Makalah Seminar SeNTIK 2017 – STMIK JAKARTA STI&K
26 Juli 2017

# IMPLEMENTASI POLA DESAIN DEKORATOR UNTUK MENINGKATKAN EFISIENSI ALGORITME KODE PROGRAM

Raden Budiarto, Dita Ningtyas
⋮
Sistem Informasi, STMIK Jakarta STI&K
Jl. BRI Radio Dalam, Jakarta Indonesia
(raden@jak-stik.ac.id)
⋮

**Abstrak**

*Pada hampir setiap komputasi terdapat berbagai macam pertimbangan yang harus dilakukan. Salah satu objek yang menjadi bahan pertimbangan penting adalah menentukan jenis algoritme yang akan mengurangi waktu seminimal mungkin yang diperlukan untuk menyelesaikan suatu komputasi. Penelitian ini mengeksplorasi teknik pola desain (design pattern) pada algoritme pemrograman kemudian menerapkan pada contoh kasus untuk diteliti hasil efisien algoritmenya. Penelitian ini mencoba untuk membuktikan secara empiris penerapan pola desain dapat meningkatkan efisiensi kode program secara signifikan. Penelitian ini menjabarkan teknik pola desain dekorator dimanfaatkan lebih jauh yakni untuk dianalisis efektivitas dan efisiensi penggunaannya terhadap performa sebuah program secara keseluruhan. Metode penelitian yang digunakan adalah metode eksperimen di mana data diambil langsung dari hasil percobaan. Variabel yang diukur pada penelitian ini adalah pemakaian sumber daya CPU dan memori, jumlah baris kode sumber dan kecepatan eksekusi program. Data penelitian diambil terutama berdasarkan dari hasil pengamatan langsung. Hasil pengolahan data menunjukkan adanya penurun yang signifikan terhadap pemakaian sumber daya CPU dan memori juga jumlah baris program secara keseluruhan.*
***Kata Kunci**:pola desain, pola dekorator, efisiensi algoritme, rekayasa perangkat lunak*

## Pendahuluan

Salah satu topik lanjutan dalam pemrograman berorientasi objek adalah *design pattern* (pola desain). Dalam dunia pemrograman berorientasi objek, pola desain adalah solusi umum yang dapat digunakan secara berulang untuk masalah sering terjadi dalam desain perangkat lunak [1]. Sebuah pola desain bukanlah seperti *plugin* atau *library* yang dapat ditambahkan langsung ke kode. Melainkan pola desain layaknya sebuah *template* atau sebuah resep untuk dapat memecahkan masalah yang umum terjadi dalam berbagai situasi. Dunia pemrograman telah berkembang ke tahap yang matang, hal ini membuat para *programmer* yang terlebih dahulu menghadapi masalah yang sering kali ditemui saat membuat program kemudian membuatkan solusinya ke dalam bentuk sebuah formula.

Saat ini istilah dalam pola desain sudah diterima secara luas di kalangan *programmer*. Jika seseorang menghadapi kasus di mana ketika ada sebuah objek yang di-*update* maka mengirimkan notifikasi ke beberapa objek lain (seperti *newsletterupdate*) maka pada saat bersamaan sebenarnya banyak *programmer* yang hadapi masalah serupa. Dari berbagai masalah yang umum ditemui ini, diformulasikan sebuah solusi ke dalam sebuah *pattern* (pola) sehingga mudah untuk diterapkan secara berulang. Ke depan formula dari solusi tersebut akan digunakan kembali untuk memecahkan masalah yang serupa. Sebagai contoh saat berhadapan dengan masalah sebuah status sebuah objek yang harus dikembalikan (*undo*) ke status sebelumnya maka dapat digunakan teknik "*momento"* [2]. Demikian pula teknik pola desain lainnya, ke semuanya digunakan untuk mempermudah memecahkan masalah tertentu.

Pola desain memiliki dua manfaat utama. Pertama, pola desain memberikan sebuah solusi untuk memecahkan masalah yang terkait dengan pengembangan perangkat lunak menggunakan solusi yang telah terbukti berhasil. Pendekatan

solusi ini akan memudahkan pengembangan modul yang sangat kohesif dengan kopling minimal. Kedua, pola desain membuat komunikasi antar desainer lebih efisien [3]. Profesional perangkat lunak dapat saling berkomunikasi efektif pada tingkat rancangan ketika nama pola desain yang digunakan untuk memecahkan masalah tertentu saat membahas perancangan sistem. Bagaimana pun manfaat yang sedemikian besar ini tidak akan dimanfaat dengan sangat minimnya penelitian dan publikasi yang membahas pemanfaatannya.

