Basic Structure of Java Program

1. Package
2. Class
3. main() Method
4. Comments :
   Single line - //
   Multiline - /*...*/

```
package helloworld;
public class HelloWorld
{  public static void main(String[] args)
{  //Print the words Hello World on the screen
 System.out.println("Hello World");
 } }
```

Variables

Names given to data that we need to store and manipulate in our programs.

<u>Primitive data type</u>

1. byte : byte userAge = 20;
2. short : short numberOfStudents = 45;
3. int : int numberOfEmployees = 500;
4. long - fraction : long numberOfInhabitants = 21021313012678L;
5. float - decimal : float hourlyRate = 60.5F;
6. double - negative : double numberOfHours = 5120.5;
7. char - special characters : char grade = 'A';
8. boolean - true or false : boolean promote = true;

Basic Operators

1. Addition: x + y = 9
2. Subtraction: x - y = 5
3. Multiplication: x*y = 14
4. Division: x/y = 3 (rounds down the answer to the nearest integer)
5. Modulus: x%y = 1 (gives the remainder when 7 is divided by 2)

More Assignment Operators

1. x=x+2 : x+=2
2. x=x-2 : x-=2

3. x=x+1 : x++
4. x=x-1 : x--

Type Casting

To convert from one data type to another : int x = (int) 20.9;

String

Piece of text

String message = "Hello World";

1. length() : tells us the total number of characters the string has.
   int myLength = "Hello World".length();
   System.out.println(myLength); = 11
2. toUpperCase() : convert a string to uppercase characters
   String uCase = "Hello World".toUpperCase();
   uCase will thus be equal to “HELLO WORLD”
3. toLowerCase() : to convert a string to lowercase characters
   String lCase = "Hello World".toLowerCase();
   lCase will thus be equal to “hello world”
4. substring() : extract a substring from a longer string
   index starts with a value of 0
   String firstSubstring = "Hello World".substring(6);
   firstSubstring is thus equal to “World”
   String secondSubstring = “Hello World” .substring(1, 8);
   secondSubstring is thus equal to “ello Wo”
5. charAt() : returns a single character at a specified location
   char myChar = "Hello World".charAt(1);
   myChar is equal to ‘e’
6. equals() : used to compare if two strings are identical
   boolean equalsOrNot = "This is Jamie".equals("This is Jamie"); = true
   boolean equalsOrNot2 = "This is Jamie".equals("Hello World"); = false
7. split() : splits a string into substrings based on a user-defined separator (also known as a delimiter)
   String names = "Peter, John, Andy, David";
   String[] splitNames = names.split(", ");
   Result : {"Peter", "John", "Andy", "David"}

Array

Collection of data that are normally related to each other.

1. equals() : if two arrays are equal to each other
   int[] arr1 = {0,2,4,6,8,10};
   int[] arr2 = {0,2,4,6,8,10};
   int[] arr3 = {10,8,6,4,2,0};
   boolean result1 = Arrays.equals(arr1, arr2); =true
   boolean result2 = Arrays.equals(arr1, arr3); =false
2. copyOfRange() : copy the contents of one array into another
   int [] source = {12, 1, 5, -2, 16, 14, 18, 20, 25};
   int[] dest = Arrays.copyOfRange(source, 3, 7);
   dest becomes {-2, 16, 14, 18}
3. toString() : returns a String that represents the contents of an array
   System.out.println(Arrays.toString(numbers));
   Output is [1, 2, 3, 4, 5]
4. sort() : sort our arrays
   int [] numbers2 = {12, 1, 5, -2, 16, 14};
   Arrays.sort(numbers2);
   System.out.println(Arrays.toString(numbers2));
   Output is [-2, 1, 5, 12, 14, 16]
5. binarySearch() : search for a specific value in a sorted array
   int[] myInt = {21, 23, 34, 45, 56, 78, 99};
   int foundIndex = Arrays.binarySearch(myInt, 99);
   foundIndex will be equal to 6
6. Finding Array Length : find the length of an array
   int [] userAge = {21, 22, 26, 32, 40};
   userAge.length is equal to 5

Primitive Type : stores its own data.
(byte, short, int, long, float, double, char and boolean)
int myNumber = 5; the variable myNumber stores the actual value 5.

Reference Type : does not store the actual data
(strings and arrays)
stores a reference to the data. It does not tell the compiler what the value of the data is; it tells the compiler where to find the actual data.
String message = "Hello"; string “Hello” is created and stored elsewhere in the computer’s memory

Display Output

1. println() : moves the cursor down to the next line after displaying the message
2. print() : keeps the cursor on the same line
3. System.out.println("Hello ");
   System.out.println ("How are you?");
   Hello
   How are you?
4. System.out.print("Hello ");
   System.out.print("How are you?");
   Hello How are you ?

