MAKALAH MATA KULIAH STRUKTUR DATA

"SOURCE CODE LINKED LIST"

Dosen Pengampu : Rizqi Putri Nourma Budiarti, S.T.,M.T

Penyusun

Yuliana (NIM.3130021006)

PROGRAM STUDI SISTEM INFORMASI

UNIVERSITAS NAHDLATUL ULAMA

SURABAYA

TAHUN 2022

KATA PENGANTAR

Bismillahirohmanirohim
Assalamualaikum Wr. Wb.

Puji syukur saya ucapkan kehadirat **ALLAH SWT** atas segala rahmat-Nya sehingga makalah untuk memenuhi tugas Mata Kuliah Pengantar Sistem Informasi ini dapat tersusun sampai dengan selesai. Tugas ini ditujukan untuk memenuhi tugas Struktur Data.

Tidak lupa saya sampaikan terima kasih terhadap **ALLAH SWT** yang telah memberikan taufik, rahmat, dan hidayahNya kepada saya serta terima kasih atas bantuan dari pihak yang telah berkontribusi dengan memberikan sumbangan baik pikiran maupun materinya.

Makalah ini memberikan penjelasan pendapat mengenai penjabaran salah satu *Source Code*. Saya sebagai penulis menyadari dalam penulisan makalah ini yang mungkin belum sempurna, maka saya mengharapkan kritik dan saran yang bersifat membangun dan positif untuk mencapai kesempurnaan makalah ini. Semoga makalah ini memberi manfaat. Aamiin.

Wassalamualaikum Wr. Wb.

Surabaya, 27 Mei 2022

Penyusun

DAFTAR ISI

DAFTAR GAMBAR

BAB I

PENDAHULUAN

1.1 Latar Belakang

Struktur data adalah cara menyimpan atau merepresentasikan data di dalam komputer agar bisa dipakai secara efisien Sedangkan data adalah representasi dari fakta dunia nyata. Saat *programmer* ingin menyimpan data pada array, *programmer* diharuskan untuk mendefinisikan besar array terlebih dahulu, sedangkan *programmer* sering mengalokasikan array yang terbilang cukup besar. Hal ini tidak efektif karena seringkali yang dipakai tidak sebesar itu. Dan apabila *programmer* ingin menyimpan data lebih dari seratus data, maka hal itu tidak dapat dimungkinkan karena sifat array yang besarnya statik. Linked list adalah salah satu struktur data yang mampu menutupi kelemahan tersebut.

Senarai berkait (*linked list*) adalah suatu simpul (*node*) yang dikaitkan dengan simpul yang lain dalam suatu urutan tertentu. *Linked List* juga disebut bentuk struktur data yang berisi kumpulan data yang tersusun secara sekuensial, saling bersambungan, dinamis dan terbatas. Suatu simpul dapat berbentuk suatu struktur atau *class*. Simpul harus mempunyai satu atau lebih elemen struktur atau *class* yang berisi data.

Senarai berkait lebih efisien di dalam melaksanakan penyisipan-penyisipan dan penghapusan-penghapusan. Senarai berkait juga menggunakan alokasi penyimpanan secara dinamis, yang merupakan penyimpanan yang dialokasikan pada *runtime*. Karena di dalam banyak aplikasi, ukuran dari data itu tidak diketahui pada saat kompile, hal ini bisa merupakan suatu atribut yang baik juga.

1.2 Rumusan Masalah

Berdasarkan latar belakang diatas, maka bisa dirumuskan masalah sebagai berikut :

1.2.1 Bagaimana *source code* dari *Single Linked List* ?
1.2.2 Bagaimana *source code* dari *Double Linked List* ?
1.2.3 Bagaimana *source code* dari *Circular Linked List* ?
1.2.4 Bagaimana *source code* dari *Multiple Linked List* ?

1.3 Tujuan

Berdasarkan rumusan masalah yang telah dibuat, maka tujuan yang hendak dicapai ini adalah sebagai berikut :

1.3.1 Mengetahui dan menjelaskan *source code* dari *Single Linked List*
1.3.2 Mengetahui dan menjelaskan *source code* dari *Double Linked List*
1.3.3 Mengetahui dan menjelaskan *source code* dari *Circular Linked List*
1.3.4 Mengetahui dan menjelaskan *source code* dari *Multiple Linked List*

1.4 Manfaat

Berdasarkan rumusan masalah serta tujuan yang telah tercantum diatas, manfaat yang didapatkan yaitu untuk memberikan pemahaman mengenai logika dan cara kerja dari berbagai jenis *Linked List.* Selain itu kita dapat mengimplementasikan struktur data dari berbagai jenis *Linked List.*

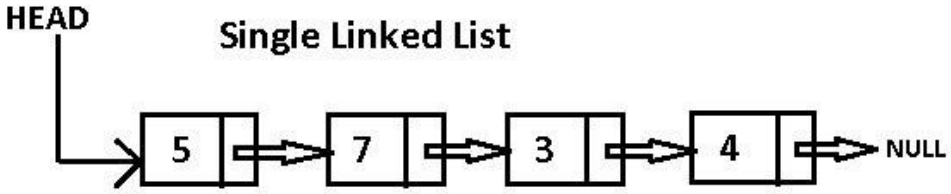
BAB II
PEMBAHASAN

2.1 Single Linked List

2.1.1 Pengertian

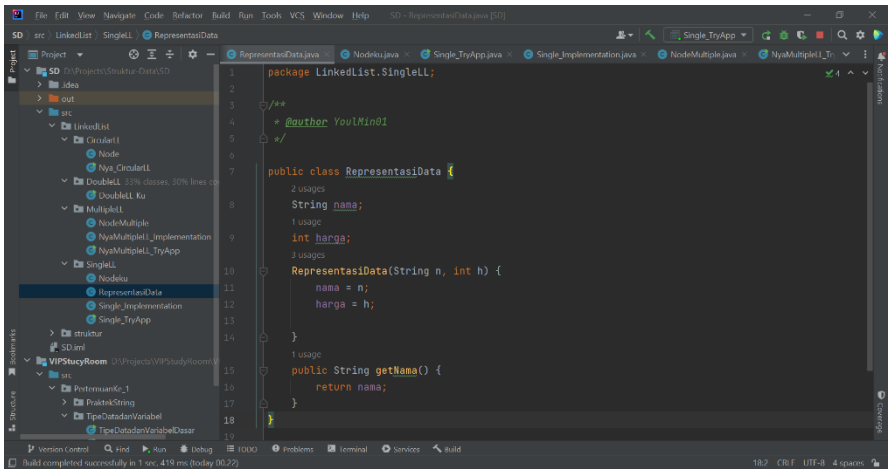
Single Linked List adalah salah satu *Linked List* yang hanya memiliki 1 variabel pointer dan menunjuk ke node selanjutnya

