MODUL II

STACK DAN QUEUE

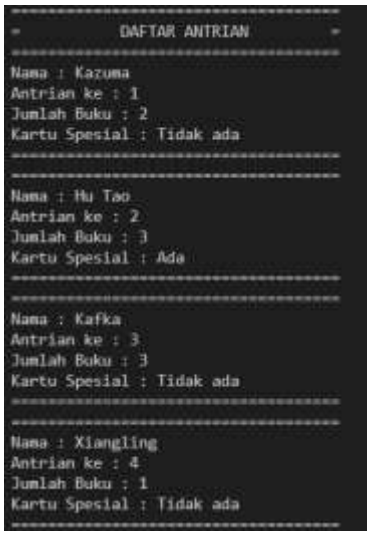
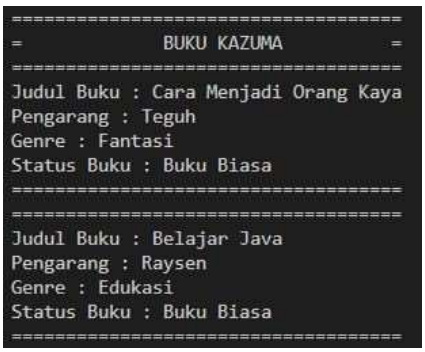
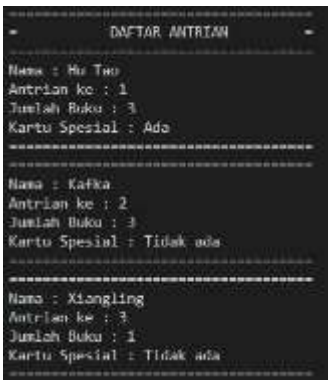
2.1 PERMASALAHAN

2.1.1 Dunia Jojo Bizzare

Dihari berikutnya Yanto kembali menjadi volunteer penjaga perpustakaan sedirian dikarenakan yanti tidak dapat hadir, tidak lama setelah Yanto memulai shift-nya beberapa orang mulai muncul untuk meminjam buku. Sesudah orang orang itu memilih buku yang ingin mereka pinjam, merekapun berbaris di depan meja Yanto dengan barisan seperti gambar disamping.

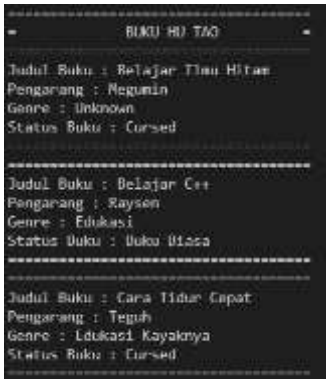
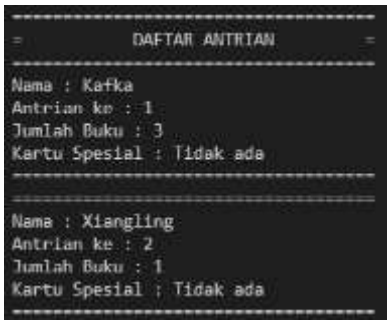
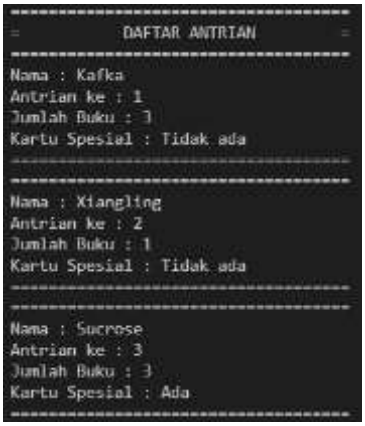
Kazuma maju dan menunjukkan buku buku yang ingin dia pinjam ke pada Yanto, Kazuma meinjam 2 buku yang berjudul "Belajar Java" dan "Cara Menjadi Orang Kaya", Atribut lengkap bukunnya dapat dilihat pada gambar disamping. Setelah Kazuma menyelesaikan peminjamannya, ia pun pergi. Sekarang tinggal tersisa 3 orang dalam antrian, dan antrian setelah Kazuma menjadi antrian pertama.

Note : Buat Antrian Menggunakan Queue dan setiap orang dalam antrian memiliki tumpukan buku (Stack). Note : "Antrian Ke" dan "Jumlah Buku" harus dinamis

Daftar Antrian	Daftar Buku Kazuma	Daftar Antrian Setelah Penghapusan Kazuma
		

Sekarang giliran Hu Tao untuk menunjukkan buku yang ingin dia pinjam kepada Yanto, Hu Tao meminjam 3 buku yang berjudul "Cara Tidur Cepat", "Belajar C++" dan "Belajar Ilmu Hitam", Atribut lengkap bukunnya dapat dilihat pada gambar disamping.

Di perpustkaan ini terdapat dua jenis buku, buku pertama adalah buku biasa dan buku kedua adalah buku terkutuk. Buku terkutuk adalah buku yang memiliki informasi tentang pengetahuan – pengetahuan terlarang jadi untuk meminjam buku terkutuk, peminjam harus memiliki kartu spesial. Karena Hu Tao memiliki kartu spesial, dia dapat meminjam buku terkutuk dan setelah menyelesaikan peminjamannya, Hu Tao langsung pergi. Di antrian sekarang tertinggal 2 orang.

