```java
import java.io.File;
import java.util.ArrayDeque;

/**
 * Calculates the total size of a file system object and displays result.
 * @author 1828799
 *
 */
public class TotalSize
{
   public static void main(String[] args)
   {
      // Stack to keep track of the files
      ArrayDeque<File> fileStack = new ArrayDeque<File>();

      int totalSize = 0; // The total size

      boolean hasUnreadableFile = false; // Boolean for end display message

      File currentFile = null; // The current file

      File[] directoryFiles = null; // Array to hold array of

      // ERROR CHECK: No argument passed in
      if (args.length == 0)
      {
         System.out.println("Please enter a file path.");
         System.exit(0);
      }

      // Initialize the current file to be the root passed in
      currentFile = new File(args[0]);

      // Push the first file onto the stack
      fileStack.push(currentFile);

      while (!fileStack.isEmpty()) // While stack is not empty
      {
         // Pop the stack for the most recent file
         currentFile = fileStack.pop();

         if (currentFile.canRead()) // If the file/directory is readable
         {

            // Add size of directory/file
            totalSize = (int) (totalSize + currentFile.length());

            if (currentFile.isDirectory())
            {
               // Access the files in the directory
               directoryFiles = currentFile.listFiles();

               // If it is an accessible directory
               if (directoryFiles != null)
               {
                  for (File aFile: directoryFiles)
                  {
                     fileStack.push(aFile); // Push files onto the stack
                  }
               }
               else // The directory is not accessible
               {
                  // Set boolean true to later display message
                  hasUnreadableFile = true;
               }
```

```
66:             }
67:          }
68:          else // Else there is an unreadable file
69:          {
70:             hasUnreadableFile = true;
71:          }
72:       }
73:
74:       // Display size in bytes to screen
75:       System.out.println("Total size: " + totalSize + " bytes");
76:
77:       if (hasUnreadableFile) // If there are unreadable files display message
78:       {
79:          System.out.println("(Note: some files/directories were unreadable)");
80:       }
81:    }
82: }
```

**Output for TotalSize – C:\windows**

Total size: 1142983585 bytes
(Note: some files/directories were unreadable)

**12.10**

a.) Implementation of equals() for BinaryTree<E>

```
public boolean equals(Object otherTree)
{
	// The result
	boolean result = true;

	// Iterators for each tree
	Iterator treeOne = null;
	Iterator treeTwo = null;

	// The current place in each tree
	BinaryTree<E> treeOneCurrent = null;
	BinaryTree<E> treeTwoCurrent = null;

	// If the otherTree is null or is not a BinaryTree<E> then they are not equal
	if (otherTree == null || !otherTree.instanceOf(BinaryTree<E>))
	{
		result = false; // Set result to false
	}
	else
	{
		// Initialize the in-order iterators for each tree
		treeOne = new iterator();
		treeTwo = new otherTree.iterator();

		// While the result is true and both iterators still have a next continue traversing
		while (result && treeOne.hasNext() && treeTwo.hasNext())
		{
			result = false; // Set the result to false

			treeOneCurrent = treeOne.next(); // The next node in the first tree
			treeTwoCurrent = treeTwo.next(); // The next node in the second tree

			// If the nodes in each tree are equal (both left, both right, or both the root
			// and contents equal)
			if ((treeOneCurrent.isLeft() && treeTwoCurrent.isLeft())
			    || (treeOneCurrent.isRight() && treeTwoCurrent.isRight())
			    || (treeOneCurrent.parent() == null
```

```
                    && treeTwoCurrent.parent() == null))
                {
                    // Set result equal to the result of whether the values are equal
                    result = (treeOneCurrent.value()).equals(treeTwoCurrent.value());
                }

            } // End while

            // When the while loop is exited because one iterator doesn't have a next then the
            // trees are not equal
            if (result && (treeOne.hasNext() != treeTwo.hasNext())
            {
                result = false;
            }

        } // End else

        return result;

} // End
```

b.) Time complexity: (n = size of smallest tree being compared)

Best Case: O(1) → when the first leaf in the in-order traversal of the tree is not equal
Average Case: O(n)
Worst Case: O(n) → when the trees are equal up until the last leaf to be visited in the in-order traversal of the smaller tree

**12.22:** How to perform an in-order traversal using a single reference

```
previous = null
current = given starting point (root)

while there are unvisited nodes
    if previous is current's left child or current has no children
        visit current
        go to right child (adjust current and previous accordingly)
    else if previous is current's right child
        go up to parent (adjust current and previous accordingly )
    else if current has left child
        go left (adjust current and previous accordingly)
```

**Problem #4**

a. public interface Visitor

```
{
    public void visit(T o);
}
```

b. Implementation of *visit* method that prints out the Object to System.out

```
public class SomeClass implements Visitor
{
    public void visit(T o)
    {
            System.out.println(o.toString());
    }
}
```

c. Implementation of the *inorderTraversal* method

```
public void inorderTraversal(Visitor v)
{
    if (v != null)
    {
            inorderTraversal(v._leftChild);
            visit(v);
            inorderTraversal(v._rightChild);
    }
}
```