LAPORAN JOBSHEET 7

Algoritma dan Struktur Data

Disusun Oleh :

DHANISA PUTRI MASHILFA

NIM. 2341720212

TI-1E/07

D-IV TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

Jl. Soekarno Hatta No. 9 Jatimulyo, Kecamatan Lowokwaru, Jatimulyo, Kec. Lowokwaru, Kota Malang, Jawa Timur 6514

2.1 Percobaan 1: Penyimpanan Tumpukan Barang dalam Gudang

2.1.1 Langkah-langkah Percobaan

2.1.2 Verifikasi Hasil Percobaan

```
Menu :
1. Tambah Barang
2. Ambil Barang
3. tampilkan Tumpukan Barang
4. Keluar
Pilih Operasi : 1
Masukan Kode Barang : 21
Masukan Nama Barang : majalah
Masukan 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
Masukan Kode Barang : 26
Masukan Nama Barang : jaket
Masukan 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
Masukan Kode Barang : 33
Masukan Nama Barang : pizza
Masukan 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)

Menu :
1. Tambah Barang
2. Ambil Barang
3. tampilkan Tumpukan Barang
4. Keluar
Pilih Operasi :
```

2.1.3 Pertanyaan

1. Lakukan perbaikan pada kode program, sehingga keluaran yang dihasilkan sama dengan verifikasi hasil percobaan! Bagian mana saja yang perlu diperbaiki?
   → Pada bagian tampilkanBarang line 65 diganti menjadi **for (int = top; i >= 0; i–)**, sehingga outputnya sesuai dengan verifikasi hasil percobaan

```
public void tampilkanBarang() {
        if (!cekKosong()) {
            System.out.println("Rincian tumpukan barang di Gudang :
");
            for (int i = top; i >= 0; i--){
            //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.");
        }
    }
```

```
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!
   → Data yang dapat ditampung dalam tumpukan adalah 7. Kodenya berada di class **Utama07.java**

```
Gudang07 gudang = new Gudang07(7);
```

3. Mengapa perlu pengecekan kondisi **!cekKosong()** pada method **tampilkanBarang**? Kalau kondisi tersebut dihapus, apa dampaknya?
   → Untuk mencegah kesalahan apabila tidak ada tumpukan kosong, sehingga tidak ada barang untuk ditampilkan. Jika kondisi tersebut dihapus, maka saat program dijalankan akan terhenti secara otomatis dan akan menyebabkan kesalahan **NullPointerException**

4. Modifikasi kode program pada class **Utama** sehingga pengguna juga dapat memilih operasi lihat barang teratas, serta dapat secara bebas menentukan kapasitas gudang!

```
Masukan Kapasitas Gudang : 3

Menu :
1. Tambah Barang
2. Ambil Barang
3. Tampilkan Tumpukan Barang
4. Tampilkan Barang Teratas
5. Keluar
Pilih Operasi : 1
Masukan Kode Barang : 1
Masukan Nama Barang : api
Masukan Nama Kategori : api
Barang api berhasil ditambahkan ke gudang

Menu :
1. Tambah Barang
2. Ambil Barang
3. Tampilkan Tumpukan Barang
4. Tampilkan Barang Teratas
5. Keluar
Pilih Operasi : 1
Masukan Kode Barang : 2
Masukan Nama Barang : air
Masukan Nama Kategori : air
Barang air berhasil ditambahkan ke gudang

Menu :
1. Tambah Barang
2. Ambil Barang
3. Tampilkan Tumpukan Barang
4. Tampilkan Barang Teratas
5. Keluar
Pilih Operasi : 1
Masukan Kode Barang : 3
Masukan Nama Barang : angin
Masukan Nama Kategori : angin
```

```
Masukan Kode Barang : 3
Masukan Nama Barang : angin
Masukan Nama Kategori : angin
Barang angin berhasil ditambahkan ke gudang

Menu :
1. Tambah Barang
2. Ambil Barang
3. Tampilkan Tumpukan Barang
4. Tampilkan Barang Teratas
5. Keluar
Pilih Operasi : 1
Masukan Kode Barang : 4
Masukan Nama Barang : sapu
Masukan Nama Kategori : sapu
Gagal! Tumpukan barang di Gudang sudah penuh

Menu :
1. Tambah Barang
2. Ambil Barang
3. Tampilkan Tumpukan Barang
4. Tampilkan Barang Teratas
5. Keluar
Pilih Operasi : 4
Barang teratas : angin
```

5. Commit dan push kode program ke Github

2.2 Percobaan 2: Konversi Kode Barang ke Biner

2.2.1 Langkah-langkah Percobaan

2.2.2 Hasil Percobaan

```
Masukan Kapasitas Gudang : 1

Menu :
1. Tambah Barang
2. Ambil Barang
3. Tampilkan Tumpukan Barang
4. Tampilkan Barang Teratas
5. Keluar
Pilih Operasi : 1
Masukan Kode Barang : 13
Masukan Nama Barang : setrika
Masukan 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
```

2.2.3 Pertanyaan

1. Pada method **konversiDesimalKeBiner**, ubah kondisi perulangan menjadi **while (kode != 0)**, bagaimana hasilnya? Jelaskan alasannya!
   → Hasilnya masih sama seperti yang sebelumnya, itu karena

   ```
   Masukan Kapasitas Gudang : 1

