**Node Class:**

public class Node {  
 Book entry;  
 Node next;  
}

**LinkedList Class:**

public class LinkedList {  
 Node head;  
 private int count;  
 LinkedList(){  
 head=null;  
 count=0;  
 }  
 boolean isListEmpty(){  
 return (count==0);  
 }  
 int listSize(){  
 return count;  
 }  
 void insertLast(Book book){  
 Node newNode=new Node();  
 newNode.entry=book;  
 newNode.next=null;  
 if (head==null){  
 head=newNode;  
 }  
 else {  
 Node n=head;  
 while (n.next!=null){  
 n=n.next;  
 }  
 n.next=newNode;  
 }  
 count++;  
 }  
 void insert(int p,Book book){  
 Node newNode=new Node();  
 newNode.entry=book;  
 newNode.next=null;  
 if (p<0 || p>listSize()){  
 System.*out*.println("Not in the range");  
 }  
 else {  
 Node n=head;  
 for (int i=0;i<p-1;i++){  
 n=n.next;  
 }  
 newNode.next=n.next;  
 n.next=newNode;  
 count++;  
 }  
 }  
 void delete(int p){  
 if (isListEmpty()){  
 System.*out*.println("List is empty.");  
 } else if (p<0 || p>listSize()) {  
 System.*out*.println("Not in the range");  
 } else if (p==0) {  
 head=head.next;  
 count--;  
 }  
 else {  
 Node n=head;  
 Node n1=null;  
 for (int i=0;i<p-1;i++){  
 n=n.next;  
 }  
 n1=n.next;  
 n.next=n1.next;  
 n1=null;  
 count--;  
 }  
 }  
 void traverseList(){  
 Node n=head;  
 while (n.next!=null){  
 System.*out*.println(n.entry);  
 n=n.next;  
 }  
 System.*out*.println(n.entry);  
 }  
}

**LinkedQueue Class:**

public class LinkedQueue {  
 Node front;  
 private Node rear;  
 private int count;  
 LinkedQueue(){  
 front=null;  
 rear=null;  
 count=0;  
 }  
 boolean isQueueEmpty(){  
 return (count==0);  
 }  
 public Book serve(){  
 if (isQueueEmpty()){  
 System.*out*.println("Queue is empty.");  
 return null;  
 }  
 else {  
 Book element=front.entry;  
 front=front.next;  
 count--;  
 return element;  
 }  
 }  
 void append(Book book){  
 Node oldRear=rear;  
 rear=new Node();  
 rear.entry=book;  
 rear.next=null;  
 if (isQueueEmpty()){  
 front=rear;  
 }  
 else {  
 oldRear.next=rear;  
 }  
 count++;  
 }  
}

**Book Class:**

public class Book {  
 private String bookId;  
 private String bookTitle;  
 private int availableCopies;  
 private int numberOfTimesBorrowed;  
 private int numberOfRequests;  
 public Book(String bookId, String bookTitle, int availableCopies, int  
 numberOfTimesBorrowed) {  
 this.bookId = bookId;  
 this.bookTitle = bookTitle;  
 this.availableCopies = availableCopies;  
 this.numberOfTimesBorrowed = numberOfTimesBorrowed;  
 }  
 public Book(String bookId, String bookTitle, int numberOfRequests) {  
 this.bookId = bookId;  
 this.bookTitle = bookTitle;  
 this.numberOfRequests=numberOfRequests;  
 }  
 public String getBookId() {  
 return bookId;  
 }  
 public String getBookTitle() {  
 return bookTitle;  
 }  
 public int getAvailableCopies() {  
 return availableCopies;  
 }  
 public void setAvailableCopies(int availableCopies) {  
 this.availableCopies = availableCopies;  
 }  
 public int getNumberOfTimesBorrowed() {  
 return numberOfTimesBorrowed;  
 }  
 public void setNumberOfTimesBorrowed(int numberOfTimesBorrowed) {  
 this.numberOfTimesBorrowed = numberOfTimesBorrowed;  
 }  
 public int getNumberOfRequests() {  
 return numberOfRequests;  
 }  
 public void setNumberOfRequests(int numberOfRequests) {  
 this.numberOfRequests = numberOfRequests;  
 }  
}

**Main Class:**

public class Main {  
 private static LinkedList *list*=new LinkedList();  
 private static LinkedQueue *queue*=new LinkedQueue();  
 public static void addBook(Book book){  
 *list*.insertLast(book);  
 }  
 public static void processNextRequest(){  
 if (!*queue*.isQueueEmpty()){  
 Book bookRequest=*queue*.serve();  
 Node currentNode=*list*.head;  
 while (currentNode!=null){  
 if  
 (currentNode.entry.getBookId().equals(bookRequest.getBookId())){  
 if (currentNode.entry.getAvailableCopies()>0){  
  
 currentNode.entry.setAvailableCopies(currentNode.entry.getAvailableCopies()  
 - bookRequest.getNumberOfRequests());  
  
 currentNode.entry.setNumberOfTimesBorrowed(currentNode.entry.getNumberOfTimesBorrowed()+1);  
 break;  
 }  
 else {  
 System.*out*.println("No available copies.");  
 }  
 }  
 currentNode=currentNode.next;  
 }  
 }  
 }  
 public static int findAvailableCopies(String bookTitle){  
 Node currentNode=*list*.head;  
 while (currentNode!=null){  
 if (currentNode.entry.getBookTitle().equals(bookTitle)){  
 return currentNode.entry.getAvailableCopies();  
 }  
 currentNode=currentNode.next;  
 }  
 return 0;  
 }  
 public static void mostBorrowedBook(){  
 int maxCount=0;  
 Book maxBook=null;  
 Node currentNode=*queue*.front;  
 while (currentNode!=null){  
 if (currentNode.entry.getNumberOfRequests()>maxCount){  
 maxCount=currentNode.entry.getNumberOfRequests();  
 maxBook=currentNode.entry;  
 }  
 currentNode=currentNode.next;  
 }  
 System.*out*.println("Most borrowed book is: "+maxBook.getBookTitle());  
 }  
 public static void printBookAvailability(String bookTitle){  
 int num=*findAvailableCopies*(bookTitle);  
 if (num>0){  
 System.*out*.println("Book: "+bookTitle+", Available copies: "+num);  
 }  
 else {  
 System.*out*.println("Book not found: "+bookTitle);  
 }  
 }  
 public static void main(String[] args) {  
 *list*.insertLast(new Book("B101","Introduction to Programming",5,0));  
 *list*.insertLast(new Book("B102","History of science",4,0));  
 *list*.insertLast(new Book("B103","The Lord of the Ring",6,0));  
 *list*.insertLast(new Book("B104","Jane Eyre",7,0));  
 *list*.insertLast(new Book("B105","David Copperfield",1,0));  
 *queue*.append(new Book("B101","Introduction to Programming",2));  
 *queue*.append(new Book("B102","History of science",1));  
 *queue*.append(new Book("B103","The Lord of the Ring",4));  
 *queue*.append(new Book("B104","Jane Eyre",0));  
 *queue*.append(new Book("B105","David Copperfield",0));  
 int numberOfCopies=*findAvailableCopies*("Introduction to Programming");  
 System.*out*.println("Number of copies are: "+numberOfCopies);  
 *mostBorrowedBook*();  
 *processNextRequest*();  
 *printBookAvailability*("Introduction to Programming");  
 }  
}

**Output:**

