CS 2336 – PROJECT 4 – Redbox Inventory System

Project Due: 11/21 by 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission:

- **The file containing main must be named Main.java. (-5 points)**
- **The project files must be in packages that start with RedBox.* (-5 points)**
- All project deliverables are to be submitted in eLearning
    - Zip the contents of the src directory into a single zipped file
    - Make sure the zipped file has a .zip extension (not .tar, .rar, .7z, etc.) (-5 points)
- Programs must compile and run with Java SE 23.
- Each student is responsible for developing unit test cases to ensure the program works as expected.
- **Type your name and netID in the comments at the top of all files submitted.** (-5 points)

Objectives:

- Implement a binary search tree class in Java
- Utilize a binary search tree
- Create a modular code solution with multi-packages
- Use the Singleton pattern to create and manage the Scanner object
- Use Java Generics to create generic classes and methods

Problem: Redbox needs a program to track inventory and to generate a report for their DVD rental kiosks. Given a log of transactions including renting and returning DVDs as well as adding and removing DVD titles, the program will need to process each transaction and create a report after all transactions have been processed. The generated report will list all DVD titles stored in the kiosk as well as how many of each disc are in the kiosk.

Details:

- The inventory will be held in a generic binary search tree that you will create (-15 points if not)
- Use the DVD title to determine node placement in the tree
- The binary tree will be seeded with an inventory file
- Once seeded, the program will parse a transaction log to update the inventory
- There are five possible transactions:
    - Add
        - Add a new title
            - Create a new node and insert it into the tree
        - Add copies to an existing title
            - Find the title in the tree and increase the number of available copies by the amount listed
    - Remove copies of *an existing title*
        - Find the title in the tree and reduce the number of available copies by the amount listed
        - If number available is zero and no copies are rented out, delete the node from the tree
        - There will not be more copies removed than available.

- Rent a DVD
  - Reduce available amount of *an existing title* by one and increase rented amount by one
- Return a DVD
  - Increase available amount of *an existing title* by one and reduce rented amount by one

Classes:

- Node
  - Members
    - Data field that will hold a Movie object [Title (string – case insensitive), Available (integer), Rented (integer)]
    - Left (node pointer)
    - Right (node pointer)
  - Methods
    - Overloaded constructor
    - Mutators
    - Accessors
- Binary Search Tree
  - Binary Search Tree class must use generics (-5 points if not)
  - Members
    - Root (node pointer)
  - Methods
    - Mutator
    - Accessor
    - Insert (recursive) (-5 points if not)
    - Search (recursive) (-5 points if not)
    - Delete
    - Other methods that are necessary to interact with a binary search tree
      - Remember that methods should be generic to be used on a binary tree regardless of the problem and data type.

User Interface and Input: There is no user interface for this program. Input will be loaded from two files, `inventory.dat` and `transaction.log`. The `inventory.dat` is assumed to be error free and it will be read first by your program and loaded into a binary search tree. Each line (except the last line which may not have a newline character) in `inventory.dat` will be formatted as follows (NOTE: <text> represents a variable value):
`"<title>",<quantity available>, <quantity rented> //May contain space(s) around commas`
After processing the inventory file, begin processing `transaction.log`. Each line of the file should follow one of the following formats (Note: Command is case insensitive):

- `aDd "<title>", <number to add> //Command lines may contain space(s) around commas`
- `ReMove "<title>",<number to remove>`
- `rent "<title>"`
- `retUrn "<title>"`

The transaction file may contain errors due to network disruptions from the main server. For each line in the transaction log, validate that it follows one of the formats listed above. If it is the correct format, process the

transaction. If the <*number to add*>, <*number to remove*>, or the line format is invalid write the line to an error file (as described below). All numbers are expected to be integers. To be valid, the line must follow the format exactly. Also, do not assume that a title in the transaction log will be in the tree.

Output: A file named `error.log` will be created only if any lines of `transaction.log` are invalid. `Error.log` will contain all invalid entries of the transaction file.
At the end of the program, create a formatted report file that contains each title, the number of copies available to rent for that title as well as the number of copies that are currently rented. The titles should be listed in alphabetical order (without the double quotes). The report should be arranged in three formatted columns that line up the data nicely:

- Title
- Copies available
- Copies rented

```
Title                          Available          Rented
-------------------------------------------------------------
Collateral Beauty              2                  2
Doctor Strange                 1                  2
Hacksaw Ridge                  0                  1
```

Write the report to a file named `redbox_kiosk.txt`

Program Flow:

- Program starts in main() method
- Load `inventory.dat`
- Create and populate the Binary Search Tree
- Load `transaction.log`
- Process all transactions
- Output final inventory to file

Unit Testing:
A unit test method is required to test each of the commands listed in the write-up above. These methods will be used by the unit testing framework to test the accuracy of your code.

- `AddTitle(...)`
- `RemoveTitle(...)`
- `RentTitle(...)`
- `ReturnTitle(...)`

Grading:

- Coding standards, style and comments **(20 Points)**
    - Create a multi-package code solution (8 Points)
    - Use Singleton pattern for scanner instance (2 Points)
    - Coding standards and comments (10 Points)
- Unit testing methods x 4, one for each of the methods mentioned above **(12 Points)**
- The rest of the grade will be broken down as follows:
- `AddTitle(...)` **(12 Points)**
- `RemoveTitle(...)` **(12 Points)**
- `RentTitle(...)` **(12 Points)**
- `ReturnTitle(...)` **(12 Points)**
- `Generic Binary Search Tree` **(20 Points)**

Notes:

- No startup code solution will be provided. You are expected to start a new solution from scratch and layer your solution in a way similar to project #3.
- All *.dat files are assumed to be local to the current executable.
- Your program is expected to have basic file validation.
- Your program must use a Singleton pattern for Scanner instance. This ensures that only one instance of the Scanner object is created throughout your program.
- You must implement and utilize a generic binary search tree class. Your class should not inherit any existing Java binary tree class.
- Deliverables: A functioning **program** including **Unit Testing** methods.