Escape Sequence
1. System.out.println("Hello\tWorld");
   Hello   World
2. To prints a newline (\n)
   System.out.println("Hello\nWorld");
   Hello
   World
3. To print the backslash character itself (\\)
   System.out.println("\\");
   \
4. To print double quotes (\") so that the double quote does not end the string
   System.out.println("I am 5'9\" tall");
   I am 5'9" tall

Accepting User Input
1. import java.util.Scanner;
2. Scanner reader = new Scanner(System.in);
3. nextInt(), nextDouble() and nextLine() for reading int, double and String data types respectively.

Control Flow Statements
1. To control the flow of your program
2. decision-making statements (if, switch), looping statements (for, while, do-while), and branching statements (break, continue)

Comparison Operators
involve doing some form of comparison
1. Not equal (!=)
   5 != 2 is true

6 != 6 is false

2. Greater than (>)

   5 > 2 is true

   3 > 6 is false

3. Smaller than (<)

   1 < 7 is true

   9 < 6 is false

4. Greater than or equal to (>=)

   5 >= 2 is true

   5 >= 5 is true

   3 >= 6 is false

5. Smaller than or equal to (<=)

   5 <= 7 is true

   7 <= 7 is true

   9 <= 6 is false

6. The AND operator (&&) : Returns true if all conditions are met

   5==5 && 2>1 && 3!=7 is true

   5==5 && 2<1 && 3!=7 is false

7. The OR operator (||) : Returns true if at least one condition is met

   5==5 || 2<1 || 3==7 is true

   5==6 || 2<1 || 3==7 is false

Decision Making Statements

If Statement : allows the program to evaluate if a certain condition is met and perform the appropriate action based on the result of the evaluation

```
if (condition 1 is met)
 {  do Task A  }
 else if
{  do Task B  }
else if
 { do Task C }
else
{  do Task D  }
```

Eg :

```
package ifdemo;
import java.util.Scanner;
 public class IfDemo
```

```
{ public static void main(String[] arg)
{ Scanner input = new Scanner(System.in);
System.out.print("\nPlease enter your age: ");
int userAge = input.nextInt();
if (userAge < 0 || userAge > 100)
{ System.out.println("Invalid Age");
System.out.println("Age must be between 0 and 100");  }
else if (userAge < 18)
System.out.println("Sorry you are underage");
else if (userAge < 21)
System.out.println("You need parental consent");
else
{ System.out.println("Congratulations!");
System.out.println("You may sign up for the event!"); } } }
```

<u>Ternary operator (?)</u> : simpler form of an if statement that is very convenient if you want to assign a value to a variable depending on the result of a condition

1. condition ? value to return if condition is true : value to return if condition is false;
2. Eg : int myNum = 3>2 ? 10 : 5;

   myNum will be assigned the value 10.

<u>Switch Statemen</u>t : requires each case to be based on a single value

```
switch (variable used for switching)
 { case firstCase:
do A;
break;
case second
Case: do B;
break;
default:
do C;
break; }
```

Eg :

```
package switchdemo;
import java.util.Scanner;
public class SwitchDemo
{ public static void main(String[] args)
{ Scanner input = new Scanner(System.in);
System.out.print("Enter your grade: ");
String userGrade = input.nextLine().toUpperCase();
switch (userGrade)
 { case "A+":
case "A":
System.out.println("Distinction");
break;
case "B":
System.out.println("B Grade");
break;
case "C":
System.out.println("C Grade");
break;
default:
System.out.println("Fail");
break; } } }
```

<u>Looping Statements</u>

<u>For Statement</u> : executes a block of code repeatedly until the test condition is no longer valid

1. for (initial value; test condition; modification to value)
   { //Do Some Task }
2. Eg : for (int i = 0; i < 5; i++)
   { System.out.println(i); }
   ```
   0
   1
   2
   3
   4
   ```

<u>Enhanced For Statement</u> : get information from an array without making any changes to it

1. for (variable declaration : name of array) { }
2. Eg : int[] myNumbers = {10, 20, 30, 40, 50};
   for (int item : myNumbers)
   System.out.println(item);
   ```
   10
   20
   30
   40
   50
   ```

While Statement : repeatedly executes instructions inside the loop while a certain condition remains valid

1. while (condition is true) { do A }
2. int counter = 5;

```
while (counter > 0)
{ System.out.println("Counter = " + counter);
counter = counter - 1; }
Counter = 5
Counter = 4
Counter = 3
Counter = 2
Counter = 1
```

