Project Four Pseudocode

Main Class {

Main Function {

Create a BinTree object called DVD_inventory , the generic tupe of this BinTree will be DVD objects.

Get the filename for inventory from User via a Scanner object.

For every line in the inventory file {
Read the line and split it using ‘,’ character into an array and get the following information from the line array:

title = first element in the array without the quotation marks
available = line_array[1] as int
rented = line_array[2] as int
Create a DVD object called item using the created title, available and rented fields.
Insert the item into the DVD_inventory Binary Tree using BinTree insert method.

}

Close the inventory file
Get the filename for Transactions from User via a Scanner object.

For every line in the Transaction file {

Read a single line from the file.
Store the operator and Operand fields into respective variables

if(Operator == add) {

Split the operand string variable using ‘,’ character into an array called str_array.

title = str_array[0], make sure to ignore the “ “

if the title is not in the Binary Tree DVD_inventory (use search function here)

```
                {
                   Create a new DVD object with correct title and available.
                   Insert the object into Binary Tree DVD_inventory

                }
               else
               {
                   if the DVD is already inside the tree then increment its
                   available value with the given amount.
               }

  } end of operator = add
 if(operator == remove) {


              Split the operand string variable using ‘,’ character into an array
              called str_array.
              title = str_array[0], make sure to ignore the “ “

             if the title is not in the Binary Tree DVD_inventory  ( use search
      function here)
                {

                    if the DVD is already inside the tree then decrement its
                   available value with the given amount.

                   If the available value becomes 0 and rent is also equal to 0 then
                   delete the DVD entry from the tree

                    }
              else do nothing DVD not found

} end of operator == remove

if(operator== rent) {

      title = Get the title from the Operand variable.

            Get the DVD from the Binary Tree DVD_inventory, increment rent value and
decrement available value.

               } end of operator == rent

if(operator == return)) {

        title = Get the title from the Operand variable.

      Get the DVD from the Binary Tree DVD_inventory, increment available value and
decrement rent value.

        } end of operator == return

} end of reading Transaction file
```

```
All Transactions Processed. Display the in-order report Using print_Tree onto the
console and Print the Preorder report into a file called inventory.out


	} end of main Function
} end of the Main class




class BinTree<AnyType extends Comparable<? super AnyType>> {

	private Node<AnyType> root;


	public AnyType search( AnyType x) {
		Should return the results of recursive private method search
	}

	private AnyType search(AnyType x, Node<AnyType> t) {
This is a recursive private method that will be called by the public method

		if(t is null)
			return null

		compareResult  = x.compareTo(t.Payload), both x and t.Payload should be
same object and should be comparable.

		if(compareResult < 0)
			return search(x, t.left)

		else if(compareResult > 0)
			return search(x, t.right)

		else
			return t.Payload;  Match VERY IMPORTANT Don't send x because idea
is to send the pointer to this element

	}
	public void insert(AnyType x) {
		root  = insert(x, root);
	}
	private Node<AnyType> insert(AnyType x, Node<AnyType> t) {

		This method is exactly like search expect we will return the inserted
node.

		if(t is null)
			return new Node<AnyType>(x, null, null), we are trying to insert
a null node.

		compareResult  = x.compareTo(t.Payload), both x and t.Payload should be
same object and should be comparable.
```

```
            if(compareResult < 0)
                  t.left =  insert(x, t.left)

            else if(compareResult > 0)
                  t.right =  insert(x, t.right)


            return t ,this means we are trying to insert duplicate
      }
```

```
  private Node<AnyType> Leftmost(Node<AnyType> t){

       This method is used for finding the minimum while deleting 2 child case
            if(t is null)
                  return null
            else if (t.left is null)
                  return t
            return Leftmost(t.left)
      }
```

```
      public void delete(AnyType x) {
            root  = delete(x, root);
      }
      private Node<AnyType> delete(AnyType x, Node<AnyType> t){
```

In this method we will delete the element from the tree. If the element is in a Node with no children or one child we will delete the Node by skipping it.

If the element is in the Node with 2 children, then we will not delete the Node. We will replace the Node with the minimum (leftmost) element of its right child and then we will delete the duplicate Node. Note a leftmost or minimum of a tree will not have a left child, so it cannot be a 2-child case.

```
            if (t is null)
                  return t
            compareResult  = x.compareTo(t.Payload), both x and t.Payload should be
same object and should be comparable.



            if(compareResult < 0)
                  t.left =  delete(x, t.left)

            else if(compareResult > 0)
                  t.right  = delete(x, t.right)


            else if(t.left != null && t.right != null) {

                  We reach here then item has been found and we will delete it now.
```

```
                  This is 2 Children Case, we will replace Node item with LeftMost
element of the right child
                   t.Payload =Leftmost(t.right).Payload
                   t.right = delete(t.Payload, t.right), make sure to delete the
element from the right child, as we have already copied it.
             }
      else
      {

      This is either one child or 0 child case, here we just make the parent point
to the child that is not null and skip the node, which in essence deletes it.


}
             return t
Deletion Successful

      }end of delete function




      public void print_Tree() {
             print_Tree(root), again calling private recursive routine
      }

private void print_Tree(Node<AnyType> t) {
  This method does in-order printing onto the console

             if(t is not null) {
                   print_Tree(t.left);
                   println(t.Payload);
                   print_Tree(t.right);
             }
      }



      public void print_Tree_preOrder(PrintStream f) {

             print_Tree_preOrder(f,root), again calling private recursive routine.

      }

      private void print_Tree_preOrder(PrintStream  f, Node<AnyType> t) {
This method does Preorder prining onto a PrintStream passed as a argument.


             if(t is not null) {

                   f.println(t.Payload.toString());

                   print_Tree_preOrder(f, t.left);
                   print_Tree_preOrder(f, t.right);
```

```
            }

      }
} end of BinTree class
```

DVD class and Node Class are very simple.

Node class will be generic with appropriate Mutators and Accessors.

DVD class should implement Comparable and have correct compareTo function (use the String compareTo function on its title). It should also have correct toString method that displays in proper column display of DVD information.