  
 Name : Azaria Cindy Sahasika

Number Id : 2341760169 / 06

Class : 1G – Business Information System

Lesson : Algorithm and Data Structure

Material : Material 12 – Graph

Github Link : <https://github.com/azariacindy/algorithm-ds>

JOBSHEET XII

Graph

1. Tujuan Praktikum

Setelah melakukan praktikum ini, mahasiswa mampu:

1. memahami model graph;
2. membuat dan mendeklarasikan struktur algoritma graph;
3. menerapkan algoritma dasar graph dalam beberapa studi kasus.
4. Praktikum

12.1 Implementasi Graph menggunakan Linked List

12.1.1 Tahapan Percobaan

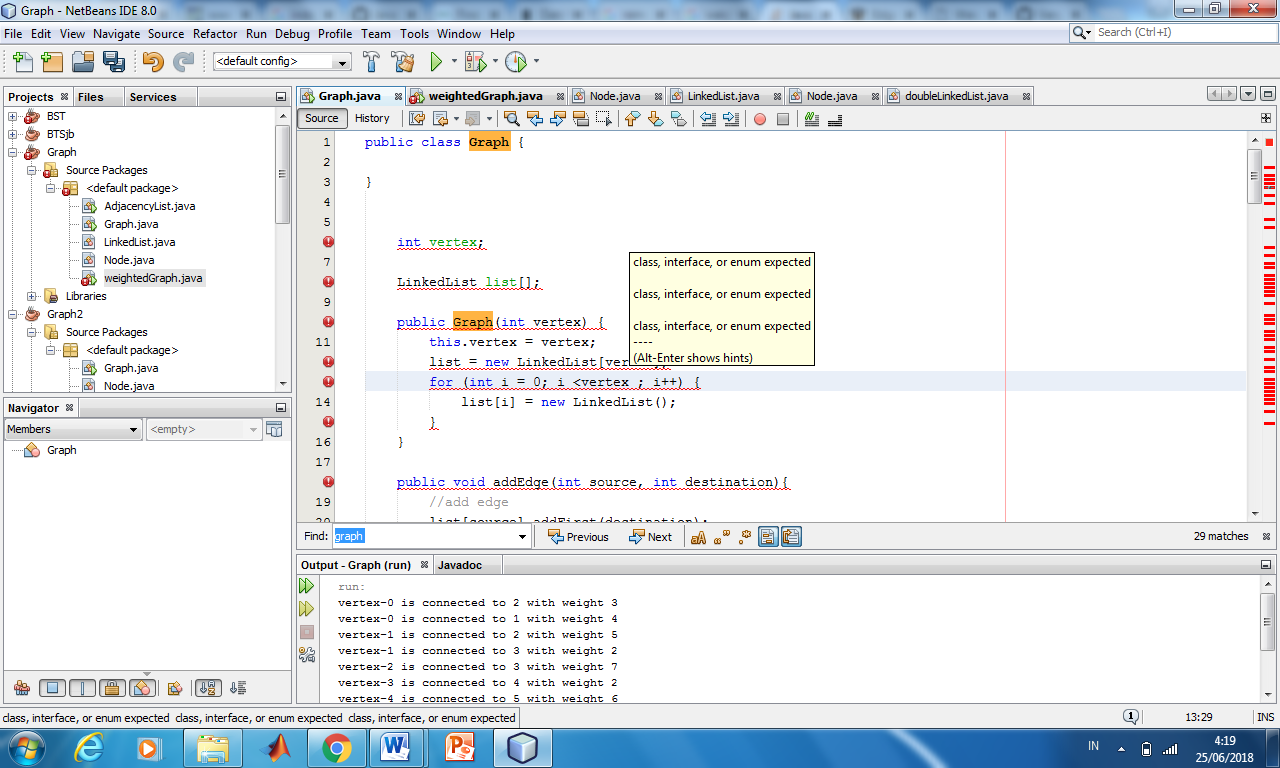
**Waktu percobaan (30 menit)**

Pada percobaan ini akan diimplementasikan Graph menggunakan Linked Lists untuk merepresentasikan graph adjacency. Silakan lakukan langkah-langkah praktikum sebagai berikut.

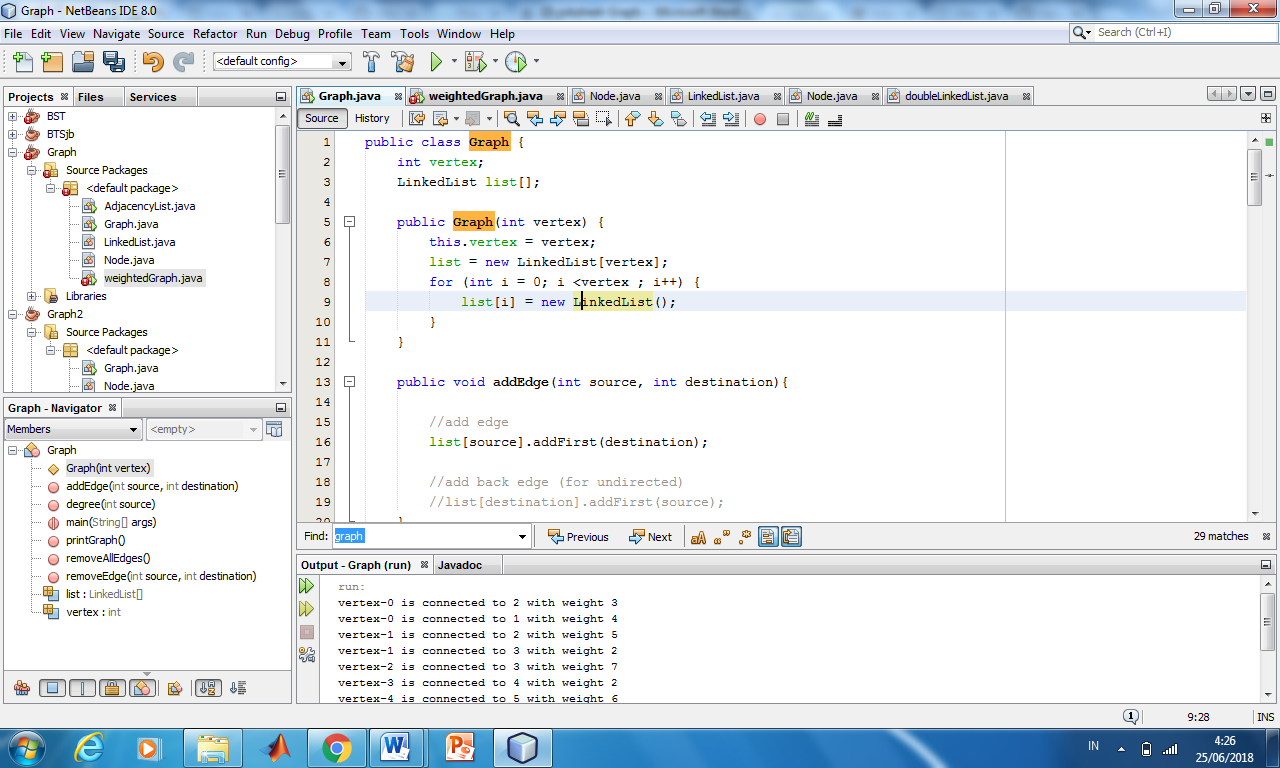
1. Buatlah class **Node**, dan class **Linked Lists** sesuai dengan praktikum **Double** **Linked Lists.**

|  |
| --- |
| Graph |
| vertex: int  LinkedList: List  right: Node |
| addEdge(source: int, destination: int): void  degree(source: int): void  removeEdge(source: int, destination: int): void  removeAllEdges()  printGraph() |

