JOBSHEET VII

**STACK**

# Tujuan Praktikum

Setelah melakukan materi praktikum ini, mahasiswa mampu:

* 1. Membuat struktur data Stack
  2. Menerapkan algoritma Stack ke dalam program

# Praktikum

* 1. **Percobaan 1: Penyimpanan Tumpukan Barang dalam Gudang**

## Waktu Percobaan : 90 Menit

Sejumlah barang akan disimpan ke dalam gudang secara bertumpuk dengan menerapkan prinsip Stack. Perhatikan Class Diagram Barang berikut ini:

|  |
| --- |
| Barang**<NoAbsen>** |
| kode: int nama: String  kategori: String |
| Barang**<NoAbsen>**(kode: int, nama: String, kategori: String) |

Selanjutnya, untuk menyimpan barang di dalam gudang, diperlukan class Gudang yang berperan sebagai Stack tempat penyimpanan data barang. Atribut dan method yang terdapat di dalam class Gudang merepresentasikan pengolahan data menggunakan struktur Stack. Perhatikan Class Diagram Gudang berikut ini:

|  |
| --- |
| Gudang**<NoAbsen>** |
| tumpukan: Barang[] size: int  top: int |
| Gudang**<NoAbsen>**(kapasitas: int) cekKosong(): boolean  cekPenuh(): boolean tambahBarang(brg): void ambilBarang(): Barang lihatBarangTeratas(): Barang  tampilkanBarang(): void |

***Catatan****: Tipe data pada variabel* ***tumpukan*** *menyesuaikan dengan data yang akan disimpan di dalam Stack. Pada percobaan ini, data yang akan disimpan merupakan array of object dari* ***Barang****, sehingga tipe data yang digunakan adalah* ***Barang****.*

Berdasarkan dua class diagram tersebut, program menggunakan bahasa Java.

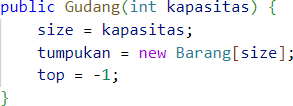
## Langkah-langkah Percobaan

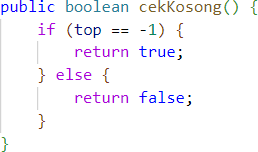
* + - 1. **Class Barang**

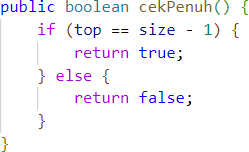
1. Buka text editor. Buat file baru, beri nama **Barang<NoAbsen>.java**
2. Lengkapi class **Barang** dengan atribut yang telah digambarkan di dalam class diagram Barang, yang terdiri dari atribut **kode**, **nama**, dan **kategori**
3. Tambahkan konstruktor berparameter pada class Barang sesuai dengan class diagram Barang

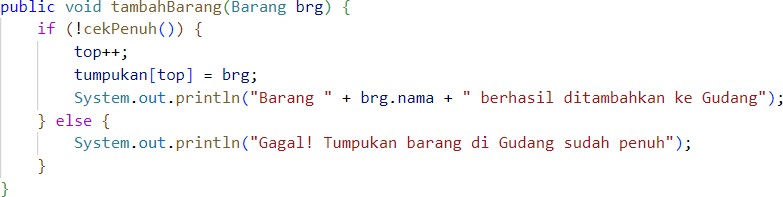
## 2. Class Gudang

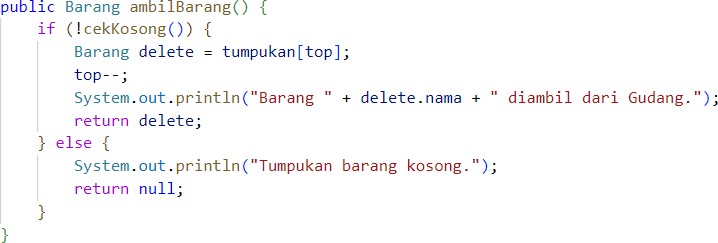
1. Setelah membuat class Barang, selanjutnya perlu dibuat class **Gudang<NoAbsen>.java** sebagai tempat untuk mengelola tumpukan barang. Class Gudang merupakan penerapan dari Stack
2. Lengkapi class **Gudang** dengan atribut yang telah digambarkan di dalam class diagram Gudang, yang terdiri dari atribut **tumpukan**, **size**, dan **top**
3. Tambahkan konstruktor berparameter pada class Gudang untuk melakukan inisialisasi kapasitas maksimum data barang yang dapat disimpan di dalam stack, serta mengeset indeks awal dari pointer top