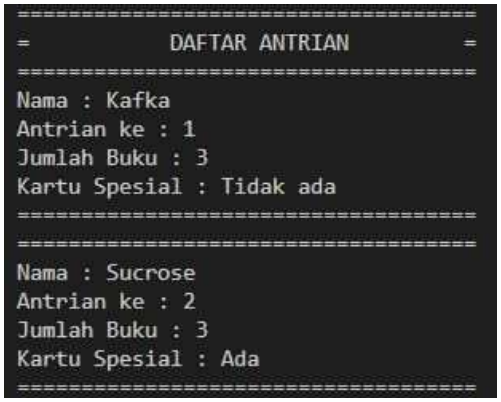
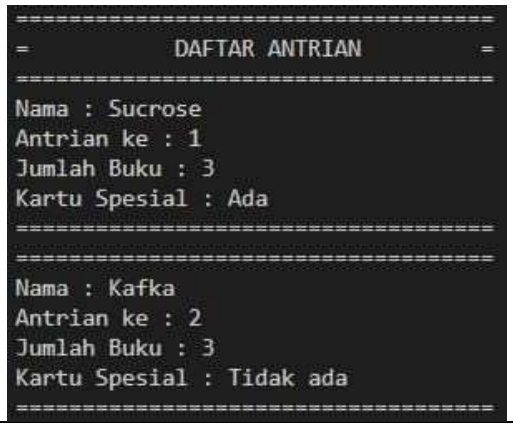
Daftar Buku Hu Tao	Daftar Antrian	Daftar Antrian
		

Xiangling tiba tiba mendapat panggilan dari Zhongli, Yanto bisa mendengar samar samar percakapan xiangling di telefonnya, Yanto mendengar bahwa Gouba sedang mengamuk di Liyue dan butuh bantuan xiangling secepat mungkin. Karena urusan mendesak itu xiangling menyimpan Kembali bukunya dan langsung pergi. Tinggal tersisa dua orang di antrian.

Sucrose juga nampaknya sedang terburu buru, ia pun menanyakan kepada kafka apakah dia boleh bertukar tempat dengan kafka. Kafka setuju, Sucrose dan Kafka pun bertukar tempat.

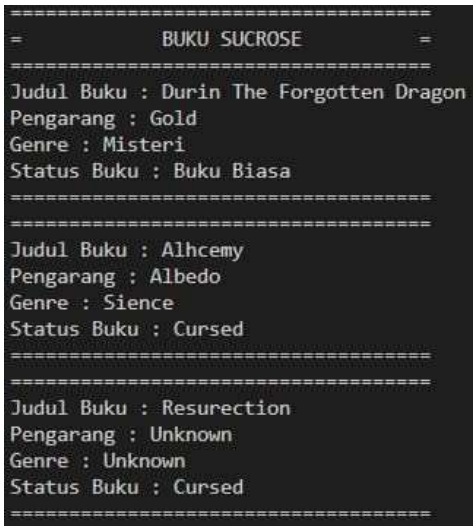
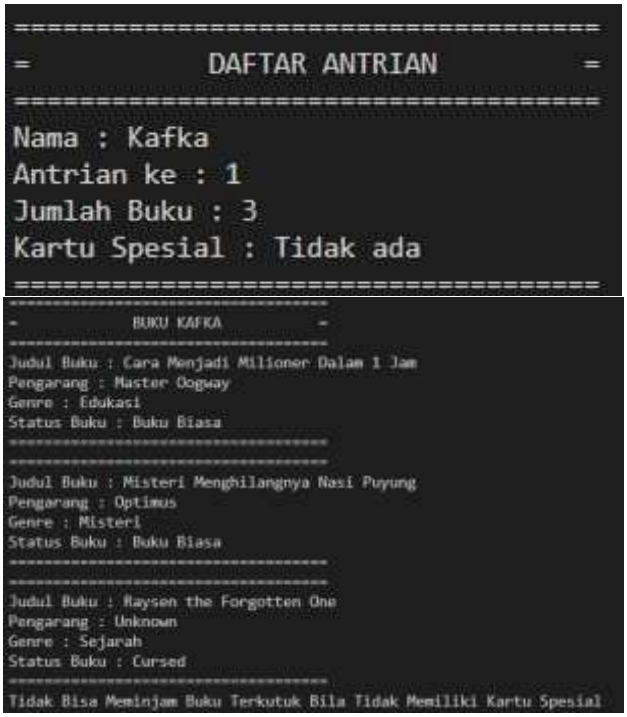
Note : Gunakan temp Stack untuk menghapus node ditengah Stack

Note : Gunakan Method Swap untuk menukar posisi node dalam Queue

Daftar Antrian	Daftar Antrian
	

Sucrose menunjukkan buku-buku yang ingin ia pinjam ke Yanto, Sucrose meminjam 3 buku dengan judul "Resurection", "Alhcemy" dan "Durin The Forgotten Dragon", Atribut lengkap bukunnya dapat dilihat pada gambar disamping.

Setelah Sucrose menyelesaikan peminjamannya, ia pun pergi dengan terburu-buru. Sekarang tinggal tersisa 1 orang dalam antrian yaitu Kafka.

Daftar Buku Sucrose	Daftar Antrian Buku Kafka
	

Yanto memeriksa satu per-satu buku yang ingin di pinjam kafka, dan salah satu buku yang ingin kafka pinjam adalah buku terkutuk. Yanto menanyakan kepada Kafka apakah dia memiliki kartu special, dan kafka menjawab bahwa dia tidak memiliki kartu special. Yanto pun memberi tahu Kafka bahwa Kafka memerlukan kartu special untuk meminjam buku terkutuk. Yanto pun memindahkan buku yang terkutuk ke bagian paling atas tumpukkan dan meminta kafka untuk mengembalikkan buku tersebut. Setelah kafka mengembalikkan buku yang terkutuk, ia menyelesaikan proses peminjaman. Sudah tidak ada orang dalam antrian.