2.1.2 Gambaran



Gambar 2.1 *Single Linked List*

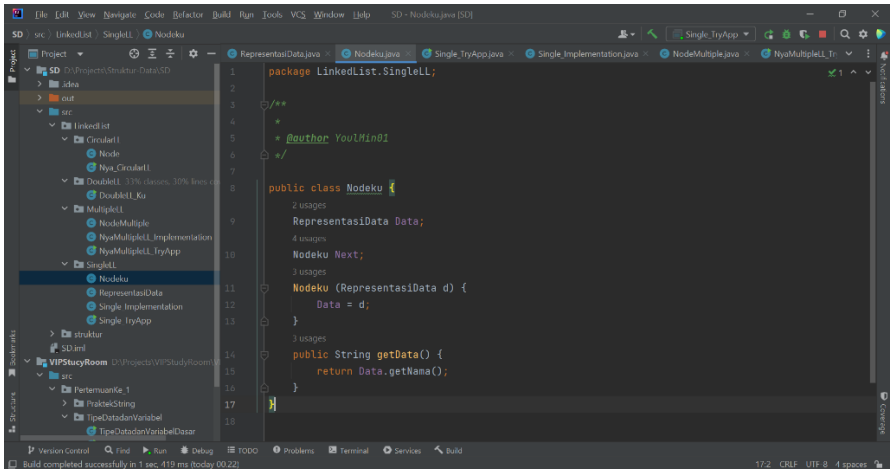
2.1.3 Source Code dan Penjelasan



```
package LinkedList.SingleLL;

/**
 * @author YoulMin01
 */

public class RepresentasiData {
    2 usages
    String nama;
    1 usage
    int harga;
    3 usages
    RepresentasiData(String n, int h) {
        nama = n;
        harga = h;

    }
    1 usage
    public String getNama() {
        return nama;
    }
}
```

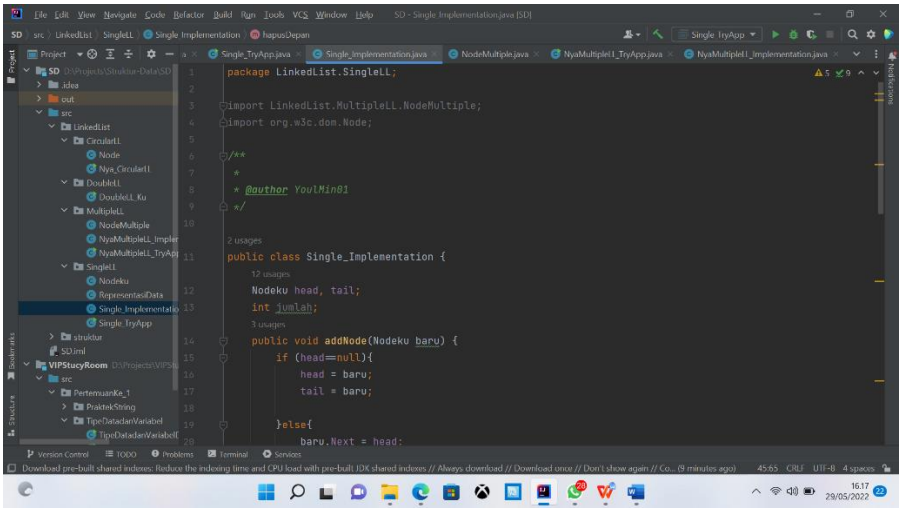
Gambar 2.2 Representasi Data

Dalam pembuatan *Single Linked List* membuat 4 Class yaitu Representasi Data, Class Node, Class Implementation (Class *Linked List*), dan Class App (Class Main). Pertama yaitu representasi data sebagai bentuk sederhana dari node class. Pada *Source Code Single Linked List* yang saya buat, saya menggunakan tipe data *String* dan *Integer*.



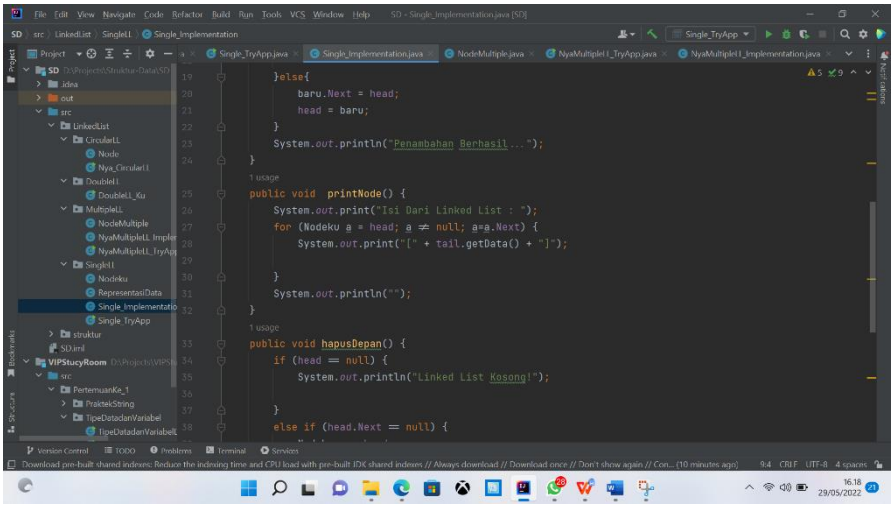
Gambar 2.3 Class Node *Single Linked List*

Selanjutnya terdapat Class Node adalah komponen penyusun untuk membuat *Single Linked List.* Pada bagian ini terdapat minimal 2 bagian yaitu data untuk menyimpan data dan *pointer* sebagai penghubungnya. Lalu kelas Node yang memiliki atribut data dan next, lalu constructor Node dengan parameter d.



Gambar 2.4 Class Single Linked List 1

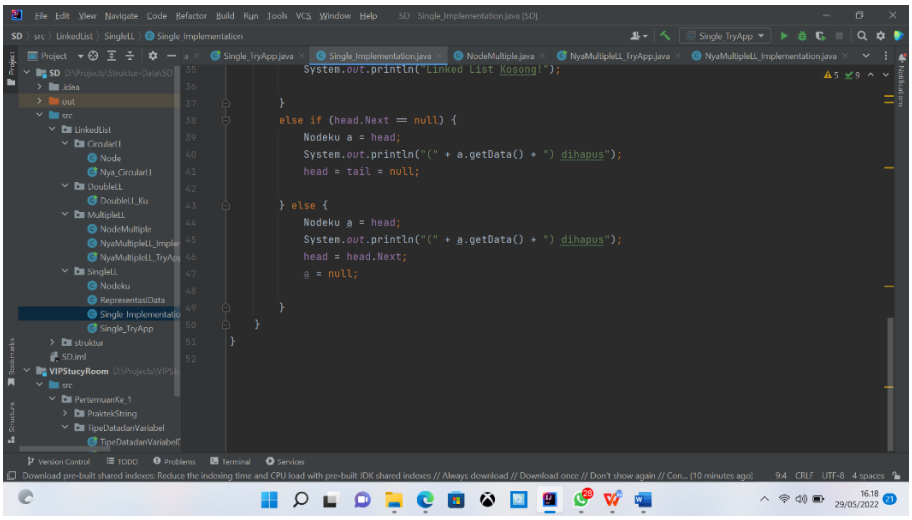
Langkah selanjutnya yaitu membuat kelas *Linked List* yang terdapat fungsi addNod, printNodes, dan hapusDepan. Buat variabel head untuk menyimpan ujung simpul/Node. Kemudian lengkapi dengan jumlah untuk menentukan jumlah variabel. Dimana terdapat fungsi addNode dengan parameter item yang berguna untuk menambahkan data sebagai Node. Tambahkan kondisi minimal 2 kondisi, yang pertama yaitu jika tail sama dengan NULL maka *Linked List* masih kosong. Maka head dan tail menunjuk ke Node yang sama dan baru. Namun jika head tidak sama dengan NULL artinya *Linked List* sudah berisi. Dengan itu Node yang baru akan selalu di depan karena tambah depan yaitu di depan head. Supaya konsisten, pointer head selalu diposisikan didepan/baru. Lalu head digeser ke baru, karena selalu menunjuk ke Node yang paling depan. Jadi setiap ada penambahan akan bergeser ke depan. Jangan sampai terbalik.



```
        }else{
            baru.Next = head;
            head = baru;
        }
        System.out.println("Penambahan Berhasil...");
    }
    1 usage
    public void  printNode() {
        System.out.print("Isi Dari Linked List : ");
        for (Nodeku a = head; a != null; a=a.Next) {
            System.out.print("[" + tail.getData() + "]");

        }
        System.out.println("");
    }
    1 usage
    public void hapusDepan() {
        if (head == null) {
            System.out.println("Linked List Kosong!");

        }
        else if (head.Next == null) {
```