1. Tambahkan class **Graph** yang akan menyimpan method-method dalam graph dan juga method main().



1. Di dalam class **Graph**, tambahkan atribut **vertex** bertipe integerdan **list[]** bertipe LinkedList.

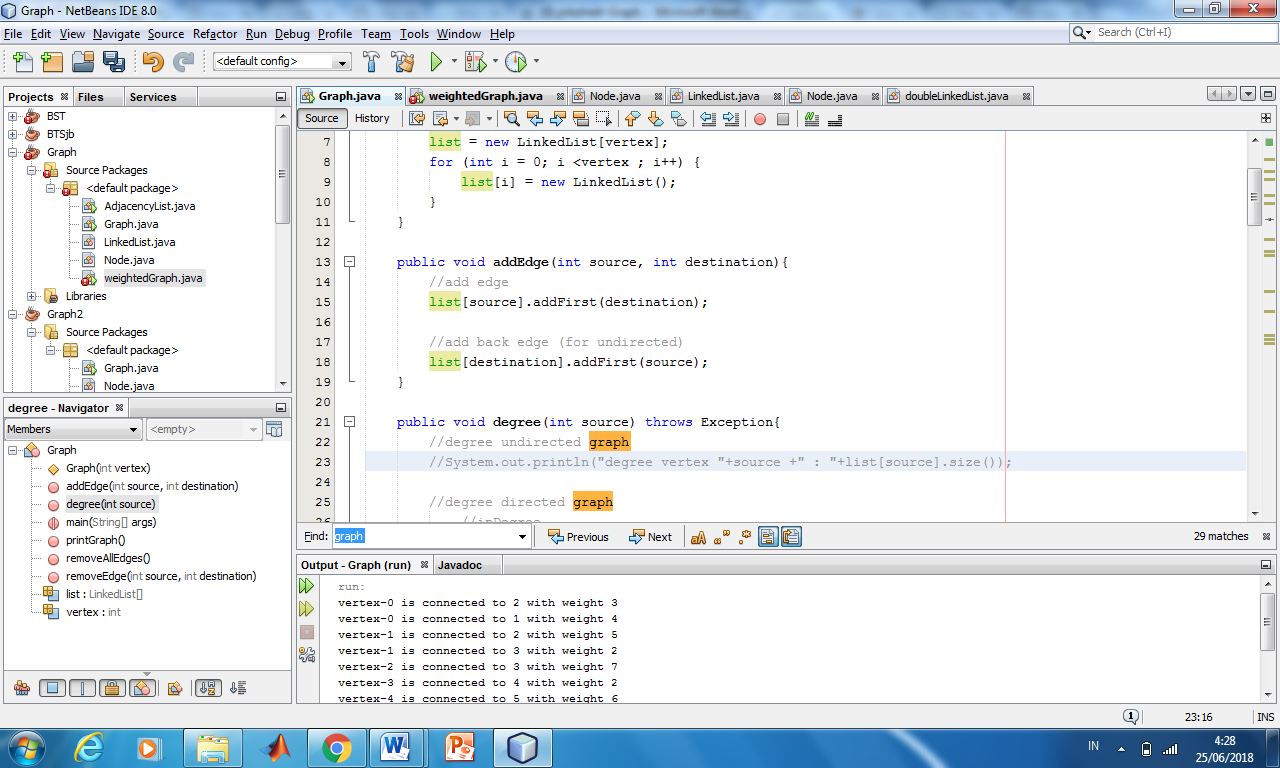


1. Tambahkan konstruktor default untuk menginisialisasi variabel vertex dan menambahkan perulangan untuk jumlah vertex sesuai dengan jumlah length array yang telah ditentukan.

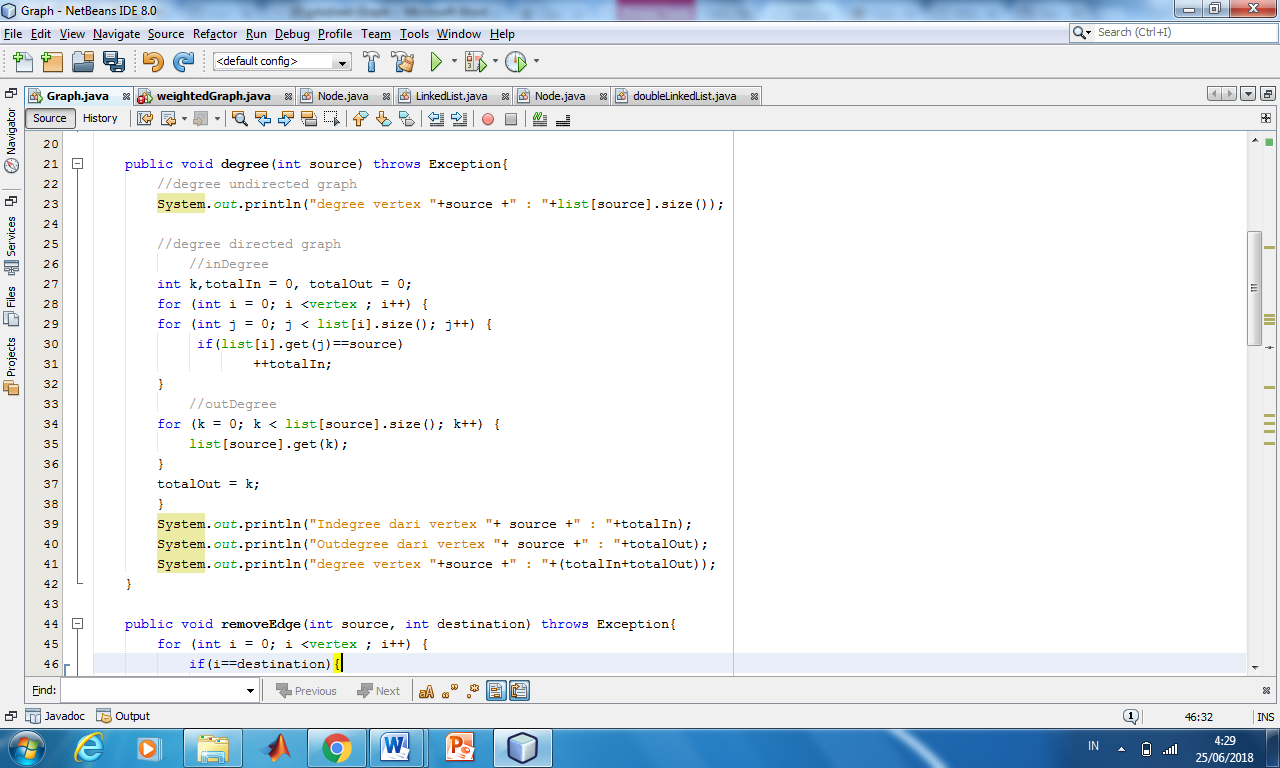
Graphical user interface, text, application

Description automatically generated

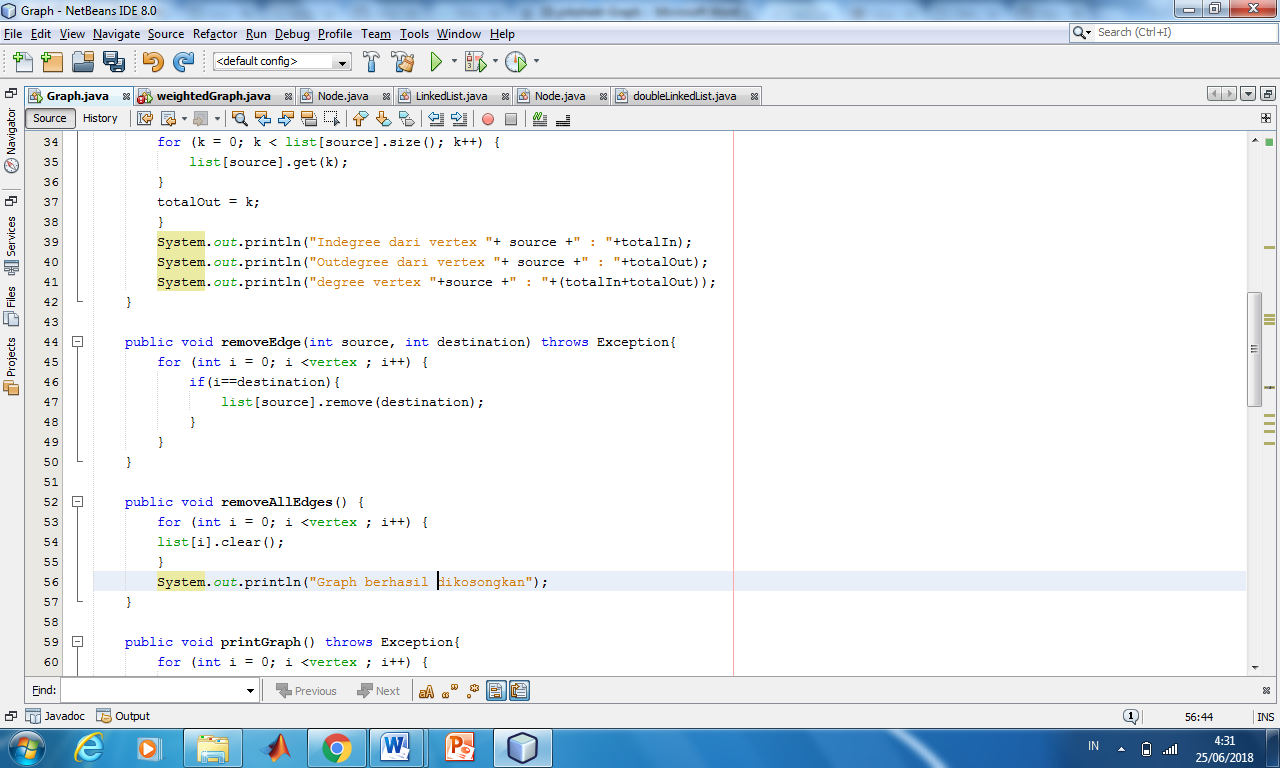
1. Tambahkan method **addEdge()**. Jika yang akan dibuat adalah graph berarah, maka yang dijalankan hanya baris pertama saja. Jika graph tidak berarah yang dijalankan semua baris pada method **addEdge()**.