Note : Gunakan method swap untuk menukar posisi buku dalam Stack

Daftar Buku Kafka	Kondisi Setelah Node Dihapus	Daftar Antrian
		

2.2 HASIL PERCOBAAN

2.2.1 Penjaga Perpustakaan

1. Algoritma
    a. Start
    b. Buat class `Book` untuk menyimpan atribut buku seperti judul, tipe (biasa atau terkutuk), dan status kartu spesial.
    c. Buat class `Character` untuk deklarasi atribut setiap karakter dalam antrian, termasuk nama dan tumpukan buku yang ingin dipinjam (menggunakan Stack).
    d. Buat class `Queue` untuk mengelola antrian karakter berupa deklarasikan metode untuk menambah karakter baru ke antrian.
    e. Buat metode untuk menghapus karakter dari antrian setelah selesai meminjam buku. Lalu buat metode untuk menukar posisi karakter di antrian.
    f. Buat class `Library` untuk menyimpan daftar antrian karakter. Jika antrian kosong, tambahkan karakter baru sebagai head dan tail. Jika tidak kosong, tambahkan karakter baru di akhir antrian.
    g. Traverse dari head antrian untuk menemukan karakter yang ingin dihapus. Jika ditemukan, hapus karakter dari antrian.
    h. Saat karakter ingin meminjam buku terkutuk, cek apakah karakter memiliki kartu spesial. Jika ya, lanjutkan peminjaman. Jika tidak, minta karakter untuk mengembalikan buku terkutuk ke posisi teratas dalam tumpukan.
    i. Menyatukan list karakter. Jika list kedua kosong, langsung satukan. Jika tidak, gabungkan kedua list dengan menambah list kedua di belakang karakter yang dicari.
    j. Menukar posisi karakter. Cari dua karakter yang ingin ditukar di antrian. Jika ditemukan, tukar posisi kedua karakter tersebut.
    k. Menukar posisi buku dalam tumpukan Cari dua buku dalam stack dan tukar posisinya menggunakan `temp stack`.
    l. Tampilkan daftar karakter dalam antrian dari head hingga tail dan cetak atribut tiap karakter.
    m. Finish