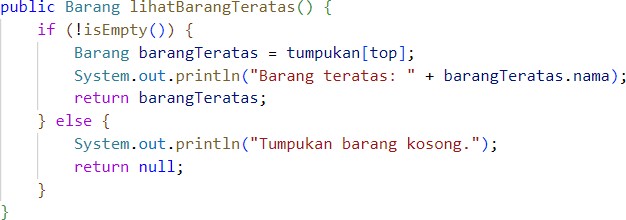
1. Pada class Gudang, buat method **cekKosong** bertipe boolean untuk mengecek apakah tumpukan barang di dalam gudang masih kosong
2. Selanjutnya, buat method **cekPenuh** bertipe boolean untuk mengecek apakah tumpukan barang di dalam gudang sudah terisi penuh sesuai kapasitas



1. Untuk dapat menambahkan barang ke dalam tumpukan di gudang, maka buat method **tambahBarang** yang merepresentasikan **push**. Method ini menerima parameter **brg** yang berupa object **Barang**
2. Pengambilan barang dari tumpukan di gudang dilakukan dengan menggunakan method **ambilBarang** yang merepresentasikan **pop**. Method ini tidak menerima parameter apapun karena barang yang diambil atau dikeluarkan pasti berada di posisi teratas

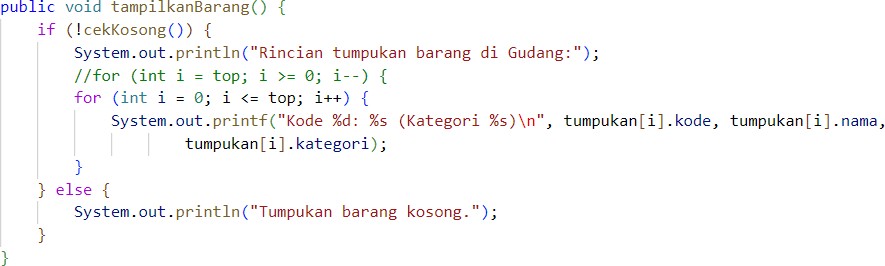


*Catatan: Apabila diperlukan informasi mengenai data barang yang diambil, maka tipe kembalian harus berupa object Barang. Sebaliknya, tipe kembalian void dapat digunakan jika data barang yang dikeluarkan tidak akan diolah atau digunakan lagi*

1. Buat method **lihatBarangTeratas** yang merepresentasikan **peek** untuk dapat mengecek tumpukan barang di posisi paling atas

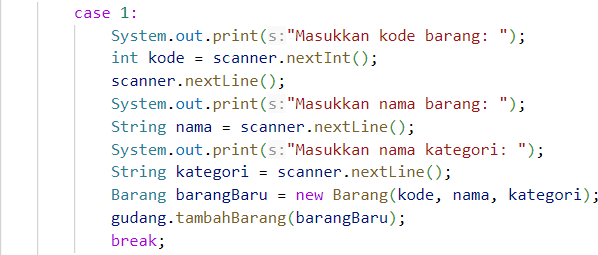
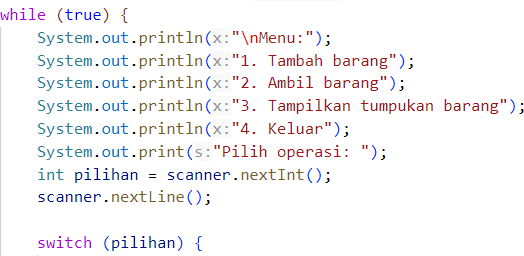
***Perbaiki kode program, jika menurut Anda terdapat kesalahan***

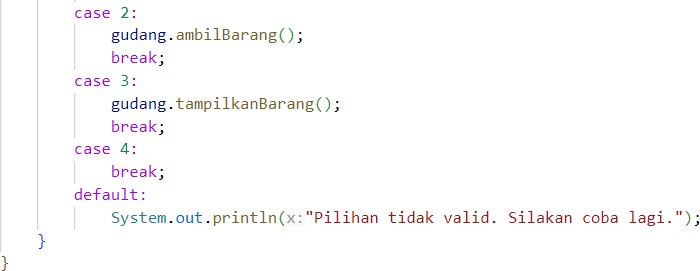
1. Tambahkan method **tampilkanBarang** untuk dapat menampilkan semua rincian tumpukan barang di gudang