Gambar 2.5 Class Single Linked List 2

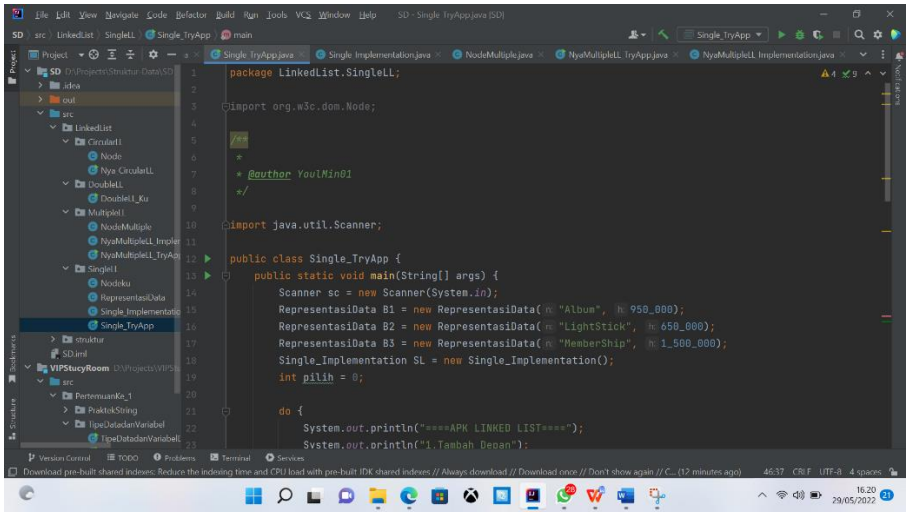
Selain fungsi addNode, pada Class *Linked List* juga terdapat fungsi printNode yang berguna untuk menampilkan. Gunakan perulangan for. Setelah itu cetak Node yang dibaca.



```
            System.out.println("Linked List Kosong!");

        }
        else if (head.Next == null) {
            Nodeku a = head;
            System.out.println("(" + a.getData() + ") dihapus");
            head = tail = null;

        } else {
            Nodeku a = head;
            System.out.println("(" + a.getData() + ") dihapus");
            head = head.Next;
            a = null;

        }
    }
}
```

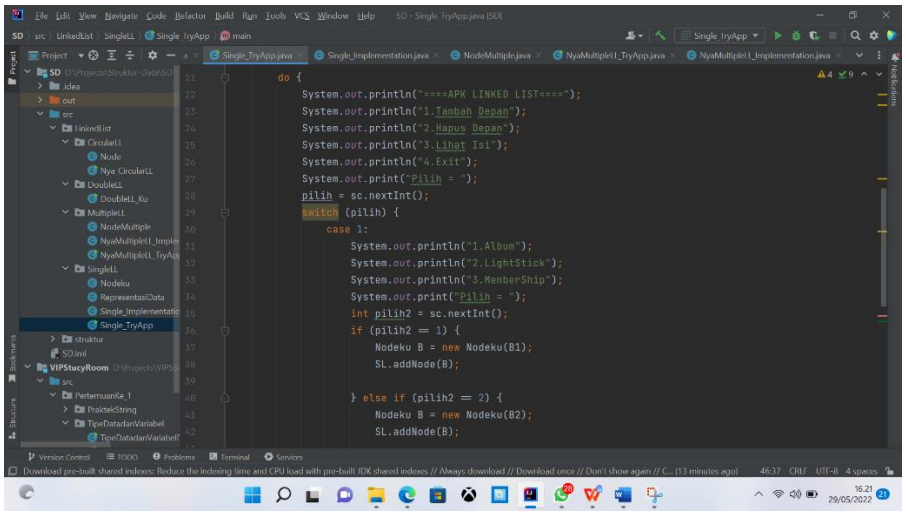
Gambar 2.6 Class Single Linked List 3

Fungsi hapusDepan berguna untuk menghapus data.



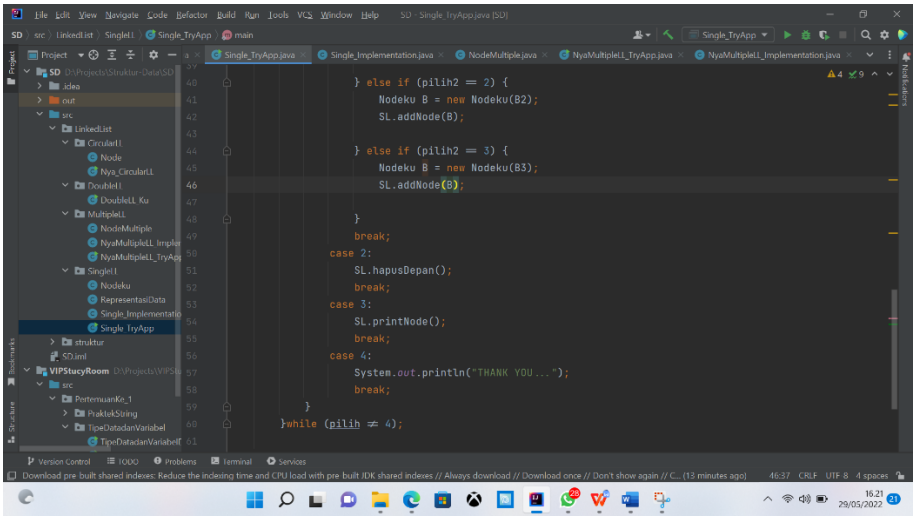
Gambar 2.7 Class Single Main 1

Kemudian yang terakhir adalah fungsi main. Lalu buatlah object baru dari *Single Linked List* seperti pada tampilan diatas. Setelah itu instansiasi data ke dalam sebuah Node.



Gambar 2.8 Class Single Main 2

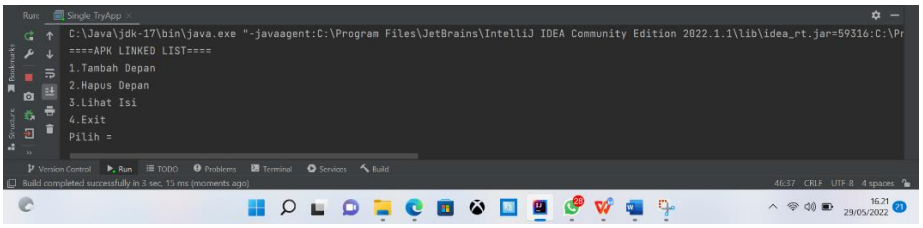
Buatlah menu program dan gunakan java.util.Scanner, pada tampilan di atas terdapat 4 menu. Gunakan perulangan do-while.



```
                } else if (pilih2 == 2) {
                    Nodeku B = new Nodeku(B2);
                    SL.addNode(B);

                } else if (pilih2 == 3) {
                    Nodeku B = new Nodeku(B3);
                    SL.addNode(B);

                }
                break;
            case 2:
                SL.hapusDepan();
                break;
            case 3:
                SL.printNode();
                break;
            case 4:
                System.out.println("THANK YOU...");
                break;
        }
    }while (pilih != 4);
```