Do-while Statement : executed at least once

```
int counter = 100;
do { System.out.println("Counter = " + counter);
counter++; }
while (counter<0);
```

Branching Statements : instructs the program to branch to another line of code

Break Statement : causes the program to exit a loop prematurely when a certain condition is met

```
 for (int i = 0; i < 5; i++)
{  System.out.println("i = " + i);
 if (i == 2)
break; }
i = 0
i = 1
i = 2
```

Continue Statement : rest of the loop after the word is skipped for that iteration

```
for (int i = 0; i<5; i++)
 {  System.out.println("i = " + i);
if (i == 2)
continue;
System.out.println("I will not be printed if i=2."); }
i = 0
I will not be printed if i=2
i = 1
I will not be printed if i=2
i = 2
```

```
i = 3
I will not be printed if i=2
i = 4
I will not be printed if i=2
```

Exception Handling : control the flow of a program when an error occurs

```
try
{ do something }
catch (type of error)
{ do something else when an error occurs }
finally
{ do this regardless of whether the try or catch condition is met. }
```

Eg:

```
package errordemo;
import java.util.Scanner;
public class ErrorDemo
{ public static void main(String[] args)
{ int num, deno;
Scanner input = new Scanner(System.in);
try
{ System.out.print("Please enter the numerator: ");
num = input.nextInt();
System.out.print("Please enter the denominator: ");
deno = input.nextInt();
System.out.println("The result is " + num/deno); }
catch (Exception e)
{ System.out.println(e.getMessage()); }
finally
{ System.out.println("---- End of Error Handling Example ----");  }}}
```

enter 12 and 4

```
The result is 3
---- End of Error Handling Example ----
```

run the program again and enter 12 and 0

```
/ by zero
---- End of Error Handling Example ----
```

Specific Errors : want to perform specific tasks depending on the error caught, or display your own error messages

```
package errordemo2;
import java.util.InputMismatchException;
import java.util.Scanner;
public class ErrorDemo2
{ public static void main(String[] args)
{ int choice = 0;
Scanner input = new Scanner(System.in);
int[] numbers = { 10, 11, 12, 13, 14, 15 };
System.out.print("Please enter the index of the array: ");
try
{ choice = input.nextInt();
System.out.printf("numbers[%d] = %d%n",
choice, numbers[choice]); }
catch (ArrayIndexOutOfBoundsException e)
{ System.out.println("Error: Index is invalid."); }
catch (InputMismatchException e)
{ System.out.println("Error: You did not enter an integer."); }
catch (Exception e)
{System.out.printf(e.getMessage()); } } }
```

Enter 10 - Error: Index is invalid.

Enter Hello - Error: You did not enter an integer

Throwing Exceptions : define our own conditions for when an error should occur

```
package errordemo2;
import java.util.InputMismatchException;
import java.util.Scanner;
public class ErrorDemo2
{ public static void main(String[] args)
{ int choice = 0;
Scanner input = new Scanner(System.in);
int[] numbers = { 10, 11, 12, 13, 14, 15 };
if (choice == 0)
throw new ArrayIndexOutOfBoundsException();
choice = input.nextInt();
System.out.print("Please enter the index of the array: ");
try
{ choice = input.nextInt();
System.out.printf("numbers[%d] = %d%n",
choice, numbers[choice]); }
catch (ArrayIndexOutOfBoundsException e)
{ System.out.println("Error: Index is invalid."); }
catch (InputMismatchException e)
{ System.out.println("Error: You did not enter an integer."); }
catch (Exception e)
{System.out.printf(e.getMessage()); } } }
```
Enter 0 - catch(ArrayIndexOutOfBoundsException e) block is executed instead.

Fields : declare a field as either private, public or protected Eg : private String nameOfStaff;
Methods : code block that performs a certain task. Eg : public void printMessage()
Overloading : create two methods of the same name as long as they have different signatures.
Getter and Setter Methods : greater control over what rights other classes have when assessing these private fields
Constructors : commonly used to initialize the fields of the class. Eg : public Staff(String firstName, String lastName)

Polymorphism : program's ability to use the correct method for an object based on its run-time type.
```
Member[] clubMembers = new Member[3];
clubMembers[0] = new NormalMember("James", 1, 2010);
clubMembers[1] = new NormalMember("Andy", 2, 2011);
clubMembers[2] = new NormalMember("Bill", 3, 2011);
clubMembers[3] = new VIPMember("Carol", 4, 2012);
```
Abstract class: special type of class that is created strictly to be a base class for other classes to derive from.