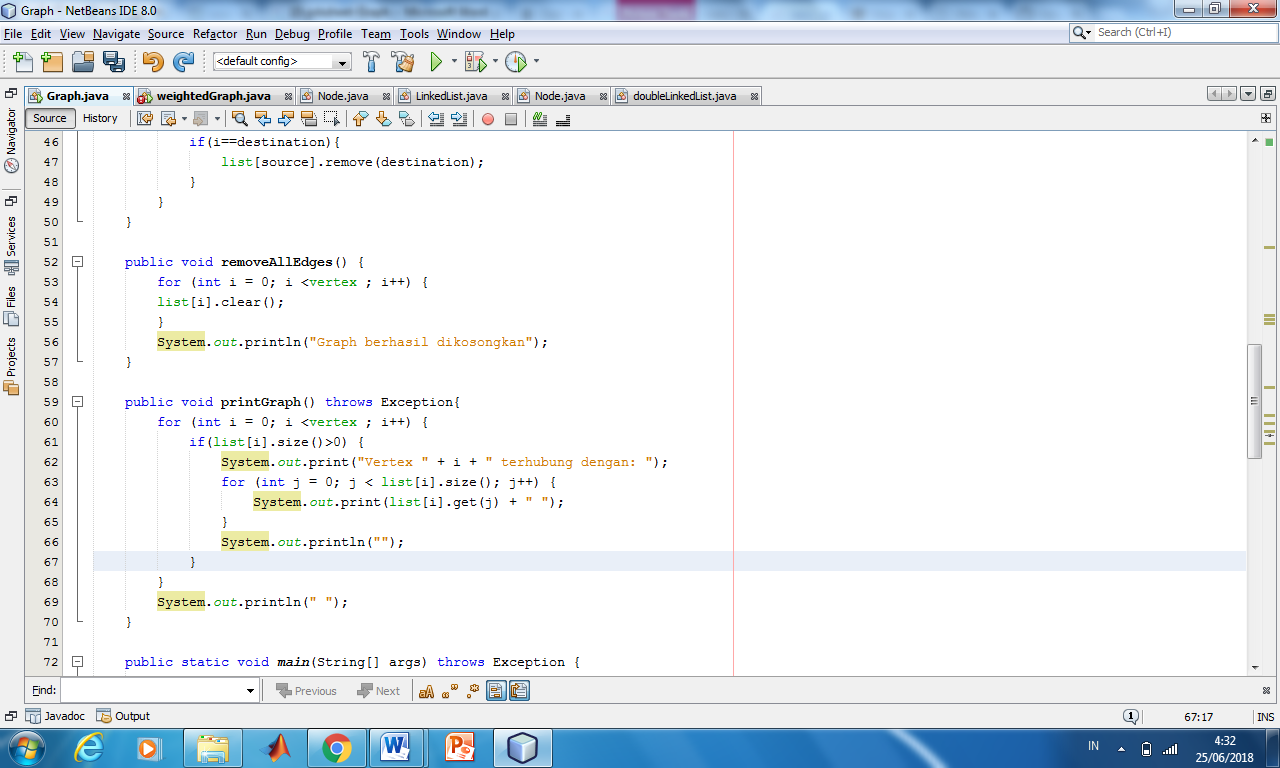
1. Tambahkan method **degree()** untuk menampilkan jumlah derajat lintasan pada suatu vertex. Di dalam metode ini juga dibedakan manakah statement yang digunakan untuk graph berarah atau graph tidak berarah. Eksekusi hanya sesuai kebutuhan saja.



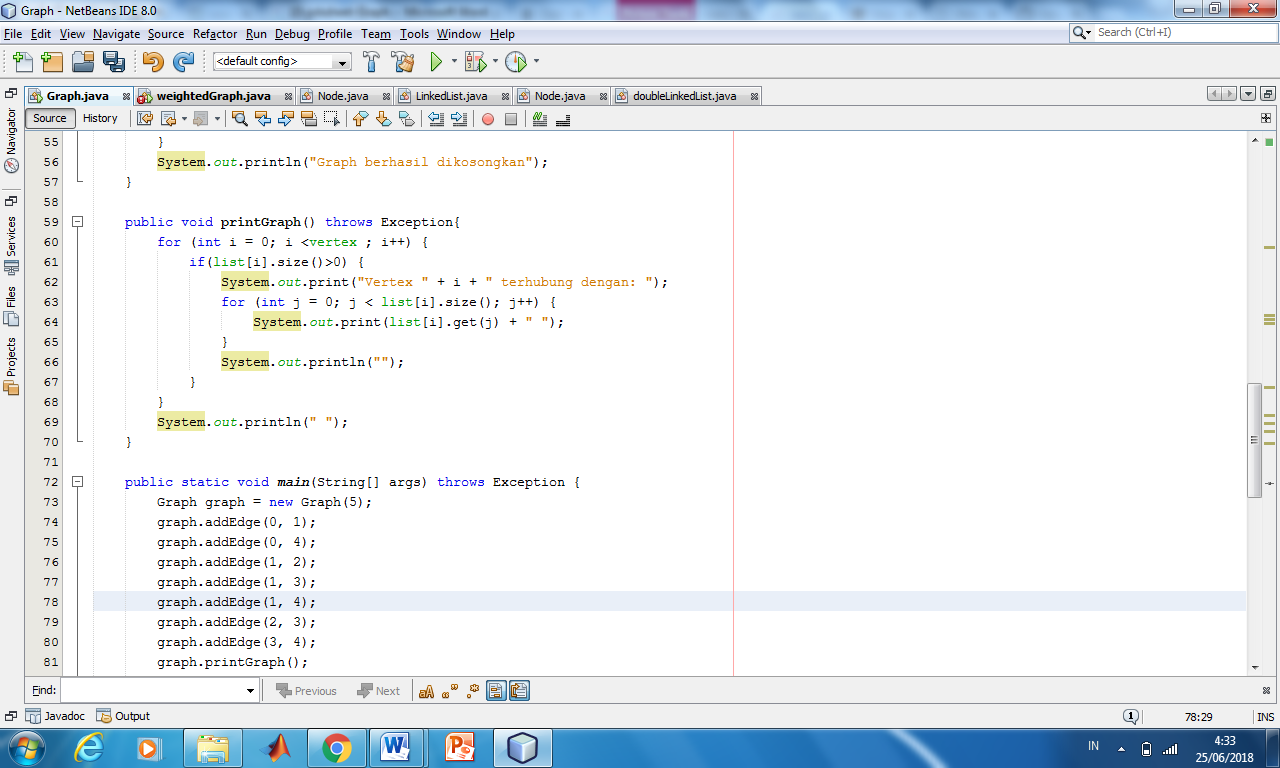
1. Tambahkan method **removeEdge()**. Method ini akan menghapus lintasan ada suatu graph. Oleh karena itu, dibutuhkan 2 parameter untuk menghapus lintasan yaitu source dan destination.