Gambar 2.9 Class Single Main 3

Membuat case untuk meletakkan input. Lalu buat pengkodisian, semisal pilih 2 sama dengan 1, maka nanti akan berupa object dari B1.



```
C:\Java\jdk-17\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.1.1\lib\idea_rt.jar=59316:C:\Pr
====APK LINKED LIST====
1.Tambah Depan
2.Hapus Depan
3.Lihat Isi
4.Exit
Pilih =
```

Gambar 2.10 Hasil Run Single

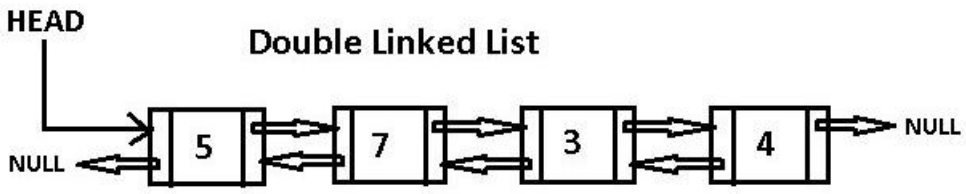
Setelah melakukan berbagai tahapan di atas maka lakukan “RUN” pada file sehingga muncul tampilan seperti yang terlihat pada tampilan. Dari tampilan tersebut bisa dilakukan input, lalu jika ingin mencetaknya lakukan klik “Enter” untuk menampilkan/”RUN”.

2.2 Double Linked List

2.2.1 Pengertian

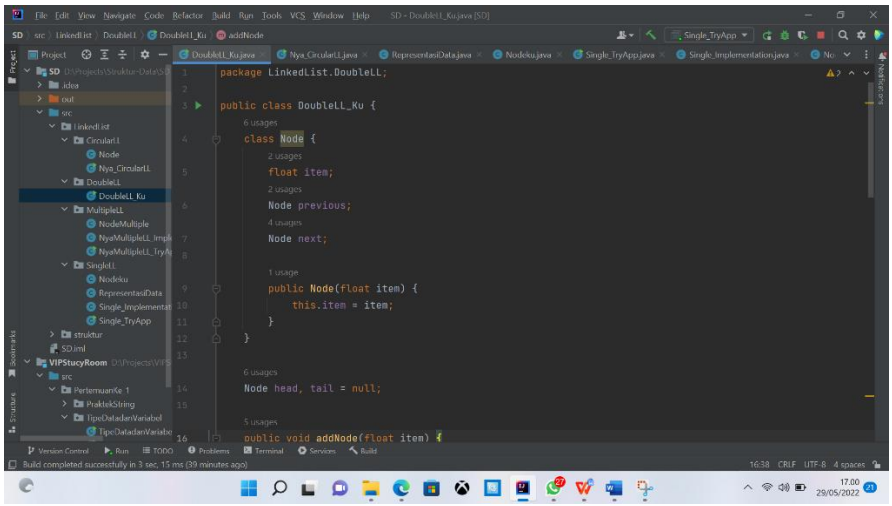
Double Linked List adalah salah satu *Linked List* yang memiliki 2 variabel pointer yang menunjuk ke node selanjutnya dan sebelumnya serta head dan tail menunjuk ke NULL.

2.2.2 Gambaran



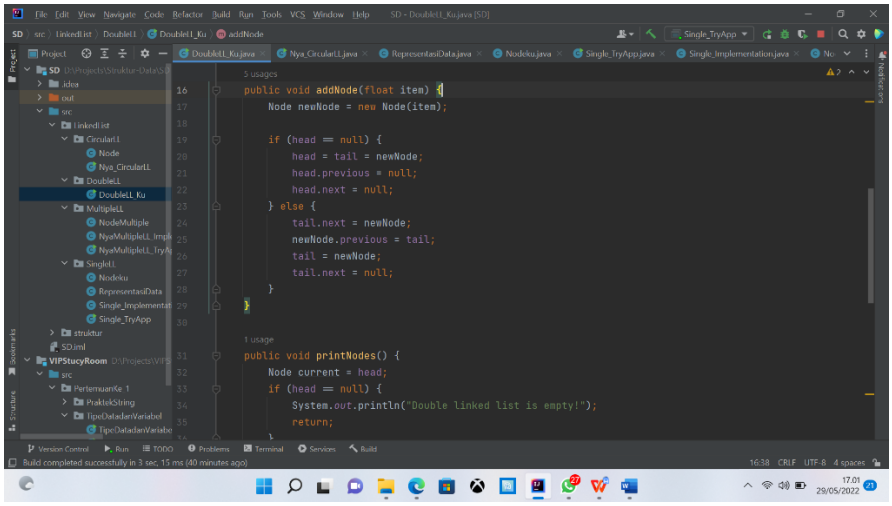
Gambar 2.11 Double Linked List

2.2.3 Source Code dan Penjelasan



Gambar 2.12 Class Node Double Linked List

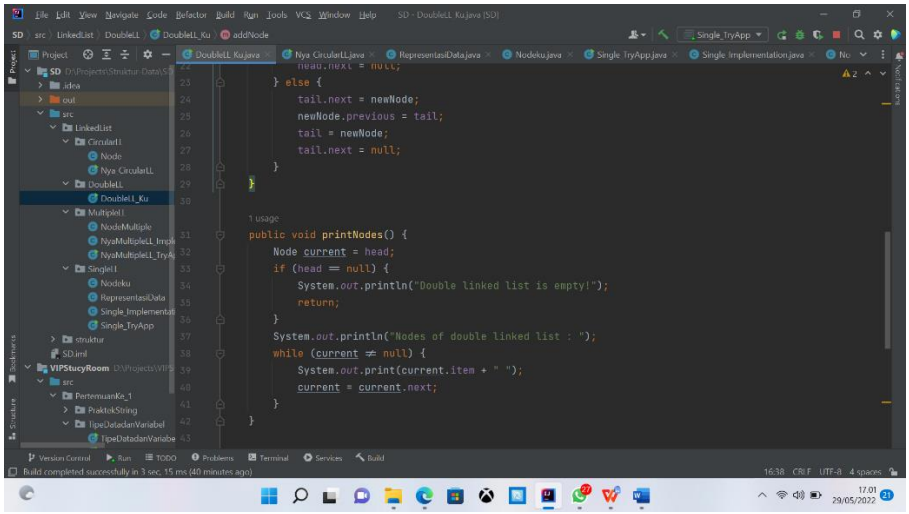
Dalam pembuatan *Double Linked List* membuat 3 Class yaitu Class Node, Class Implementation (Class *Linked List*), dan Class App (Class Main). Pertama yaitu membuat Node Class. Pada *Source Code Double Linked List* yang saya buat, saya menggunakan tipe data *Float*. Pada Class Node terdapat atribut item, previous, dan next. Lalu constructor Node dengan parameter item dimana atribut item akan berisi data dari parameter. Lalu variabel head dan tail berisi nilai NULL.