Java Collections Framework : set of pre-written classes and interfaces that Java provides to help us organize and manipulate groups of objects.

Automatic conversion :
Autoboxing. : We simply assign the value 100 to intObject, We do not need to pass this int value to the Integer constructor, Java does it for us behind the scene
To convert from int to Integer, instead of writing - Integer intObject = new Integer(100);
Simply write - Integer intObject = 100;

Unboxing : do not need to explicitly use the intValue() method. When we assign an Integer object to an int variable, Java automatically converts the Integer object to int type.
To convert from Integer to int, instead of writing - int m = intObject.intValue();
Simply write - int m = intObject;

ArrayList class : pre-written class that implements the List interface.

1. add() : add members to a list
   userAgeList.add(40);
   userAgeList.add(53);
   userAgeList.add(45);
   System.out.println(userAgeList); - [40, 53, 45]

   add members at a specific position:
   userAgeList.add(2, 51);
   userAgeList now becomes [40, 53, 51, 45]
2. set() : replace an element at a specified position with another element
   userAgeList.set(3, 49);
   userAgeList now becomes [40, 53, 51, 49]
3. remove() : remove member at a specific position from the list
   userAgeList.remove(3);
   userAgeList becomes [40, 53, 51]
4. get() : get the element at a specific position
   userAgeList.get(2);
   gives us the number 51.
5. size() : find out the number of elements in the list
   userAgeList.size() gives us 3
6. contains() : check if a list contains a certain member
   userAgeList.contains(51); - true
   userAgeList.contains(12); - false
7. indexOf() : get the index of the first occurrence of a certain element
   userAgeList.indexOf(53); = 1
   userAgeList.indexOf(12); = -1
8. toArray() : get all the elements in the ArrayList
   Integer[] myIntArray = userAgeList.toArray(new Integer[0]); - get integer
9. clear() : remove all items in a list
   userAgeList.clear();
   no elements left in the list

A LinkedList stores the addresses of the elements before and after each element.
use a LinkedList when there is a need to add or remove elements frequently.
higher memory consumption

1. poll() : returns the first element of the list and removes the element from the list.
   returns null if the list is empty.
   userAgeList is currently [40, 53, 51, 53]
   System.out.println(userAgeList.poll()); = 40
   print out the elements of the userAgeList again = [53, 51, 53]
2. peek() : returns the first element of the list but does not remove the element from the list.
   returns null if the list is empty.
3. getFirst() : returns the first element of the list and does not remove the element. gives a NoSuchElementException exception when the list is empty
4. getLast() : returns the last element of the list and does not remove the element. It gives a NoSuchElementException exception when the list is empty.

accept an ArrayList as a parameter, we declare the method as

```
public void methodOne(ArrayList m)
{ //Some implementation code }
```

return an ArrayList from a method, we declare the method as

```
public ArrayList methodTwo()
 { ArrayList a = new ArrayList<>();
//Some implementation code
return a; }
```

both methodOne() and methodTwo() are in a class called MyClass

```
MyClass mc = new MyClass();
```

To call methodOne(),

```
ArrayList b = new ArrayList<>();
mc.methodOne(b);
```

To call methodTwo()

```
ArrayList c = mc.methodTwo();
```

Functional interface : interface that contains one and only one abstract method

Project : basic membership management program for a fitness centre. This fitness centre has three outlets:two types of members: A single club member has access to only one of the three clubs. A multi club member, on the other hand, has access to all three clubs.
For single club members, the fees also depend on which club he/she has access to. Finally, multi club members are awarded membership points for joining the club. Upon sign up, they are awarded 100 points
Classes

1. *Member*
2. *SingleClubMember extends Member*
3. *MultiClubMember extends Member*
4. *MembershipManagement*
5. *FileHandler*
6. *Java Project*

Interface : Calculator

<u>Member Class</u> : This class contains basic information about each member.

1. Fields :
   private char memberType;
   private int memberID;
   private String name;
   private double fees;
2. Constructor :
   char pMemberType
   int pMemberID
   String pName
   double pFees
3. Methods :
   public void setMemberType(char pMemberType)
   public void setMemberID(int pMemberID)
   public void setName(String pName)
   public void setFees(double pFees)

   public char getMemberType()
   public int getMemberID()
   public String getName()
   public double getFees()

   public String toString()