3. *Source Code*

```
class BookNode {
    String title;
    String author;
    String genre;
    boolean isCursed;
    BookNode next;

    public BookNode(String title, String author, String genre,
boolean isCursed) {
        this.title = title;
        this.author = author;
        this.genre = genre;
        this.isCursed = isCursed;
        this.next = null;
    }
}

class Stack {
    BookNode top;

    public void push(String title, String author, String genre,
boolean isCursed) {
        BookNode newBook = new BookNode(title, author, genre,
isCursed);
        newBook.next = top;
        top = newBook;
    }

    public BookNode pop() {
        if (top == null) throw new IllegalStateException("Stack is
empty");
        BookNode removedBook = top;
        top = top.next;
        return removedBook;
    }

    public void display() {
        BookNode current = top;
        while (current != null) {

System.out.println("====================================");
            System.out.println("Judul Buku : " + current.title);
            System.out.println("Pengarang : " + current.author);
            System.out.println("Genre : " + current.genre);
            System.out.println("Status Buku : " + (current.isCursed
? "Cursed" : "Normal"));

System.out.println("====================================");
            current = current.next;
        }
    }

    // Move the first cursed book found to the top of the stack
    public boolean moveTwoBooks(String title1, String title2) {
        if (top == null || top.next == null) return false;

        BookNode prev1 = null, node1 = top;
        BookNode prev2 = null, node2 = top;
```

```
        // Find the first book with title1
        while (node1 != null && !node1.title.equals(title1)) {
            prev1 = node1;
            node1 = node1.next;
        }

        // Find the second book with title2
        while (node2 != null && !node2.title.equals(title2)) {
            prev2 = node2;
            node2 = node2.next;
        }

        if (node1 == null || node2 == null) return false; // Either
book not found

        // Swap the nodes
        if (prev1 != null) prev1.next = node2; else top = node2;
        if (prev2 != null) prev2.next = node1; else top = node1;

        BookNode temp = node1.next;
        node1.next = node2.next;
        node2.next = temp;

        return true;
    }

    public boolean removeBook(String title) {
        if (top == null) return false; // Stack is empty

        if (top.title.equals(title)) { // Book is at the top of the
stack
            top = top.next;
            return true;
        }

        // Search for the book in the stack
        BookNode current = top;
        while (current.next != null &&
!current.next.title.equals(title)) {
            current = current.next;
        }

        if (current.next == null) {
            return false; // Book not found
        }

        // Remove the book
        current.next = current.next.next;
        return true;
    }


}

class QueueNode {
    String name;
    boolean hasSpecialCard;
    Stack borrowedBooks;
    QueueNode next;

    public QueueNode(String name, boolean hasSpecialCard) {
```

```
        this.name = name;
        this.hasSpecialCard = hasSpecialCard;
        this.borrowedBooks = new Stack();
        this.next = null;
    }
}

class Queue {
    QueueNode front;
    QueueNode rear;

    public void enqueue(String name, boolean hasSpecialCard) {
        QueueNode newUser = new QueueNode(name, hasSpecialCard);
        if (rear == null) {
            front = rear = newUser;
        } else {
            rear.next = newUser;
            rear = newUser;
        }
    }

    public QueueNode dequeue() {
        if (front == null) throw new IllegalStateException("Queue
is empty");
        QueueNode removedUser = front;
        front = front.next;
        if (front == null) rear = null;
        return removedUser;
    }

    public boolean swap(String name1, String name2) {
        if (front == null || name1.equals(name2)) return false;

        // Initialize pointers to find nodes and their previous
nodes
        QueueNode prev1 = null, node1 = front;
        QueueNode prev2 = null, node2 = front;

        // Search for the first node
        while (node1 != null && !node1.name.equals(name1)) {
            prev1 = node1;
            node1 = node1.next;
        }

        // Search for the second node
        while (node2 != null && !node2.name.equals(name2)) {
            prev2 = node2;
            node2 = node2.next;
        }

        // Check if both nodes were found
        if (node1 == null || node2 == null) return false;

        // If node1 is not at the front, link its previous node to
node2
        if (prev1 != null) prev1.next = node2;
        else front = node2; // If node1 is the front, make node2
the new front

        // If node2 is not at the front, link its previous node to
node1
```

```
        if (prev2 != null) prev2.next = node1;
        else front = node1; // If node2 is the front, make node1
the new front

        // Swap the next pointers
        QueueNode temp = node1.next;
        node1.next = node2.next;
        node2.next = temp;

        // Update rear if necessary
        if (node1.next == null) rear = node1;
        else if (node2.next == null) rear = node2;

        return true;
    }

    public void displayQueueOnly() {
        QueueNode current = front;
        int position = 1;

        int bookCount = 0;
        BookNode temp = current.borrowedBooks.top;
        while (temp != null) {
            bookCount++;
            temp = temp.next;
        }

        while (current != null) {

System.out.println("====================================");
            System.out.println("Nama : " + current.name);
            System.out.println("Antrian ke : " + position);
            System.out.println("Jumlah Buku : " + bookCount);
            System.out.println("Kartu Spesial : " +
(current.hasSpecialCard ? "Ya" : "Tidak"));

System.out.println("====================================");

            current = current.next;
            position++;
        }
    }

    public void displayUserBooks(QueueNode user) {

System.out.println("====================================");
        System.out.println("=          BUKU " + user.name + "
=");
        user.borrowedBooks.display();

System.out.println("====================================");
    }
    public boolean removeByName(String name) {
        if (front == null) return false; // Queue is empty

        if (front.name.equals(name)) {
            front = front.next;
            if (front == null) rear = null; // If the queue becomes
empty, set rear to null
            return true;
        }
```

```
        QueueNode current = front;
        while (current.next != null &&
!current.next.name.equals(name)) {
            current = current.next;
        }

        if (current.next == null) {
            return false; // Name not found
        }

        // Remove the node
        current.next = current.next.next;
        if (current.next == null) {
            rear = current; // Update rear if we removed the last
node
        }
        return true;
    }

    public boolean moveTwoBooks(String userName, String title1,
String title2) {
        QueueNode current = front;

        // Find the user with the specified name
        while (current != null) {
            if (current.name.equals(userName)) {
                // Call moveTwoBooks on this user's borrowedBooks
stack
                return current.borrowedBooks.moveTwoBooks(title1,
title2);
            }
            current = current.next;
        }
        return false; // User not found
    }
}

public class tes2 {
    public static void main(String[] args) {
        Queue queue = new Queue();
        // Stack stack = new Stack();

        // Menambahkan pengguna ke dalam queue
        queue.enqueue("Kazuma", false);
        queue.front.borrowedBooks.push("Belajar Java", "Raysen",
"Edukasi", false);
        queue.front.borrowedBooks.push("Cara Menjadi Orang Kaya",
"Teguh", "Fantasi", false);

        queue.enqueue("Hu Tao", true);
        queue.front.next.borrowedBooks.push("Cara Tidur Cepat",
"Teguh", "Edukasi kayaknya", true);
        queue.front.next.borrowedBooks.push("Belajar C++",
"Raysen", "Edukasi", false);
        queue.front.next.borrowedBooks.push("Belajar Ilmu Hitam",
"Megumin", "Unknown", true);

        queue.enqueue("Kafka", false);
        queue.front.next.next.borrowedBooks.push("Raysen the
Forgotton One", "Unknown", "Sejarah", true);
```

```
        queue.front.next.next.borrowedBooks.push("Misteri
Menghilangnya Nasi Puyung", "Optimus", "Misteri", false);
        queue.front.next.next.borrowedBooks.push("Cara Menjadi
Milioner Dalam 1 Jam", "Master Oogway", "Edukasi", false);

        queue.enqueue("Xiangling", false);
        queue.front.next.next.next.borrowedBooks.push("gatau",
"gatau", "gatau", true);


System.out.println("====================================");
        System.out.println("=          DAFTAR ANTRIAN
=");
        queue.displayQueueOnly();

        while (queue.front != null) {
            QueueNode user = queue.dequeue();
            System.out.println("\nMengeluarkan " + user.name + "
dari antrian...");
            queue.displayUserBooks(user);
            break;
        }


System.out.println("====================================");
        System.out.println("=         DAFTAR ANTRIAN           =");
        queue.displayQueueOnly();

        while (queue.front != null) {
            QueueNode user = queue.dequeue();
            System.out.println("\nMengeluarkan " + user.name + "
dari antrian...");
            queue.displayUserBooks(user);
            break;
        }

        queue.enqueue("Sucrose", true);
        queue.rear.borrowedBooks.push("Resurection", "Unknown",
"Unknown", true);
        queue.rear.borrowedBooks.push("Alcemy", "Albedo",
"Science", true);
        queue.rear.borrowedBooks.push("Durin The Forgotton Dragon",
"Gold", "Misteri", false);


System.out.println("====================================");
        System.out.println("=         DAFTAR ANTRIAN           =");
        queue.displayQueueOnly();

        queue.removeByName("Xiangling");


System.out.println("====================================");
        System.out.println("=         DAFTAR ANTRIAN           =");
        queue.displayQueueOnly();

        queue.swap("Kafka", "Sucrose");


System.out.println("====================================");
        System.out.println("=         DAFTAR ANTRIAN           =");
```

```
        queue.displayQueueOnly();

        while (queue.front != null) {
            QueueNode user = queue.dequeue();
            System.out.println("\nMengeluarkan " + user.name + "
dari antrian...");
            queue.displayUserBooks(user);
            break; }


System.out.println("====================================");
        System.out.println("=     DAFTAR ANTRIAN        =");
        queue.displayQueueOnly();

        while (queue.front != null) {
            QueueNode user = queue.front;  // Access the current
front user without dequeuing

            // Check if the user has a special card
            if (!user.hasSpecialCard) {
                QueueNode current = queue.front;
                while (current != null) {
                    System.out.println("\nMenampilkan buku dari " +
current.name + "...");
                    queue.displayUserBooks(current); // Tampilkan
buku tanpa dequeue
                    break; // Pindah ke pengguna berikutnya dalam
antrian   }

                // Check if any book in the user's stack is cursed
                BookNode currentBook = user.borrowedBooks.top;
                boolean hasCursedBook = false;
                while (currentBook != null) {
                    if (currentBook.isCursed) {
                        hasCursedBook = true;
                        break; }
                    currentBook = currentBook.next;
                }

                if (hasCursedBook) {
                    System.out.println("Tidak Bisa meminjam buku");
                    break; }
            }

        }

        queue.moveTwoBooks("Kafka", "Raysen the Forgotton One",
"Cara Menjadi Milioner Dalam 1 Jam");

        QueueNode current = queue.front;
        while (current != null) {
            System.out.println("\nMenampilkan buku dari " +
current.name + "...");
            queue.displayUserBooks(current); // Tampilkan buku
tanpa dequeue
            current = current.next; // Pindah ke pengguna
berikutnya dalam antrian
        }
    }
}
```

2.3 ANALISIS DATA

2.3.1 Penjaga Perpustakaan

```
class BookNode {
    String title;
    String author;
    String genre;
    boolean isCursed;
    BookNode next;

    public BookNode(String title, String author, String genre, boolean
isCursed) {
        this.title = title;
        this.author = author;
        this.genre = genre;
        this.isCursed = isCursed;
        this.next = null;
    }
}
```