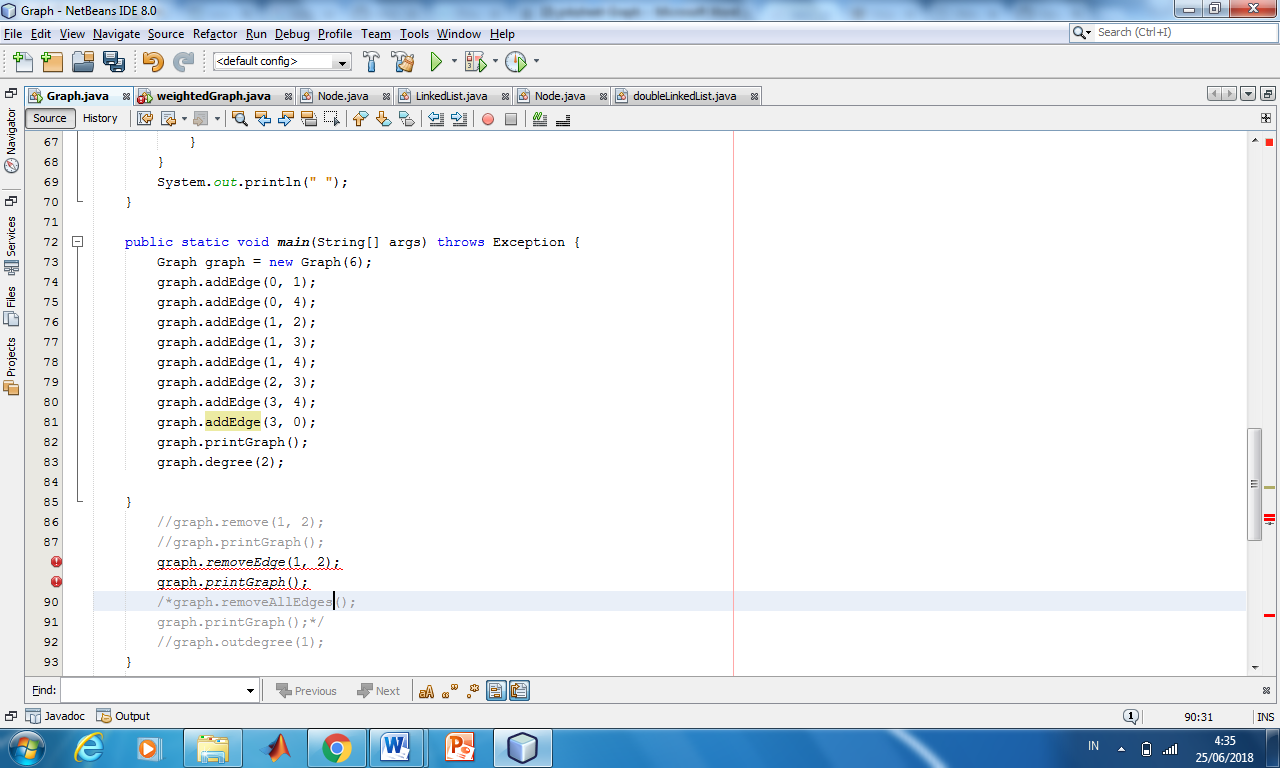
1. Tambahkan method **removeAllEdges()** untuk menghapus semua vertex yang ada di dalam graph.



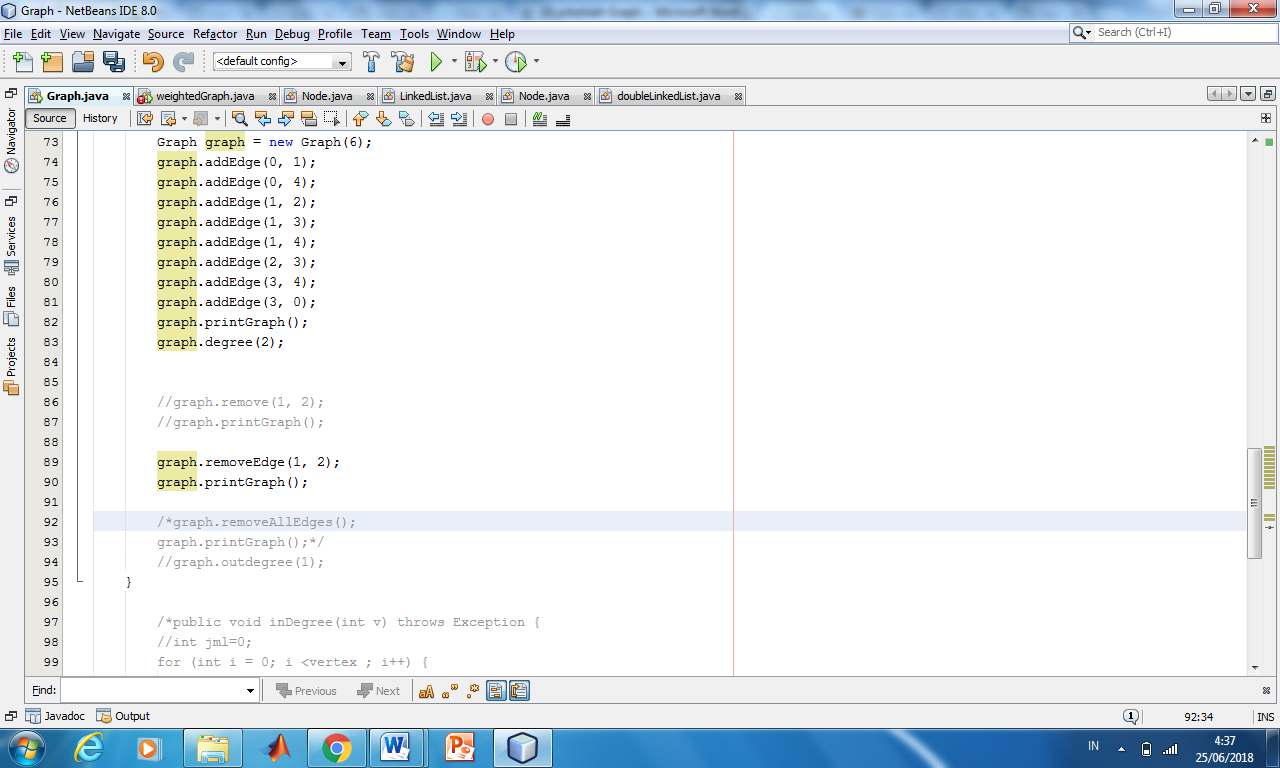
1. Tambahkan method **printGraph()** untuk mencatak graph ter-*update.*



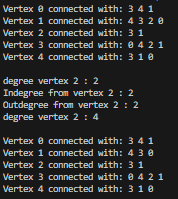
1. Compile dan jalankan method **main()** dalam class **Graph** untuk menambahkan beberapa edge pada graph, kemudian tampilkan. Setelah itu keluarkan hasilnya menggunakan pemanggilan method main(). **Keterangan:** degree harus disesuaikan dengan jenis graph yang telah dibuat (directed/undirected).



1. Amati hasil running tersebut.
2. Tambahkan pemanggilan method **removeEdge()** sesuai potongan code di bawah ini pada method main(). Kemudian tampilkan graph tersebut.



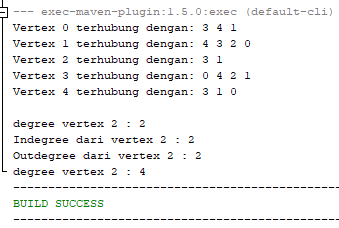
1. Amati hasil running tersebut.
2. Uji coba penghapusan lintasan yang lain! Amati hasilnya!



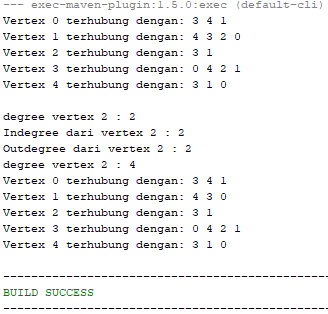
2.1.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

**Hasil running pada langkah ke-11**



**Hasil running pada langkah ke-13**



2.1.3 Pertanyaan Percobaan

1. Sebutkan beberapa jenis (minimal 3) algoritma yang menggunakan dasar Graph, dan apakah kegunaan algoritma-algoritma tersebut?

* Breadth-First Search (BFS), untuk traversing atau mencari semua node di sebuah graph secara level-wise. Biasa digunakan untuk menemukan jarak terpendek pada graph tak berbobot dan dalam melakukan pencarian pada jaringan yang sangat luas seperti web crawling.
* Depth-First Search (DFS), untuk traversing atau mencari semua node di sebuah graph secara depth-wise. Biasa digunakan untuk algoritma penyelesaian masalah seperti menemukan komponen terhubung, topologi sorting dan menemukan semua path dalam suatu graph.
* Dijkstra’s Algorithm, untuk menemukan jalur terpendek dari satu node ke node lainnya dalam graph berbobot yang tidak memiliki bobot tepi negatif. Sngat berguna dalam aplikasi seperi routing jaringan dan sistem navigasi.

1. Pada class Graph terdapat array bertipe LinkedList, yaitu LinkedList list[]. Apakah tujuan pembuatan variabel tersebut?

* Bertujuan untuk merepresentasikan adjacency list dari sebuah graph. Setiap elemen array adalah linked list yang menyimpan semua node tetangga dari node tertentu. Hal ini merupakan cara yang efisien untuk menyimpan dan mengelola struktur data graph karena memungkinkan penambahan dan penghapusan edge yang cepat dan traversal yang efektif.

