import java.io.\*;

import java.util.\*;

import java.util.concurrent.locks.\*;

class Book implements Serializable {

private String title;

private String author;

private String ISBN;

private boolean isReserved;

public Book(String title, String author, String ISBN) {

this.title = title;

this.author = author;

this.ISBN = ISBN;

this.isReserved = false;

}

public String getTitle() { return title; }

public String getAuthor() { return author; }

public String getISBN() { return ISBN; }

public boolean isReserved() { return isReserved; }

public void setReserved(boolean reserved) { isReserved = reserved; }

}

class User implements Serializable {

private String name;

private String userID;

private List<Book> borrowedBooks;

private static final int MAX\_BOOKS = 3;

public User(String name, String userID) {

this.name = name;

this.userID = userID;

this.borrowedBooks = new ArrayList<>();

}

public String getName() { return name; }

public String getUserID() { return userID; }

public List<Book> getBorrowedBooks() { return borrowedBooks; }

public boolean canBorrow() { return borrowedBooks.size() < MAX\_BOOKS; }

}

interface ILibrary {

void borrowBook(String ISBN, String userID) throws Exception;

void returnBook(String ISBN, String userID) throws Exception;

void reserveBook(String ISBN, String userID) throws Exception;

Book searchBook(String title);

}

abstract class LibrarySystem implements ILibrary {

protected List<Book> books = new ArrayList<>();

protected List<User> users = new ArrayList<>();

protected Lock lock = new ReentrantLock();

public abstract void addBook(Book book);

public abstract void addUser(User user);

}

class LibraryManager extends LibrarySystem {

@Override

public void addBook(Book book) { books.add(book); }

@Override

public void addUser(User user) { users.add(user); }

@Override

public void borrowBook(String ISBN, String userID) throws Exception {

lock.lock();

try {

User user = users.stream().filter(u -> u.getUserID().equals(userID)).findFirst().orElseThrow(() -> new Exception("User not found"));

if (!user.canBorrow()) throw new Exception("Max books allowed exceeded");

Book book = books.stream().filter(b -> b.getISBN().equals(ISBN)).findFirst().orElseThrow(() -> new Exception("Book not found"));

user.getBorrowedBooks().add(book);

} finally {

lock.unlock();

}

}

@Override

public void returnBook(String ISBN, String userID) throws Exception {

lock.lock();

try {

User user = users.stream().filter(u -> u.getUserID().equals(userID)).findFirst().orElseThrow(() -> new Exception("User not found"));

Book book = user.getBorrowedBooks().stream().filter(b -> b.getISBN().equals(ISBN)).findFirst().orElseThrow(() -> new Exception("Book not borrowed by user"));

user.getBorrowedBooks().remove(book);

} finally {

lock.unlock();

}

}

@Override

public void reserveBook(String ISBN, String userID) throws Exception {

lock.lock();

try {

Book book = books.stream().filter(b -> b.getISBN().equals(ISBN)).findFirst().orElseThrow(() -> new Exception("Book not found"));

if (book.isReserved()) throw new Exception("Book already reserved");

book.setReserved(true);

} finally {

lock.unlock();

}

}

@Override

public Book searchBook(String title) {

return books.stream().filter(b -> b.getTitle().equalsIgnoreCase(title)).findFirst().orElse(null);

}

}

public class Main {

public static void main(String[] args) {

LibraryManager libManager = new LibraryManager();

libManager.addBook(new Book("Java Programming", "James Gosling", "12345"));

libManager.addUser(new User("Alice", "U001"));

Thread t1 = new Thread(() -> {

try {

libManager.borrowBook("12345", "U001");

System.out.println("Alice borrowed Java Programming");

} catch (Exception e) {

System.out.println(e.getMessage());

}

});

Thread t2 = new Thread(() -> {

try {

libManager.returnBook("12345", "U001");

System.out.println("Alice returned Java Programming");

} catch (Exception e) {

System.out.println(e.getMessage());

}

});

t1.start();

t2.start();

}

}