Hal yang mendorong pola desain terus berkembang adalah faktor efisiensi algoritme kode program. Faktor ini selalu berbanding lurus dengan kinerja program secara keseluruhan. Milton [4] dalam publikasinya menyatakan bahwa setidaknya ada 3 indikator utama yang menyatakan bahwa sebuah algoritme kode program dapat dikatakan efisien, indikator tersebut adalah: jumlah baris kode, pemakaian sumber daya dan waktu eksekusi program. Sekalipun dengan algoritme yang lebih efisien juga berarti lebih efisien dalam pemanfaatan sumber daya baik itu CPU maupun memori ke semua hal tersebut bukan berarti perkembangan efisien algoritme berjalan cepat. David May [5] menyatakan sebuah teori(dikenal dengan hukum May) yang bermakna "Efisiensi perangkat lunak berkurang menjadi separuh setiap 18 bulan". Pernyataan ini yang bertolak belakang dengan hukum Moore, sekaligus menjadi kompensasi dari perkembangan perangkat keras.

Penelitian ini mencoba untuk mencari hubungan antara penerapan pola desain dengan efisiensi algoritme kode program. Langkah pengujian eksperimental dilakukan untuk membuktikan secara empiris penerapan pola desain dapat meningkatkan efisiensi kode program secara signifikan. Untuk membatasi lingkup penelitian maka diambil salah satu teknik pola desain yakni *decorator pattern*. Pola dekorator dipilih karena teknik ini merupakan teknik yang masih relatif baru dan belum populer digunakan. Dengan publikasi penelitian ini diharapkan pola dekorasi dapat semakin dikenal penggunaan dan pemanfaatannya. Penelitian tentang algoritme ini diharapkan dapat memberikan wawasan dalam penyelesaian masalah yang lebih efisien dan efektif namun tetap independen dari bahasa pemrograman atau platform tertentu.

## Tinjauan Pustaka

Pola desain mendapatkan popularitas dalam bidang ilmu komputer setelah buku "*Design Pattern: Elemen Reusable Software Object-Oriented*" diterbitkan pada tahun 1994 oleh empat sekawan: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (*"gang of four*"). Mereka merumuskan 23 pola desain yang terbagi ke dalam 3 kategori yakni *creational pattern* (pola desain yang berhubungan dengan penciptaan objek), *structural pattern* (pola desain yang berhubungan dengan struktur hubungan antar kelas seperti pewarisan dan komposisi) dan *behavioral pattern* (pola desain yang berhubungan dengan komunikasi antar objek) [6]. Buku ini kemudian mendapat berbagai kritik dari berbagai pengamat dan praktisi rekayasa perangkat lunak terutama dikarenakan buku tersebut fokus pada bahasa pemrograman C++ (tidak semuanya dapat diimplementasikan ke dalam bahasa pemrograman lainnya. Meskipun demikian istilah "*design pattern*" mendapat sambutan positif yang luas di kalangan praktisi rekayasa perangkat lunak, banyak di antaranya yang mencoba menambah koleksi pola desain terhadap berbagai masalah dari berbagai bahasa pemrograman yang berbeda [7].

Pola desain dekorator (*decorator pattern*) adalah pola desain yang memungkinkan untuk menambahkan perilaku ke objek individu baik statis atau dinamis, tanpa mempengaruhi perilaku lainnya yang sudah ada [8]. Pola dekorator berkaitan dengan masalah menambahkan fungsi pada sebuah objek serta dapat pula digunakan untuk memperluas fungsi dari objek statis, atau dalam beberapa kasus pada saat *run-time*. Fungsi-fungsi tersebut secara independen berasal dari kelas yang sama. Hal ini dicapai dengan merancang kelas dekorator baru yang membungkus kelas yang asli [9].

