****

University of Central Punjab

***Data Structure and Algorithms***

***Assignment\_#\_03***

**(BINARY SEARCH TREE)**

**Name:** Abdullah Maqbool

**Reg no:** L1F22BSSE0391

**Section:** P4

**Submitted to:** Ma`am Afifa Hameed

***======================== Scenario\_01 =======================***

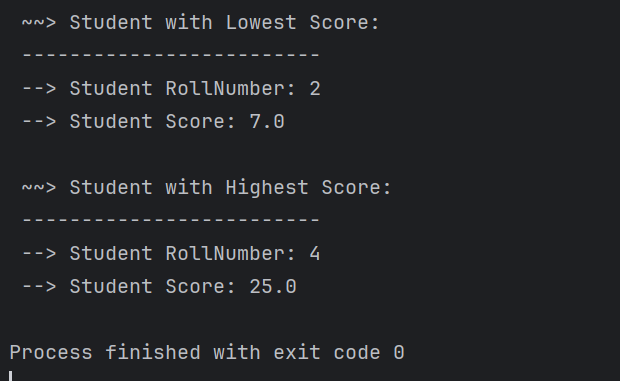
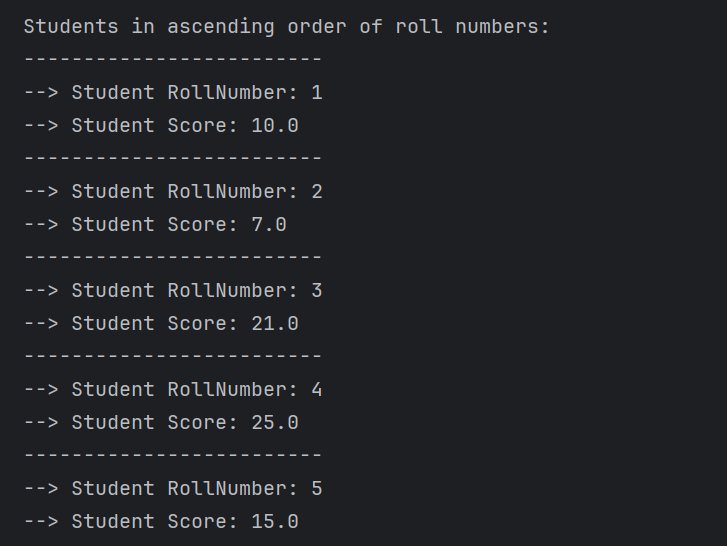
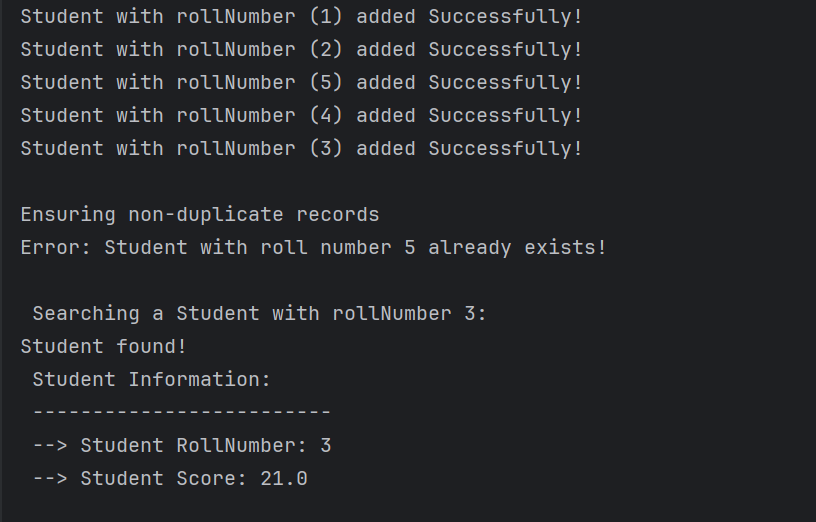
(University Students Record Management)

Java Code:

class Student{  
 int rollNumber;  
 double score;  
 Student Left,Right;  
  
 public Student(int rollNumber, double score) {  
 this.rollNumber = rollNumber;  
 this.score = score;  
 Left = null;  
 Right = null;  
 }  
  
