

PRAKTIKUM ALGORITMA DAN STRUKTUR DATA
JOB SHEET PERTEMUAN KE-8



RIO TRI PRAYOGO

TI 1A

26

2341720236

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG
2024

Stack

Praktikum 1 : Penyimpanan Tumpukan Barang dalam Gudang

Percobaan :

Main

```
package minggu8;

import java.util.Scanner;

public class Utama26 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        Gudang26 gudang = new Gudang26(7);

        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Tambah barang");
            System.out.println("2. Ambil barang");
            System.out.println("3. Tampilkan tumpukan barang");
            System.out.println("4. Keluar");
            System.out.print("Pilih operasi: ");
            int pilihan = scan.nextInt();
            scan.nextLine();

            switch (pilihan) {
                case 1:
                    System.out.print("Masukkan kode barang: ");
                    int kode = scan.nextInt();
                    scan.nextLine();
                    System.out.print("Masukkan nama barang: ");
                    String nama = scan.nextLine();
                    System.out.print("Masukkan nama kategori: ");
                    String kategori = scan.nextLine();
                    Barang26 barangBaru = new Barang26(kode, nama,
kategori);
                    gudang.tambahBarang(barangBaru);
```

```

        break;
    case 2:
        gudang.ambilBarang();
        break;
    case 3:
        gudang.tampilkanBarang();
        break;
    case 4:
        break;
    default:
        System.out.println("Pilihan tidak valid. Silahkan coba
lagi.");
    }
}
}
}
}

```

Gudang

```

package minggu8;

public class Gudang26 {
    Barang26[] tumpukan;
    int size;
    int top;

    public Gudang26(int kapasitas) {
        size = kapasitas;
        tumpukan = new Barang26[size];
        top = -1;
    }

    public boolean cekKosong() {
        if (top == -1) {
            return true;
        } else {
            return false;
        }
    }
}

```

```

public boolean cekPenuh() {
    if (top == size - 1) {
        return true;
    } else {
        return false;
    }
}

public void tambahBarang(Barang26 brg) {
    if (!cekPenuh()) {
        top++;
        tumpukan[top] = brg;
        System.out.println("Barang " + brg.nama + " berhasil
ditambahkan ke Gudang");
    } else {
        System.out.println("Gagal! Tumpukan barang di Gudang sudah
penuh");
    }
}

public Barang26 ambilBarang() {
    if (!cekKosong()) {
        Barang26 delete = tumpukan[top];
        top--;
        System.out.println("Barang " + delete.nama + " diambil dari
Gudang.");
        return delete;
    } else {
        System.out.println("Tumpukan barang kosong.");
        return null;
    }
}

public Barang26 lihatBarangTeratas() {
    if (!cekKosong()) {
        Barang26 barangTeratas = tumpukan[top];

```

```

        System.out.println("Barang teratas: " + barangTeratas.nama);
        return barangTeratas;
    } else {
        System.out.println("Tumpukan barang kosong.");
        return null;
    }
}

public void tampilkanBarang() {
    if (!cekKosong()) {
        System.out.println("Rincian tumpukan barang di Gudang:");
        for (int i = 0; i <= top; i++) {
            System.out.printf("Kode %d: %s (Kategori %s)\n",
tumpukan[i].kode, tumpukan[i].nama,
                tumpukan[i].kategori);
        }
    } else {
        System.out.println("Tumpukan barang kosong.");
    }
}
}

```

Barang

```

package minggu8;

public class Barang26 {
    int kode;
    String nama;
    String kategori;

    public Barang26(int kode, String nama, String kategori) {
        this.kode = kode;
        this.nama = nama;
        this.kategori = kategori;
    }
}