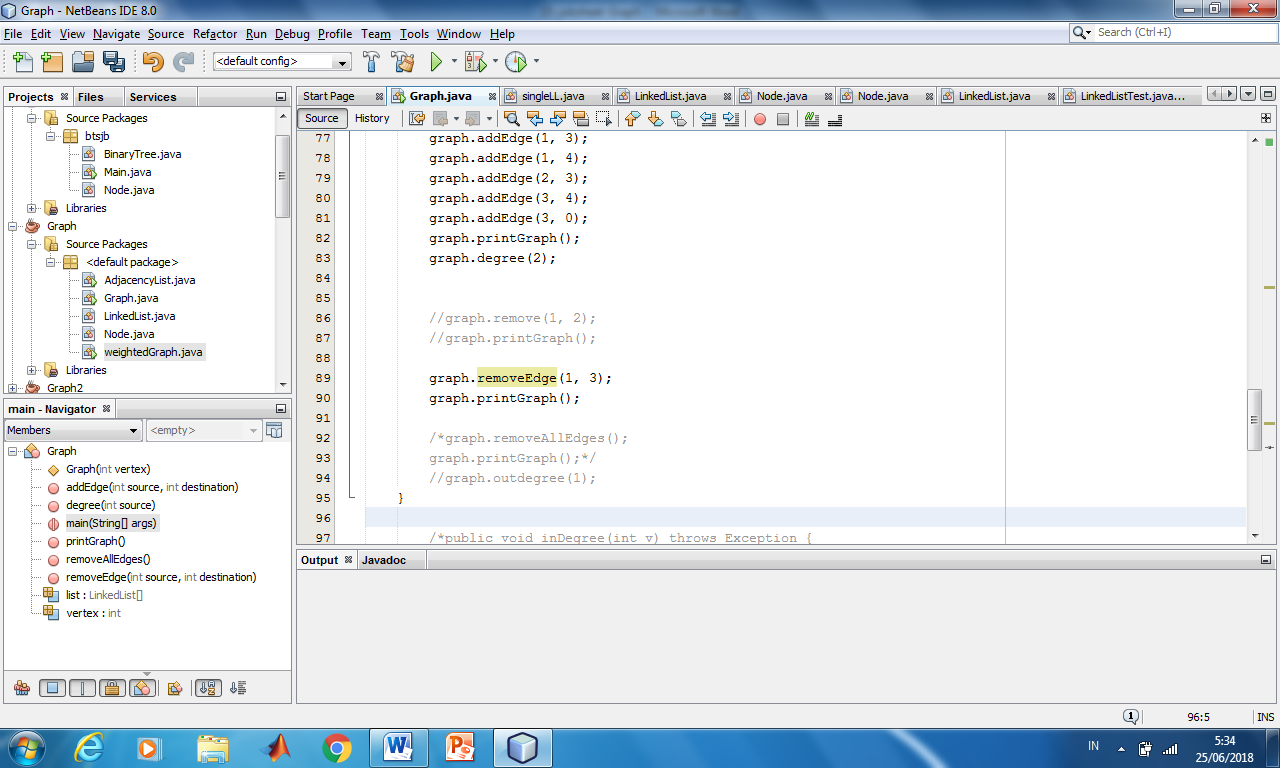
1. Apakah alasan pemanggilan method **addFirst()** untuk menambahkan data, bukan method add jenis lain pada linked list ketika digunakan pada method addEdge pada **class Graph?**

* **‘addFirst()’** digunakan untuk menambahkan elemen baru diawal linked list, juga digunakan untuk memastikan bahwa penambahan edge yang dilakukan dengan cara yang efisien, karena penambahan diawal linked list memiliki kompleksitias O(1). Hal ini juga memastikan jika urutan penambahan edge adalah konsisten dan predictable.

1. Bagaimana cara mendeteksi prev pointer pada saat akan melakukan penghapusan suatu edge pada graph ?

* Untuk mendeteksi prev pointer pada saat penghapusan edge dalam adjacency list yang direpresentasikan dengan linked list, kita perlu traversal linked list tersebut dari head hingga menemukan node yang akan dihapus. Selama traversal, kita simpan referensi ke node sebelumnya (prev). Setelah menemukan node yang akan dihapus, kita atur prev.next ke current.next untuk menghapus node dari linked list.

1. Kenapa pada praktikum 2.1.1 langkah ke-12 untuk menghapus path yang bukan merupakan lintasan pertama kali menghasilkan output yang salah ? Bagaimana solusinya ?



* Mungkin karena tidak eksis atau penghapusan edge dilakukan pada adjacency list yang tidak singkron dengan graph sebenarnya. Bisa menyebablan inkonsistensi pada representasi graph.

2.2 Implementasi Graph menggunakan Matriks

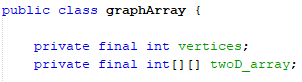
Kegiatan praktikum 2 merupakan implementasi Graph dengan Matriks. Silakan lakukan langkah-langkah percobaan praktikum berikut ini, kemudian verifikasi hasilnya. Setelah itu jawablah pertanyaan terkait percobaan yang telah Anda lakukan.

2.2.1 Tahapan Percobaan

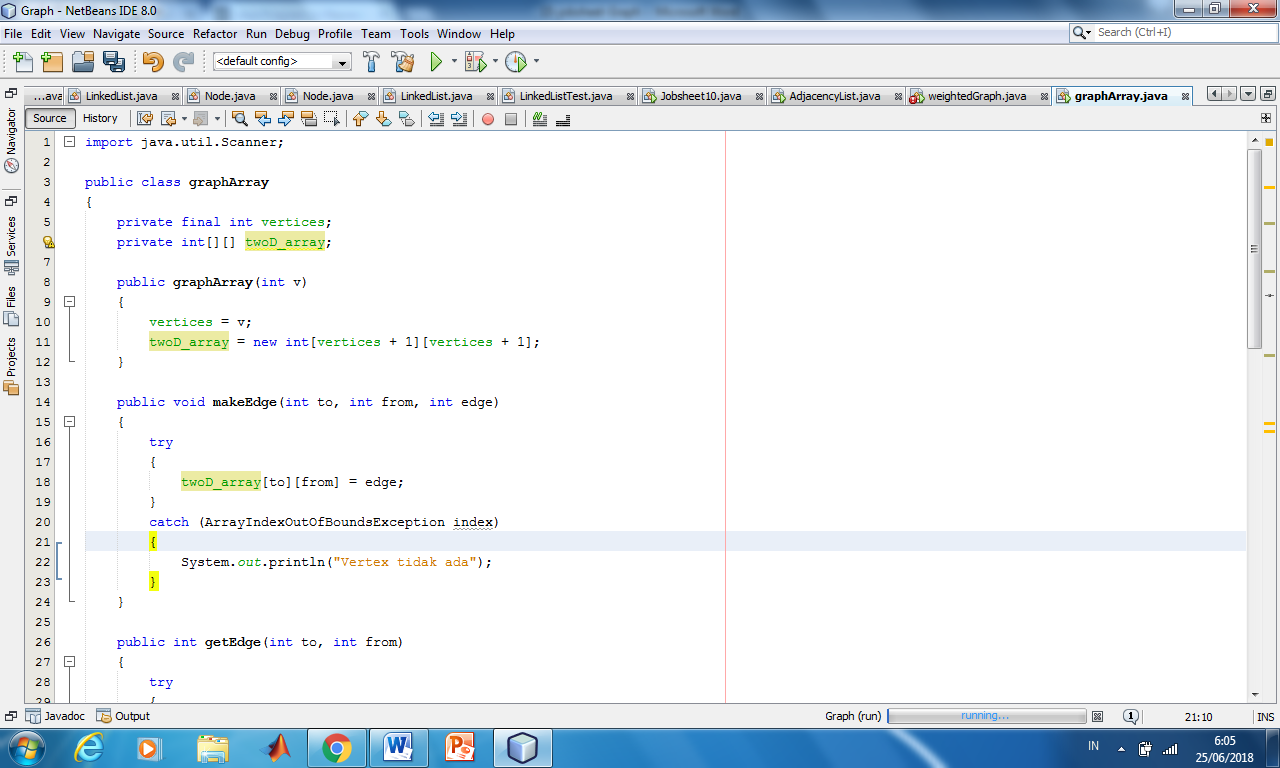
**Waktu percobaan: 30 menit**

Pada praktikum 2.2 ini akan diimplementasikan Graph menggunakan matriks untuk merepresentasikan graph adjacency. Silakan lakukan langkah-langkah praktikum sebagai berikut.

