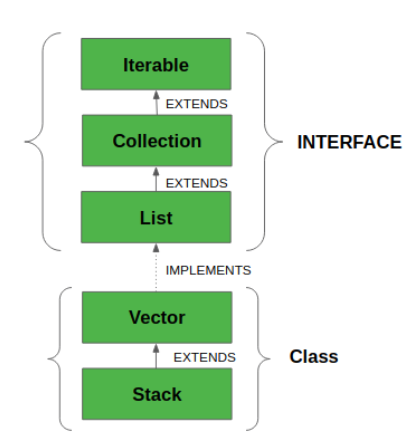
1. **STACK CLASS**

Framework Java Collection menyediakan kelas Stack yang memodelkan dan mengimplementasikan struktur data Stack. Kelas didasarkan pada prinsip dasar last-in-first-out. Selain operasi push dan pop dasar, kelas ini menyediakan tiga fungsi lagi yaitu kosong, cari, dan intip. Kelas juga dapat dikatakan memperluas Vektor dan memperlakukan kelas sebagai tumpukan dengan lima fungsi yang disebutkan. Kelas juga dapat disebut sebagai subclass dari Vector.

Diagram di bawah menunjukkan hierarki kelas Stack:



Kelas mendukung satu Stack konstruktor default () yang digunakan untuk membuat tumpukan

kosong.

Pernyataan:

“Public class stack<E> memperluas Vector<E>”

Semua Antarmuka yang Diimplementasikan:

* Serializable: Ini adalah antarmuka penanda yang harus diimplementasikan kelas jika ingin diserialisasi dan deserialisasi.
* Cloneable: Ini adalah antarmuka di Java yang perlu diimplementasikan oleh kelas untuk memungkinkan objeknya dikloning.
* Iterable<E>: Antarmuka ini mewakili kumpulan objek yang dapat diulang - artinya dapat diulang.
* Koleksi<E>: Koleksi mewakili sekelompok objek yang dikenal sebagai elemennya. Antarmuka Koleksi digunakan untuk menyebarkan koleksi objek di mana keumuman maksimum diinginkan.
* Daftar<E>: Antarmuka Daftar menyediakan cara untuk menyimpan koleksi yang dipesan. Ini adalah antarmuka anak Koleksi.
* RandomAccess: Ini adalah antarmuka penanda yang digunakan oleh implementasi Daftar untuk menunjukkan bahwa mereka mendukung akses acak cepat (umumnya waktu konstan).

Bagaimana Cara Membuat Tumpukan?

Untuk membuat tumpukan, kita harus mengimpor paket java.util.stack dan menggunakan konstruktor Stack() dari kelas ini. Contoh di bawah membuat Stack kosong.

“Stack<E> stack = new stack<E>();” (Di sini E adalah jenis Objek.)

**Contoh:**

**import** java.io.\*;

**import** java.util.\*;

**class** Test

{

// Pushing element on the top of the stack

**static** **void** stack\_push(Stack<Integer> stack)

{

**for**(**int** i = 0; i < 5; i++)

{

stack.push(i);

}

}

// Popping element from the top of the stack

**static** **void** stack\_pop(Stack<Integer> stack)

{

System.out.println("Pop Operation:");

**for**(**int** i = 0; i < 5; i++)

{

Integer y = (Integer) stack.pop();

System.out.println(y);

}

}

// Displaying element on the top of the stack

**static** **void** stack\_peek(Stack<Integer> stack)

{

Integer element = (Integer) stack.peek();

System.out.println("Element on stack top: " + element);

}

// Searching element in the stack

**static** **void** stack\_search(Stack<Integer> stack, **int** element)

{

Integer pos = (Integer) stack.search(element);

**if**(pos == -1)

System.out.println("Element not found");

**else**

System.out.println("Element is found at position: " + pos);

}

**public** **static** **void** main (String[] args)

{

Stack<Integer> stack = **new** Stack<Integer>();

stack\_push(stack);

stack\_pop(stack);

stack\_push(stack);

stack\_peek(stack);

stack\_search(stack, 2);

stack\_search(stack, 6);