Script "`class BookNode {`" adalah sebuah blueprint untuk merepresentasikan buku dalam bentuk node pada struktur data. Setiap BookNode memiliki beberapa atribut, yaitu title (judul buku), `author` (penulis buku), `genre` (jenis atau kategori buku), dan `isCursed` (indikator apakah buku tersebut terkutuk atau tidak, bertipe boolean). Selain itu, terdapat atribut next yang berfungsi sebagai referensi ke node buku berikutnya dalam sebuah daftar berantai (linked list). Atribut ini membantu menghubungkan setiap `BookNode` dengan node lain, memungkinkan penyimpanan beberapa buku dalam urutan tertentu. Konstruktor `BookNode` menerima empat parameter untuk menginisialisasi atribut `title`, `author`, `genre`, dan `isCursed`, sementara atribut next diinisialisasi ke `null` untuk menunjukkan bahwa node ini belum terhubung dengan node lainnya pada awalnya.

```
class Stack {
    BookNode top;

    public void push(String title, String author, String genre, boolean
isCursed) {
        BookNode newBook = new BookNode(title, author, genre,
isCursed);
        newBook.next = top;
        top = newBook;
    }
```

Script "`class Stack {`" merupakan adalah struktur data tumpukan yang menggunakan pendekatan LIFO (Last In, First Out), di mana elemen terakhir yang ditambahkan akan menjadi elemen pertama yang diambil. Pada Stack, atribut `top` berfungsi sebagai penunjuk ke elemen teratas dalam tumpukan, yaitu `BookNode` yang paling akhir ditambahkan.

Method `push` digunakan untuk menambahkan buku baru ke dalam tumpukan. Metode ini menerima empat parameter (`title`, `author`, `genre`, dan `isCursed`) yang digunakan

untuk membuat objek `BookNode` baru. Setelah `newBook` dibuat, atribut `next` dari `newBook` diatur untuk mengarah ke node teratas saat ini (`top`), sehingga `newBook` ditempatkan di bagian atas tumpukan. Kemudian, `top` diperbarui untuk menunjuk ke `newBook`, menjadikannya node teratas yang baru dalam `Stack`. Dengan demikian, setiap pemanggilan `push` akan menambahkan buku di posisi teratas, sementara buku-buku sebelumnya akan terdorong ke bawah.

```
  public BookNode pop() {
    if (top == null) throw new IllegalStateException("Stack is empty");
        BookNode removedBook = top;
        top = top.next;
        return removedBook;
    }

    public void display() {
        BookNode current = top;
        while (current != null) {

System.out.println("====================================");
     System.out.println("Judul Buku : " + current.title);
     System.out.println("Pengarang : " + current.author);
     System.out.println("Genre : " + current.genre);
     System.out.println("Status Buku : " + (current.isCursed ? "Cursed"
: "Normal"));

System.out.println("====================================");
            current = current.next;
        }
    }
```

Script "`public BookNode pop() { dan public void display {`" memungkinkan pengelolaan dan penampilan isi tumpukan `BookNode`. Method `pop` digunakan untuk menghapus dan mengembalikan buku yang berada di bagian atas tumpukan. Pertama, metode ini memeriksa apakah `top` bernilai `null` (tanda bahwa tumpukan kosong). Jika ya, maka `IllegalStateException` dengan pesan "Stack is empty" akan dilemparkan. Jika tumpukan tidak kosong, `pop` menyimpan referensi ke buku teratas dalam variabel `removedBook`, lalu memperbarui `top` ke buku di bawahnya (`top`.next). Dengan begitu, buku teratas sebelumnya berhasil dihapus dan dikembalikan oleh metode ini. Method `display` digunakan untuk menampilkan semua buku dalam tumpukan dari atas ke bawah. Dimulai dari `top`, setiap buku diperiksa, dan atributnya (judul, pengarang, genre, dan status terkutuk atau tidak) dicetak. Metode ini terus menelusuri tumpukan menggunakan referensi `next` hingga mencapai akhir, memastikan semua buku ditampilkan dengan garis pembatas untuk memperjelas pemisahan antara setiap buku.

```
public boolean moveTwoBooks(String title1, String title2) {
        if (top == null || top.next == null) return false;
```

```
        BookNode prev1 = null, node1 = top;
        BookNode prev2 = null, node2 = top;

        // Find the first book with title1
        while (node1 != null && !node1.title.equals(title1)) {
            prev1 = node1;
            node1 = node1.next;
        }

        // Find the second book with title2
        while (node2 != null && !node2.title.equals(title2)) {
            prev2 = node2;
            node2 = node2.next;
        }

        if (node1 == null || node2 == null) return false; // Either
book not found

        // Swap the nodes
        if (prev1 != null) prev1.next = node2; else top = node2;
        if (prev2 != null) prev2.next = node1; else top = node1;

        BookNode temp = node1.next;
        node1.next = node2.next;
        node2.next = temp;

        return true;
    }
```

Script "`public boolean moveTwoBooks(String title1, String title2) {`" digunakan untuk menukar posisi dua buku di dalam tumpukan berdasarkan judulnya. Method ini menerima dua parameter, `title1` dan `title2`, yang merepresentasikan judul buku yang ingin ditukar posisinya. Pertama, metode ini memeriksa apakah tumpukan kosong atau hanya berisi satu buku (dalam kedua kasus tersebut, pertukaran tidak dapat dilakukan), dan mengembalikan false. Selanjutnya, dua referensi node1 dan node2 digunakan untuk mencari buku dengan judul `title1` dan `title2`, sementara `prev1` dan `prev2` menjaga referensi ke node sebelumnya. Metode ini melakukan pencarian node yang cocok melalui iterasi pada tumpukan, dan jika salah satu atau kedua judul tidak ditemukan, `false` akan dikembalikan. Jika kedua buku ditemukan, metode ini melakukan pertukaran posisi kedua node. Jika `node1` bukan node teratas, referensi `prev1`.next diatur untuk mengarah ke `node2` sebaliknya, jika `node1` adalah `top`, `top` diperbarui menjadi `node2`. Proses serupa dilakukan untuk `node2` dan `prev2`. Terakhir, koneksi `next` antara `node1` dan `node2` juga ditukar untuk menyelesaikan pertukaran, dan `true` dikembalikan untuk menandakan keberhasilan pertukaran.

```
BookNode current = top;
        while (current.next != null &&
!current.next.title.equals(title)) {
            current = current.next;
        }
```

```
        if (current.next == null) {
            return false; // Book not found
        }

        // Remove the book
        current.next = current.next.next;
        return true;
    }

}
```

Script "`BookNode current = top`" berfungsi untuk mencari dan menghapus buku tertentu dalam tumpukan `Stack` berdasarkan judulnya. Dimulai dari `top` (buku teratas), variabel `current` melakukan iterasi melalui tumpukan menggunakan perulangan `while`, yang berjalan selama node `current`.next tidak `null` dan judul `current`.next tidak cocok dengan judul buku yang dicari. Dengan kata lain, `current` bergerak ke bawah tumpukan sampai menemukan buku dengan judul yang sesuai atau mencapai akhir tumpukan.