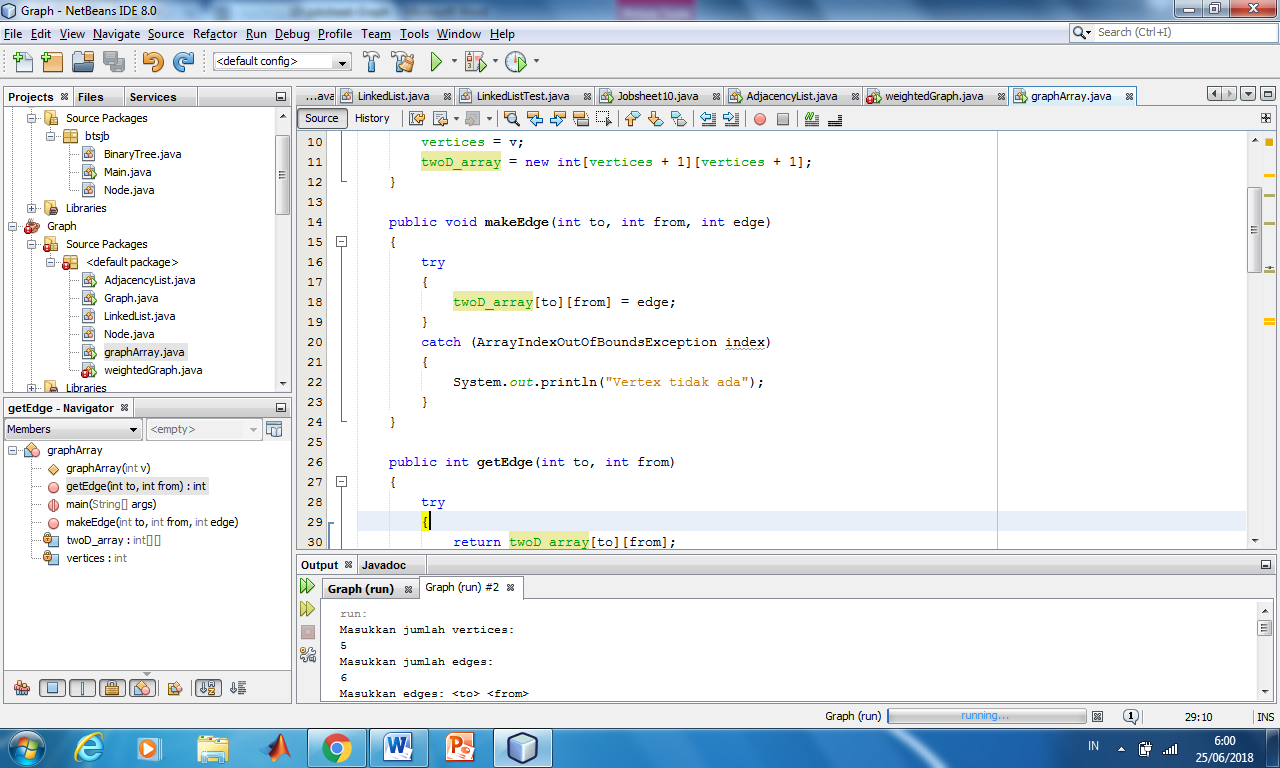
* + - 1. Uji coba graph bagian 2.2 menggunakan array 2 dimensi sebagai representasi graph. Buatlah class **graphArray** yang didalamnya terdapat variabel **vertices** dan **array twoD\_array**!



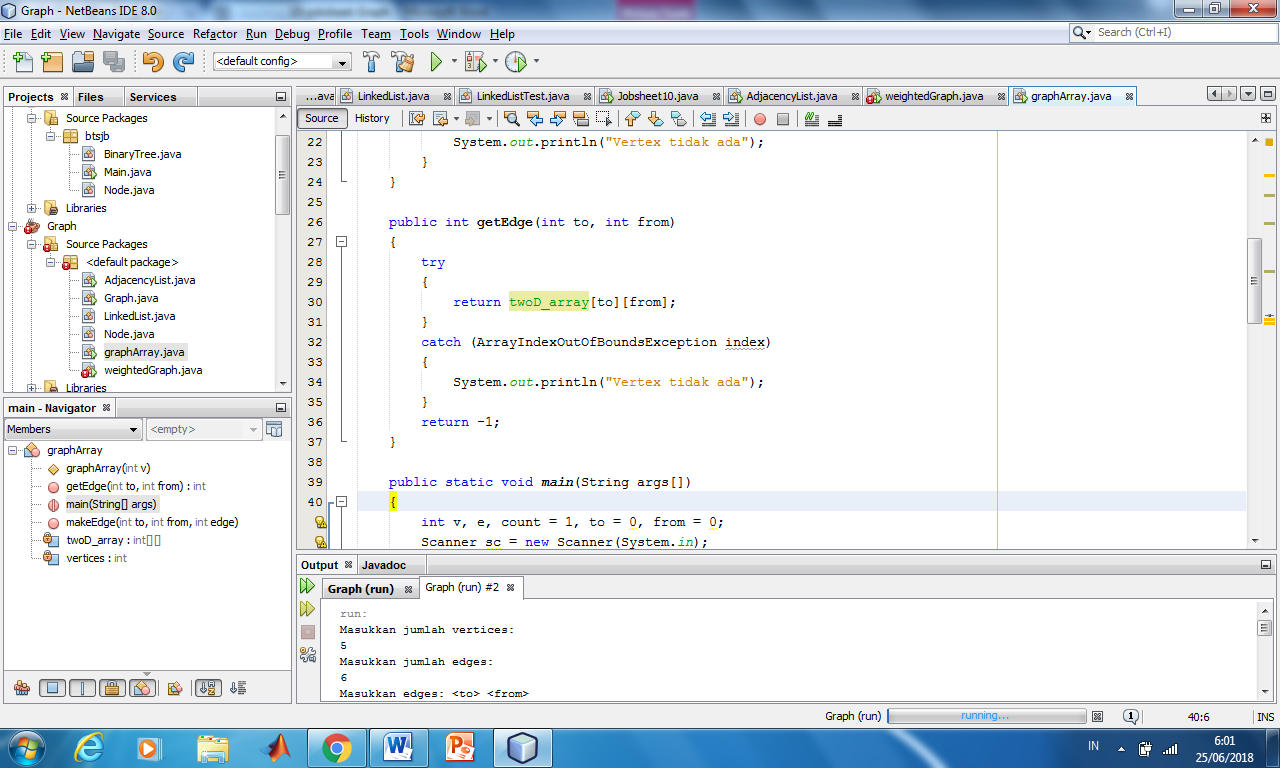
* + - 1. Buatlah konstruktor **graphArray** sebagai berikut!



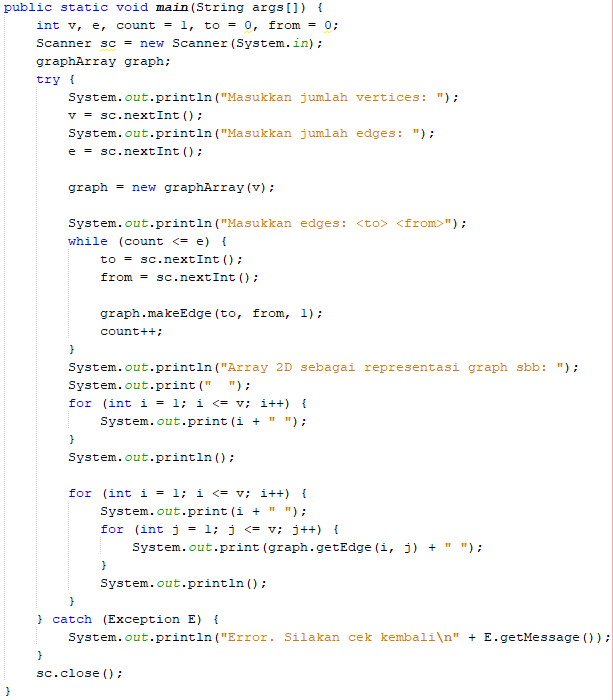
* + - 1. Untuk membuat suatu lintasan maka dibuat method **makeEdge()** sebagai berikut.



Untuk menampilkan suatu lintasan diperlukan pembuatan method **getEdge()** berikut.



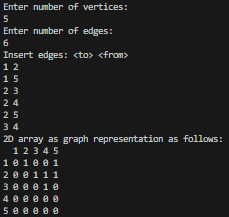
* + - 1. Kemudian buatlah method **main()** seperti berikut ini.