Actual Code

```
package javaproject;
public class Member
{ char memberType;
int memberID;
String name;
double fees;

Member(char pMemberType, int pMemberID, String pName, double pFees)

{  memberType = pMemberType;
memberID = pMemberID;
name = pName;
fees = pFees; }

public void setMemberType(char pMemberType)
{ memberType = pMemberType; }

public char getMemberType()
{ return memberType; }

public void setMemberID(int pMemberID)
{ memberID = pMemberID; }

public int getMemberID()
{ return memberID; }

public void setName(String pName)
{ name = pName; }

public String getName()
{ return name; }

 public void setFees(double pFees)
{ fees = pFees; }

public double getFees()
{ return fees; }

@Override public String toString()
{ return memberType + ", " + memberID + ", " + name + ", " + fees; }}
```

The SingleClubMember Class :

```
package javaproject;

public class SingleClubMember extends Member

{ private int club;

SingleClubMember(char pMemberType, int pMemberID, String pName, double pFees, int
pClub)
{ super(pMemberType, pMemberID, pName, pFees);
club = pClub; }

public void setClub(int pClub)
{ club = pClub; }

public int getClub()
{ return club; }

@Override public String toString()
{ return super.toString() + ", " + club; } }
```

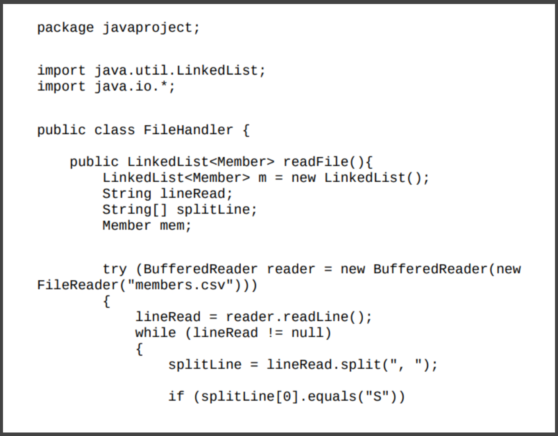
The MultiClubMember Class :

```
package javaproject;

public class MultiClubMember extends Member

{ private int membershipPoints;

MultiClubMember(char pMemberType, int pMemberID, String pName, double pFees, int
pMembershipPoints){ super(pMemberType, pMemberID, pName, pFees);
membershipPoints = pMembershipPoints; }

public void setMembershipPoints(int pMembershipPoints)
{ membershipPoints = pMembershipPoints; }

public int getMembershipPoints()
{ return membershipPoints; }

@Override public String toString()
{ return super.toString() + ", " + membershipPoints; } }
```

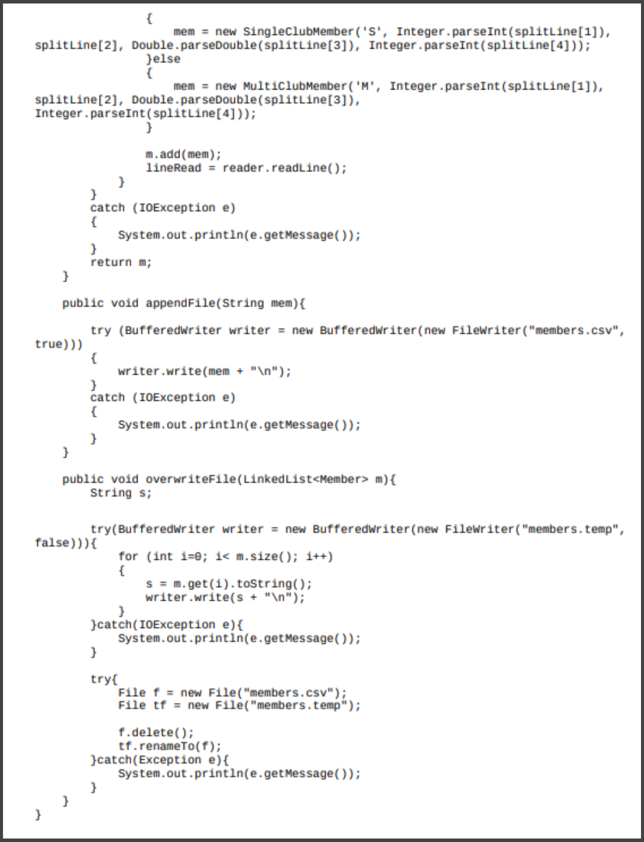
The Calculator Interface :

```
package javaproject;
public interface Calculator
{ double calculateFees(T clubID); }
```

The FileHandler Class :