```

Output :

```
Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Keluar
Pilih operasi: 1
Masukkan kode barang: 21
Masukkan nama barang: Majalah
Masukkan nama kategori: Buku
Barang Majalah berhasil ditambahkan ke Gudang

Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Keluar
Pilih operasi: 1
Masukkan kode barang: 26
Masukkan nama barang: Jaket
Masukkan nama kategori: Pakaian
Barang Jaket berhasil ditambahkan ke Gudang

Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Keluar
Pilih operasi: 2
Barang Jaket diambil dari Gudang.
```

```
Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Keluar
Pilih operasi: 1
Masukkan kode barang: 33
Masukkan nama barang: Pizza
Masukkan nama kategori: Makanan
Barang Pizza berhasil ditambahkan ke Gudang

Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Keluar
Pilih operasi: 3
Rincian tumpukan barang di Gudang:
Kode 21: Majalah (Kategori Buku)
Kode 33: Pizza (Kategori Makanan)
```

Pertanyaan :

1. Lakukan perbaikan pada kode program, sehingga keluaran yang dihasilkan sama dengan verifikasi hasil percobaan! Bagian mana saja yang perlu diperbaiki?
 - Output yang dihasilkan pada kode percobaan tidak sama dengan verifikasi hasil percobaan dimana kode percobaan menampilkan barang terbaru dibawah barang sebelumnya, sementara pada verifikasi hasil percobaan memiliki output dimana barang terbaru berada diatas. Kesalahan tersebut bisa diperbaiki dengan mengganti perulangan pada method `tampilkanBarang()` dari object `Gudang26` seperti kode dibawah ini:

```
...
public void tampilkanBarang() {
    if (!cekKosong()) {
        System.out.println("Rincian tumpukan barang di Gudang:");
        for (int i = top; i >= 0; i--) {
            System.out.printf("Kode %d: %s (Kategori %s)\n",
                tumpukan[i].code, tumpukan[i].nama,
                tumpukan[i].kategori);
        }
    } else {
        System.out.println("Tumpukan barang kosong.");
    }
}
...

```

Output :

```
Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Keluar
Pilih operasi: 1
Masukkan kode barang: 33
Masukkan nama barang: Pizza
Masukkan nama kategori: Makanan
Barang Pizza berhasil ditambahkan ke Gudang

Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Keluar
Pilih operasi: 3
Rincian tumpukan barang di Gudang:
Kode 33: Pizza (Kategori Makanan)
Kode 21: Majalah (Kategori Buku)
```

2. Berapa banyak data barang yang dapat ditampung di dalam tumpukan? Tunjukkan potongan kode programnya!
- Banyak data barang yang dapat ditampung dalam tumpukan adalah 7 barang. Hal ini bisa dilihat dari potongan kode pada `Main` dimana kapasitas diisi sebanyak 7:

```
...
Gudang26 gudang = new Gudang26 (7);
...
```

3. Mengapa perlu pengecekan kondisi `!cekKosong()` pada method `tampilkanBarang`? Kalau kondisi tersebut dihapus, apa dampaknya?
- Pengecekan kondisi `!cekKosong()` pada method `tampilkanBarang` berfungsi untuk memberi tahu apakah tumpukan barang kosong atau tidak. Jika pengecekan kondisi tersebut dihapus maka output saat menampilkan barang jika kosong tidak akan ada apa-apa.

Output dengan kondisi <code>!cekKosong()</code>	Output tanpa kondisi <code>!cekKosong()</code>
<pre>Menu: 1. Tambah barang 2. Ambil barang 3. Tampilkan tumpukan barang 4. Keluar Pilih operasi: 3 Tumpukan barang kosong.</pre>	<pre>Menu: 1. Tambah barang 2. Ambil barang 3. Tampilkan tumpukan barang 4. Keluar Pilih operasi: 3 Rincian tumpukan barang di Gudang:</pre>

4. Modifikasi kode program pada class `Utama` sehingga pengguna juga dapat memilih operasi lihat barang teratas, serta dapat secara bebas menentukan kapasitas gudang!
- Untuk melihat barang teratas dapat ditambahkan case baru untuk mengecek dan juga membuat method dari object `Gudang` untuk melihat barang teratas, seperti kode dibawah:
- `Main`

```
...
System.out.println("\nMenu:");
System.out.println("1. Tambah barang");
```

```

        System.out.println("2. Ambil barang");
        System.out.println("3. Tampilkan tumpukan barang");
        System.out.println("4. Tampilkan barang teratas");
        System.out.println("5. Keluar");
        System.out.print("Pilih operasi: ");
        int pilihan = scan.nextInt();

...

        case 4:
            gudang.lihatBarangTeratas();
            break;

...

