PRAKTIKUM ALGORITMA DAN STRUKTUR DATA

JOBSHEET PERTEMUAN KE-7

NAMA : ALVINO VALERIAN D.R

KELAS : 1A

NO. ABSEN : 05

NIM : 2341720027

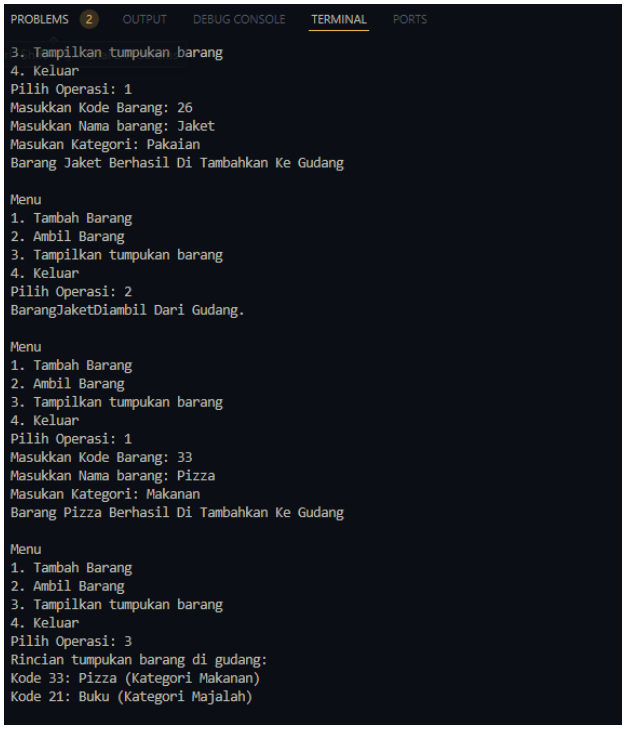
PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2024

2.1.2 Verifikasi Hasil Percobaan



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?

2. Berapa banyak data barang yang dapat ditampung di dalam tumpukan? Tunjukkan potongan kode programnya!

3. Mengapa perlu pengecekan kondisi **!cekKosong()** pada method **tampilkanBarang**? Kalau kondisi tersebut dihapus, apa dampaknya?

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

JAWABAN:

1.

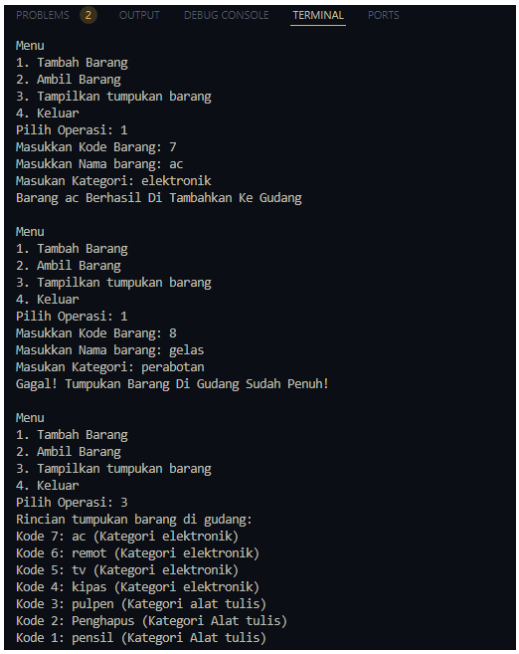
```
for (int i=top;i>=0;i--){
                // for (int i=0;i< top;i++){
```

Diganti yang bawah menjadi comment supaya barang pizza tampil

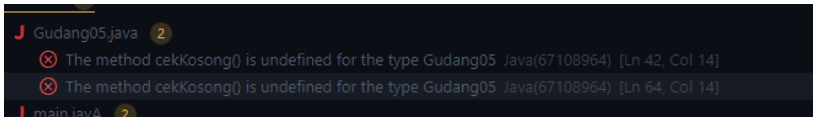
2.

```
Gudang05 gudang =new Gudang05(7);
```

Karena daya tamping atau kapasitas hanya 7 maka hanya bisa menampilkan 7 barang di tumpukan



3. engecekan kondisi !cekKosong() pada method tampilkanBarang bertujuan untuk memastikan bahwa tumpukan tidak kosong sebelum melakukan iterasi dan menampilkan barang-barangnya. Hal ini penting untuk menghindari kesalahan saat mencoba mengakses elemen pada tumpukan kosong. Jika cek kosong dihapus maka kodeakan error karena method tampil dll terhubung dengan cekKosong().