 @Override  
 public String toString(){  
 return " -------------------------\n" +  
 " --> Student RollNumber: " + rollNumber + "\n --> Student Score: " + score;  
 }  
}  
  
class ScoreManagementSystem\_BST{  
 public Student root;  
  
 // Method to insert student data:  
 public Student insertStudentData(Student root,int rollNumber,double score){  
 if (root == null){  
 System.*out*.println("Student with rollNumber ("+rollNumber+") added Successfully!");  
 return new Student(rollNumber,score);  
 } else if (rollNumber < root.rollNumber) {  
 root.Left = insertStudentData(root.Left,rollNumber,score);  
 }else if (rollNumber > root.rollNumber){  
 root.Right = insertStudentData(root.Right,rollNumber,score);  
 }else {  
 System.*out*.println("Error: Student with roll number " + rollNumber + " already exists!");  
 }  
 return root;  
 }  
  
 // Method to Search for a student by roll number and displaying its info:  
 public boolean search(Student root,int rollNumber){  
 if (root == null){  
 System.*out*.println("Student not Found!");  
 return false;  
 }  
 if (rollNumber == root.rollNumber){  
 System.*out*.println("Student found! \n Student Information: \n"+ root);  
 return true;  
 }  
 else if (rollNumber < root.rollNumber){  
 return search(root.Left,rollNumber);  
 }  
 else{  
 return search(root.Right,rollNumber);  
 }  
 }  
  
 // Method to Display all students in ascending order of roll number (using inOrder traversal technique bcz this technique give sorted data):  
 public void inorderTraversal(Student root) {  
 if (root != null) {  
 inorderTraversal(root.Left);  
 System.*out*.println(root);  
 inorderTraversal(root.Right);  
 }  
 }  
  
 // Method to find the student with the lowest score  
 public Student findStudentWithLowestScore(Student root) {  
 if (root == null) return null;  
  
 Student minStudent = root;  
 Student leftMin = findStudentWithLowestScore(root.Left);  
 Student rightMin = findStudentWithLowestScore(root.Right);  
  
 if (leftMin != null && leftMin.score < minStudent.score) {  
 minStudent = leftMin;  
 }  
 if (rightMin != null && rightMin.score < minStudent.score) {  
 minStudent = rightMin;  
 }  
 return minStudent;  
 }  
  
 // Method to find the student with the highest score  
 public Student findStudentWithHighestScore(Student root) {  
 if (root == null) {  
 return null;  
 }  
 Student maxStudent = root;  
 Student leftMax = findStudentWithHighestScore(root.Left);  
 Student rightMax = findStudentWithHighestScore(root.Right);  
  
 if (leftMax != null && leftMax.score > maxStudent.score) {  
 maxStudent = leftMax;  
 }  
 if (rightMax != null && rightMax.score > maxStudent.score) {  
 maxStudent = rightMax;  
 }  
 return maxStudent;  
 }  
}  
  
public class Scenario01 {  
 public static void main(String[] args) {  
 ScoreManagementSystem\_BST system = new ScoreManagementSystem\_BST();  
  
 // adding 5 records in BST(where unique key is the rollNumber):  
 system.root = system.insertStudentData(system.root,1,10);  
 system.root = system.insertStudentData(system.root,2,7);  
 system.root = system.insertStudentData(system.root,5,15);  
 system.root = system.insertStudentData(system.root,4,25);  
 system.root = system.insertStudentData(system.root,3,21);  
  
 System.*out*.println("\nEnsuring non-duplicate records");  
 // ensuring non-duplicate records  
 system.root = system.insertStudentData(system.root,5,14); // it will give error (as rollNumber 5 already added before!)  
  
 System.*out*.println("\n Searching a Student with rollNumber 3: ");  
 // searching a Student and displaying their data:  
 system.search(system.root,3);  
  
 // displaying all students in ascending order of roll numbers:  
 System.*out*.println("\n\n Students in ascending order of roll numbers:");  
 system.inorderTraversal(system.root);  
  
 System.*out*.println("\n ~~> Student with Lowest Score: ");  
 System.*out*.println(system.findStudentWithLowestScore(system.root));  
  
