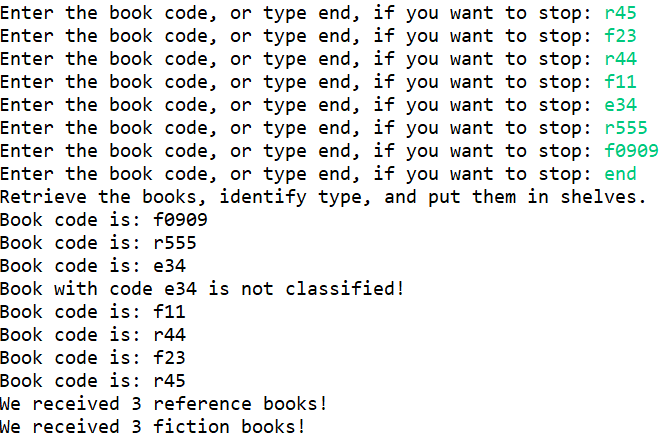
**2**

Main Output



Code

**package** com.tsl.library\_helper;

**import** java.util.Scanner;

/\*\*

\* **@author** EMILIA BUTU

\* version 1.0

\* since 2020-05

\*

\* Student name: Tom Lever

\* Completion date: 05/27/21

\*

\* LibraryHelperDriver.java: demonstrates the use of LibraryHelper class

\*

\* Creates and pushes elements of type String in a stack,

\* Retrieves the elements, and counts the ones that have code starting with R or F

\*

\* Student tasks: complete tasks specified in the file

\*/

**public** **class** LibraryHelperDriver {

**public** **static** **void** main(String[] args) {

/\*\*

\* main is the entry point of the program. The program requests that a user enter books codes, followed by "end". The

\* program prints the book codes, the number of book codes corresponding to reference books, and the number of book

\* codes corresponding to fiction books. There are no preconditions to running this program.

\* **@param** args

\*/

//\*\*\* Task #1: define a libraryStack variable of type StackInterface with element type String

StackInterface<String> libraryStack;

//\*\*\* Task #2: instantiate the libraryStack of LibraryHelper type with 10 elements cap

libraryStack = **new** LibraryHelper<String>(10);

// prepare the Scanner object to enter data from the user

Scanner input = **new** Scanner(System.***in***);

String bookCode="";

//\*\*\* Task #3: create a loop in which you read book codes until user enters "end"

**while**(!bookCode.equalsIgnoreCase("end")) {

System.***out***.print("Enter the book code, or type end, if you want to stop: ");

bookCode=input.next();

**if**(!bookCode.equalsIgnoreCase("end")) {

libraryStack.push(bookCode);

}

}

input.close();

System.***out***.println("Retrieve the books, identify type, and put them in shelves.");

//\*\*\* Task #4: define and initialize variables to count reference and fiction books

**int** refNo=0;

**int** fictionNo=0;

//\*\*\* Task #5: pop the elements off the stack

// count the elements that start with 'R' or 'r' and elements that start with 'F' or 'f'

// if the code starts with other letter, display a message announcing that the code is not classified

**while**(!libraryStack.isEmpty())

{

bookCode=libraryStack.top();

libraryStack.pop();

System.***out***.println("Book code is: " + bookCode);

**if**(bookCode.charAt(0)=='R' ||bookCode.charAt(0)=='r') {

refNo++;

}

**else** {

**if**(bookCode.charAt(0)=='F' ||bookCode.charAt(0)=='f') {

fictionNo++;

}

**else** {

System.***out***.println("Book with code " + bookCode + " is not classified!");

}

}

}

//\*\*\* Task #6: display the number of reference book, and the number of fiction books

System.***out***.println("We received " + refNo + " reference books!");

System.***out***.println("We received " + fictionNo + " fiction books!");

}

}

**package** com.tsl.library\_helper;

/\*\*

\* StackInterface.java

\* Interface for a class that implements a stack of <T>.

\* A stack is a last-in, first-out structure.

\*/

**public** **interface** StackInterface<T> {

**void** push(T element) **throws** StackOverflowException;

// Throws StackOverflowException if this stack is full,

// otherwise places element at the top of this stack.

**void** pop() **throws** StackUnderflowException;

// Throws StackUnderflowException if this stack is empty,

// otherwise removes top element from this stack.

T top() **throws** StackUnderflowException;

// Throws StackUnderflowException if this stack is empty,

// otherwise returns top element of this stack.

**boolean** isEmpty();

// Returns true if this stack is empty, otherwise returns false.

**boolean** isFull();

// Returns true if this stack is full, otherwise returns false.

}

**package** com.tsl.library\_helper;

/\*\*

\* StackOverflowException represents the structure for an exception that occurs when a push onto a stack is requested,

\* but the stack is full.