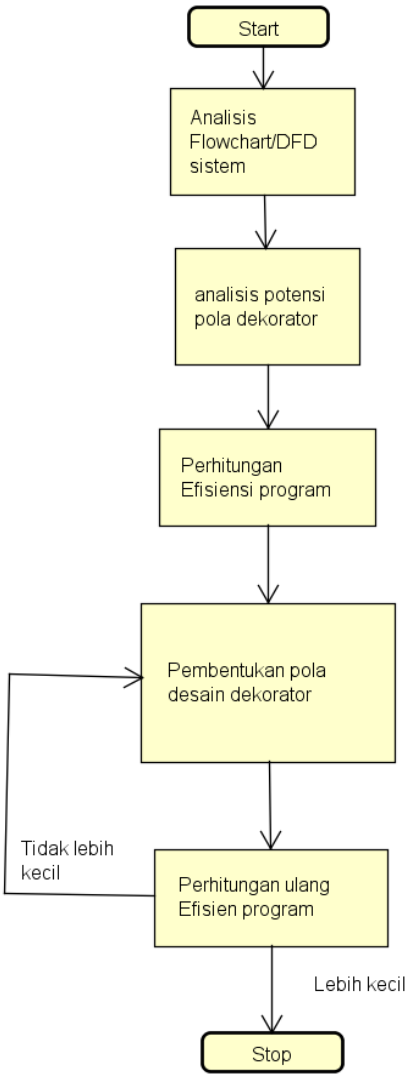
Pada penelitian ini, pola dekorator dimanfaatkan lebih jauh yakni untuk dianalisis efektivitas dan efisiensi penggunaannya terhadap performa sebuah program secara keseluruhan. Desain algoritme efisien untuk memecahkan masalah yang berhubungan dengan algoritme merupakan salah satu bidang penelitian yang paling penting dalam ilmu komputer [10]. Dalam mengukur efisiensi kode program terdapat berbagai parameter tolak ukur yang dapat digunakan. Secara umum kompleksitas dari sebuah algoritme dapat diukur dari segi ruang dan waktu. Ruang kompleksitas diukur dengan unsur-unsur seperti jumlah memori yang digunakan, jumlah baris kode program dan ukuran data struktur yang digunakan. Sementara kompleksitas waktu diukur dengan lama waktu yang diperlukan selama pelaksanaan algoritme. pada sisi lain terdapat kesalahpahaman yang ditemui seputar efisiensi misalnya, sering terdapat mitos kecepatan komputer, mengatakan bahwa komputer begitu sangat cepat sehingga tidak ada masalah dengan waktu pemrosesan algoritme. Banyak contoh dapat ditemukan untuk

menunjukkan bahwa apa pun kecepatan komputer atau akan, masih ada pentingnya dalam mempercepat pelaksanaan algoritme. Sampai saat ini masih terdapat banyak masalah komputasi yang sulit untuk dikerjakan bahkan dengan menggunakan kecepatan super komputer [11]. Selain itu, hasil dari sebuah algoritme mungkin saja terlalu lama dan tidak efisien sehingga tidak dapat diterima oleh pengguna.

## Metode Penelitian

Dilihat dari jenis masalah yang dihadapi dalam penelitian ini maka jenis penelitian ini termasuk ke dalam kategori penelitian eksperimental yaitu metode penelitian yang memungkinkan peneliti memanipulasi variabel dan meneliti akibat-akibatnya. Dalam sebuah metode penelitian eksperimental peneliti dapat mencari hubungan sebab akibat dengan memanipulasikan satu atau lebih variabel yang sedang diteliti. Berdasarkan jenis data yang dikumpulkan dan diolah maka penelitian ini termasuk dalam penelitian empiris. Pada penelitian empiris, penelitian dilakukan terhadap fakta yang diperoleh dari hasil observasi atau pengalaman. Objek yang diteliti lebih ditekankan pada kejadian yang sebenarnya ketimbang persepsi orang terhadap kejadian. Data yang ambil merupakan data aktual yang terjadi pada saat penelitian dari hasil observasi terhadap sistem yang berjalan. Jenis data dalam penelitian ini adalah data objektif yaitu jenis data penelitian yang berupa hasil pengamatan langsung terhadap objek yang diteliti dan bukan merupakan data subjektif yang berupa opini, sikap atau pengalaman dari seseorang.

Pada penelitian ini melibatkan beberapa langkah yang disusun dalam satuan kerangka penelitian. Alur penelitian ini dapat dilihat pada. Alur penelitian ini menunjukkan langkah demi langkah dari awal sampai akhir yang dilalui saat proses penelitian (gambar 1). Alur penelitian dimulai dari analisis aliran data dan proses yang ada dengan meninjau diagram *flowchart* dan data *flow* diagram sistem informasi pada objek yang diteliti. Selanjutnya dari setiap proses yang ada akan dianalisis untuk potensi penerapan teknik pola dekorator yang mungkin terjadi. Untuk memudahkan mendaftar semua kemungkinan potensi penerapan pola dekorator pada sistem maka daftar potensi tersebut dipilah menjadi dua kategori yakni kategori dapat diterapkan dan tidak.