 System.*out*.println("\n ~~> Student with Highest Score: ");  
 System.*out*.println(system.findStudentWithHighestScore(system.root));  
 }  
}

|  |  |
| --- | --- |
| Method | Time Complexity |
| insertStudentData | O(log n) average, O(n) worst |
| search | O(log n) average, O(n) worst |
| inorderTraversal | O(n) |
| findStudentWithLowestScore | O(n) |
| findStudentWithHighestScore | O(n) |

**Output Screenshots:**



***======================== Scenario\_02 =======================***

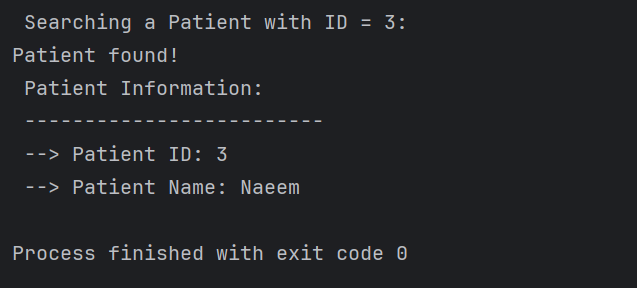
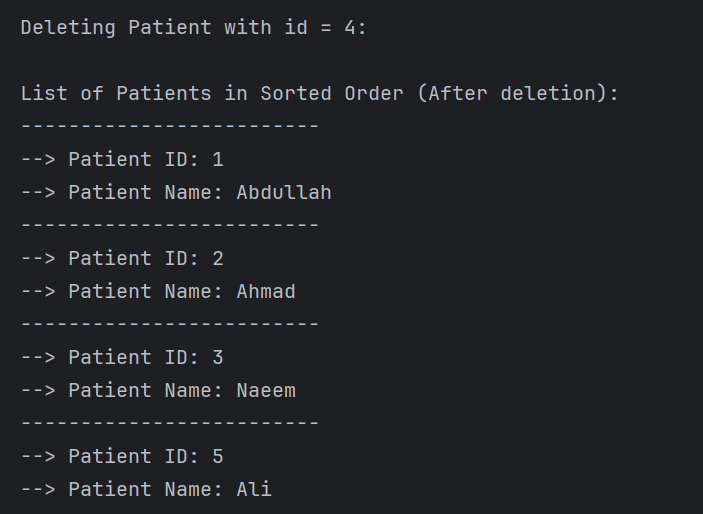
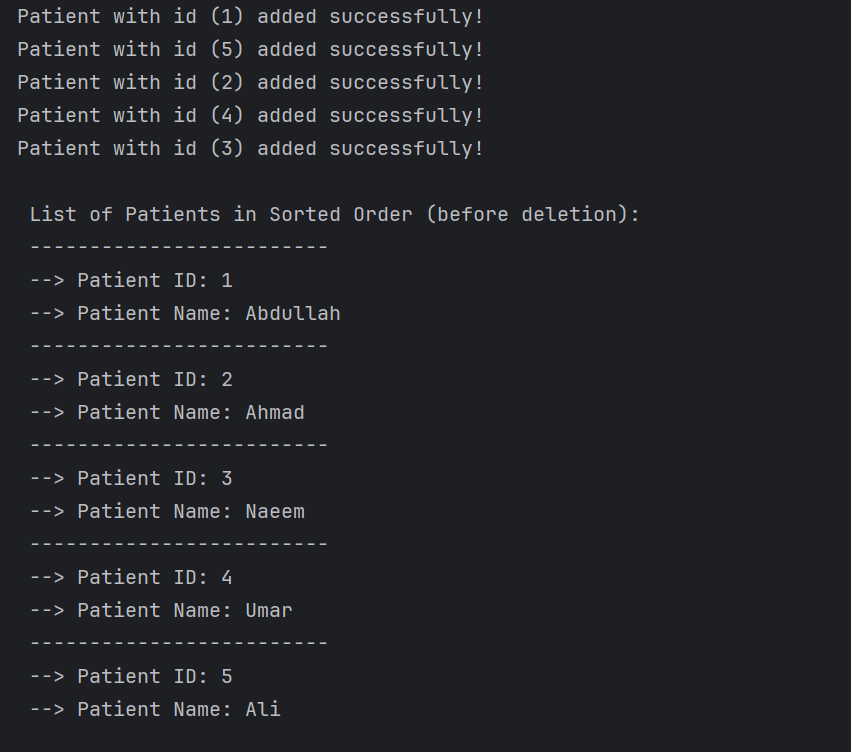
(Patients Record Management)