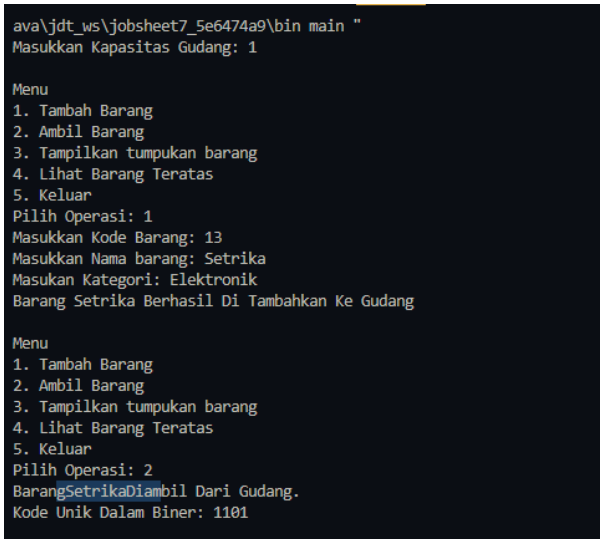


4.

```
Scanner scanner = new Scanner(System.in);
        System.out.print("Masukkan Kapasitas Gudang: ");
        int kapasitas = scanner.nextInt();
        Gudang05 gudang =new Gudang05(kapasitas);
```

```
  System.out.println("4. Lihat Barang Teratas");
            System.out.println("5. Keluar");
case 4:
                    gudang.lihatBarangTeratas();
                    break;
```

2.2.2 Verifikasi Hasil Percobaan



2.2.3 Pertanyaan

1. Pada method **konversiDesimalKeBiner**, ubah kondisi perulangan menjadi **while (kode != 0)**, bagaimana hasilnya? Jelaskan alasannya!

2. Jelaskan alur kerja dari method **konversiDesimalKeBiner**!

JAWABAN:

1. Perbedaan dalam hasil mungkin terjadi ketika kode awalnya adalah 0 atau bilangan negatif. Potongan pertama tidak akan menjalankan iterasi jika kode awalnya adalah 0, sementara potongan kedua akan menjalankan setidaknya satu iterasi jika kode awalnya adalah 0. Namun, jika kode awalnya adalah bilangan negatif, kedua potongan kode tersebut tidak akan menjalankan iterasi sama sekali.

2.

- Selama kode tidak sama dengan 0, lakukan langkah-langkah berikut:
- Hitung sisa pembagian kode dengan 2 dan simpan hasilnya ke dalam variabel sisa.
- Masukkan sisa ke dalam stack. Bagi nilai kode dengan 2 untuk iterasi berikutnya.
- Setelah loop pertama selesai, buat sebuah string kosong biner yang akan digunakan
- untuk menyimpan hasil konversi biner.
- Selama stack tidak kosong, lakukan langkah-langkah berikut:
- Keluarkan nilai dari stack menggunakan metode pop() dan tambahkan ke string biner.
- Kembalikan nilai biner yang telah terisi dengan representasi biner dari bilangan
- desimal awal.

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?

2. Jelaskan alur kerja method konversi!

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

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

JAWABAN:

1. Alasan Nilai Sama: Pada method derajat, beberapa operator memiliki nilai balik (tingkat precedence) yang sama (misalnya '*' dan '/'). Hal ini karena operator tersebut memiliki prioritas yang sama dalam kebanyakan ekspresi matematika. Ini berarti operator tersebut dievaluasi dari kiri ke kanan ketika ditemui dalam ekspresi infiks.

Dampak dari Nilai Berbeda: Mengubah nilai balik menjadi unik untuk setiap operator berpotensi mengubah logika konversi. Jika tingkat precedence yang baru tidak mengikuti aturan matematika standar, ekspresi postfix yang dihasilkan mungkin salah.

2.

Inisialisasi:

Buat string kosong P untuk menyimpan ekspresi postfix.Inisialisasi variabel karakter c untuk menyimpan karakter yang sedang diproses.Masukkan kurung buka (ke dalam stack.

Iterasi Lewat Infix:

Operan: Jika c adalah operan, tambahkan ke P.Kurung Buka: Jika c adalah (, masukkan ke dalam stack.Kurung Tutup: Jika c adalah), keluarkan elemen dari stack dan tambahkan ke P hingga bertemu (, lalu buang).Operator: Jika c adalah operator:Keluarkan elemen dari stack dan tambahkan ke P hingga elemen teratas memiliki precedence lebih kecil dari c.Masukkan c ke dalam stack.

Penyelesaian:

Keluarkan semua elemen dari stack dan tambahkan ke P.

Pengembalian:

Kembalikan string P yang berisi ekspresi postfix.

3. kode c = Q.charAt(i); memainkan peran vital dalam memungkinkan method konversi untuk secara efektif mengubah ekspresi infiks ke bentuk postfix yang sesuai. Dengan mengekstrak dan memproses karakter individual, method ini dapat secara akurat menentukan jenis simbol dan menerapkan aturan konversi yang tepat.