   Menu :
   1. Tambah Barang
   2. Ambil Barang
   3. Tampilkan Tumpukan Barang
   4. Tampilkan Barang Teratas
   5. Keluar
   Pilih Operasi : 1
   Masukan Kode Barang : 13
   Masukan Nama Barang : setrika
   Masukan 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
   ```

2. Jelaskan alur kerja dari method **konversiDesimalKeBiner**!
   a. Inisialisai : Deklarasikan variabel yang dapat menyimpan bilangan desimal dan biner dalam string kosong.
   b. Pembagian yang Berulang :
      →Jika nilai desimal lebih besar dari atau sama dengan 0, lakukan perulangan.
      → Jika nilai desimal lebih besar dari atau sama dengan 2, simpan sisa hasil bagi sebagai bit biner di awal string biner. Perbarui nilai desimal dengan hasil pembagian.
      → Pembentukan Bilangan Biner: Ulangi langkah 2.b dan 2.c sampai nilai desimal menjadi 0.
   c. Untuk mengubah string biner : balikkan bit paling kiri ke bit paling kanan (signifikan).
   d. Temuan : Nilai variabel biner yang telah dibalik sama dengan bilangan desimal.

2.3 Percobaan 3: Konversi Notasi Infix ke Postfix

2.3.1 Langkah-langkah Percobaan

2.3.2 Hasil Percobaan

```
Masukan ekspresi matematika (infix) :a+b*(c+d-e)/f
Postfix : abcd+e-*f/+
PS C:\Users\asus\Documents\Semester2\Algoritma dan
```

2.3.3 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?

   Karena beberapa operator memiliki tingkat prioritas yang sama, tingkat prioritas operator akan berubah ketika return value diubah, yang menghasilkan hasil yang tidak sesuai dengan aturan prioritas operator yang umum. Jika operasi perkalian memiliki prioritas lebih rendah daripada operasi penjumlahan, hasil konversi dari notasi infix ke postfix akan menghasilkan kesalahan.

2. Jelaskan alur kerja method **konversi**!
   a. Inisialisasi string kosong P yang akan menyimpan hasil konversi dari notasi infix menjadi notasi postfix.
   ```
   String p = "";
   ```
   b. Iterasi melalui setiap karakter dalam string Q menggunakan loop for.
   ```
   for (int i = 0; i < n; i++) {
               c = Q.charAt(i);
   ```
   c. Pengecekan apakah karakter c merupakan operand (angka atau huruf).
   ```
   if (IsOperand(c)){
                   p = p + c;
   ```
   d. Pengecekan apakah karakter c merupakan kurung buka (. Jika ya, karakter tersebut dimasukkan ke dalam stack.
   ```
   if (c == '(') {
                   push(c);
   ```
   e. Pengecekan apakah karakter c merupakan kurung tutup). Jika ya, dilakukan pengambilan karakter-karakter dari stack dan dimasukkan ke dalam string P sampai ditemukan kurung buka yang sesuai.
   ```
   if (c == ')') {
                   while (stack[top] != '(') {
                       p = p + pop();
                   }
                   pop();
               }
   ```
   f. Pengecekan apakah karakter c merupakan operator
   ```
   if (IsOperator(c)) {
   ```

7. Jika karakter c merupakan operator, dilakukan pengecekan terhadap operator operator yang sudah ada dalam stack. Jika operator di stack memiliki tingkat prioritas yang lebih tinggi atau sama dengan operator yang sedang diproses, operator-operator tersebut diambil dari stack dan dimasukkan ke dalam string P. Lalu operator yang sedang diproses ditambahkan ke dalam stack.
return P;

```
if (IsOperator(c)) {
                while (derajat(stack[top]) >= derajat(c)) {
                    p = p + pop();
                }
                push(c);
            }
        }
```

8. Mengembalikan string P sebagai hasil konversi dari notasi infix menjadi notasi postfix.

3. Pada method konversi, apa fungsi dari potongan kode berikut?

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

→ mengambil karakter pada indeks ke-i dari string O dan menyimpannya ke variable c

2.4 Latihan Praktikum

Waktu : 60 Menit

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

```
Menu :
1. Tambah Barang
2. Ambil Barang
3. Tampilkan Tumpukan Barang
4. Tampilkan Barang Teratas
5. Tampilkan Barang Terbawah
6. Keluar
Pilih Operasi : 1
Masukan Kode Barang : 3
Masukan Nama Barang : 3
Masukan Nama Kategori : 3
Barang 3 berhasil ditambahkan ke gudang

Menu :
1. Tambah Barang
2. Ambil Barang
3. Tampilkan Tumpukan Barang
4. Tampilkan Barang Teratas
5. Tampilkan Barang Terbawah
6. Keluar
Pilih Operasi : 5
Barang terbawah : 1
```

→ Method cariBarang digunakan untuk mencari ada atau tidaknya barang berdasarkan kode barangnya atau nama barangnya

```
Masukan Kapasitas Gudang : 1

Menu :
1. Tambah Barang
2. Ambil Barang
3. Tampilkan Tumpukan Barang
4. Tampilkan Barang Teratas
5. Tampilkan Barang Terbawah
6. Cari Barang
7. Keluar
Pilih Operasi : 1
Masukan Kode Barang : 2
Masukan Nama Barang : minyak
Masukan Nama Kategori : makanan
Barang minyak berhasil ditambahkan ke gudang

Menu :
1. Tambah Barang
2. Ambil Barang
3. Tampilkan Tumpukan Barang
4. Tampilkan Barang Teratas
5. Tampilkan Barang Terbawah
6. Cari Barang
7. Keluar
Pilih Operasi : 6
Masukan kode barang : 2
Barang yang ditemukan adalah minyak
```