Jika `current`.next bernilai `null` setelah loop, itu berarti buku yang dicari tidak ditemukan di tumpukan, dan `false` akan dikembalikan. Namun, jika `current`.next berisi node buku yang sesuai, node ini akan dihapus dengan mengatur `current`.next untuk menunjuk ke node setelahnya (current.next.next). Ini memutus referensi ke node buku yang akan dihapus, sehingga buku tersebut tidak lagi menjadi bagian dari tumpukan. Setelah penghapusan, `true` dikembalikan untuk menunjukkan bahwa buku telah berhasil ditemukan dan dihapus dari tumpukan.

```
class QueueNode {
    String name;
    boolean hasSpecialCard;
    Stack borrowedBooks;
    QueueNode next;

    public QueueNode(String name, boolean hasSpecialCard) {
        this.name = name;
        this.hasSpecialCard = hasSpecialCard;
        this.borrowedBooks = new Stack();
        this.next = null;
    }
}
```

Script "`class QueueNode {`" dirancang erepresentasikan setiap orang dalam antrian yang ingin meminjam buku. Setiap `QueueNode` memiliki beberapa atribut, yaitu `name` (nama orang), `hasSpecialCard` (boolean yang menunjukkan apakah orang tersebut memiliki kartu spesial untuk meminjam buku terkutuk), `borrowedBooks` (tumpukan buku yang dipinjam oleh orang tersebut, direpresentasikan oleh objek Stack), dan `next`, yang merupakan referensi ke node berikutnya dalam antrian.

Konstruktor `QueueNode` menerima dua parameter, `name` dan `hasSpecialCard`, untuk menginisialisasi nama dan status kartu spesial seseorang. Pada saat objek `QueueNode` dibuat, `borrowedBooks` diinisialisasi sebagai objek Stack baru untuk menyimpan buku yang dipinjam oleh orang tersebut, sementara `next` diatur menjadi `null` sebagai penanda bahwa node ini belum terhubung ke node lainnya dalam antrian. Struktur ini memungkinkan antrian dibuat dalam bentuk daftar berantai (linked list) di mana setiap node merepresentasikan seseorang yang ingin meminjam buku di perpustakaan.

```
class Queue {
    QueueNode front;
    QueueNode rear;

    public void enqueue(String name, boolean hasSpecialCard) {
        QueueNode newUser = new QueueNode(name, hasSpecialCard);
        if (rear == null) {
            front = rear = newUser;
        } else {
            rear.next = newUser;
            rear = newUser;
        }
    }
```

Script "`class Queue {`" merupakan struktur data antrian yang menggunakan pendekatan FIFO (First In, First Out), di mana elemen pertama yang masuk akan menjadi elemen pertama yang keluar. Pada `Queue`, terdapat dua atribut utama: `front` dan `rear`. `front` menunjuk ke node pertama dalam antrian (orang yang paling depan dalam antrean), sedangkan `rear` menunjuk ke node terakhir (orang yang paling belakang dalam antrean). Method `enqueue` digunakan untuk menambahkan seseorang ke bagian belakang antrian. Metode ini menerima dua parameter, `name` dan `hasSpecialCard`, yang digunakan untuk membuat objek `QueueNode` baru bernama `newUser`. Jika `rear` bernilai `null`, ini berarti antrian kosong, sehingga `front` dan `rear` sama-sama diatur ke `newUser`, yang menjadi satu-satunya node dalam antrian. Jika antrian tidak kosong, `newUser` ditambahkan di belakang antrian dengan mengatur rear.next untuk menunjuk ke newUser, kemudian memperbarui `rear` untuk menunjuk ke `newUser` sebagai node terakhir. Dengan cara ini, method `enqueue` memungkinkan penambahan node secara teratur di bagian belakang antrian.

```
public QueueNode dequeue() {
        if (front == null) throw new IllegalStateException("Queue is
empty");
        QueueNode removedUser = front;
        front = front.next;
        if (front == null) rear = null;
        return removedUser;
    }

    public boolean swap(String name1, String name2) {
        if (front == null || name1.equals(name2)) return false;
```

```
        // Initialize pointers to find nodes and their previous nodes
        QueueNode prev1 = null, node1 = front;
        QueueNode prev2 = null, node2 = front;

        // Search for the first node
        while (node1 != null && !node1.name.equals(name1)) {
            prev1 = node1;
            node1 = node1.next;
        }

        // Search for the second node
        while (node2 != null && !node2.name.equals(name2)) {
            prev2 = node2;
            node2 = node2.next;
        }

        // Check if both nodes were found
        if (node1 == null || node2 == null) return false;

        // If node1 is not at the front, link its previous node to
node2
        if (prev1 != null) prev1.next = node2;
        else front = node2; // If node1 is the front, make node2 the
new front

        // If node2 is not at the front, link its previous node to
node1
        if (prev2 != null) prev2.next = node1;
        else front = node1; // If node2 is the front, make node1 the
new front

        // Swap the next pointers
        QueueNode temp = node1.next;
        node1.next = node2.next;
        node2.next = temp;

        // Update rear if necessary
        if (node1.next == null) rear = node1;
        else if (node2.next == null) rear = node2;

        return true;
    }
```