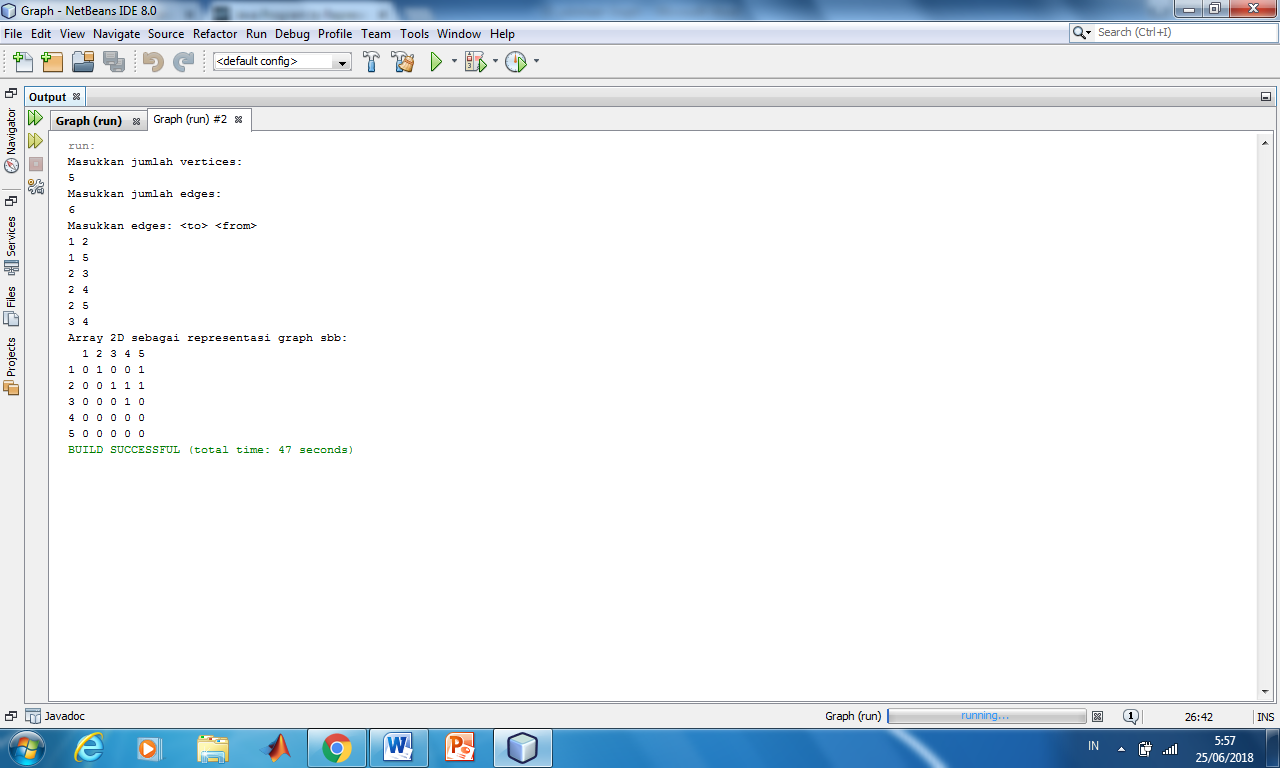


* + - 1. Jalankan class **graphArray** dan amati hasilnya!

2.2.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.



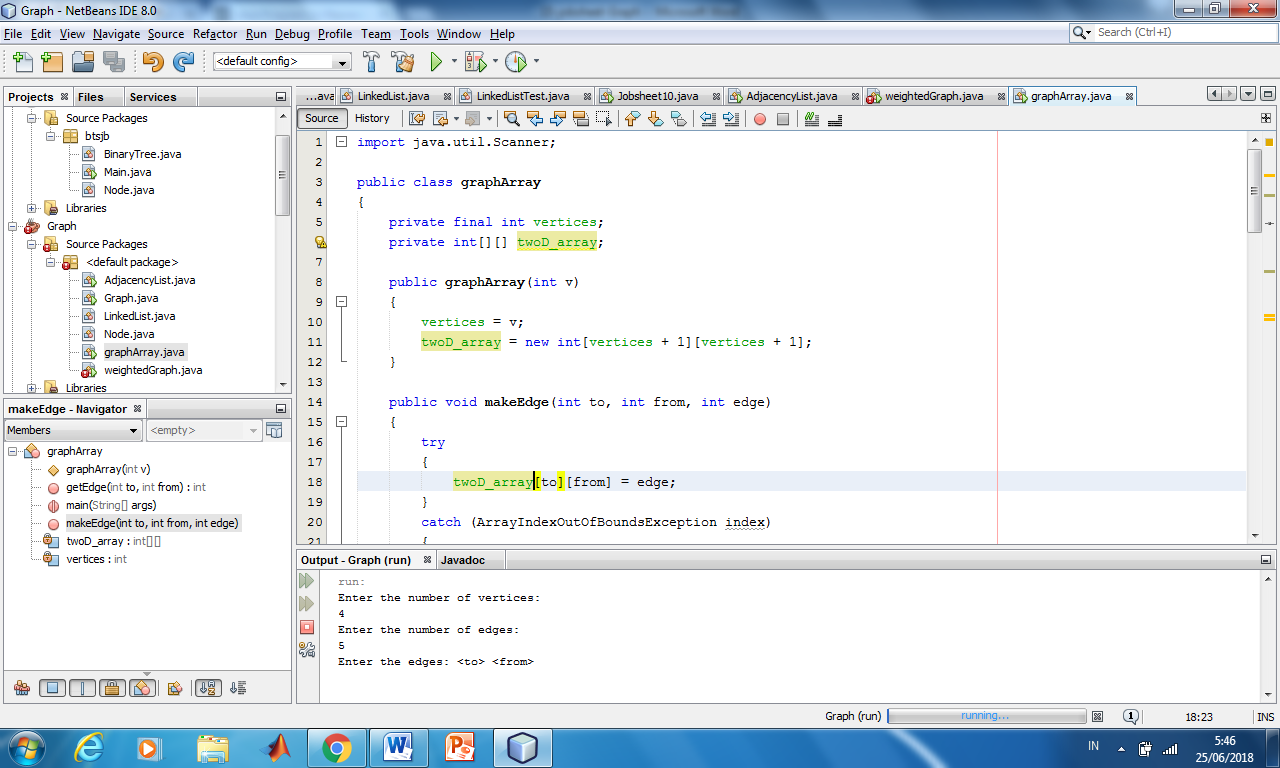


2.2.3 Pertanyaan Percobaan

1. Apakah perbedaan degree/derajat pada *directed* dan *undirected graph*?

* Undirected graph, degree adalah jumlah edge yang terhubung ke sebuah node.
* Directed grph
  + - * 1. In-degree, jumlah edge yang masuk ke sebuah node
        2. Out-degree, jumlah edge yang keluar dari sebuah node

1. Pada implementasi graph menggunakan adjacency matriks. Kenapa jumlah vertices harus ditambahkan dengan 1 pada indeks array berikut?



* Untuk membuat adjecency matrix yang indeksnya mulai dari 1, bukan dari 0. Hal ini untuk memudahkan pemahaman dan manipulasi graph berdasarkan indeks yang lebih intuitif. Misal, jika jumlah vertices =5, maka ukuran array yang dibuat adalah 6x6, sehingga indeks mulai dari 1 sampai 5 dapat digunakan secara langsung.

1. Apakah kegunaan method **getEdge()** ?

* Untuk mengembalikan nilai atau bobot edge antara dua node yang ditentukan dan, jika ada.

1. Termasuk jenis graph apakah uji coba pada praktikum 2.2?

* Menggunkan metode ‘makeEdge’, edge dibuat dengan arah tertentu dari ‘to’ ke ‘from’. Jadi, setiap kali menambahkan edge, memiliki arah yang jelas, merupakan karakteristik dari directed graph.
* Graph direpresentasikan menggunakan adjacency matrix ‘twoD\_array’, di mana nilai ‘twoD\_array[to][from]’ diset dengan nilai edge. Ini menunjukkan bahwa ada arah dari vertex ‘to’ ke vertex ‘from’.

1. Mengapa pada method main harus menggunakan *try-catch Exception* ?

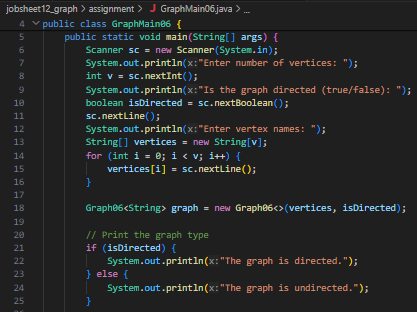
* Karena untuk menangani kemungkinan kesalahan yang bisa terjadi selama eksekusi program. Agar program tidak crash dan dapat memberikan pesan error yang informatif kepada user. Memudahkan dalam debugging karena exception yang dilempar bisa ditangkap dan dilaporkan dengan detail yang lebih baik.