```

Gudang

```

...

    public Barang26 lihatBarangTeratas() {
        if (!cekKosong()) {
            Barang26 barangTeratas = tumpukan[top];
            System.out.println("Barang teratas: " +
barangTeratas.nama);
            return barangTeratas;
        } else {
            System.out.println("Tumpukan barang kosong.");
            return null;
        }
    }

...

```

Output :

```

Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Tampilkan barang teratas
5. Keluar
Pilih operasi: 3
Rincian tumpukan barang di Gudang:
Kode 33: Pizza (Kategori Makanan)
Kode 21: Majalah (Kategori Buku)

Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Tampilkan barang teratas
5. Keluar
Pilih operasi: 4
Barang teratas: Pizza

```


- Supaya user bisa dengan bebas menentukan kapasitas bisa ditambahkan inputan pada Main

```
...  
        System.out.print("Tentukan kapasitas gudang: ");  
        int n = scan.nextInt();  
        Gudang26 gudang = new Gudang26(n);  
...
```

Output :

```
Tentukan kapasitas gudang: 10  
  
Menu:  
1. Tambah barang  
2. Ambil barang  
3. Tampilkan tumpukan barang  
4. Tampilkan barang teratas  
5. Keluar  
Pilih operasi: █
```

5. Commit dan push kode program ke Github



Praktikum 2 : Konversi Kode Barang ke Biner

Percobaan :

"Baris kode di bawah merupakan tambahan yang dimasukkan ke dalam kode Praktikum 1"

Gudang

```
...  
    public Barang26 ambilBarang() {  
        if (!cekKosong()) {  
            Barang26 delete = tumpukan[top];  
            top--;  
            System.out.println("Barang " + delete.nama + " diambil dari  
Gudang.");  
            System.out.println("Kode unik dalam biner:  
"+konversiDesimalKeBiner(delete.kode));  
            return delete;  
        } else {  
            System.out.println("Tumpukan barang kosong.");  
            return null;  
        }  
    }  
}
```

```

...
...
    public String konversiDesimalKeBiner(int kode) {
        StackKonversi26 stack = new StackKonversi26();
        while (kode > 0) {
            int sisa = kode % 2;
            stack.push(sisa);
            kode = kode / 2;
        }
        String biner = new String();
        while (!stack.isEmpty()) {
            biner += stack.pop();
        }
        return biner;
    }
...

```

StackKonversi

```

package minggu8;

public class StackKonversi26 {
    int size;
    int[] tumpukanBiner;
    int top;

    public StackKonversi26() {
        this.size = 32;
        tumpukanBiner = new int[size];
        top = -1;
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public boolean isFull() {
        return top == size - 1;
    }
}

```

```

    public void push(int data) {
        if (isFull()) {
            System.out.println("Stack penuh");
        } else {
            top++;
            tumpukanBiner[top] = data;
        }
    }

    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack kosong");
            return -1;
        } else {
            int data = tumpukanBiner[top];
            top--;
            return data;
        }
    }
}

```

Output :

```

Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Tampilkan barang teratas
5. Keluar
Pilih operasi: 1
Masukkan kode barang: 13
Masukkan nama barang: Setrika
Masukkan nama kategori: Elektronik
Barang Setrika berhasil ditambahkan ke Gudang

Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Tampilkan barang teratas
5. Keluar
Pilih operasi: 2
Barang Setrika diambil dari Gudang.
Kode unik dalam biner: 1101