Method `dequeue` dan `swap` dalam `class Queue` berfungsi untuk mengelola node dalam antrian. Method `dequeue` digunakan untuk menghapus dan mengembalikan node yang berada di depan antrian. Metode ini pertama-tama memeriksa apakah `front` bernilai `null`, yang menunjukkan bahwa antrian kosong; jika ya, maka `IllegalStateException` dilemparkan. Jika antrian tidak kosong, node depan disimpan dalam variabel `removedUser`, dan `front` diperbarui untuk menunjuk ke node berikutnya. Jika setelah penghapusan front menjadi null, `rear` juga diatur ke `null`, menandakan bahwa antrian kini kosong. Sebaliknya, method `swap` memungkinkan penukaran posisi dua node dalam antrian berdasarkan nama yang diberikan. Metode ini memulai dengan mencari dua node menggunakan pointer `node1` dan `node2`, serta menyimpan referensi ke node sebelumnya masing-masing. Jika salah satu

atau kedua node tidak ditemukan, `false` akan dikembalikan. Jika kedua node ditemukan, referensi `next` dari node sebelumnya diatur untuk menunjuk ke node yang lain, serta `next` dari kedua node ditukar untuk menyelesaikan pertukaran. Jika penukaran berhasil, metode ini mengembalikan `true`, menunjukkan bahwa posisi kedua node telah berhasil ditukar dalam antrian.

```
public void displayQueueOnly() {
    QueueNode current = front;
    int position = 1;

    int bookCount = 0;
    BookNode temp = current.borrowedBooks.top;
    while (temp != null) {
        bookCount++;
        temp = temp.next;
    }

    while (current != null) {
        System.out.println("====================================");
        System.out.println("Nama : " + current.name);
        System.out.println("Antrian ke : " + position);
        System.out.println("Jumlah Buku : " + bookCount);
        System.out.println("Kartu Spesial : " + (current.hasSpecialCard
? "Ya" : "Tidak"));
        System.out.println("====================================");

        current = current.next;
        position++;
    }
}
```

Script "`public void displayQueueOnly() {` berfungsi untuk menampilkan informasi tentang setiap orang dalam antrian tanpa mencetak detail buku yang dipinjam. Metode ini mulai dengan inisialisasi variabel `current` yang menunjuk pada node `front`, serta variabel `position` untuk melacak urutan antrian. Pertama, method ini menghitung jumlah buku yang dipinjam oleh orang yang berada di depan antrian. Ini dilakukan dengan menelusuri tumpukan borrowedBooks menggunakan variabel sementara temp yang dimulai dari current.`borrowedBooks.top`. Dengan perulangan `while`, jumlah buku dihitung hingga mencapai akhir tumpukan, dengan setiap iterasi menambah nilai `bookCount`.

Setelah menghitung jumlah buku, metode ini kemudian melanjutkan untuk menampilkan informasi untuk setiap node dalam antrian. Dalam perulangan `while`, detail tentang nama orang, posisi dalam antrian, jumlah buku yang dipinjam, dan status kartu spesial (apakah mereka memiliki kartu spesial atau tidak) dicetak. Setiap informasi dipisahkan dengan garis pembatas untuk meningkatkan keterbacaan. Akhirnya, `current` diperbarui untuk menunjuk ke node berikutnya dalam antrian, dan `position` ditingkatkan untuk menyesuaikan urutan saat iterasi dilanjutkan hingga mencapai akhir antrian. Method

ini memberikan gambaran umum tentang status orang-orang yang menunggu di antrian untuk meminjam buku tanpa masuk ke detail tumpukan buku mereka.

```
public void displayUserBooks(QueueNode user) {
    System.out.println("====================================");
    System.out.println("=           BUKU " + user.name + "
=");
    user.borrowedBooks.display();
    System.out.println("====================================");
}
public boolean removeByName(String name) {
    if (front == null) return false; // Queue is empty

    if (front.name.equals(name)) {
        front = front.next;
        if (front == null) rear = null; // If the queue becomes empty,
set rear to null
        return true;
    }

    QueueNode current = front;
    while (current.next != null && !current.next.name.equals(name)) {
        current = current.next;
    }

    if (current.next == null) {
        return false; // Name not found
    }

    // Remove the node
    current.next = current.next.next;
    if (current.next == null) {
        rear = current; // Update rear if we removed the last node
    }
    return true;
}
public boolean moveTwoBooks(String userName, String title1, String
title2) {
        QueueNode current = front;

        // Find the user with the specified name
        while (current != null) {
            if (current.name.equals(userName)) {
                // Call moveTwoBooks on this user's borrowedBooks stack
                return current.borrowedBooks.moveTwoBooks(title1,
title2);
            }
            current = current.next;
        }
        return false; // User not found
    }
}
```

Tiga method dalam class `Queue`, yaitu `displayUserBooks`, `removeByName`, dan `moveTwoBooks`, masing-masing memiliki peran penting dalam mengelola dan menampilkan informasi pengguna dalam antrian. Method `displayUserBooks` digunakan untuk menampilkan daftar buku yang dipinjam oleh pengguna tertentu, dengan mencetak