Gambar 2.13 Class Double Linked List 1

Langkah selanjutnya yaitu membuat kelas *Linked List* yang terdapat fungsi addNode dan printNodes. Dimana terdapat fungsi addNode dengan parameter item yang berguna untuk menambahkan data sebagai Node. Pada kelas Node terdapat newNode dengan menambahakan parameter item. Kemudian dicek apakah head memiliki nilai NULL. Jika benar, maka head dan tail diisi dengan newNode lalu previous dan next diisi dengan NULL, karena head masih bernilai NULL yang berarti data yang dimasukan adalah data pertama. Lalu jika kondisi salah atau sudah terdapat data pada head data dimasukkan pada posisi tail

next atau setelah data paling akhir dengan newNode, lalu newNode previous sebagai tail, dan newNode atau data terbaru akan menjadi tail, lalu tail next data setelah tail di isi dengan NULL.



```
        } else {
            tail.next = newNode;
            newNode.previous = tail;
            tail = newNode;
            tail.next = null;
        }
    }

    1 usage
    public void printNodes() {
        Node current = head;
        if (head == null) {
            System.out.println("Double linked list is empty!");
            return;
        }
        System.out.println("Nodes of double linked list : ");
        while (current != null) {
            System.out.print(current.item + " ");
            current = current.next;
        }
    }
```

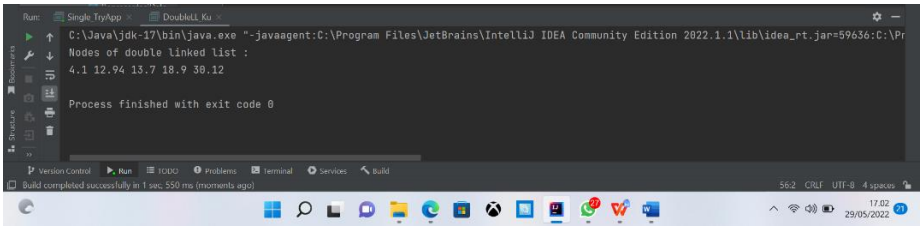
Gambar 2.14 Class Double Linked List 2

Selain addNode pada kelas *Linked List* ini terdapat fungsi printNodes. Fungsi printNodes ini berguna untuk menampilkan Nodes yang telah di tambahkan sebelumnya. Dalam fungsi ini menggunakan variabel current sebagai acuan data saat ini yang dimulai dari head atau data yang paling awal. Lalu dilakukan pengecekan jika headnya NULL maka saya mencetak bahwa *Double Linked List* tersebut kosong. Lalu while digunakan untuk melakukan perulangan dengan kondisi current tidak sama dengan NULL, cetak dengan current.item untuk memanggil item Node, kemudian lalukan current.next untuk memindahkan posisi ke data selanjutnya.

```
    public static void main(String[] args) {
        DoubleLL_Ku Dll = new DoubleLL_Ku();

        Dll.addNode( item: 4.10f);
        Dll.addNode( item: 12.94f);
        Dll.addNode( item: 13.7f);
        Dll.addNode( item: 18.9f);
        Dll.addNode( item: 30.12f);

        Dll.printNodes();
        System.out.println("");
    }
}
```

Gambar 2.15 Class Double Main

Pada tahap berikutnya saya membuat kelas baru yaitu kelas main. Pertama-pertama saya melakukan instansiasi kelas DoubleLL_Ku dengan nama Dll, lalu memanggil fungsi addNode untuk menambahkan Node dengan parameter sebagai value atau nilai datanya. Tahap terakhir dilakukan printNodes untuk mencetak Nodes yang sudah ditambahkan.



Gambar 2.16 Hasil Run Double

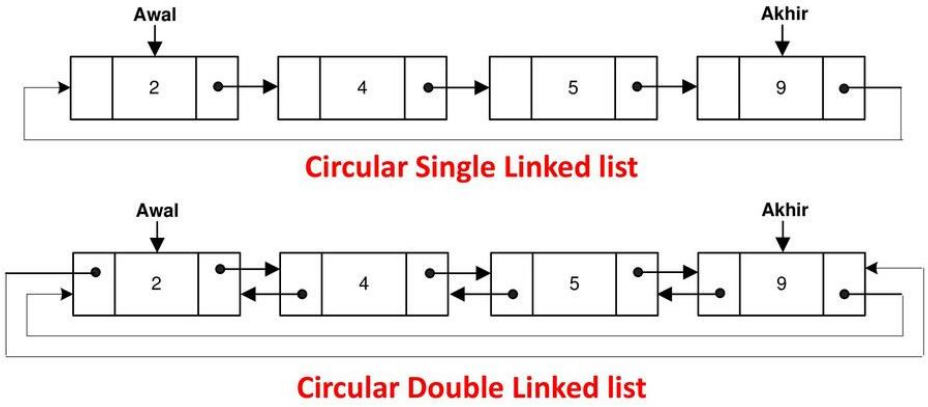
Setelah melakukan beberapa tahapan dari Class Node, Class *Linked List*¸dan Class Main saya melakukan "RUN" pada program dengan hasil seperti pada gambar. Hasil "RUN" pada gambar diatas menunjukkan bahwa program sudah sukses.

2.3 Circular Linked List

2.3.1 Pengertian

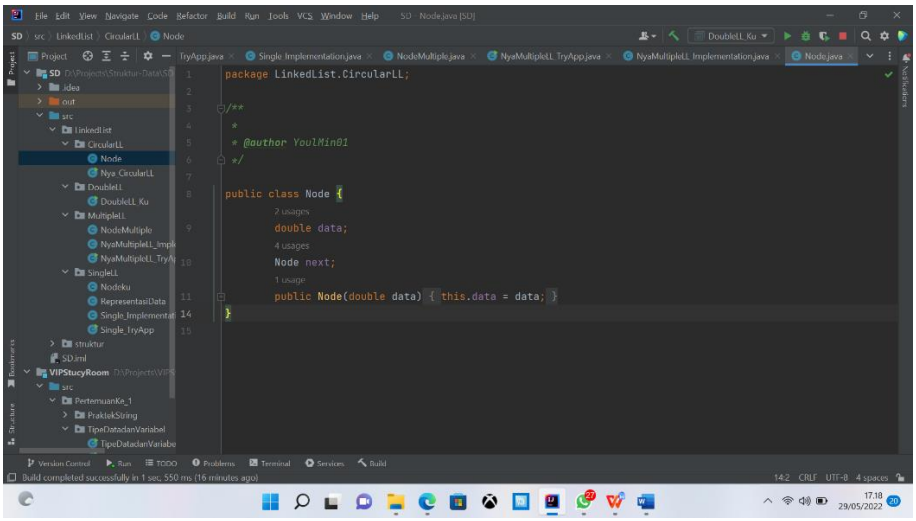
Circular Linked List adalah salah satu *Linked List* dimana node terakhir menunjuk ke node pertama. Jadi tidak ada pointer yang menunjuk NULL. Perlu diketahui bahwa *Circular Linked List* ada 2 yaitu *Circular Single Linked List* dan *Circular Double Linked List*. Namun dalam makalah ini saya menggunakan *Circular Single Linked List.*