Java Code:

class Patient{  
 int patientID;  
 String patientName;  
 Patient left,right;  
 public Patient(int patientID, String patientName) {  
 this.patientID = patientID;  
 this.patientName = patientName;  
 left = right = null;  
 }  
  
 @Override  
 public String toString(){  
 return " -------------------------\n" +  
 " --> Patient ID: " + patientID + "\n --> Patient Name: " + patientName;  
 }  
}  
  
class PatientManagementSystem\_BST {  
 public Patient root;  
  
 // Method to insert a Patient  
 public Patient insertPatient(Patient root, int id, String name) {  
 if (root == null) {  
 System.*out*.println("Patient with id (" + id + ") added successfully!");  
 return new Patient(id, name);  
 } else if (id < root.patientID) {  
 root.left = insertPatient(root.left, id, name);  
 } else if (id > root.patientID) {  
 root.right = insertPatient(root.right, id, name);  
 } else {  
 System.*out*.println("Error: Patient with ID " + id + " already exist!");  
 }  
 return root;  
 }  
  
 // method to delete a Patient by its ID  
 public Patient deleteByID(Patient root, int id) {  
 if (root == null) {  
 return null;  
 }  
 if (id < root.patientID) {  
 root.left = deleteByID(root.left, id);  
 } else if (id > root.patientID) {  
 root.right = deleteByID(root.right, id);  
 } else {  
 // Node with only one child or no child  
 if (root.left == null) {  
 return root.right;  
 } else if (root.right == null) {  
 return root.left;  
 }  
  
 // Node with two children: get inorder successor (smallest in the right subtree)  
 root.patientID = getMinValue(root.right);  
  
 // Delete the inorder successor  
 root.right = deleteByID(root.right, root.patientID);  
 }  
 return root;  
 }  
  
 int getMinValue(Patient root) {  
 int min = root.patientID;  
 while (root.left != null) {  
 min = root.left.patientID;  
 root = root.left;  
 }  
 return min;  
 }  
  
 // Method to Search for a Patient by its ID and displaying its info:  
 public boolean searchByID(Patient root,int id){  
 if (root == null){  
 System.*out*.println("Patient not Found!");  
 return false;  
 }  
 if (id == root.patientID){  
 System.*out*.println("Patient found! \n Patient Information: \n"+ root);  
 return true;  
 }  
 else if (id < root.patientID){  
 return searchByID(root.left,id);  
 }  
 else{  
 return searchByID(root.right,id);  
 }  
 }  
  
 // Method to Display all students in ascending order of PatientIDs (using inOrder traversal technique bcz this technique give sorted data):  
 public void inorderTraversal(Patient root) {  
 if (root != null) {  
 inorderTraversal(root.left);  
 System.*out*.println(root);  
 inorderTraversal(root.right);  
 }  
 }  
}  
  
  
public class Scenario02 {  
 public static void main(String[] args) {  
 PatientManagementSystem\_BST system = new PatientManagementSystem\_BST();  
  
 // inserting 5 Patient Records:  
 system.root = system.insertPatient(system.root,1,"Abdullah");  
 system.root = system.insertPatient(system.root,5,"Ali");  
 system.root = system.insertPatient(system.root,2,"Ahmad");  
 system.root = system.insertPatient(system.root,4,"Umar");  
 system.root = system.insertPatient(system.root,3,"Naeem");  
  
  
 System.*out*.println("\n List of Patients in Sorted Order (before deletion):");  
 system.inorderTraversal(system.root);  
  
 System.*out*.println("\n Deleting Patient with id = 4:");  
 system.root = system.deleteByID(system.root,4);  
  
 System.*out*.println("\n List of Patients in Sorted Order (After deletion):");  
 system.inorderTraversal(system.root);  
  
 System.*out*.println("\n Searching a Patient with ID = 3:");  
 system.searchByID(system.root,3);  
 }  
}

|  |  |
| --- | --- |
| Method | Time Complexity |
| insertPatient | O(log n) average, O(n) worst |
| deleteByID | O(log n) average, O(n) worst |
| searchByID | O(log n) average, O(n) worst |
| inorderTraversal | O(n) |

Output Screenshots:



***======================== Scenario\_03 =======================***

(“Book Store” Record Management)