Gambar 1. Alur Penelitian

Pada tahap berikutnya dilakukan perhitungan efisiensi algoritme kode program. Dalam penelitian ini digunakan 3 kategori utama [12] sebagai indikator dari efisien sebuah kode program yakni:

1. Jumlah baris kode program (*Lines of Code*)
2. Pemakaian sumber daya CPU, Memori RAM dan *disk*
3. Kecepatan eksekusi program

Ke semuanya indikator tersebut akan dihitung dan dijadikan acuan untuk perhitungan efisien algoritme program. Berikutnya tahap paling penting dalam penelitian ini yang juga merupakan *output* utama adalah modifikasi atau pembentukan pola dekorator. Setelah pola desain ini dirancang dan diterapkan maka terlebih dahulu fungsi dan tampilan program diperiksa untuk melihat apakah berjalan sama seperti sebelumnya atau ada perubahan. Jika tidak ada

perubahan fungsi atau tampilan maka selanjutnya akan dilakukan perhitungan ulang efisien algoritme kode program sama seperti pada tahap sebelumnya. Diharapkan setelah penerapan pola desain dekorator efisiensi kode program akan meningkat secara signifikan. Parameter ini ditunjukkan dengan jumlah baris kode program yang lebih sedikit, pemakaian sumber daya yang lebih kecil dan waktu eksekusi program yang lebih cepat.

Pola desain dekorator berkaitan dengan masalah menambahkan fungsi statis (atribut) atau dinamis (perilaku) pada sebuah objek. Sebagai contoh kasus dalam penelitian ini digunakan sebuah program untuk menentukan harga minuman "es *bubble*", minuman ini memiliki tambahan berupa taburan *topping* bervariasi seperti meses, susu, keju, biskuit, *whip cream*, dan *choco chip*. Minuman es *bubble* ini juga memiliki banyak rasa seperti avokad, jeruk, melon, stroberi, cokelat dan sebagainya. Maka pada contoh kasus ini merupakan objek yang tepat untuk mendekorasi minuman es *bubble* dengan dekorator *topping*. Terdapat sebuah kafe xyz yang beroperasi di wilayah Kebayoran Baru, Jakarta mempunyai program aplikasi sebagaimana dengan contoh yang disebutkan.

Data penelitian yang digunakan pada penelitian ini adalah data primer yang diambil langsung dari hasil pengamatan terhadap objek yang diteliti. Pengumpulan data dilakukan selama 2 Minggu, yaitu dari tanggal 23 Januari 2017 Januari sampai dengan 6 Februari 2017. Adapun data-data yang dibutuhkan dalam penelitian ini di antaranya:

1. *Flowchart* program
2. *Class* diagram
3. Kode sumber (*source code*) program
4. Data akses ke administrasi sistem

Adapun teknik pengumpulan data yang dilakukan dalam penelitian ini terdiri dari 3 macam, yaitu:

1. Studi pustaka

Tahap pertama yang dilakukan adalah mempelajari tentang teori dan topik yang akan dibahas. Dalam proses ini, semua teori yang berhubungan dengan topik "*Design Pattern*" dikumpulkan dari berbagai sumber seperti buku, jurnal, internet, dan sebagainya.

2. Pengamatan secara langsung

Pengamatan secara langsung terhadap sistem bertujuan untuk mempelajari bagaimana proses aliran data menjadi informasi dalam objek penelitian. Selanjutnya pengamatan langsung juga dilakukan untuk menghitung besaran skala efisiensi program.

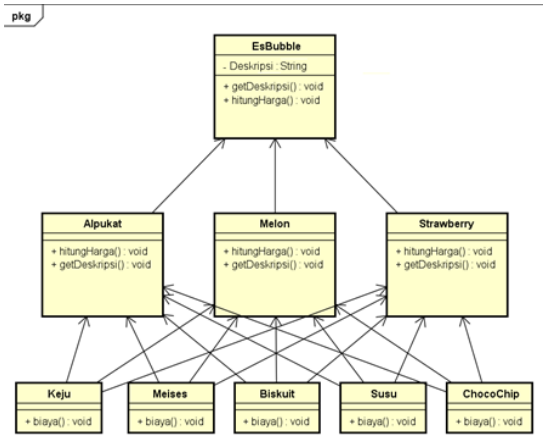
3. Wawancara dan tanya jawab

Wawancara dan tanya jawab ini dilakukan dengan berbagai pihak yang berhubungan dengan program aplikasi pada kafe xyz. Teknik wawancara dilakukan terhadap beberapa pihak yang terkait di antaranya pengembang aplikasi (*developer*), pemilik kafe dan pengguna akhir aplikasi yang dalam hal ini adalah kasir kafe untuk mengumpulkan semua data-data yang diperlukan dalam penelitian ini terutama skema aplikasi yang sedang berjalan. Proses ini juga bertujuan untuk mendapatkan gambaran serta uraian detail tentang cara kerja aplikasi yang sedang diteliti.