}

}

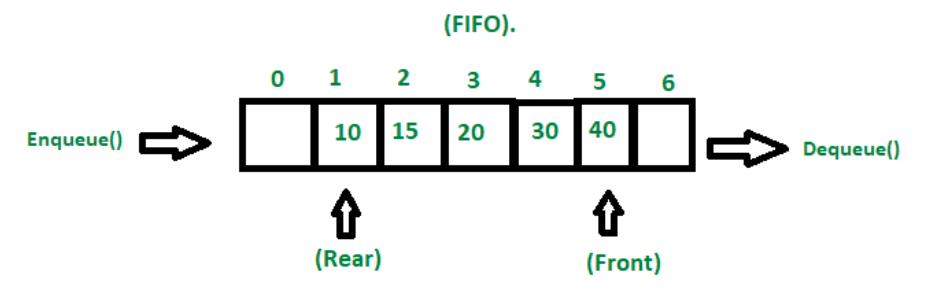
**2. QUEUE**

Antrean(queue) didefinisikan sebagai struktur data linier yang terbuka di kedua ujungnya dan operasi dilakukan dalam urutan First In First Out (FIFO), sebagai daftar di mana semua penambahan ke daftar dibuat di satu ujung, dan semua penghapusan dari daftar dibuat di ujung lainnya. Elemen yang pertama didorong ke dalam urutan, operasi pertama kali dilakukan pada itu.



Prinsip Antrian FIFO:

1. Antrean seperti antrean yang menunggu untuk membeli tiket, di mana orang pertama dalam antrean adalah orang pertama yang dilayani. (mis. First come first serve).
2. Posisi entri dalam antrian yang siap dilayani, yaitu entri pertama yang akan dikeluarkan dari antrian, disebut bagian depan antrian (kadang-kadang kepala antrian), demikian pula posisi entri terakhir dalam antrian, yaitu yang terakhir ditambahkan, disebut bagian belakang (atau ekor) dari antrian. Lihat gambar di bawah ini.



Karakteristik Antrian:

1. Antrean dapat menangani banyak data.
2. Kita dapat mengakses kedua ujungnya.
3. Mereka cepat dan fleksibel.

Representasi Antrian:

Seperti tumpukan, Queue juga dapat direpresentasikan dalam array: Dalam representasi ini, Queue diimplementasikan menggunakan array. Variabel yang digunakan dalam hal ini adalah

1. Queue: nama array yang menyimpan elemen antrian.
2. Depan: indeks tempat elemen pertama disimpan dalam larik yang mewakili antrian.
3. Belakang: indeks tempat elemen terakhir disimpan dalam larik yang mewakili antrean.

contoh code queue:

import java.lang.reflect.Array;

import java.util.Scanner;

public class Queue<T> {

public Integer pointer;

private Integer sizeArray;

private T[] element;

public Queue(Class<?> namaClass, Integer size) {

pointer = -1;

sizeArray = size;

element = (T[]) Array.newInstance(namaClass, size);

}

public void enqueue(T data) {

if (pointer < (sizeArray - 1)) {

pointer++;

element[pointer] = data;

} else {

System.out.println("Queue Sudah Penuh");

}

}

public void dequeue() {

if (pointer >= 0) {

for (Integer i = 0; i <= pointer; i++) {

element[i] = element[i + 1];

}

pointer--;

} else {

System.out.println("Queue Sudah Kosong");

}

}

public void viewQueue() {

for (Integer i = 0; i <= pointer; i++) {

System.out.print(element[i]);

System.out.print(" ");

}

}

public void clearQueue() {

pointer = -1;

}

public static void main(String[] args) {

Integer pil;

Queue<String> queue = new Queue<>(String.class, 10);

Scanner scanner = new Scanner(System.in);

do {

queue.viewQueue();

System.out.println();

System.out.println("1. Enqueue");

System.out.println("2. Dequeue");

System.out.println("3. Clear");

System.out.println("4. Exit");

pil = scanner.nextInt();

if (pil > 0 && pil < 4) {

if (pil == 1) {

String data;

System.out.print("Masukkan Data: ");

data = scanner.next();

queue.enqueue(data);

} else if (pil == 2)

queue.dequeue();

else if (pil == 3)

queue.clearQueue();

} else

break;

} while (true);}}