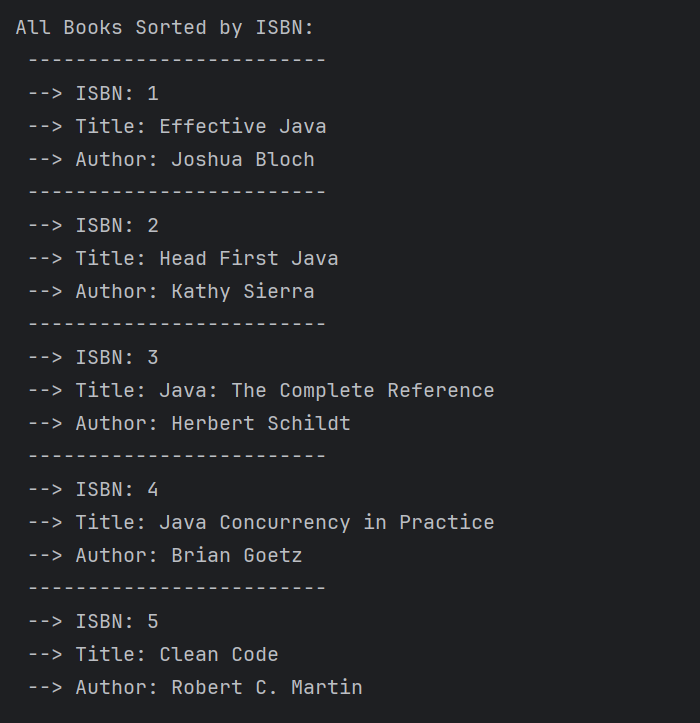
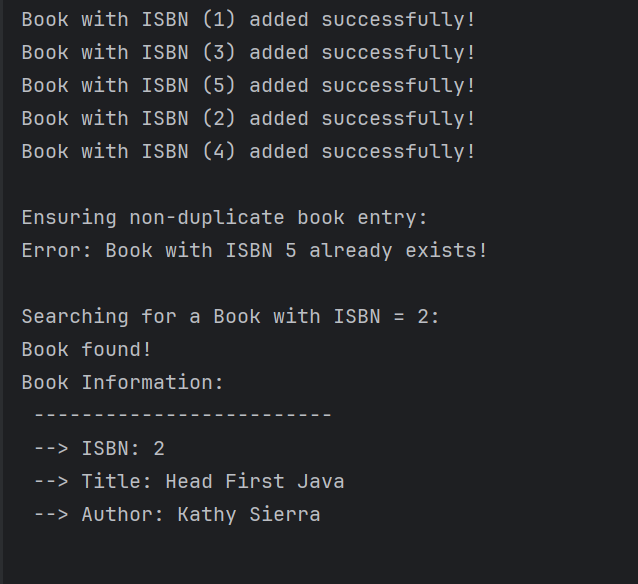
Java Code:

class Book {  
 int isbn;  
 String title, author;  
 Book Left, Right;  
  
 public Book(int isbn, String title, String author) {  
 this.isbn = isbn;  
 this.title = title;  
 this.author = author;  
 this.Left = null;  
 this.Right = null;  
 }  
  
 @Override  
 public String toString() {  
 return " -------------------------\n" +  
 " --> ISBN: " + isbn +  
 "\n --> Title: " + title +  
 "\n --> Author: " + author;  
 }  
}  
  
class LibraryBST {  
 public Book root;  
  
 // Method to insert a new book  
 public Book addBook(Book root, int isbn, String title, String author) {  
 if (root == null) {  
 System.*out*.println("Book with ISBN (" + isbn + ") added successfully!");  
 return new Book(isbn, title, author);  
 } else if (isbn < root.isbn) {  
 root.Left = addBook(root.Left, isbn, title, author);  
 } else if (isbn > root.isbn) {  
 root.Right = addBook(root.Right, isbn, title, author);  
 } else {  
 System.*out*.println("Error: Book with ISBN " + isbn + " already exists!");  
 }  
 return root;  
 }  
  
 // Method to search a book by ISBN  
 public boolean searchBook(Book root, int isbn) {  
 if (root == null) {  
 System.*out*.println("Book not found!");  
 return false;  
 }  
 if (isbn == root.isbn) {  
 System.*out*.println("Book found! \nBook Information: \n" + root);  
 return true;  
 } else if (isbn < root.isbn) {  
 return searchBook(root.Left, isbn);  
 } else {  
 return searchBook(root.Right, isbn);  
 }  
 }  
  