\* **@author** Tom

\*

\*/

**public** **class** StackOverflowException **extends** RuntimeException {

**public** StackOverflowException() {

/\*\*

\* StackOverflowException() is a conventional zero-parameter constructor for StackOverflowException, which calls

\* Exception's zero-parameter constructor.

\*/

**super**();

}

**public** StackOverflowException(String message) {

/\*\*

\* StackOverflowException(String message) is a one-parameter constructor for StackOverflowException, which passes an

\* error message to Exception's one-parameter constructor with a message parameter.

\* **@param** message

\*/

**super**(message);

}

}

**package** com.tsl.library\_helper;

/\*\*

\* StackUnderflowException represents the structure for an exception that occurs when a pop off a stack is requested,

\* or a look at the top element in the stack is requested, but the stack is empty.

\* **@author** Tom Lever

\* **@since** 05/27/21

\*

\*/

**public** **class** StackUnderflowException **extends** RuntimeException {

**public** StackUnderflowException() {

/\*\*

\* StackUnderflowException() is a conventional zero-parameter constructor for StackUnderflowException, which calls

\* Exception's zero-parameter constructor.

\*/

**super**();

}

**public** StackUnderflowException(String message) {

/\*\*

\* StackUnderflowException(String message) is a one-parameter constructor for StackUnderflowException, which passes an

\* error message to Exception's one-parameter constructor with a message parameter.

\* **@param** message

\*/

**super**(message);

}

}

**package** com.tsl.library\_helper;

/\*\*

\* **@author** EMILIA BUTU

\* version 1.0

\* since 2020-05

\*

\* Student name: Tom Lever

\* Completion date: 05/27/21

\*

\* LibraryHelper.txt: save it as LibraryHelper.java

\* Implements StackInterface using an array to hold the stack elements.

\*

\* Two constructors are provided: one that creates an array of a

\* default size and one that allows the calling program to

\* specify the size.

\*

\* Student tasks: complete tasks specified in the file

\*/

**public** **class** LibraryHelper<T> **implements** StackInterface<T> {

**protected** **final** **int** DEFCAP = 100; // default capacity

**protected** T[] elements; // holds stack elements

**protected** **int** topIndex = -1; // index of top element in stack

**public** LibraryHelper() {

/\*\*

\* LibraryHelper() is a zero-parameter constructor for LibraryHelper that sets this stack's array of elements to a

\* new array of objects of type T, with a number of elements equal to this stack's default capacity.

\*/

**this**.elements = (T[]) **new** Object[**this**.DEFCAP];

}

**public** LibraryHelper(**int** maxSize) {

/\*\*

\* LibraryHelper(int maxSize) is a one-parameter constructor for LibraryHelper that sets this stack's array of

\* elements to a new array of objects of type T, with a number of elements equal to argument maxSize.

\* **@param** maxSize

\*/

//\*\*\* Task #1: implement this constructor

**this**.elements = (T[]) **new** Object[maxSize];

}

**public** **void** push(T element) {

// Throws StackOverflowException if this stack is full,

// otherwise places element at the top of this stack.

//\*\*\* Task #2: implement this method throwing the right exception if necessary

**if** (isFull()) {

**throw** **new** StackOverflowException(

"Exception: push onto a stack of type LibraryHelper failed as the stack was full.");

}

**this**.topIndex++;

**this**.elements[**this**.topIndex] = element;

}

**public** **void** pop() {

// Throws StackUnderflowException if this stack is empty,

// otherwise removes top element from this stack.

//\*\*\* Task #3: implement this method throwing the right exception if necessary

**if** (isEmpty()) {

**throw** **new** StackUnderflowException(

"Exception: pop from a stack of type LibraryHelper failed as the stack was empty.");

}

**this**.elements[**this**.topIndex] = **null**;

**this**.topIndex--;

}

**public** T top() {

// Throws StackUnderflowException if this stack is empty,

// otherwise returns top element of this stack.

//\*\*\* Task #4: implement this method throwing the right exception if necessary

**if** (isEmpty()) {

**throw** **new** StackUnderflowException(

"Exception: top from a stack of type LibraryHelper failed as the stack was empty.");

}

**return** **this**.elements[**this**.topIndex];

}

**public** **boolean** isEmpty() {

// Returns true if this stack is empty, otherwise returns false.

//\*\*\* Task #5: implement this method

**return** (**this**.topIndex == -1);

}

**public** **boolean** isFull() {

// Returns true if this stack is full, otherwise returns false.

//\*\*\* Task #6: implement this method

**return** (**this**.topIndex == (**this**.elements.length - 1));

}

}

**3**