## 3. Class Utama

1. Buat file baru, beri nama **Utama<NoAbsen>.java**
2. Tuliskan struktur dasar bahasa pemrograman Java yang terdiri dari fungsi **main**
3. Di dalam fungsi **main,** lakukan instansiasi object Gudang bernama **gudang** dengan nilai parameternya adalah 7.
4. Deklarasikan Scanner dengan nama variabel **scanner**
5. Tambahkan menu untuk memfasilitasi pengguna dalam memilih operasi Stack dalam mengelola data barang di gudang menggunakan struktur perulangan **While**



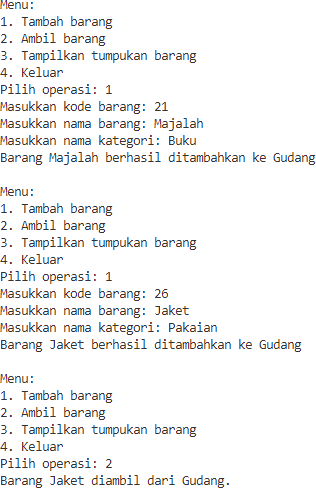


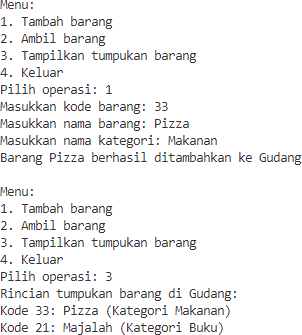
## Commit dan push kode program ke Github

1. Compile dan run program.

## Verifikasi Hasil Percobaan

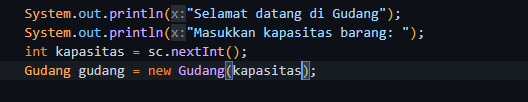
Cocokkan hasil compile kode program Anda dengan gambar berikut ini.





## Pertanyaan

1. Lakukan perbaikan pada kode program, sehingga keluaran yang dihasilkan sama dengan verifikasi hasil percobaan! Bagian mana saja yang perlu diperbaiki? T*dak ada yang harus diperbaiki.*
2. Berapa banyak data barang yang dapat ditampung di dalam tumpukan? Tunjukkan potongan kode programnya! 7 barang
3. Mengapa perlu pengecekan kondisi **!cekKosong()** pada method **tampilkanBarang**? Kalau kondisi tersebut dihapus, apa dampaknya? Method tersebut digunakan untuk mengecek apakah ada barang dalam tumpukan, jika method tersebut dihapus maka tidak ada pengecekan, dan apabila data kosong tidak ada pemberitahuan, dan hanya tampil kosong.
4. Modifikasi kode program pada class **Utama** sehingga pengguna juga dapat memilih operasi lihat barang teratas, serta dapat secara bebas menentukan kapasitas gudang!



1. **Commit dan push kode program ke Github**

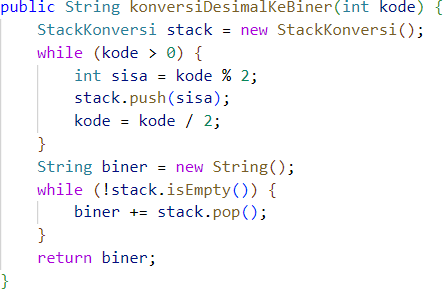
# Percobaan 2: Konversi Kode Barang ke Biner

## Waktu Percobaan: 30 Menit

Sampai tahap ini, proses pengelolaan data barang menggunakan konsep Stack telah berhasil dibuat pada Percobaan 1. Selanjutnya, pada Percobaan 2 ini ditambahkan method baru yang berfungsi untuk mengonversi kode barang bertipe int ke dalam bentuk biner saat barang tersebut diambil atau dikeluarkan dari tumpukan.

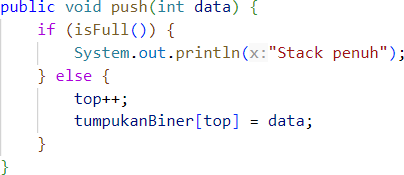
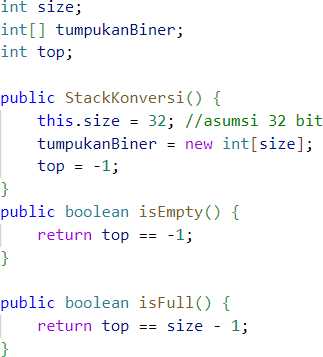
## Langkah-langkah Percobaan