 // Method to print all books sorted by ISBN (in-order traversal)  
 public void printBooks(Book root) {  
 if (root != null) {  
 printBooks(root.Left);  
 System.*out*.println(root);  
 printBooks(root.Right);  
 }  
 }  
  
 // Method to delete a book by ISBN  
 public Book deleteBook(Book root, int isbn) {  
 if (root == null) {  
 System.*out*.println("Error: Book with ISBN " + isbn + " not found!");  
 return null;  
 }  
  
 if (isbn < root.isbn) {  
 root.Left = deleteBook(root.Left, isbn);  
 } else if (isbn > root.isbn) {  
 root.Right = deleteBook(root.Right, isbn);  
 } else {  
 // Node with only one child or no child  
 if (root.Left == null) {  
 System.*out*.println("Book with ISBN " + isbn + " deleted successfully!");  
 return root.Right;  
 } else if (root.Right == null) {  
 System.*out*.println("Book with ISBN " + isbn + " deleted successfully!");  
 return root.Left;  
 }  
  
 // Node with two children: Get the inorder successor (smallest in the right subtree)  
 Book minNode = findMin(root.Right);  
 root.isbn = minNode.isbn;  
 root.title = minNode.title;  
 root.author = minNode.author;  
  
 // Delete the inorder successor  
 root.Right = deleteBook(root.Right, minNode.isbn);  
 }  
 return root;  
 }  
  
 private Book findMin(Book root) {  
 while (root.Left != null) {  
 root = root.Left;  
 }  
 return root;  
 }  
}  
  
public class Scenario03 {  
 public static void main(String[] args) {  
 LibraryBST library = new LibraryBST();  
 // Adding books  
 library.root = library.addBook(library.root, 1, "Effective Java", "Joshua Bloch");  
 library.root = library.addBook(library.root, 3, "Java: The Complete Reference", "Herbert Schildt");  
 library.root = library.addBook(library.root, 5, "Clean Code", "Robert C. Martin");  
 library.root = library.addBook(library.root, 2, "Head First Java", "Kathy Sierra");  
 library.root = library.addBook(library.root, 4, "Java Concurrency in Practice", "Brian Goetz");  
  
 System.*out*.println("\nEnsuring non-duplicate book entry:");  
 library.root = library.addBook(library.root, 5, "Effective Java 2nd Edition", "Joshua Bloch");  
  
 System.*out*.println("\nSearching for a Book with ISBN = 2:");  
 library.searchBook(library.root, 2);  
  
 System.*out*.println("\n\nAll Books Sorted by ISBN:");  
 library.printBooks(library.root);  
  
 System.*out*.println("\nDeleting Book with ISBN = 3:");  
 library.root = library.deleteBook(library.root, 3);

System.*out*.println("\nAll Books After Deletion:");  
 library.printBooks(library.root);  
 }  
}

|  |  |
| --- | --- |
| Method | Time Complexity |
| addBook | O(log n) average, O(n) worst |
| searchBook | O(log n) average, O(n) worst |
| printBooks | O(n) |
| deleteBook | O(log n) average, O(n) worst |