2.3.2 Gambaran



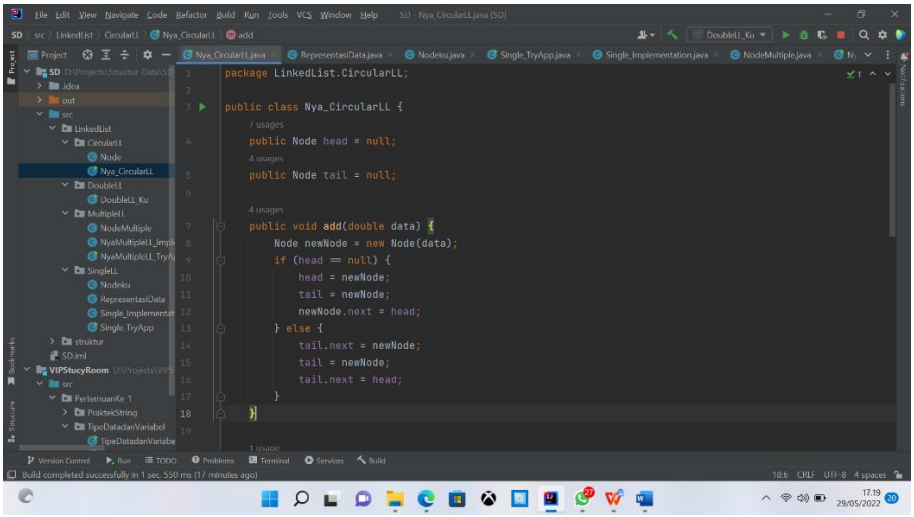
Gambar 2.17 Circular Linked List

2.3.3 Source Code dan Penjelasan



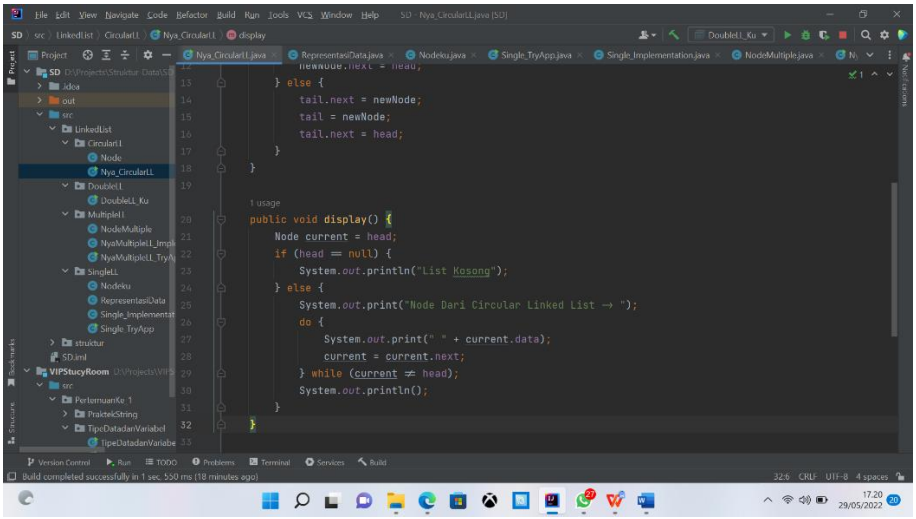
Gambar 2.18 Class Node Circular Linked List

Dalam pembuatan *Circular Linked List* membuat 3 Class yaitu Class Node, Class Implementation (Class *Linked List*), dan Class App (Class Main). Pertama yaitu membuat Node Class. Pada *Source Code Circular Linked List* yang saya buat, saya menggunakan tipe data *Double.* Pada Class Node terdapat atribut data dan next serta constructor Node dengan parameter data, dimana atribut data akan di isi oleh parameter data. Lalu terdapat deklarasi variabel head dan tail dengan NULL.



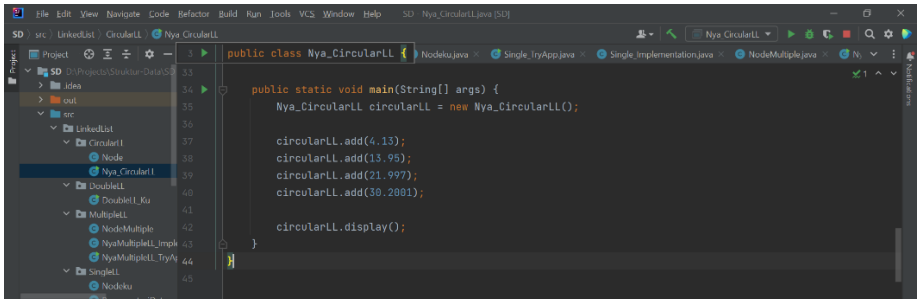
Gambar 2.19 Class Circular Linked List 1

Langkah selanjutnya yaitu membuat Class *Linked List* yang terdapat fungsi add dan fungsi display. Fungsi add ini berguna untuk menambahkan Node dengan parameter data. Pertama terdapat instansiasi kelas Node dengan nama newNode dan parameter data. Lalu melakukan pengecekan jika head sama dengan NULL, maka head dan tail diisi dengan newNode lalu newNode.next sebagai head. Jika head sudah memiliki data maka tail.next diisi dengan newNode, tail berada pada posisi newNode, lalu tail.next disambungkan dengan head.



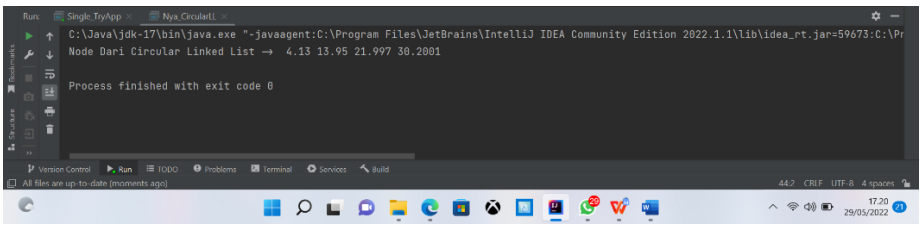
Gambar 2.20 Class Circular Linked List 2

Selain fungsi add pada kelas *Linked List* ini terdapat fungsi display. Fungsi display berguna untuk menampilkan data Node. Di dalamnya variabel current yang berisi posisi head. Lalu dilakukan pengecekan. Jika head kosong, maka cetak bahwa "List Kosong". Namun jika tidak lakukan pengulangan do while dengan kondisi current tidak sama dengan head. Lalu data dicetak dan melakukan current.next untuk memindahkan posisi saat ini ke data selanjutnya.



Gambar 2.21 Class Circular Main

Pada tahap berikutnya saya membuat kelas baru yaitu kelas main. Pada kelas ini terdapat instansiasi kelas Nya_CircularLL dengan nama circularLL, lalu memanggil fungsi add beserta data nya, lalu memanggil fungsi display untuk menampilkan data Nodenya.



Gambar 2.22 Hasil Run Circular

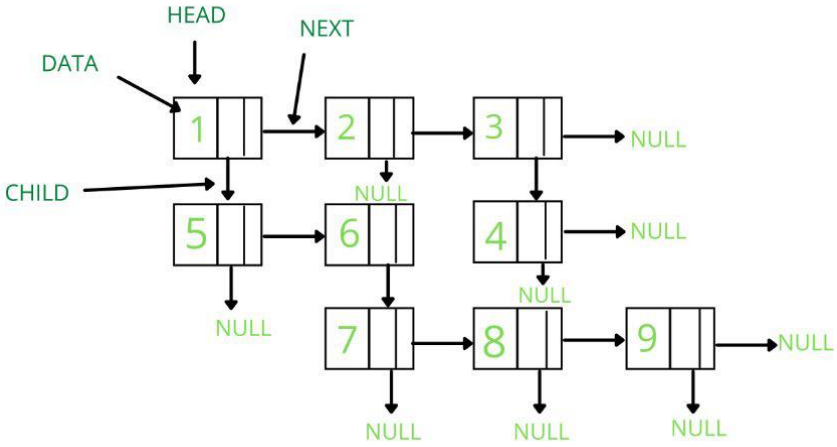
Setelah melakukan beberapa tahapan di atas, lakukan proses "RUN" program hingga muncul tampilan sesuai pada gambar di atas.