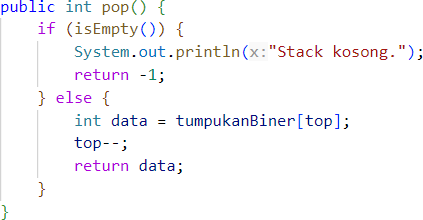
1. Buka kembali file **Gudang<NoAbsen>.java**
2. Tambahkan method **konversiDesimalKeBiner** dengan menerima parameter **kode** bertipe int



Pada method ini, terdapat penggunaan **StackKonversi** yang merupakan penerapan Stack, sama halnya dengan class Gudang. Hal ini bertujuan agar Stack untuk barang berbeda dengan Stack yang digunakan untuk biner. Oleh karena itu, buat file baru bernama **StackKonversi<NoAbsen>.java**

*Catatan: Perlu diingat bahwa pada dasarnya semua class Stack mempunyai operasi (method) yang sama. Hal yang membedakan adalah aktivitas spesifik yang perlu dilakukan, misalnya setelah menambah atau mengeluarkan data.*

1. Tambahkan empat method yaitu **isEmpty**, **isFull**, **push**, dan **pull** sebagai operasi utama Stack pada class **StackKonversi**



1. Agar kode barang dikonversi ke dalam bentuk biner saat barang tersebut diambil atau dikeluarkan dari tumpukan, maka tambahkan baris kode program pada method **ambilBarang**

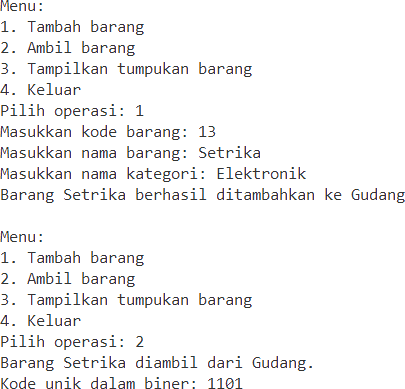


1. Compile dan run program.

## Commit dan push kode program ke Github

* + 1. **Verifikasi Hasil Percobaan**

Cocokkan hasil compile kode program Anda dengan gambar berikut ini.



## Pertanyaan

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

Sama aja, karna kode tidak sama 0

1. Jelaskan alur kerja dari method **konversiDesimalKeBiner**!
   * + 1. Instansiasi stack dari kelas StackKonversi
       2. Melakukan perulangan saat kode lebih dari 0, lalu mengambil sisa dari kode yang dibagi 2, sisa tersebut dipush ke stack. Lalu variabel kode diganti dengan hasil dari kkode dibagi 2.
       3. Membuat string kosong, lalu dia mengambil string dari stack, selagi stack belum kosong maka akan diambil terus dan di append ke string.

# Percobaan 3: Konversi Notasi Infix ke Postfix

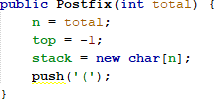
## Waktu Percobaan: 90 Menit

Pada percobaan ini, dilakukan pembuatan kode program untuk melakukan konversi notasi infix menjadi notasi postfix. Perhatikan Class Diagram Postfix berikut ini:

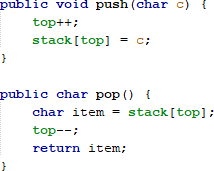
|  |
| --- |
| Postfix**<NoAbsen>** |
| n: int top: int  stack: char[] |
| Postfix(total: int) push(c: char): void pop(): void  IsOperand(c: char): boolean IsOperator(c: char): boolean derajat(c: char): int  konversi(Q: String): string |

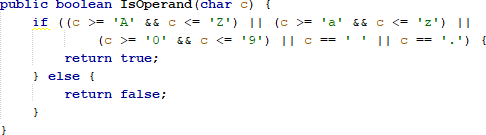
## Langkah-langkah Percobaan

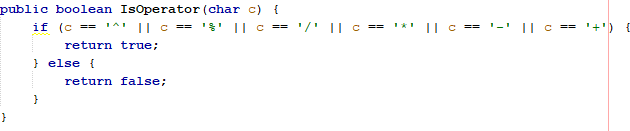
1. Buat file baru bernama **Postfix<NoAbsen>.java**
2. Tambahkan atribut **n**, **top**, dan **stack** sesuai Class Diagram Postfix tersebut
3. Tambahkan pula konstruktor berparameter seperti gambar berikut ini.