3. Tugas Praktikum

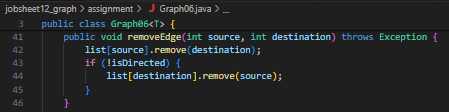
1. Tambahkan scanner pada method AddEdge praktikum 2.1 untuk menerima input Edge



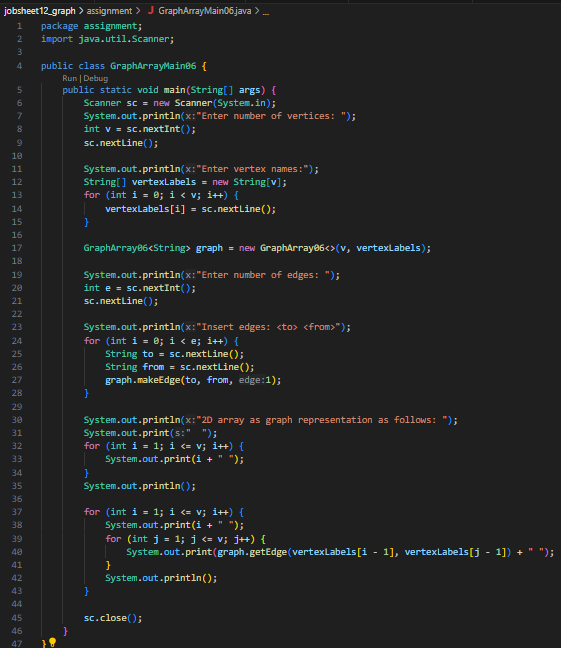
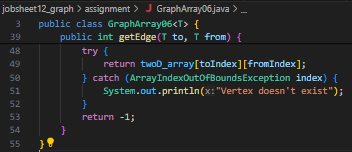
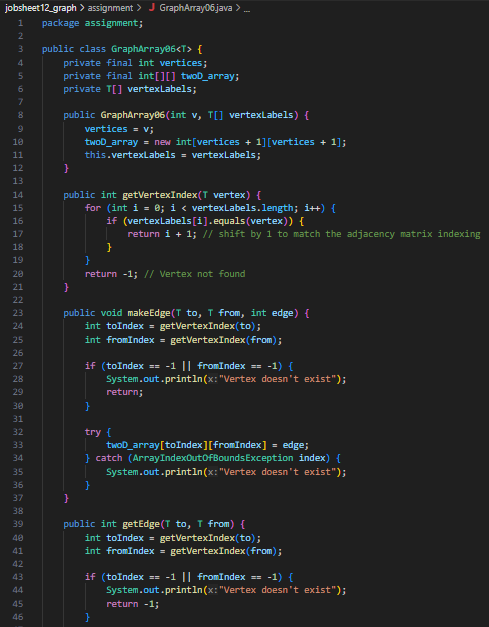
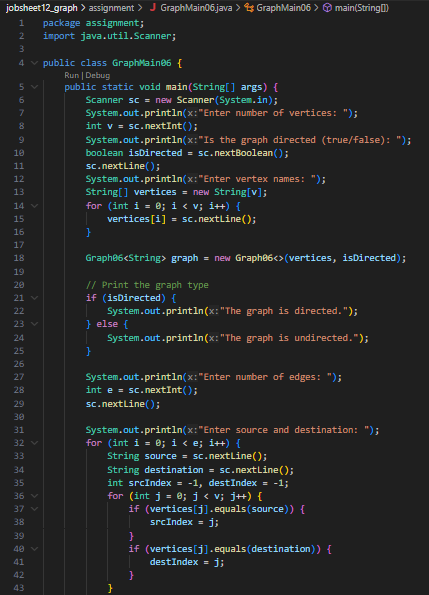
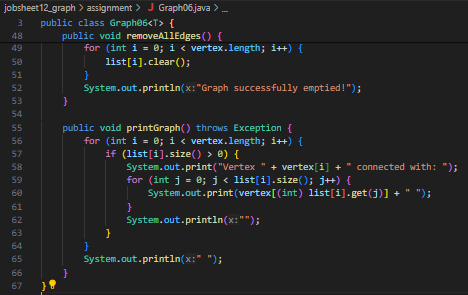
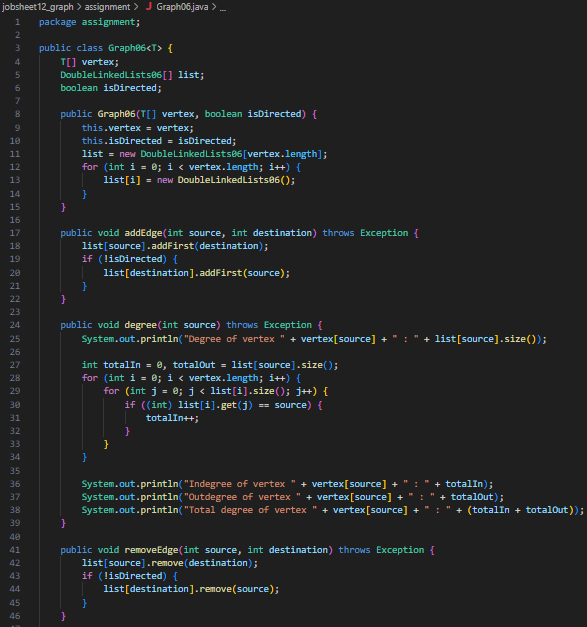
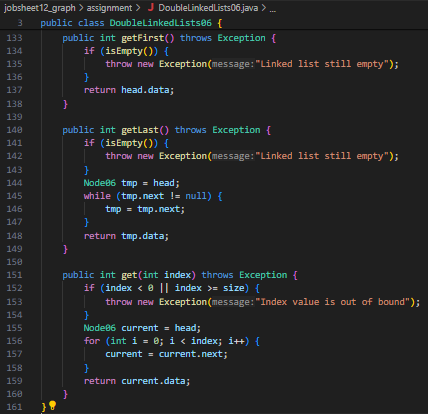
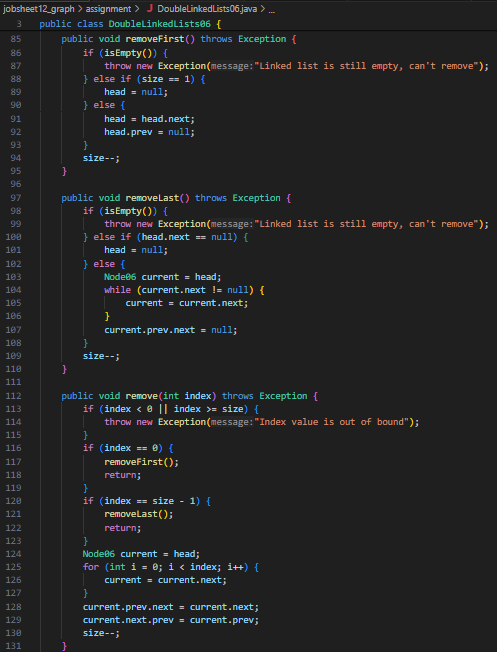
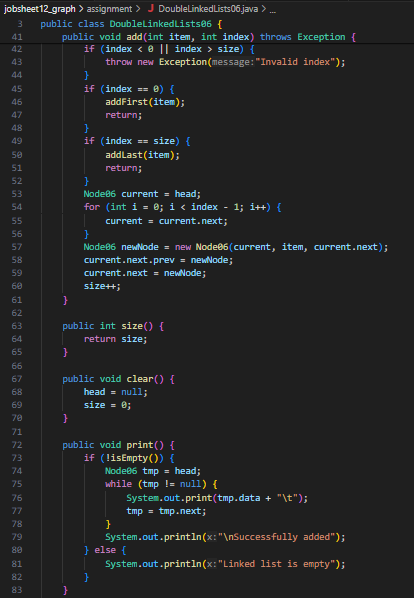
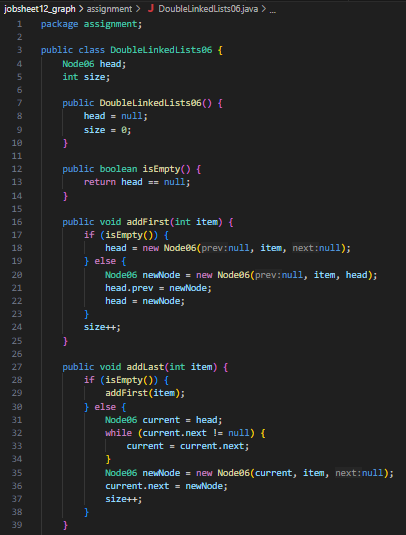
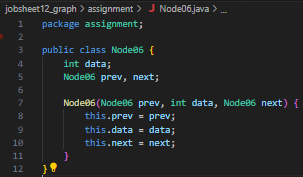
1. Tambahkan method **graphType** dengan tipe boolean yang akan membedakan *graph* termasuk *directed* atau *undirected* *graph*. Kemudian update seluruh method yang berelasi dengan method **graphType** tersebut (hanya menjalankan statement sesuai dengan jenis graph) pada praktikum 2.1



1. Modifikasi method **removeEdge()** pada praktikum 2.1 agar tidak menghasilkan output yang salah untuk path selain path pertama kali!



1. Ubahlah tipe data *vertex* pada seluruh graph pada praktikum 2.1 dan 2.2 dari Integer menjadi tipe generic agar dapat menerima semua tipe data dasar Java! Misalnya setiap *vertex* yang awalnya berupa angka 0,1,2,3, dst. selanjutnya ubah menjadi suatu nama daerah seperti Gresik, Bandung, Yogya, Malang, dst.



--- \*\*\* ---