2.4 Multiple Linked List

2.4.1 Pengertian

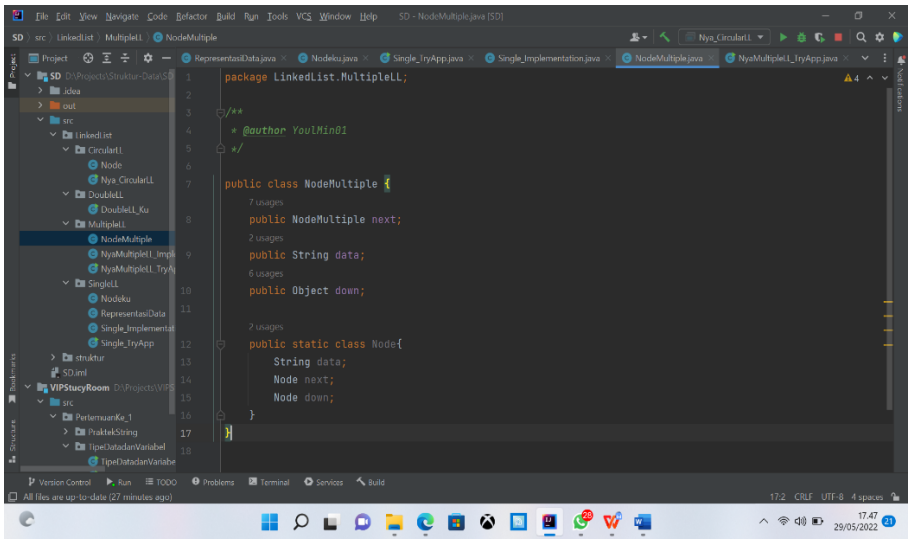
Multiple Linked List adalah salah satu *Linked List* yang memiliki lebih dari 2 variabel pointer.

2.4.2 Gambaran



Gambar 2.23 Multiple Linked List

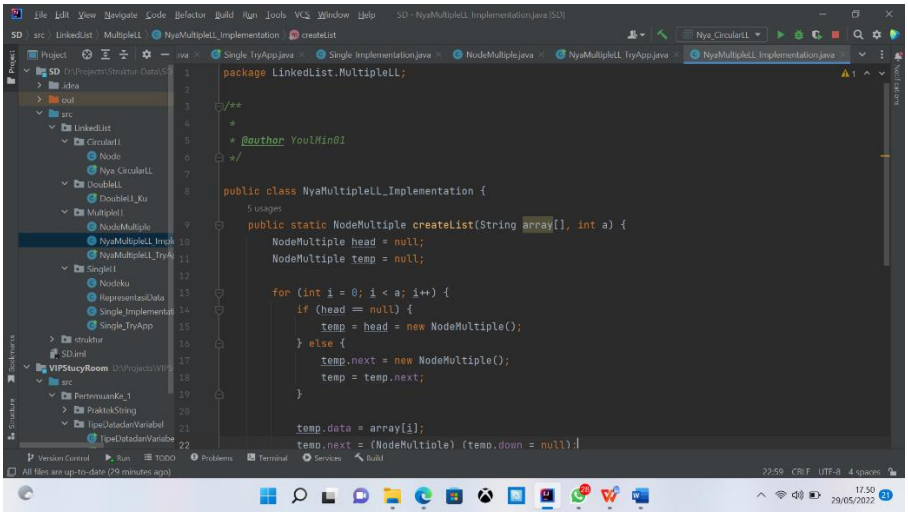
2.4.3 Source Code dan Penjelasan



```
package LinkedList.MultipleLL;

/**
 * @author YoulMin01
 */

public class NodeMultiple {
    public NodeMultiple next;
    public String data;
    public Object down;

    public static class Node{
        String data;
        Node next;
        Node down;
    }
}
```

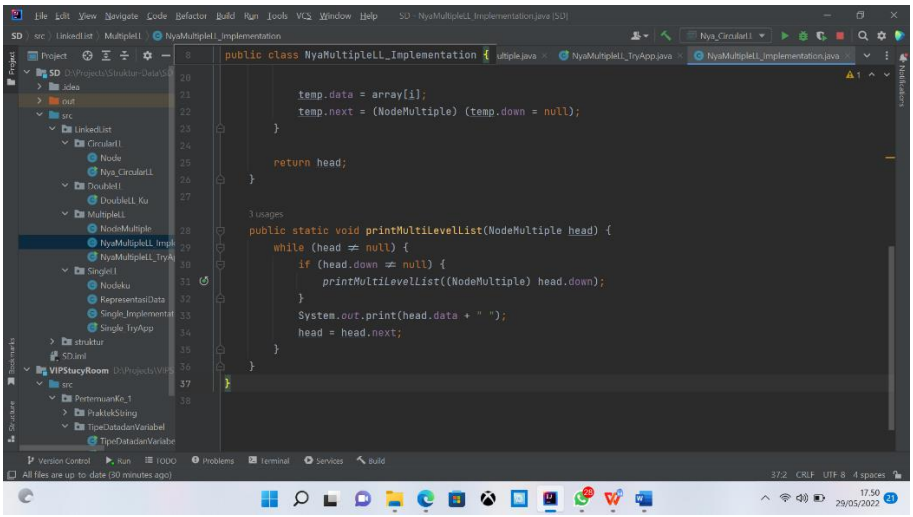
Gambar 2.24 Class Node Multiple Linked List

Dalam pembuatan *Multiple Linked List* membuat 3 Class yaitu Node Class, Class Implementation(Class *Linked List*), dan Class App (Class Main). Pertama yaitu membuat Node Class. Pada *Source Code Multiple Linked List* yang saya buat, saya menggunakan tipe data *String.* Pertama terdapat kelas Node yang memiliki atribut data, next, dan down.



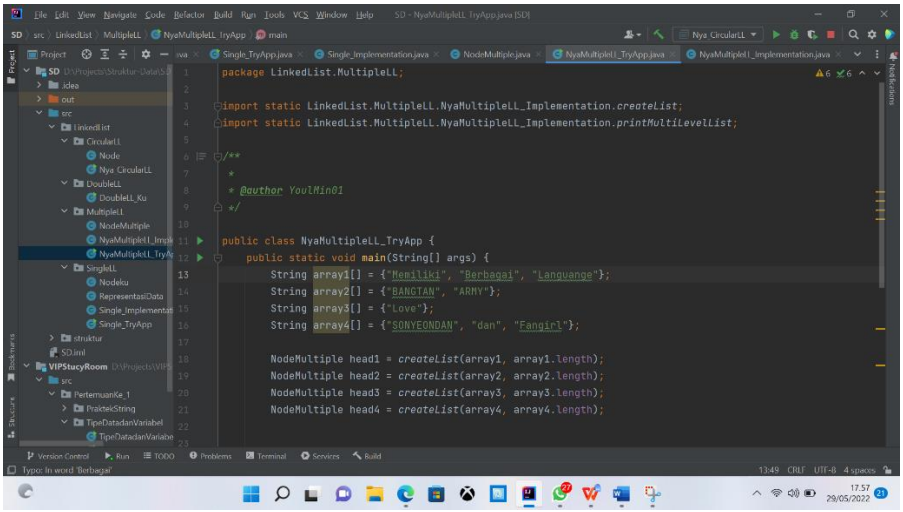
Gambar 2.25 Class Multiple Linked List 1

Langkah selanjutnya yaitu membuat Class *Linked List* yang terdapat fungsi createList dan fungsi printMultilevelList. Fungsi createList memiliki parameter array dan "a" yang di dalamnya terdapat variabel head dan temp yang berisi NULL, lalu dilakukan perulangan sampai nilai ke "a". Di dalam perulangan terdapat pengkondisian jika head NULL, maka isi temp dan head dengan new Node. Namun jika tidak, isi temp.next dengan new Node lalu posisi temp dimajukan dengan next. Setelah itu temp.data berisi array dengan index ke i, iterasi dari array, lalu next dan down di isi dengan NULL. Fungsi inilah yang mengembalikan nilai head.