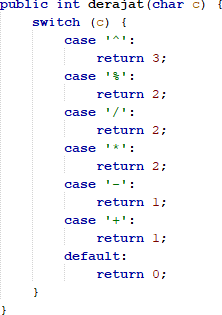


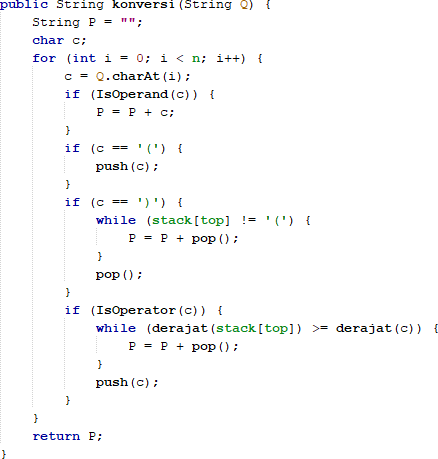
1. Buat method **push** dan **pop** bertipe void.

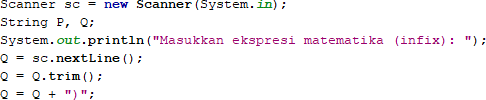


1. Buat method **IsOperand** dengan tipe boolean yang digunakan untuk mengecek apakah elemen data berupa operand.
2. Buat method **IsOperator** dengan tipe boolean yang digunakan untuk mengecek apakah elemen data berupa operator.



1. Buat method **derajat** yang mempunyai nilai kembalian integer untuk menentukan derajat operator.
2. Buat method **konversi** untuk melakukan konversi notasi infix menjadi notasi postfix dengan cara mengecek satu persatu elemen data pada **String Q** sebagai parameter masukan.



1. Selanjutnya, buat class baru dengan nama **PostfixMain<NoAbsen>.java**. Buat method main, kemudian buat variabel **P** dan **Q**. Variabel P digunakan untuk menyimpan hasil akhir notasi postfix setelah dikonversi, sedangkan variabel Q digunakan untuk menyimpan masukan dari pengguna berupa ekspresi matematika dengan notasi infix. Deklarasikan variabel Scanner dengan nama sc, kemudian panggil fungsi *built-in* **trim** yang digunakan untuk menghapus adanya spasi di depan atau di belakang teks dari teks persamaan yang dimasukkan oleh pengguna.

Penambahan string **“)”** digunakan untuk memastikan semua simbol/karakter yang masih berada di stack setelah semua persamaan terbaca, akan dikeluarkan dan dipindahkan ke postfix.

1. Buat variabel total untuk menghitung banyaknya karakter pada variabel Q.



1. Lakukan instansiasi objek dengan nama **post** dan nilai parameternya adalah total. Kemudian panggil method **konversi** untuk melakukan konversi notasi infix Q menjadi notasi postfix P.

A close up of text  Description automatically generated

1. Compile dan run program.

## Commit dan push kode program ke Github

* + 1. **Verifikasi Hasil Percobaan**

Cocokkan hasil compile kode program Anda dengan gambar berikut ini.

A screenshot of a computer  Description automatically generated

## 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 ada beberapa operasi yang nilainya sama seperti perkalia, pembagian, dimana itu lebih besar dari pengurangan atau penjumlahan. Apabila nilainya berbeda beda dan dan tidak tepat maka hasil operasinya juga tidak tepat.

* + - 1. Jelaskan alur kerja method **konversi**!
  1. Membuat variable P yang berisi string, dan variabel char c.
  2. Melakukan perulangan dan mengganti isi dari variabel c dengan charatcter dari string Q.
  3. Melakukan beberapa pengecekan, jika isOperand maka c akan di append ke string P.
  4. Jika c = ( maka c akan di push
  5. Jika c = ) maka akan melakukan looping untuk meng pop stack dan diappend ke string P sampai stacknya == )
  6. Jika tidak maka akan melakukan looping selama derajat dari c kurnag dari sama dengan derajat dari stack teratas. Lalu satcknya di pop dan di append ke string P. Lalu setelah perulangan selesai makan c akan di push ke stack.
  7. Return string P.
     + 1. Pada method konversi, apa fungsi dari potongan kode berikut?



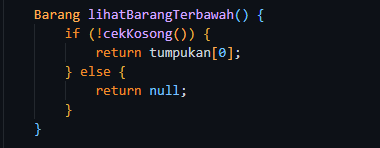
Mengambil 1 karakter string dari p dengan index = i

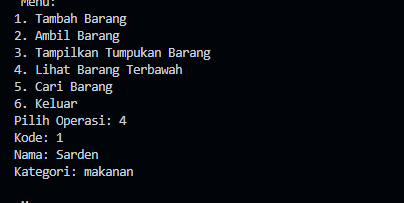
# Latihan Praktikum

## 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





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