header dan kemudian memanggil method `display` dari objek `borrowedBooks` untuk menampilkan detail buku yang dipinjam, diakhiri dengan garis pembatas. Sementara itu, method `removeByName` bertanggung jawab untuk menghapus node pengguna dari antrian berdasarkan nama yang diberikan; ia memeriksa apakah antrian kosong dan mencari pengguna baik di depan maupun di tengah antrian untuk dihapus, memperbarui referensi `front` dan `rear` sesuai kebutuhan. Terakhir, method `moveTwoBooks` memungkinkan pemindahan posisi dua buku dalam tumpukan buku yang dipinjam oleh pengguna tertentu, dengan mencari pengguna berdasarkan nama dan memanggil method `moveTwoBooks` pada tumpukan buku mereka. Secara keseluruhan, ketiga method ini berfungsi untuk mengelola interaksi pengguna dengan antrian dan buku yang mereka pinjam, memberikan struktur yang teratur dalam sistem perpustakaan.

```
public class tes2 {
    public static void main(String[] args) {
        Queue queue = new Queue();
        // Stack stack = new Stack();

        // Menambahkan pengguna ke dalam queue
        queue.enqueue("Kazuma", false);
        queue.front.borrowedBooks.push("Belajar Java", "Raysen",
"Edukasi", false);
        queue.front.borrowedBooks.push("Cara Menjadi Orang Kaya",
"Teguh", "Fantasi", false);

        queue.enqueue("Hu Tao", true);
        queue.front.next.borrowedBooks.push("Cara Tidur Cepat",
"Teguh", "Edukasi kayaknya", true);
        queue.front.next.borrowedBooks.push("Belajar C++", "Raysen",
"Edukasi", false);
        queue.front.next.borrowedBooks.push("Belajar Ilmu Hitam",
"Megumin", "Unknown", true);

        queue.enqueue("Kafka", false);
        queue.front.next.next.borrowedBooks.push("Raysen the Forgotton
One", "Unknown", "Sejarah", true);
        queue.front.next.next.borrowedBooks.push("Misteri Menghilangnya
Nasi Puyung", "Optimus", "Misteri", false);
        queue.front.next.next.borrowedBooks.push("Cara Menjadi Milioner
Dalam 1 Jam", "Master Oogway", "Edukasi", false);

        queue.enqueue("Xiangling", false);
        queue.front.next.next.next.borrowedBooks.push("gatau", "gatau",
"gatau", true);

        System.out.println("=====================================");
        System.out.println("=          DAFTAR ANTRIAN           =");
        queue.displayQueueOnly();

        while (queue.front != null) {
            QueueNode user = queue.dequeue();
            System.out.println("\nMengeluarkan " + user.name + " dari
antrian...");
            queue.displayUserBooks(user);
```

```
            break;
        }

        System.out.println("=====================================");
        System.out.println("=          DAFTAR ANTRIAN           =");
        queue.displayQueueOnly();

        while (queue.front != null) {
            QueueNode user = queue.dequeue();
            System.out.println("\nMengeluarkan " + user.name + " dari
antrian...");
            queue.displayUserBooks(user);
            break;
        }

        queue.enqueue("Sucrose", true);
        queue.rear.borrowedBooks.push("Resurection", "Unknown",
"Unknown", true);
        queue.rear.borrowedBooks.push("Alcemy", "Albedo", "Science",
true);
        queue.rear.borrowedBooks.push("Durin The Forgotton Dragon",
"Gold", "Misteri", false);

        System.out.println("=====================================");
        System.out.println("=          DAFTAR ANTRIAN           =");
        queue.displayQueueOnly();

        queue.removeByName("Xiangling");

        System.out.println("=====================================");
        System.out.println("=          DAFTAR ANTRIAN           =");
        queue.displayQueueOnly();

        queue.swap("Kafka", "Sucrose");

        System.out.println("=====================================");
        System.out.println("=          DAFTAR ANTRIAN           =");
        queue.displayQueueOnly();

        while (queue.front != null) {
            QueueNode user = queue.dequeue();
            System.out.println("\nMengeluarkan " + user.name + " dari
antrian...");
            queue.displayUserBooks(user);
            break;
        }

        System.out.println("=====================================");
        System.out.println("=          DAFTAR ANTRIAN           =");
        queue.displayQueueOnly();

        while (queue.front != null) {
            QueueNode user = queue.front;  // Access the current front
user without dequeuing

            // Check if the user has a special card
            if (!user.hasSpecialCard) {
                QueueNode current = queue.front;
                while (current != null) {
                    System.out.println("\nMenampilkan buku dari " +
current.name + "...");
```

```
                    queue.displayUserBooks(current); // Tampilkan buku
tanpa dequeue
                    break; // Pindah ke pengguna berikutnya dalam
antrian
                }

                // Check if any book in the user's stack is cursed
                BookNode currentBook = user.borrowedBooks.top;
                boolean hasCursedBook = false;
                while (currentBook != null) {
                    if (currentBook.isCursed) {
                        hasCursedBook = true;
                        break;
                    }
                    currentBook = currentBook.next;
                }

                if (hasCursedBook) {
                    System.out.println("Tidak Bisa meminjam buku");
                    break;
                }
            }

        }

        queue.moveTwoBooks("Kafka", "Raysen the Forgotton One", "Cara
Menjadi Milioner Dalam 1 Jam");

        QueueNode current = queue.front;
        while (current != null) {
            System.out.println("\nMenampilkan buku dari " +
current.name + "...");
            queue.displayUserBooks(current); // Tampilkan buku tanpa
dequeue
            current = current.next; // Pindah ke pengguna berikutnya
dalam antrian
        }
    }
}
```

Class `tes2` berfungsi sebagai titik masuk untuk menguji implementasi antrian dan tumpukan buku dalam sistem peminjaman buku. Dalam metode `main`, pertama-tama dibuat sebuah objek `Queue`, kemudian pengguna ditambahkan ke dalam antrian menggunakan method `enqueue`. Masing-masing pengguna, seperti Kazuma, Hu Tao, Kafka, dan Xiangling, memiliki buku yang mereka pinjam yang ditambahkan ke tumpukan mereka melalui method `push`. Setelah semua pengguna dan buku dimasukkan, program menampilkan daftar antrian dengan memanggil method `displayQueueOnly`. Selanjutnya, pengguna pertama dikeluarkan dari antrian dengan memanggil method `dequeue`, dan buku yang dipinjam oleh pengguna tersebut ditampilkan. Proses ini diulang untuk pengguna berikutnya, di mana setelah mengeluarkan Xiangling, pengguna baru Sucrose ditambahkan ke dalam antrian. Program juga menguji kemampuan untuk menghapus pengguna berdasarkan nama menggunakan method `removeByName`, menukar posisi dua pengguna

dalam antrian dengan method `swap`, dan menampilkan kembali daftar antrian setelah setiap perubahan. Selain itu, terdapat pengecekan apakah pengguna yang sedang ditampilkan memiliki kartu spesial dan apakah ada buku terkutuk dalam tumpukan mereka; jika ada buku terkutuk dan pengguna tidak memiliki kartu spesial, proses peminjaman dibatalkan. Terakhir, method `moveTwoBooks` dipanggil untuk memindahkan posisi dua buku dalam tumpukan pengguna Kafka. Program berakhir dengan menampilkan semua buku yang dipinjam oleh setiap pengguna dalam antrian, menunjukkan interaksi yang kompleks dalam sistem peminjaman buku.