```

Pertanyaan :

1. Pada method **konversiDesimalKeBiner**, ubah kondisi perulangan menjadi **while (kode != 0)**, bagaimana hasilnya? Jelaskan alasannya!
 - Dengan mengubah kondisi perulangan dari `while (kode > 0)` menjadi `while (kode != 0)` output yang dihasilkan akan berbeda jika kode barang berupa bilangan negatif. Hal ini terjadi dikarenakan kondisi yang pertama akan berjalan jika kode lebih dari 0, sehingga tidak akan memunculkan hasil biner jika kode tidak lebih dari 0. Sementara kondisi kedua akan berjalan jika kode bukan 0, sehingga akan memunculkan output biner jika kode bukan 0 dan hasil binernya akan sama seperti kode barang jika kode barang adalah bilangan negatif. Dapat dilihat dari output dibawah:

<code>while (kode > 0)</code>	<code>while (kode != 0)</code>
<pre>Menu: 1. Tambah barang 2. Ambil barang 3. Tampilkan tumpukan barang 4. Tampilkan barang teratas 5. Keluar Pilih operasi: 1 Masukkan kode barang: -1 Masukkan nama barang: Setrika Masukkan nama kategori: Elektronik Barang Setrika berhasil ditambahkan ke Gudang Menu: 1. Tambah barang 2. Ambil barang 3. Tampilkan tumpukan barang 4. Tampilkan barang teratas 5. Keluar Pilih operasi: 2 Barang Setrika diambil dari Gudang. Kode unik dalam biner:</pre>	<pre>Menu: 1. Tambah barang 2. Ambil barang 3. Tampilkan tumpukan barang 4. Tampilkan barang teratas 5. Keluar Pilih operasi: 1 Masukkan kode barang: -1 Masukkan nama barang: Setrika Masukkan nama kategori: Elektronik Barang Setrika berhasil ditambahkan ke Gudang Menu: 1. Tambah barang 2. Ambil barang 3. Tampilkan tumpukan barang 4. Tampilkan barang teratas 5. Keluar Pilih operasi: 2 Barang Setrika diambil dari Gudang. Kode unik dalam biner: -1</pre>

2. Jelaskan alur kerja dari method **konversiDesimalKeBiner**!
 - Pertama menginstansiasi object `StackKonversi26` dengan nama `stack`. Setelah itu mengecek apakah kode lebih dari 0, jika iya maka masuk ke tahap selanjutnya. Tahap selanjutnya kode tersebut akan dibagi 2 terus menerus sampai habis, sisa bagi 2 akan dimasukkan ke dalam `stack` satu persatu secara urut. Setelah habis maka masuk ke tahap berikutnya. Tahap berikutnya yaitu memindahkan sisa bagi dari `stack` ke dalam variabel `biner` yang telah dideklarasikan dalam bentuk `String` satu persatu secara urut. Setelah itu di `return`/kembalikan nilai biner tersebut.

Praktikum 3 : Konversi Notasi Infix ke Postfix

Percobaan :

Main

```
package minggu8;

import java.util.Scanner;

public class PostfixMain26 {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String P, Q;
```

```

        System.out.println("Masukkan ekspresi matematika (infix): ");
        Q = scan.nextLine();
        Q = Q.trim();
        Q = Q + ")";

        int total = Q.length();

        Postfix26 post = new Postfix26(total);
        P = post.konversi(Q);
        System.out.println("Postfix: " + P);
    }
}

```

Postfix

```

package minggu8;

public class Postfix26 {
    int n;
    int top;
    char[] stack;

    public Postfix26(int total) {
        n = total;
        top = -1;
        stack = new char[n];
        push('(');
    }

    public void push(char c) {
        top++;
        stack[top] = c;
    }

    public char pop() {
        char item = stack[top];
        top--;
    }
}

