Pembahasan Lab 2

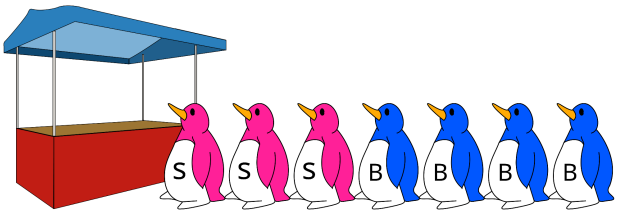
Deksripsi Singkat

Hayate memiliki toko es krim di kutub utara. Karena pesanan setiap harinya semakin banyak, Hayate ingin memiliki sistem otomatis untuk membantunya mengatasi antrean geng penguin. Sistem tersebut dapat menerima tiga event yang mungkin terjadi:

- **DATANG G_i X_i:** Sebanyak X_i penguin baru dari geng G_i datang dan antre di toko Hayate. Event ini mengeluarkan jumlah penguin pada antrean toko saat itu (termasuk penguin yang baru datang).
- **LAYANI Y_i:** Hayate melayani Y_i penguin terdepan di antrean toko. Event ini mengeluarkan nama geng terakhir yang dilayani Hayate.
- **TOTAL G_i:** Hayate ingin tahu saat ini telah melayani berapa penguin dari geng G_i. Event ini mengeluarkan nama geng terakhir yang dilayani Hayate.

Ide

Pertama, mari kita coba approach *naive* dimana penguin disimpan dalam queue secara **satu per satu** seperti ilustrasi dan potongan kode di bawah.




```
# Event DATANG
String namaGeng = in.next();
int X = in.nextInt()
for (int i = 0; i < X; i++) {
    queue.add(namaGeng);
}
```

Hal ini dapat menyebabkan munculnya verdict TLE dan SG.

- TLE, karena pada kasus terburuk ada 200.000 event. Setiap event bisa ada 10.000 penguin. Ini berarti ada 200.000 * 10.000 = $2 * 10^9$ operasi. Tentu saja ini TLE di java.
- SG, karena pada kasus terburuk yang sama, ini berarti bisa ada $2 * 10^9$ string disimpan di suatu object dalam suatu waktu. Dengan asumsi panjang maksimal (10 karakter), ini berarti ukuran penyimpanan yang dibutuhkan itu $2 * 10^{10}$ byte, atau 20 GB. Tentu melebihi batas memori yang diberikan yaitu hanya 256 MB

Akan lebih baik apabila kita menyimpan queue dengan memanfaatkan OOP. Sehingga satu geng cukup diwakili oleh satu object pada queue. Kemudian, kita juga bisa menggunakan ADT Map untuk menyimpan banyak penguin dalam Geng G_i yang sudah dilayani.

```
class Geng {
    int X;
    String name;

    public Geng(int _X, String _Name) {
        X = _X;
        name = _Name;
    }
}
```

Dengan demikian, event DATANG dapat diatasi dengan membuat objek baru dengan atribut nama dan X (banyak penguin dalam geng).

```
static private int handleDatang(String Gi, int Xi) {
        queueSize += Xi;
        q.add(new Geng(Xi, Gi));

string(         return queueSize;
    }
```

Selanjutnya, kita dapat membagi event LAYANI menjadi dua case:

- Case 1, dimana jumlah penguin yang dilayani >= jumlah penguin dalam geng terdepan
    - 1 object geng tersebut di-*poll* dari queue
- Case 2, dimana jumlah penguin yang dilayani < jumlah penguin dalam geng terdepan
    - Cukup meng-*update* jumlah penguin dalam geng tersebut sebanyak yang dilayani, **tidak** di-*poll* dari queue.

```
    static void updatePenguinLayani(String Gi, int freq) {
        int layanid = hm.getOrDefault(Gi, 0);
        queueSize -= freq;
        layanid += freq;
        hm.put(Gi, layanid);
    }

    static private String handleLayani(int Yi) {

        String lastPopped = "";
        while(Yi > 0) {
            Geng top = q.peek();
```

```
            lastPopped = top.name;

            // case 1: semua penguin disini di poll
            if(top.X <= Yi) {
                Yi -= top.X;
                updatePenguinLayani(top.name, top.X);
                q.remove();

            } else { // Yi < top.X, stop disini
                updatePenguinLayani(top.name, Yi);
                top.X -= Yi;
                Yi = 0;

                assert(q.peek().X == top.X);
            }
        }
        return lastPopped;
    }
```

Terakhir, untuk event TOTAL digunakan HashMap untuk menyimpan jumlah penguin yang sudah dilayani dari masing-masing geng. HashMap tersebut akan di-*update* setiap dipanggil method updatePenguinLayani. Sehingga ketika dibutuhkan kita cukup melakukan pemanggilan get dari HashMap yang sudah ada. Perhatikan tricky case bisa saja **TOTAL dipanggil dengan nama geng yang belum pernah muncul sebelumnya**.

```
static private int handleTotal(String Gi) {
        return hm.getOrDefault(Gi, 0);
    }
```

Analisis Kompleksitas

- Event DATANG mempunyai kompleksitas $O(1)$karena tidak ada penggunaan for-loop ataupun while-loop dalam pemanggilan handleDatang.
- Event LAYANI mempunyai kompleksitas $O(N)$secara keseluruhan karena setiap event hanya masuk dan keluar dari queue sekali.
- Event TOTAL mempunyai kompleksitas $O(1)$karena method get dari Java Map bersifat konstan.

Sehingga kompleksitas dari program ini adalah $O(1) + O(N) + O(1) = O(N)$.