Data yang telah dikumpulkan dianalisis menggunakan alat bantu Microsoft Excel 2013 untuk input data numerik dan pembuatan grafik dan alat bantu Astah Professional digunakan untuk memodelkan aplikasi objek penelitian ke dalam bentuk diagram seperti *flowchart* dan *class* diagram. Setelah melakukan pengamatan dan wawancara terhadap *developer* sistem untuk menentukan apakah data dan prosedur yang dilakukan sudah valid, maka langkah selanjutnya dilakukan pengolahan data guna membentuk potensi penerapan pola desain dekorator. Jika terdapat potensi besar untuk efisiensi dengan menerapkan pola dekorator maka akan dilakukan modifikasi kode program dengan alat bantu IDE (*Integrated Development Environment*) Netbeans 8.1. Untuk mendapatkan hasil analisis efisiensi dari program digunakan alat bantu *WindowsPerformance Analyzer*. Sedangkan dari sisi perangkat keras, pengujian dilakukan pada komputer dengan CPU Intel Core i3 6300 3,7 Ghz, Memori RAM DDR3 16000 4 GB dan hardisk SATA 500 GB 5400 r.p.m. dengan sistem operasi Windows 7.

## Hasil dan Pembahasan

Setelah semua data yang dikumpulkan dilakukan tahap analisis data. Pendekatan pertama yang dilakukan adalah menganalisis permodelan objek penelitian. Didapati bahwa program yang ada menyelesaikan masalah perhitungan harga jual minuman dengan menggunakan teknik algoritme pewarisan sederhana. Di sini minuman es *bubble* merupakan kelas abstrak yang mewarisi beberapa kelas konkret berupa varian rasa seperti melon, stroberi dan sebagainya. Berikutnya ketika menambahkan varian *topping* ke masing-masing kelas dari varian rasa digunakan *subclass*. Permodelan ini diilustrasikan dengan kelas diagram pada **Error! Reference source not found.**.

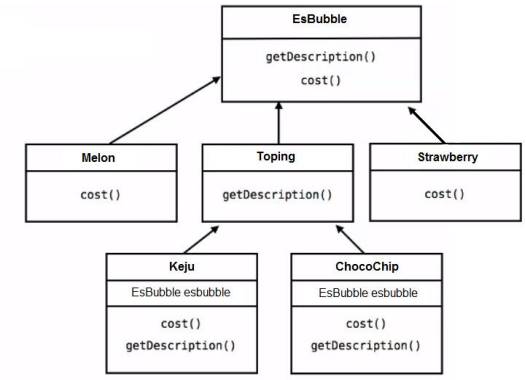



Gambar 2. Skema *Class* diagram pada aplikasi sebelumnya

Berdasarkan hasil analisis program aplikasi yang berjalan saat diteliti memiliki beberapa kelemahan di antaranya:

1. Algoritme ini akan menjadi rumit jika ke depan ada penambahan lagi varian rasa atau *topping* yang baru. Hal ini dikarenakan untuk setiap penambahan rasa baru akan dibuatkan sebuah kelas baru dan semua *topping* (*subclass*) harus dihubungkan dengan kelas baru tersebut
2. Ketika menambahkan sebuah varian *topping* yang baru maka harus membuat subclass baru kemudian dihubungkan dengan semua superclass yang ada sebelumnya.
3. Banyak duplikasi kode antar kelas dan terdapat hubungan *coupling* yang tinggi antar kelas dengan demikian kode program akan sulit di-*maintenance*

Teknik pewarisan adalah salah satu inti dari pemrograman berorientasi objek. Sering kali *programmer* pemula sering kali memiliki kecenderungan untuk berlebihan dalam menggunakan pewarisan. Bagaimana pun hal ini akan mengakibatkan desain yang tidak fleksibel dan sulit untuk di-*maintenance*. Solusi jalan keluar pada masalah aplikasi berjalan adalah menggunakan pola desain dekorator. Pada pola dekorator akan menggunakan teknik komposisi dengan cara yang berbeda dengan teknik pewarisan. Terdapat dua kelas utama dalam pola desain dekorator yakni komponen dan dekorator. Pada dasarnya cara kerja dari pola desain dekorator adalah dengan membungkus objek komponen utama dengan objek dekorator sehingga teknik ini sering kali disebut sebagai *wrapper*. Setiap objek dekorator maka akan membungkus objek dekorator sebelumnya.