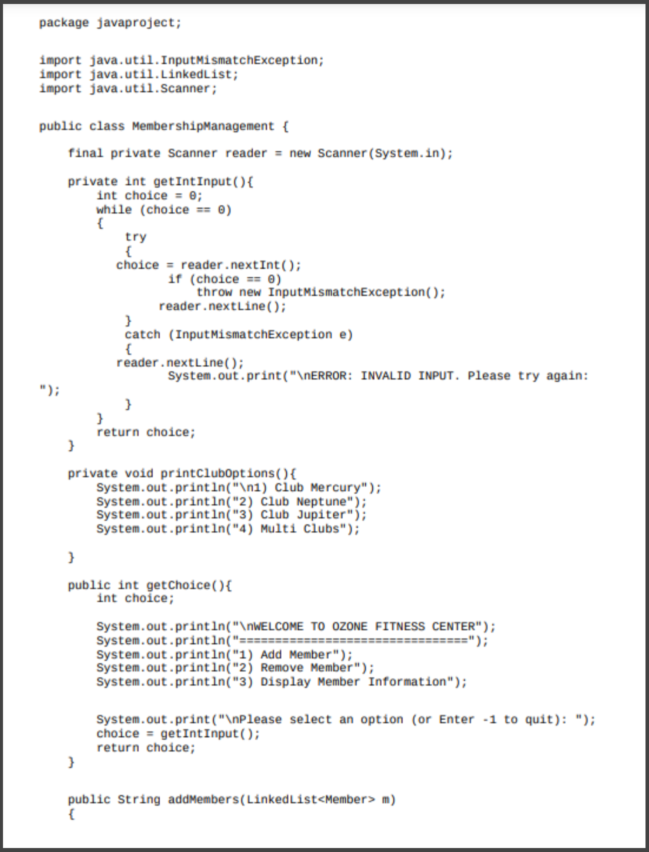


```
package javaproject;

import java.util.LinkedList;
import java.io.*;

public class FileHandler {

    public LinkedList<Member> readFile(){
        LinkedList<Member> m = new LinkedList();
        String lineRead;
        String[] splitLine;
        Member mem;


        try (BufferedReader reader = new BufferedReader(new 
FileReader("members.csv")))
        {
            lineRead = reader.readLine();
            while (lineRead != null)
            {
                splitLine = lineRead.split(", ");

                if (splitLine[0].equals("S"))
```

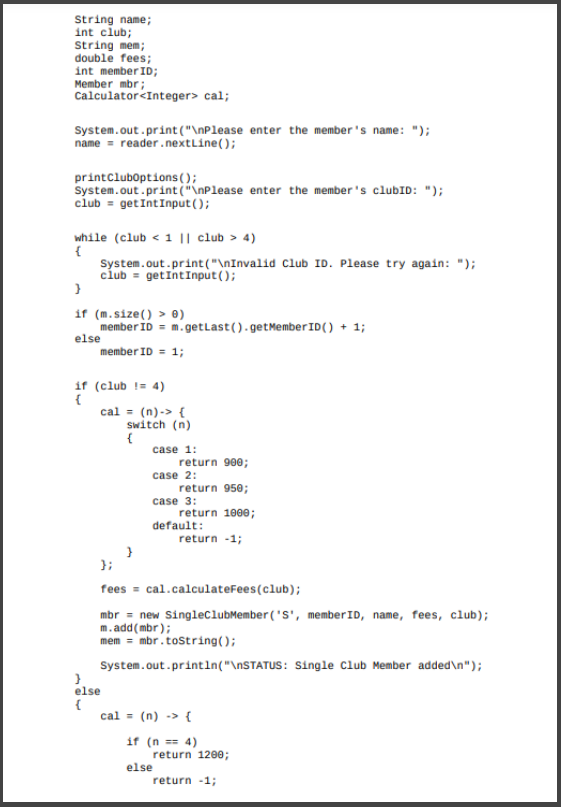


```
                {
                    mem = new SingleClubMember('S', Integer.parseInt(splitLine[1]), 
splitLine[2], Double.parseDouble(splitLine[3]), Integer.parseInt(splitLine[4]));
                }else
                {
                    mem = new MultiClubMember('M', Integer.parseInt(splitLine[1]), 
splitLine[2], Double.parseDouble(splitLine[3]), 
Integer.parseInt(splitLine[4]));
                }

                m.add(mem);
                lineRead = reader.readLine();
            }
        }
        catch (IOException e)
        {
            System.out.println(e.getMessage());
        }
        return m;
    }

    public void appendFile(String mem){

        try (BufferedWriter writer = new BufferedWriter(new FileWriter("members.csv", 
true)))
        {
            writer.write(mem + "\n");
        }
        catch (IOException e)
        {
            System.out.println(e.getMessage());
        }
    }

    public void overwriteFile(LinkedList<Member> m){
        String s;


        try(BufferedWriter writer = new BufferedWriter(new FileWriter("members.temp", 
false))){
            for (int i=0; i< m.size(); i++)
            {
                s = m.get(i).toString();
                writer.write(s + "\n");
            }
        }catch(IOException e){
            System.out.println(e.getMessage());
        }

        try{
            File f = new File("members.csv");
            File tf = new File("members.temp");

            f.delete();
            tf.renameTo(f);
        }catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```

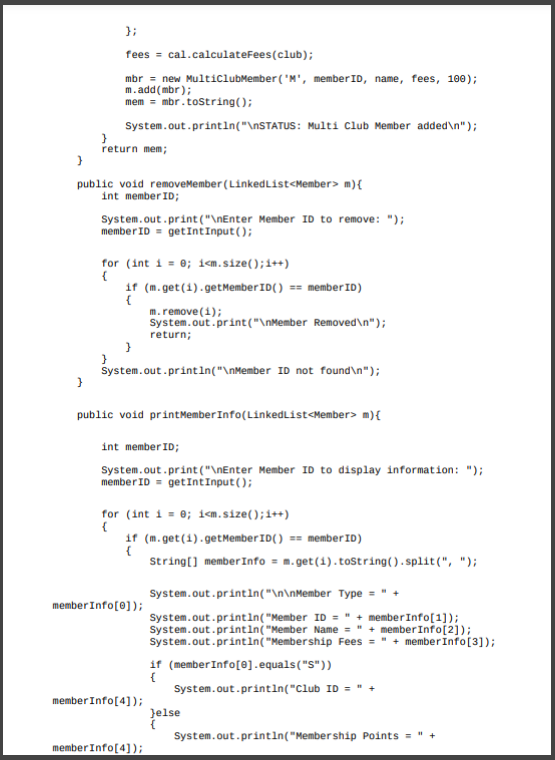
MembershipManagement Class



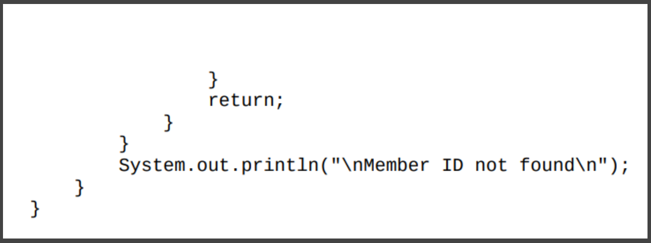
```java
package javaproject;