2.4 Latihan Praktikum

1.Method lihatBarangTerbawah

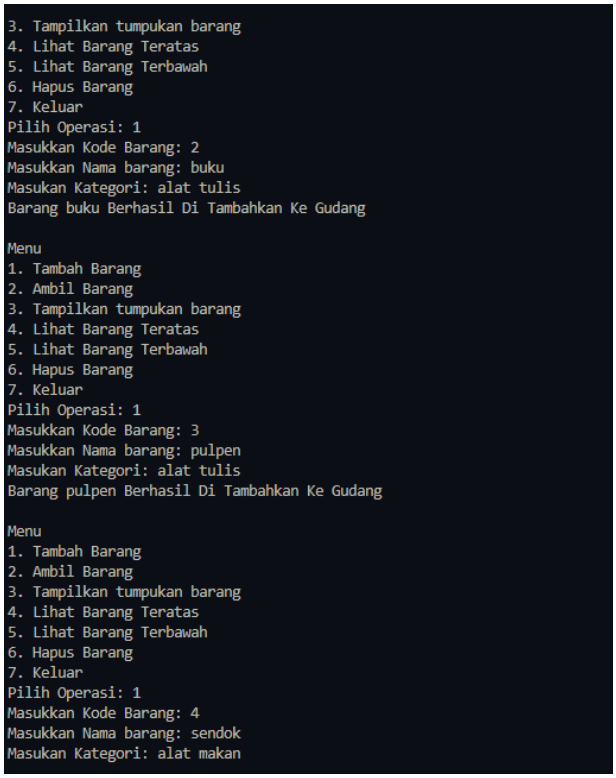
2. Method cariBarang

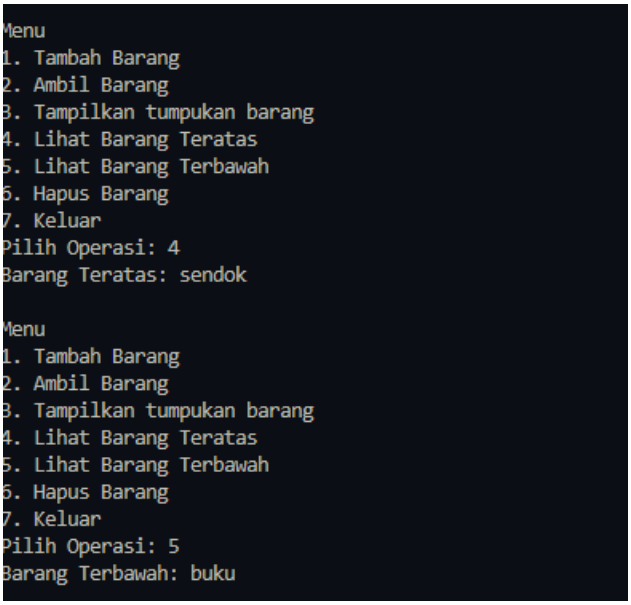
JAWABAN:

1.

```
public Barang05 lihatBarangTerbawah(){
        if (!isEmpty()){
            Barang05 barangTerbawah = tumpukan[0];
            System.out.println("Barang Terbawah:
"+barangTerbawah.nama);
            return barangTerbawah;
        }else{
            System.out.println("Tumpukan Barang Kosong.");
            return null;
        }
```

```
case 5:
                    gudang.lihatBarangTerbawah();
                    break;
```





2.

```
public void hapusBarang(int kodeBarang) {
        if (!isEmpty()) {
            int index = -1;
            // Cari index barang yang akan dihapus
            for (int i = 0; i <= top; i++) {
                if (tumpukan[i].kode == kodeBarang) {
                    index = i;
                    break;
                }
            }
            // Hapus barang jika ditemukan
            if (index != -1) {
                for (int i = index; i < top; i++) {
                    tumpukan[i] = tumpukan[i + 1];
                }
                top--;
                System.out.println("Barang dengan kode " +
kodeBarang + " berhasil dihapus.");
            } else {
                System.out.println("Barang dengan kode " +
kodeBarang + " tidak ditemukan.");
            }
        } else {
            System.out.println("Tumpukan Barang Kosong.");
        }
    }
    }
```

```
    case 6:
                    System.out.print("Masukkan Kode Barang Yang Akan
DI Hapus: ");
                    kode = scanner.nextInt();
                    gudang.hapusBarang(kode);
                    break;
```

```
7. Keluar
Pilih Operasi: 3
Rincian tumpukan barang di gudang:
Kode 4: sendok (Kategori alat makan)
Kode 3: pulpen (Kategori alat tulis)
Kode 2: buku (Kategori alat tulis)

Menu
1. Tambah Barang
2. Ambil Barang
3. Tampilkan tumpukan barang
4. Lihat Barang Teratas
5. Lihat Barang Terbawah
6. Hapus Barang
7. Keluar
Pilih Operasi: 6
Masukkan Kode Barang Yang Akan DI Hapus: 3
Barang dengan kode 3 berhasil dihapus.

Menu
1. Tambah Barang
2. Ambil Barang
3. Tampilkan tumpukan barang
4. Lihat Barang Teratas
5. Lihat Barang Terbawah
6. Hapus Barang
7. Keluar
Pilih Operasi: 3
Rincian tumpukan barang di gudang:
Kode 4: sendok (Kategori alat makan)
Kode 2: buku (Kategori alat tulis)
```