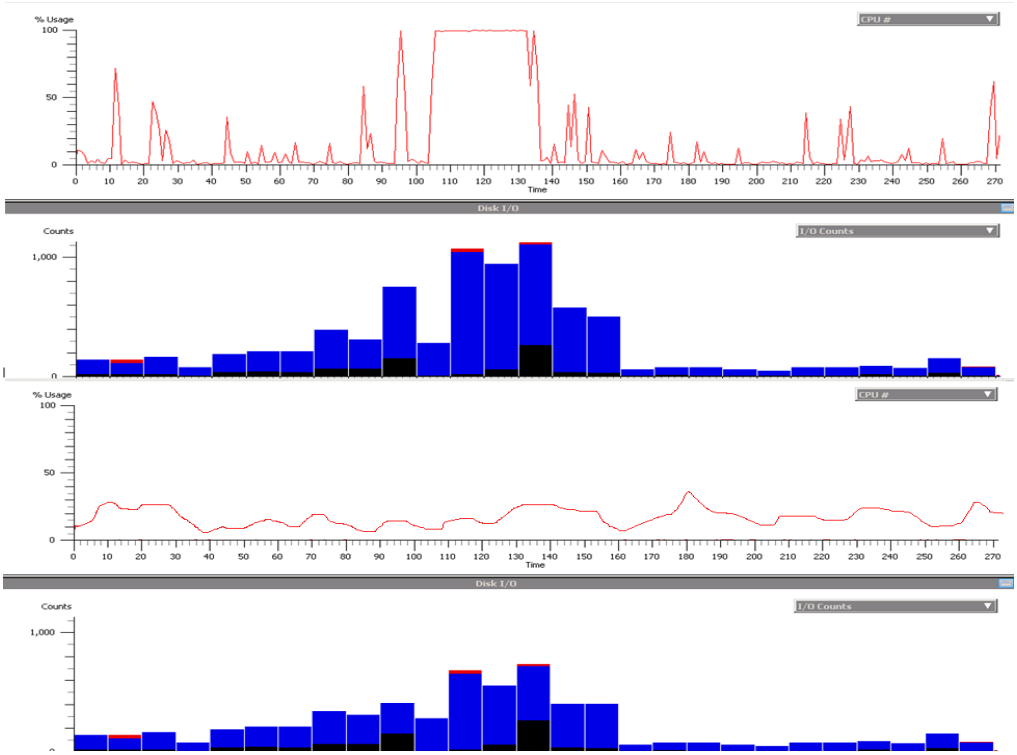

Gambar 3. Rancangan *Class* Diagram yang diusulkan

Pada kasus contoh program yang ditemui kelas *component* tidak lain adalah kelas es *bubble* yang juga merupakan kelas abstrak. Selanjutnya kelas *concrete component* di sini adalah varian rasa dari es *bubble*, masing-masing varian rasa dibuat sebagai *sub-class* dari kelas es *bubble*. Kelas dekorator di sini tidak lain adalah *topping* yang juga merupakan sebuah kelas abstrak, Untuk itu setiap varian dari *topping* menjadi kelas konkret yang diwariskan dari kelas dekorator. Sekarang desain program ini sudah menjadi jauh lebih simpel dan juga fleksibel.

Usulan desain ini juga memenuhi prinsip SOLID (*Single responsibility, Open-closed, Liskovsubstitution, Interfacesegregation, Dependencyinversion)* sepenuhnya. Hal ini akan berujung pada kemudahan untuk *maintenance* dan memperluas fitur Setiap ada penambahan varian dari es *bubble* atau *topping* baru dapat dibuat dengan cara membuat *sub-class* baru tanpa memodifikasi kode program yang telah dibuat sebelumnya. Setiap penciptaan objek varian es *bubble* dapat menggunakan teknik *wrapping* secara opsional dengan memanggil subclass untuk varian *topping*. Berikut desain kelas diagram dapat dilihat pada gambar 3.