import java.util.InputMismatchException;
import java.util.LinkedList;
import java.util.Scanner;

public class MembershipManagement {

    final private Scanner reader = new Scanner(System.in);

    private int getIntInput(){
        int choice = 0;
        while (choice == 0)
        {
            try
            {
           choice = reader.nextInt();
                if (choice == 0)
                    throw new InputMismatchException();
              reader.nextLine();
            }
            catch (InputMismatchException e)
            {
           reader.nextLine();
                System.out.print("\nERROR: INVALID INPUT. Please try again:
");
            }
        }
        return choice;
    }

    private void printClubOptions(){
        System.out.println("\n1) Club Mercury");
        System.out.println("2) Club Neptune");
        System.out.println("3) Club Jupiter");
        System.out.println("4) Multi Clubs");

    }

    public int getChoice(){
        int choice;

        System.out.println("\nWELCOME TO OZONE FITNESS CENTER");
        System.out.println("================================");
        System.out.println("1) Add Member");
        System.out.println("2) Remove Member");
        System.out.println("3) Display Member Information");


        System.out.print("\nPlease select an option (or Enter -1 to quit): ");
        choice = getIntInput();
        return choice;
    }

    public String addMembers(LinkedList<Member> m)
    {
```



```java
        String name;
        int club;
        String mem;
        double fees;
        int memberID;
        Member mbr;
        Calculator<Integer> cal;


        System.out.print("\nPlease enter the member's name: ");
        name = reader.nextLine();


        printClubOptions();
        System.out.print("\nPlease enter the member's clubID: ");
        club = getIntInput();


        while (club < 1 || club > 4)
        {
            System.out.print("\nInvalid Club ID. Please try again: ");
            club = getIntInput();
        }

        if (m.size() > 0)
            memberID = m.getLast().getMemberID() + 1;
        else
            memberID = 1;


        if (club != 4)
        {
            cal = (n)-> {
                switch (n)
                {
                    case 1:
                        return 900;
                    case 2:
                        return 950;
                    case 3:
                        return 1000;
                    default:
                        return -1;
                }
            };

            fees = cal.calculateFees(club);

            mbr = new SingleClubMember('S', memberID, name, fees, club);
            m.add(mbr);
            mem = mbr.toString();

