l);}
  public void takeABook(String t, String a, User u) {
    if (((LoanPoint)point).takeABook(new Book(t, a), u))
      System.out.println("This one is the book you have requested");
  }
  public void renderABook(String t, String a, User u) {
    if (((LoanPoint)point).renderABook(new Book(t, a), u)) System.out.println("Thank you");
  }
}

class ManagementLibrarian extends Librarian { ← bibliotecario del punto di gestione
  public ManagementLibrarian(String n, String a, Point p, Library l) {super(n,a,p,l);}
  public void newBook(String t, String a, int n) {
    Book b = new Book(t, a); ((ManagementPoint)point).insertBookCopies(b, n);
  }
}
```

Punti e Punti di Gestione


```
class Point {  ← super classe comune ai punti di gestione e prestito
  protected String name; ← attributi: nome, indirizzo e biblioteca di appartenenza
  protected String address; ↖ protected per facilitarne l'uso alle sottoclassi
  private Library library; ← privato perché settato da fuori la classe
  public void setLibrary(Library l){library = l;}
  public Library getLibrary(){return library;}
}

class ManagementPoint extends Point { ← punto di gestione
  public ManagementPoint(String n, String a) {name = n; address = a;}
    ↓ nei punti di gestione il bibliotecario si limita ad introdurre nuovi libri
  public void insertBookCopies(Book b, int n) {
    getLibrary().insertBookCopies(b, n);
  }
}
```

Punti di Prestito

```
import java.util.*; ← import per le Hashtable

class LoanPoint extends Point { ← punto di prestito
  private Hashtable borrows; ← elenco libri in prestito con relativo utente a cui sono prestati (User,Book)
  public LoanPoint(String n, String a) {name = n; address = a; borrows = new Hashtable();}
     ↓ prestito libri, controlla se l'utente può prendere libri, controlla in biblio se il libro è disponibile,
     ↓ quindi da il libro in prestito aggiornando l'elenco
  public boolean takeABook(Book b, User u) {
    Book b1 = borrows.get(u);
    if (b1 != null) { System.out.println("I'm sorry you already have taken a book"); return false; }
    if (getLibrary().takeABook(b)){ borrows.put(u, b); return true; }
    else return false;
  }
     ↓ restituzione libri, controlla se il libro restituito è quello preso in prestito, quindi aggiorna l'elenco
  public boolean renderABook(Book b, User u) {
    Book b1 = borrows.get(u);
    if (b1 == null) {
      System.out.println("You cannot have such a book! You haven't any of our books!");
      return false;
    }
    if (!b1.equals(b)) { System.out.println("We have lended you a different book!"); return false; }
    if (getLibrary().renderABook(b)){ borrows.remove(u); return true; }
    else return false;
  }
}
```

Utenti

Gli utenti hanno in comune con i bibliotecari le informazioni tipiche di una persona (nome, indirizzo) che quindi vanno raccolte in una classe apposita (**Person**) da cui erediteranno.

```
class Person {
  protected String name;
  protected String address;
  public Person(String n, String a) {name=n; address=a;}
}
```

La classe **User** non aggiunge granché alla classe **Person**

```
class User extends Person {
  public User(String n, String a) {super(n,a);}
}
```

La classe non ridefinisce **equals()** e **hashCode()** perché sfrutta le implementazioni ereditate da **Object**