Setelah perancangan usulan modifikasi algoritme program dikerjakan, tahap berikutnya adalah percobaan untuk membandingkan efisien dengan program sebelumnya. Pengujian dilakukan dengan menilai 3 variabel utama yakni jumlah baris kode, pemakaian sumber daya dan kecepatan eksekusi program. Dalam hal pemakaian sumber penilaian diwakilkan pada 3 kategori yakni pemakaian CPU, memori RAM dan memori *disk*. Ketiga variabel ini menjadi tolak ukur efisien algoritme kode program.

Pengukuran yang pertama ditujukan untuk mengukur jumlah baris kode program (*Linesof Code*). Teknik pengukuran ini dilakukan dengan cara langsung mengamati kode sumber. Setiap pernyataan, definisi variabel, *method*, *function* dihitung sebagai 1 baris sedangkan komentar,




Gambar 4.Perbandingan penggunaan sumber daya CPU dan memori aplikasi sebelum dimodifikasi (atas) dan setelah dimodifikasi (bawah)

kurung kurawal pembuka atau penutup tidak dianggap sebagai satu baris kode. Supaya pengukuran jumlah baris kode lebih valid dan *realiable* maka sebelumnya dipastikan bahwa kedua aplikasi menjalankan fungsi dan memiliki tampilan yang sama persis. Selain itu juga kedua aplikasi yang sebelum dimodifikasi atau setelah dimodifikasi menggunakan bahasa pemrograman yang sama, yakni dalam kasus ini menggunakan bahasa pemrograman Jawa. Tampilan kode program setelah menerapkan pola desain dekorator dapat dilihat pada gambar 5.

Hasilnya aplikasi yang telah dimodifikasi memiliki 829 baris kode dan aplikasi sebelum dimodifikasi memiliki 1130 baris kode. Dengan kata lain jumlah baris menurun dengan tingkat efisiensi sebesar 301 baris atau sekitar 27%. Jumlah baris kode yang lebih sedikit tidak berefek pada waktu pengerjaan dan biaya yang lebih efisien namun juga biaya keamanan dan perawatan yang lebih ringan karena lebih sedih kode program secara logika juga berarti lingkup perawatan dan celah keamanan yang lebih sempit.

Pengujian selanjutnya ditujukan untuk mengukur kecepatan eksekusi program. Kedua aplikasi sebelum dan sesudah modifikasi telah diinstal pada komputer yang sama sebelum pengukuran. Perhitungan waktu eksekusi program di sini dihitung menggunakan alat bantu *Windows performanceanalyzer*. Waktu eksekusi program dimulai dari waktu program di-*compile*, *loadtime* sampai program dapat menghasilkan *output*. Dilihat dari kecepatan eksekusi program maka program awal sebelum dimodifikasi mencacat

```
public abstract class EsBubble {
        String description = "Unknown";

        public String getDescription() {
                return description;
        }

        public abstract double cost();
}
public class Melon extends EsBubble {
        public Melon() {
                description = "Es Bubble Rasa Melon";
        }

        public double cost() {
                return 10000;
        }
}
public abstract class Toping extends EsBubble {
        public abstract String getDescription();
}

public class Keju extends Toping {
        EsBubble esbubble;

        public Keju(EsBubble esbubble) {
                this.esbubble = esbubble;
        }

        public String getDescription() {
                return beverage.getDescription() + ", Toping Keju";
        }

        public double cost() {
                return 1000 + esbuble.cost();
        }
}
public class kafeXyz {

        public static void main(String args[]) {
                EsBubble esbubble = new Melon();
                esbubble= new Keju(esbubble);
        System.out.println(esbubble.getDescription()
                        + " Rp" + esbubble.cost());
        }
}
```

Gambar 5. Tampilan sintak kode program dengan penerapan pola dekorator

waktu eksekusi 3,11 detik sedangkan program yang sudah dimodifikasi dengan pola desain dekorator unggul dengan selisih waktu 0,27 detik lebih cepat atau 2,84 detik. Dengan demikian terjadi percepatan waktu eksekusi sekitar 8,6%.