            System.out.println("\nSTATUS: Single Club Member added\n");
        }
        else
        {
            cal = (n) -> {

                if (n == 4)
                    return 1200;
                else
                    return -1;
```

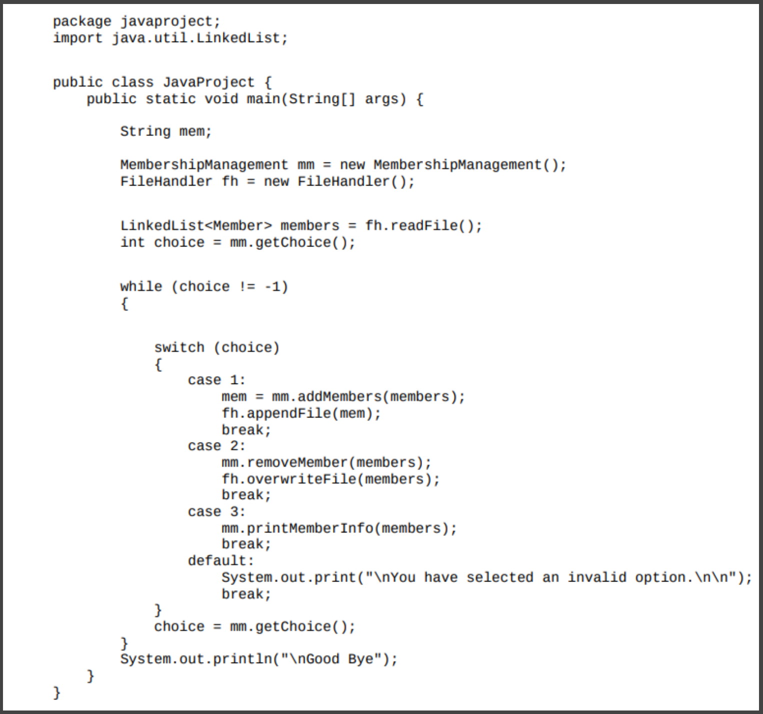


```java
            };

            fees = cal.calculateFees(club);

            mbr = new MultiClubMember('M', memberID, name, fees, 100);
            m.add(mbr);
            mem = mbr.toString();

            System.out.println("\nSTATUS: Multi Club Member added\n");
        }
        return mem;
    }

    public void removeMember(LinkedList<Member> m){
        int memberID;

        System.out.print("\nEnter Member ID to remove: ");
        memberID = getIntInput();


        for (int i = 0; i<m.size();i++)
        {
            if (m.get(i).getMemberID() == memberID)
            {
                m.remove(i);
                System.out.print("\nMember Removed\n");
                return;
            }
        }
        System.out.println("\nMember ID not found\n");
    }


    public void printMemberInfo(LinkedList<Member> m){


        int memberID;

        System.out.print("\nEnter Member ID to display information: ");
        memberID = getIntInput();


        for (int i = 0; i<m.size();i++)
        {
            if (m.get(i).getMemberID() == memberID)
            {
                String[] memberInfo = m.get(i).toString().split(", ");


                System.out.println("\n\nMember Type = " +
memberInfo[0]);
                System.out.println("Member ID = " + memberInfo[1]);
                System.out.println("Member Name = " + memberInfo[2]);
                System.out.println("Membership Fees = " + memberInfo[3]);

                if (memberInfo[0].equals("S"))
                {
                    System.out.println("Club ID = " +
memberInfo[4]);
                }else
                {
                    System.out.println("Membership Points = " +
memberInfo[4]);
```



```java
                }
                return;
            }
        }
        System.out.println("\nMember ID not found\n");
    }
}
```

JavaProject Class



```
package javaproject;
import java.util.LinkedList;


public class JavaProject {
    public static void main(String[] args) {

        String mem;

        MembershipManagement mm = new MembershipManagement();
        FileHandler fh = new FileHandler();


        LinkedList<Member> members = fh.readFile();
        int choice = mm.getChoice();


        while (choice != -1)
        {


            switch (choice)
            {
                case 1:
                    mem = mm.addMembers(members);
                    fh.appendFile(mem);
                    break;
                case 2:
                    mm.removeMember(members);
                    fh.overwriteFile(members);
                    break;
                case 3:
                    mm.printMemberInfo(members);
                    break;
                default:
                    System.out.print("\nYou have selected an invalid option.\n\n");
                    break;
            }
            choice = mm.getChoice();
        }
        System.out.println("\nGood Bye");
    }
}
```