

Palindrome recursive

```
public static boolean
palindromecheck(String temp) {
    boolean result = false;
    temp = temp.toLowerCase();
    temp = temp.replaceAll(" ",
    "");
    temp = temp.replaceAll("\n",
    "");
    temp = temp.replaceAll("\r",
    "");
    temp = temp.replaceAll("\t",
    "");
    temp = temp.replaceAll("\f",
    "");
    temp = temp.replaceAll("\r",
    "");
    temp = temp.replaceAll("\n",
    "");
    temp = temp.replaceAll("\t",
    "");
    temp = temp.replaceAll("\f",
    "");
    if (temp.length() > 1) {
        if (temp.charAt(0) == temp.charAt(temp.length()-1)) {
            palindromecheck(temp.substring(1, temp.length()-1));
            result = true;
        }
        else {
            result = false;
        }
    }
    return result;
}
```

Sorting(Simple one)

```
public static void
sorting(String[] list) {
    for (int x = 0; x < list.length; x++) {
        for (int y = x + 1; y < list.length; y++) {
            // index of array
            if (list[x].compareTo(list[y]) > 0) {
```

Sorting(Simple one) (cont)

```
> continue;
}
else if (list[x].compareTo(list[y]) == 0) {
    continue;
}
else {
    String temp = list[x];
    list[x] = list[y];
    list[y] = temp;
}
}
System.out.print("Pass: "+x); printlist(list);
}
}
```

File IO (Write)

```
public class write {
    public static void main(String[] args) {
        BufferedWriter writer = null;
        File file = new File("log.txt");
        try {
            System.out.println("test");
            writer = new BufferedWriter(new FileWriter(file));
            writer.write("Hello World.Wat ??? \n");
            writer.append("Hello Mars. \n");
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

File IO (Write) (cont)

```
> }
finally {
    try {
        if (writer != null) {
            writer.close();
        }
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}
}
```

Node recursive

```
public static void
inOrderTraverse(Node root)
{
    if (root == null) {
    }
    else {
        //System.out.println(" " + root.id); //pre order
        inOrderTraverse(root.left);
        System.out.println(" " + root.id); // in order
        inOrderTraverse(root.right);
        //System.out.println(" " + root.id); // post order
    }
}
```

getSnippet

```
public static String
getSnippet(Word root, int
window)
{
    if(root == null) return null;
    String Builder str = new
    String Builder();
    str.append("[" + root + "] ");
    Word pointer = root.getNextWord();
    for(int i = 0; i < window; i++)
    { if(pointer == null)
    {
        break;
    }
    else
    { str.insert(0, pointer.getNextWord() + " ");
    pointer = pointer.getNextWord();
    }
    }
    if(pointer != null) str.insert(0, "... ");

    pointer = root.getNextWord();
    for(int i = 0; i < window; i++)
    { if(pointer == null)
    {
        str.append("... ");
        break;
    }
    else
    { str.append(pointer.getNextWord() + " ");
    pointer = pointer.getNextWord();
    }
    }
}
```

getSnippet (cont)

```
> if(pointer != null) str.append("... ");

return str.toString().trim();
}
```

How to create array of int and double

```
int a[]=new int[5];
data = new Double [10];
```

Switch case

```
int day = 4;
switch (day) {
    case 6:
        System.out.println -
        ("Today is Saturday");
        break;
    case 7:
        System.out.println -
        ("Today is Sunday ");
        break;
    default:
        System.out.println -
        ("Looking forward to the
        Weekend");
}
// Outputs " Looking forward to
the Weekend"
```

Compareto

```
result will be int
> 0 : greater
=0 : similar
<0: least
```

isFullBinTree (Node)

```
public static boolean
isFullBinTree(Node root)
{
    boolean result = false;
    if (root == null) {
        result = true;
    }
    else {
        if (root.left != null &&
        root.right != null) {
            if (!isFullBinTree(root.left) ||
            !isFullBinTree(root.right)) {
                result = false;
            }
            else {
                result = true;
            }
        }
        else if (root.left == null &&
        root.right == null) {
            result = true;
        }
        else {
            result = false;
        }
    }
    return result;
}
```

binaryfind

```
public static boolean
binaryfind(int[][] matrix, int
window) {
    boolean check = false;
    int length = matrix -
[0].length * matrix [1].length;
    int low = 0;
    int high = sorted Words.size()
- 1;
    int mid = 0;
    while (low <= high) {
        mid = low + ((high - low)/2);
        if (sorted Words.get(mid) ==
target) {
            return true;
        }
        else if (sorted Words.get(mid) <
target) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    return check;
}
```

File IO (Read apache)

```
//load file
String text = null;
try {
    text = FileUtils.readFileToS
tring(new File(filename),
" UTF -8");
} catch (IOException e) {
    e.printStackTrace();
}
```

File IO (Read apache) (cont)

```
> //clean text
text = text.toLowerCase().replaceAll("\\W+",
"").replaceAll("\\s\\w\\s", " ");

//System.out.println(text);
//tokenize words
String[] tokens = text.split("\\s+");

//store tokens in words
sortedWords = new ArrayList<Word>();
for(int i = 0; i < tokens.length; i++)
{ sortedWords.add(new Word(tokens[i],i));
if(i > 0)
{ sortedWords.get(i).setPreviousWord(sorted-
Words.get(i-1));
sortedWords.get(i-1).setNextWord(sorted-
Words.get(i));
}
}

//sort word
Collections.sort(sortedWords);
```