Gambar 2.26 Class Multiple Linked List 2

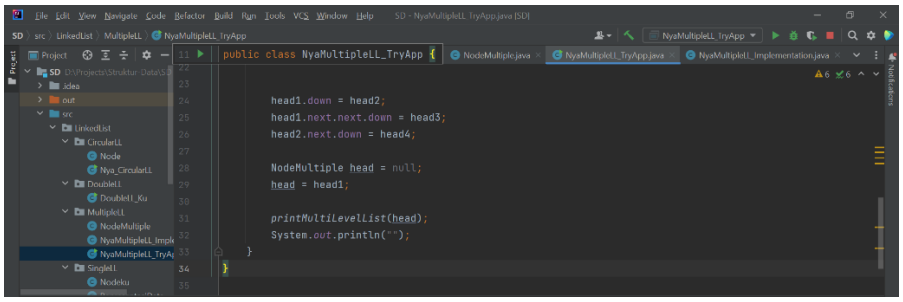
Selain fungsi creatList pada kelas *Linked List* ini terdapat juga fungsi printMultilevelList. Fungsi printMultiLevelList berguna untuk menampilkan Nodes yang telah dibuat dengan parameter head. Lalu di dalamnya terdapat perulangan dengan kondisi jika head tidak sama dengan NULL, maka lakukan pengkondisian juga. Namun jika head.down tidak sama dengan NULL, maka lakukan fungsi rekursif dengan parameter head.down. Lalu cetak head.data dan majukan posisi head dengan next.



```
package LinkedList.MultipleLL;

import static LinkedList.MultipleLL.NyaMultipleLL_Implementation.createList;
import static LinkedList.MultipleLL.NyaMultipleLL_Implementation.printMultiLevelList;

/**
 *
 * @author YoulMin01
 */

public class NyaMultipleLL_TryApp {
    public static void main(String[] args) {
        String array1[] = {"Memiliki", "Berbagai", "Languange"};
        String array2[] = {"BANGTAN", "ARMY"};
        String array3[] = {"Love"};
        String array4[] = {"SONYEONDAN", "dan", "Fangirl"};

        NodeMultiple head1 = createList(array1, array1.length);
        NodeMultiple head2 = createList(array2, array2.length);
        NodeMultiple head3 = createList(array3, array3.length);
        NodeMultiple head4 = createList(array4, array4.length);
```

Gambar 2.27 Class Multiple Main 1

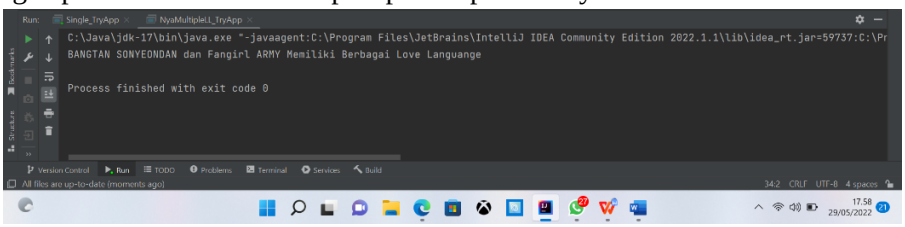
Pada tahap berikutnya saya membuat kelas baru yaitu kelas main. Pada fungsi main ini berisi deklarasi array yang akan dimasukkan menjadi Node dengan memasukkan array satu persatu dengan memanggil fungsi createList, disetiap pemanggilan fungsi dimasukkan ke dalam variabel head dengan iterasi.



```
public class NyaMultipleLL_TryApp {

        head1.down = head2;
        head1.next.next.down = head3;
        head2.next.down = head4;

        NodeMultiple head = null;
        head = head1;

        printMultiLevelList(head);
        System.out.println("");
    }
}
```

Gambar 2.28 Class Multiple Main 2

Lalu memberikan pointer untuk setiap head, seperti head1 memiliki down head2, selanjutnya head1.next.next.down yang berarti array1 element 3 memiliki down/child head3, dan terakhir head2.next.down head2 element 2 memiliki down head4. Kemudian head di kosongkan untuk mengembalikan posisi head ke head1. Selanjutnya cetak semua Node dengan memanggil fungsi printMultiLeveList dengan parameter head dari tiap-tiap head pada array.



```
C:\Java\jdk-17\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.1.1\lib\idea_rt.jar=59737:C:\Pr
BANGTAN SONYEONDAN dan Fangirl ARMY Memiliki Berbagai Love Languange

Process finished with exit code 0
```

Gambar 2.29 Hasil Run Multiple

Setelah melakukan berbagai tahapan di atas maka lakukan “RUN” pada file sehingga muncul tampilan seperti yang terlihat pada tampilan.

BAB III
PENUTUP

3.1 Kesimpulan

Senarai berkait (*linked list*) adalah suatu simpul (*node*) yang dikaitkan dengan simpul yang lain dalam suatu urutan tertentu. *Linked List* juga disebut bentuk struktur data yang berisi kumpulan data yang tersusun secara sekuensial, saling bersambungan, dinamis dan terbatas. *Linked List* memiliki beberapa jenis diantaranya *Single Linked List, Double Linked List, Circular Linked List,* dan *Multiple Linked List*.

Masing-masing memiliki perbedaan seperti diantaranya yaitu *Single Linked List* yang hanya memiliki 1 variabel pointer yang menunjuk ke node selanjutnya, sedangkan *Double Linked List* memiliki 2 variable pointer yang menunjuk ke kode selanjutnya dan kode sebelumnya. Untuk *Circular Linked List* tidak memiliki nilai NULL untuk medan sambungnya, sedangkan *Multiple Linked List* memiliki beberapa *Linked List* yang saling berkaitan serta memiliki karakteristik yang berbeda.

3.2 Saran

Adapun saran yang akan saya sampaikan yaitu perlu dilakukan pembelajaran penggunaan Linked List agar lebih memiliki pemahaman yang kurang lebih cukup untuk mengimplementasikannya. Karena menurut beberapa sumber yang saya baca telah menunjukkan keuntungan dari penggunaan Linked List yang dapat mempermudah programmer dalam menyimpan data.

DAFTAR PUSTAKA

Cipta Ramadhani,S.T.,MEng. 2015 Algoritma dan Struktur Data dengan Bahasa Java. Jakarta

Rosa. 2018. Struktur Data Terapan Dalam Berbagai Bahasa Pemrograman. Modula. Bandung

Tubagus Eza. *Linked List dan Implementasi di Jawa.*
URL : https://youtu.be/SR_tdSvGD88 diakses pada tanggal 27 Mei 2022.

Unkown3 Contoh Program Linked List Sederhana Pada Jawa.
URL : https://balog18.blogspot.com/2019/10/3-contoh-program-linked-list-sederhana.html diakses pada tanggal 27 Mei 2022.

Vicky Royibha. Makalah Struktur Data Variasi Linked List.
URL :
https://www.academia.edu/32350910/MAKALAH_STRUKTUR_DATA_VARIASI_LINKED_LIST diakses pada 27 Mei 2022.