```

```

        return item;
    }

    public boolean IsOperand(char c) {
        if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c < 'z') || (c >= '0' &&
c <= '9') || c == ' ' || c == '.') {
            return true;
        } else {
            return false;
        }
    }

    public boolean IsOperator(char c) {
        if (c == '^' || c == '%' || c == '/' || c == '*' || c == '-' || c
== '+') {
            return true;
        } else {
            return false;
        }
    }

    public int derajat(char c) {
        switch (c) {
            case '^':
                return 3;
            case '%':
                return 2;
            case '/':
                return 2;
            case '*':
                return 2;
            case '-':
                return 1;
            case '+':
                return 1;
            default:
                return 0;
        }
    }

```

```

    }

}

public String konversi(String Q) {
    String P = "";
    char c;
    for (int i = 0; i < n; i++) {
        c = Q.charAt(i);
        if (IsOperand(c)) {
            P = P + c;
        }
        if (c == '(') {
            push(c);
        }
        if (c == ')') {
            while (stack[top] != '(') {
                P = P + pop();
            }
            pop();
        }
        if (IsOperator(c)) {
            while (derajat(stack[top]) >= derajat(c)) {
                P = P + pop();
            }
            push(c);
        }
    }
    return P;
}
}

```

Output :

Masukkan ekspresi matematika (infix):
 $a+b*(c+d-e)/f$
 Postfix: $abcd+e-*f/+$

Pertanyaan :

1. Pada method **derajat**, mengapa return value beberapa case bernilai sama? Apabila return value diubah dengan nilai berbeda-beda setiap case-nya, apa yang terjadi?
 - Beberapa return value pada method **derajat** bernilai sama dikarenakan dalam operasi matematika *Operator* tersebut memiliki derajat yang sama. Jika nilai berbeda setiap case program akan tetap berjalan dengan baik dan tidak akan error karena derajat tersebut hanya berfungsi untuk memposisikan operator dalam *Postfix*. Akan tetapi hasil dari *Postfix*-nya tidak akan sesuai dikarenakan derajatnya tidak sesuai dengan derajat *Operator* dalam dunia operasi matematika yang benar.
2. Jelaskan alur kerja method **konversi**!
 - Alur kerja dimulai dengan mendeklarasikan `char` dengan nama `c` yang nanti akan menyimpan sementara setiap karakter yang diambil dari parameter `Q` atau ekspresi matematika yang diinputkan. Juga mendeklarasikan variabel `P` berupa `String` yang dimana berfungsi untuk menyimpan *Postfix*. Setelah itu melakukan perulangan untuk menghitung setiap karakter dan nanti dimasukkan ke dalam variabel `c`. Jika karakter dari ekspresi matematika berupa *Operand* maka akan langsung dimasukkan ke dalam *Postfix*. Jika karakter berupa '(' maka akan dimasukkan ke dalam *stack*. Jika karakter berupa ')' maka semua karakter yang ada didalam variabel *stack* dari yang paling atas sampai karakter '(' akan dimasukkan ke dalam *Postfix* tetapi tidak memasukkan karakter '(' atau membuangnya. Jika karakter berupa *Operator* maka akan dicek apakah derajat dari *Operator* yang paling atas didalam *stack* lebih besar atau setara dari *Operator* yang didalam variabel `c`. Jika iya maka *Operator* dari *stack* akan dikeluarkan dari *stack* dimasukkan ke dalam *Postfix*, jika tidak maka *Operator* `c` akan dimasukkan ke dalam *stack*. Setelah karakter habis maka semua *Operator* dalam *stack* akan dimasukkan ke dalam *Postfix* secara berurutan dari atas. Setelah selesai maka akan mendapatkan nilai *Postfix* berupa variabel `P`.
3. Pada method konversi, apa fungsi dari potongan kode berikut?

```
c = Q.charAt(i);
```

 - Potongan kode tersebut berfungsi untuk mengambil setiap karakter dari inputan ekspresi matematika dan menyimpannya ke dalam variabel `c`.