Tabel 1.
Hasil Perbandingan Pengujian Aplikasi

| **Parameter** | **Aplikasi Awal** | **Aplikasi setelah dimodifikasi** |
|---|---|---|
| LinesofCodes | 1130 | 829 |
| LoadTime | 3,11 | 2,84 |
| CPU usage | 1-100% | 10-35% |

Pengukuran terakhir difokuskan untuk mengukur pemakaian sumber daya selama aplikasi berjalan. Pengujian dilakukan masih pada perangkat komputer dan alat bantu yang sama. Hasil menunjukkan ada peningkatan signifikan pada pemakaian sumber daya CPU dan memori RAM (seperti ditunjukkan pada gambar 4). Pada program sebelum dimodifikasi terjadi *overload* CPU pada puncak 100 pada selang waktu 105 sampai 305 ms sejak kompilasi kode program. Pada aplikasi yang telah dimodifikasi beban pemakaian CPU jauh lebih stabil dengan kisaran pemakaian beban CPU antara 10 sampai 35%. Efisiensi pemakaian sumber juga terjadi pada pemakaian memori Ram di mana pada program awal pemakaian sumber daya memori sampai ke batas 1024 kb maka setelah program dimodifikasi anjlok ke posisi 512 kb. Dengan kata lain terjadi penghematan pemakaian memori sampai dengan 50%.

Aplikasi yang telah dimodifikasi dengan pola desain dekorator menunjukkan hasil efisiensi yang positif. Terdapat beberapa faktor yang diprediksi kuat menjadi alasan terjadinya hal ini. Pertama algoritme dengan pola desain dekorator memiliki kohesi yang tinggi dengan *coupling* rendah. Kohesi yang tinggi ditunjukkan dengan indikator algoritme modul dan kelas yang lebih ringkas dan modular pada akhirnya semua ini akan menghemat penggunaan CPU dan memori. Kedua menerapkan pola dekorator juga berarti membungkus kode utama dengan fungsi tambahan masing-masing, pada akhirnya hal ini akan berakibat kode sumber lebih ringkas namun fungsi dan tampilan program tetap dipertahankan.

## Penutup

Penelitian ini berhasil membuktikan secara empiris efisiensi algoritme kode program dengan pola desain dekorator. Terdapat kenaikan tingkat efisien kode program yang signifikan pada jumlah baris kode sumber secara keseluruhan juga pada efisiensi penggunaan CPU dan memori RAM. Di samping itu ada sedikit kenaikan pada kecepatan eksekusi program dan pengguna memori *disk*. Dengan demikian tujuan dari penelitian ini telah tercapai dengan dijabarkannya bukti-bukti empiris dari data yang telah dikumpulkan dan telah diuji

Dengan dijabarkannya manfaat efisien algoritme kode program pada penelitian ini diharapkan untuk para developer untuk menerapkan pola desain ini ketika ingin menambahkan atribut atau perilaku pada sebuah objek. Bagaimana pun pada penelitian ini hanya berfokus pada satu tipe pola desain dan masih terdapat banyak pola desain lainnya yang perlu diteliti dan dieksplorasi manfaat baik secara efisien algoritme, performa kinerja atau faktor-faktor lainnya seperti kemudahan perawatan program (*maintainability*).

## Daftar Pustaka

[1] R. Budiarto, Pemrograman Berorientasi Objek Menggunakan Java, Yogyakarta: Andi Publisher, 2017.

[2] A. Schwinn dan S. Joachim, "Design patterns for data integration," *Journal of Enterprise Information Management,* vol. 18, no. 4, pp. 471 - 482, 2015.

[3] W. Ahmad, "Object-Oriented Design Patterns for Detailed Design," *Journal of Object Technology,* vol. V, no. 2, pp. 155-169, 2016.

[4] M. Milton, Head First Data Analysis, Sebastopol: O'Reilly Media, 2009.

[5] D. May, "The Return of Innovation," *Cambrige Journal,* pp. 11-17, 2015.

[6] E. Gamma, R. Helm, R. Johnson dan J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, New Jersey: Addison-Wesley Professional, 1994.

[7] P. Graham, "Design Pattern, Revenge of the Nerds," *IEEE Computer,* pp. 62-71, 2012.

[8] H. Martin dan L. Priscila, "The World technological capacity to store, communicate and compute information," *Science,* vol. 332, no. 6025, pp. 60-65, April 2011.

[9] T. Bevis, Java Design Pattern Essentials, Ability First: California, 2010.

[10] E. Gal dan E. Zul, "The efficiency of algorithms–misconceptions.," *In proceedings of the32nd ASEE/IEEE Frontiers in Education Conference,* vol. IV, no. 38, pp. 319-329, 2012.

[11] T. Cormen, C. C. Leiserson dan C. Stein, Introduction to Algorithms, Cambrige: MIT Press, 2009.

[12] P. Cichosz, "Measuring the efficiency of algorithms," *MIT Journal of Computation,* vol. II, no. 12, pp. 54-68, 2012.