**Output Screenshots:**



***======================== Scenario\_04 =======================***

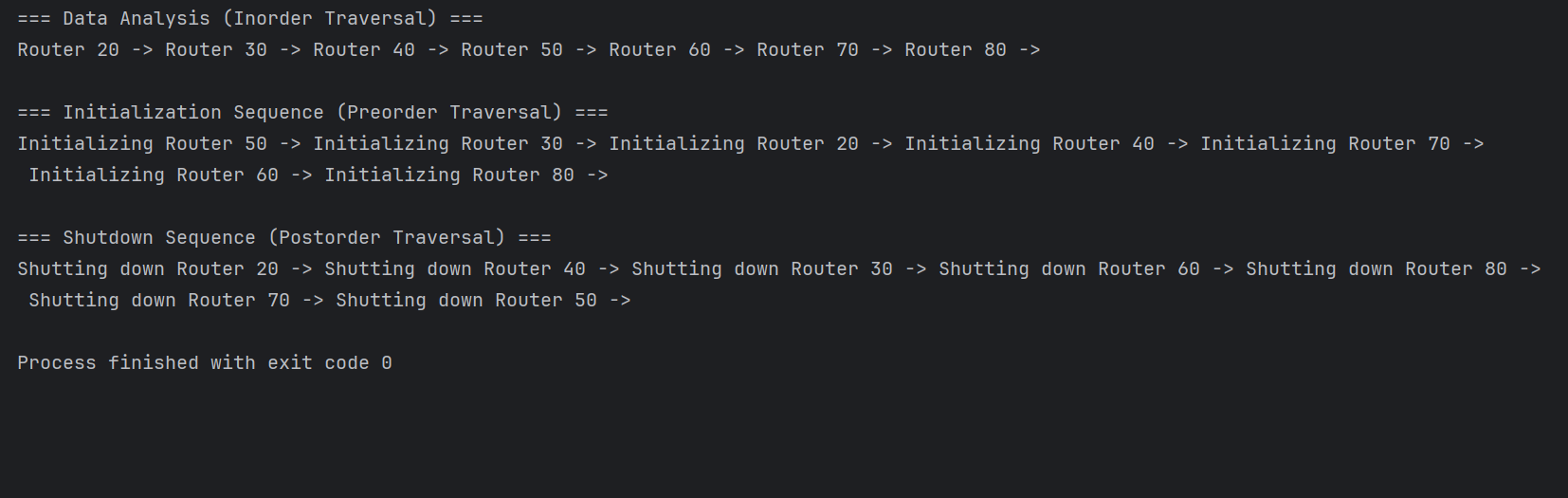
(Packet Routing System Simulation)

Java Code:

class RouterNode {  
 int routerId;  
 RouterNode left, right;  
  
 public RouterNode(int routerId) {  
 this.routerId = routerId;  
 this.left = null;  
 this.right = null;  
 }  
}  
  
class NetworkTree {  
 RouterNode root;  
  
 // Method to Insert Routers  
 public RouterNode insert(RouterNode root, int id) {  
 if (root == null) {  
 return new RouterNode(id);  
 }  
 if (id < root.routerId) {  
 root.left = insert(root.left, id);  
 } else {  
 root.right = insert(root.right, id);  
 }  
 return root;  
 }  
  
 // Inorder traversal - simulate data analysis  
 public void inorder(RouterNode root) {  
 if (root != null) {  
 inorder(root.left);  
 System.*out*.print("Router " + root.routerId + " -> ");  
 inorder(root.right);  
 }  
 }  
  
 // Preorder traversal - simulate initialization  
 public void preorder(RouterNode root) {  
 if (root != null) {  
 System.*out*.print("Initializing Router " + root.routerId + " -> ");  
 preorder(root.left);  
 preorder(root.right);  
 }  
 }  
  
 // Postorder traversal - simulate shutdown  
 public void postorder(RouterNode root) {  
 if (root != null) {  
 postorder(root.left);  
 postorder(root.right);  
 System.*out*.print("Shutting down Router " + root.routerId + " -> ");  
 }  
 }  
}  
  
// Main class to simulate  
public class Scenario04 {  
 public static void main(String[] args) {  
 NetworkTree network = new NetworkTree();  
  
 // Example router network  
 int[] routerIds = {50, 30, 70, 20, 40, 60, 80};  
 for (int id : routerIds) {  
 network.root = network.insert(network.root, id);  
 }  
  
 System.*out*.println("=== Data Analysis (Inorder Traversal) ===");  
 network.inorder(network.root);  
 System.*out*.println("\n");  
  
 System.*out*.println("=== Initialization Sequence (Preorder Traversal) ===");  
 network.preorder(network.root);  
 System.*out*.println("\n");  
  
 System.*out*.println("=== Shutdown Sequence (Postorder Traversal) ===");  
 network.postorder(network.root);  
 System.*out*.println();  
 }  
}

|  |  |
| --- | --- |
| Method | Time Complexity |
| insert | O(log n) average, O(n) worst |
| inorder | O(n) |
| preorder | O(n) |
| postorder | O(n) |

Output Screenshot:



***~~~~~~~~~~~~~~~~~~~~~~~~~ End of Assignment ~~~~~~~~~~~~~~~~~~~~~~~~~***