Latihan Praktikum

Perhatikan dan gunakan kembali kode program pada Percobaan 1. Tambahkan dua method berikut pada class Gudang:

- Method **lihatBarangTerbawah** digunakan untuk mengecek barang pada tumpukan terbawah
- Method **cariBarang** digunakan untuk mencari ada atau tidaknya barang berdasarkan **kode** barangnya atau **nama** barangnya

"Baris kode di bawah merupakan tambahan yang dimasukkan ke dalam kode Praktikum 1 & 2"

Main

```
...  
  
        System.out.println("\nMenu:");  
        System.out.println("1. Tambah barang");  
        System.out.println("2. Ambil barang");  
        System.out.println("3. Tampilkan tumpukan barang");
```



```

        System.out.println("4. Tampilkan barang teratas");
        System.out.println("5. Tampilkan barang terbawah");
        System.out.println("6. Cari barang (kode/nama)");
        System.out.println("7. Keluar");
        System.out.print("Pilih operasi: ");
        int pilihan = scan.nextInt();

...
...

        case 5:
            gudang.lihatBarangTerbawah();
            break;
        case 6:
            System.out.println("Masukkan nama/kode barang
yang dicari: ");
            String cari = scan.nextLine();
            gudang.cariBarang(cari);
            break;

...

```

- Lihat barang terbawah
- Gudang

```

...
    public Barang26 lihatBarangTerbawah() {
        if (!cekKosong()) {
            Barang26 barangTerbawah = tumpukan[0];
            System.out.println("Barang terbawah: " +
barangTerbawah.nama);
            return barangTerbawah;
        } else {
            System.out.println("Tumpukan barang kosong.");
            return null;
        }
    }
...

```

Output :

```
Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Tampilkan barang teratas
5. Tampilkan barang terbawah
6. Cari barang (kode/nama)
7. Keluar
Pilih operasi: 3
Rincian tumpukan barang di Gudang:
Kode 33: Pizza (Kategori Makanan)
Kode 21: Majalah (Kategori Buku)

Menu:
1. Tambah barang
2. Ambil barang
3. Tampilkan tumpukan barang
4. Tampilkan barang teratas
5. Tampilkan barang terbawah
6. Cari barang (kode/nama)
7. Keluar
Pilih operasi: 5
Barang terbawah: Majalah
```

- Cari barang berdasarkan kode/nama
Gudang

```
...

    public Barang26 cariBarang(String cari) {
        if (!cekKosong()) {
            for (int i = top; i >= 0; i--) {
                Barang26 barang = tumpukan[i];
                if (barang.nama.equalsIgnoreCase(cari) ||
String.valueOf(barang.kode).equalsIgnoreCase(cari)) {
                    System.out.printf("Barang ditemukan: %s
(Kode: %d, Kategori: %s)\n", barang.nama, barang.kode,
                        barang.kategori);
                    return barang;
                }
            }
            System.out.println("Barang tidak ditemukan.");
            return null;
        } else {
            System.out.println("Tumpukan barang kosong.");
            return null;
        }
    }

    ...
```

Output :

Cari menggunakan nama	Cari menggunakan kode
<pre>Menu: 1. Tambah barang 2. Ambil barang 3. Tampilkan tumpukan barang 4. Tampilkan barang teratas 5. Tampilkan barang terbawah 6. Cari barang (kode/nama) 7. Keluar Pilih operasi: 6 Masukkan nama/kode barang yang dicari: Majalah Barang ditemukan: Majalah (Kode: 21, Kategori: Buku)</pre>	<pre>Menu: 1. Tambah barang 2. Ambil barang 3. Tampilkan tumpukan barang 4. Tampilkan barang teratas 5. Tampilkan barang terbawah 6. Cari barang (kode/nama) 7. Keluar Pilih operasi: 6 Masukkan nama/kode barang yang dicari: 33 Barang ditemukan: Pizza (Kode: 33, Kategori: Makanan)</pre>