Victor Mao

vtm160030

CS 2336.003

Project 4 Pseudocode

Classes:

Node

* Package
  + package BStree;
* Variables
  + private String title;
  + private int available;
  + private int copiesRented;
  + private Node left;
  + private Node right;
* Constructor
  + public Node (String t, int a, int cr, int l, int r)
    - title = t;
    - available = a;
    - copiesRented = cr;
    - left = null;
    - right = null;
* Getter Methods
  + public String getTitle()
    - return title
  + public int getAvailable()
    - return available;
  + public int getCopiesRented()
    - return copiesRented;
  + public Node getLeft()
    - return left;
  + public Node getRight()
    - return right;
* Setter Methods
  + public void setTitle(String t)
    - title = t;
  + public void setAvailable(int a)
    - available = a;
  + public void setCopiesRented(int cr)
    - copiesRented = cr;
  + public void setLeft(Node l)
    - left = l;
  + public void setRight(Node r)
    - right = r;

BSTree

* Package
  + package BSTree;
* Implements
  + implements Comparable
* Variables
  + private Node root;
* Constructor
  + public BSTree()
    - root = null;
* Getter Method
  + public Node getRoot()
    - return root;
* Setter Method
  + public void setRoot(Node r)
    - root = r;
* Other Methods
  + Return the Node with title String t starting at Node current
  + public Node find (Node current, String t) // current, title
    - if (current == null)
      * return;
    - if (current.getTitle().compareTo(t) < 0)
      * return search (current.getLeft())
    - else if (current.getTitle().compareTo(t) > 0)
      * return search (current.getRight())
    - else //current.getTitle().equals(t)
      * return current;
  + Add a new Node into BSTree in order
  + public void addNew(String t, int a, int r) // title, available, rented
    - Node newNode = new Node(t, a, r);
    - if (root == null)
      * root = newNode;
    - else
      * Node parent = findParent(root, root, newNode)
      * if (newNode.getTitle().compareTo(parent.getTitle()) < 0)
        + parent.setLeft(newNode);
      * else
        + parent.setRight(newNode);
  + Return the parent node of the Node goal starting at Node current
  + private Node findParent(Node current, Node parent, Node goal)
    - if (current == null)
      * return;
    - if (goal.getTitle().compareTo(current.getTitle() < 0)
      * return findParent(current.getLeft(), current, goal);
    - else if (goal.getTitle().compareTo(current.getTitle() > 0)
      * return findParent(current.getRIght(), current, goal);
    - else if (goal.getTitle().compareTo(current.getTitle() = 0) // duplicate node
      * return parent;
  + public void add(String t, int num) // title, numberToAdd
    - Node current = find(root, t);
    - current.setAvailable(current.getAvailable() + r);
  + public void remove(String t, int num) // title, numberToRemove
    - Node current = find(root, t);
    - if (current.getAvailable() == 0)
      * deleteNode(current);
    - else
      * current.setAvailable(current.getAvailable() + num);
  + private void deleteNode(Node node)
    - Node parent = findParent(root, root, node);
    - Node current = find(root, node.getTitle());
    - if (current.getLeft() == null)
      * if (parent.getLeft() == current)
        + parent.setLeft(current.getRight());
      * else if (parent.getRight() == current)
        + parent.setRight(current.getRight());
    - else // current.getLeft() != null
      * Node rightMost = findRightMost(node);
      * Node parentOfRightMost = findParent(node, node, rightMost);
      * current.setTitle(rightMost.getTitle());
      * current.setAvailable(rightMost.getAvailable());
      * current.setCopiesRented(rightMost.getCopiesRented());
      * parentOfRightMost.setRight(rightMost.getLeft());
  + private Node findRightMost(Node current)
    - if (current.getRight() == null)
      * return current;
    - else
      * return findRightMost(current.getRight());
  + public void rent(String t) // title
    - Node current = find(root, t);
    - current.setAvailable(current.getAvailable() - 1);
    - current.setCopiesRented(current.getCopiesRented() + 1);
  + public void return(String t) // title
    - Node current = find(root, t);
    - current.setAvailable(current.getAvailable() + 1);
    - current.setCopiesRented(current.getCopiesRented() - 1);

Main.java

* Imports
  + import java.io.\*;
  + import java.util.Scanner;
  + import java.util.ArrayList;
  + import BSTree.\*;
* Variables
  + BSTree inventory;
* Methods
  + public static BSTree readInventory()
    - Scanner scan = new Scanner(new File(“inventory.dat”);
    - BSTree inventory = new BSTree();
    - while (not end of file)
      * scan next line
      * split by commas
      * inventory.addNew(title, available, rented);
    - close scanner
    - return inventory
  + public static void processTransactions()
    - Scanner scan = new Scanner(new File(“transaction.log”));
    - while (not end of file)
      * scan next line
      * if (add)
        + inventory.add(title, numberToAdd);
      * else if (remove)
        + inventory.remove(title, numberToRemove);
      * else if (rent)
        + inventory.rent(title);
      * else if (return)
        + inventory.return(title);
      * else //error
        + print line to “error.log”
    - close scanner
  + public static void writeReport(BSTree inventory)
    - Printwriter pw = new PrintWriter(new File(“redbox\_kiosk.txt”);
    - print column headers
    - ArrayList<Node> nodes = inOrder(inventory.getRoot(), new ArrayList<Node>())
    - for (int i=0; i<nodes.size(); i++)
      * pw print nodes.get(i).getTitle(), nodes.get(i).getAvailable(), nodes.get(i).getCopiesRented()
    - pw.close();
  + private static ArrayList<Node> inOrder(Node current, ArrayList<Node> nodes)
    - if(current == null)
      * return;
    - inOrder(current.getLeft());
    - nodes.add(current);
    - inOrder(current.getRight());
    - return nodes;
* Main
  + BSTree inventory = readInventory();
  + processTransactions();
  + writeReport(inventory);