PROYEK AKHIR

MATA KULIAH PEMODELAN BERORIENTASI OBJEK

SEMESTER GANJIL 2013-2014

PERANCANGAN DAN IMPLEMENTASI PROGRAM AUTOSHAPES DENGAN PENDEKATAN BERORIENTASI OBJEK

Disusun oleh:

Kelompok A Kelas F

Adam Adi P.	105060807111166
Choirul Huda	105060807111032
Eko Subha	105060801111067
Faris Fitrianto	105060801111065
Lipsia Cakra T.	105060801111043

Dosen Pengajar : Wayan Firdaus Mahmudy, Ph.D.

PROGRAM STUDI INFORMATIKA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa atas segala limpahan rahmat dan karunia-Nya sehingga kami dapat menyelesaikan proposal dengan judul "**Perancangan Dan Implementasi Program Autoshapes Dengan Pendekatan Berorientasi Objek**" untuk memenuhi tugas akhir Mata Kuliah Pemodelan Berorientasi Objek

Dengan selesainya penyusunan proposal ini, maka kami ingin menyampaikan ucapan terima kasih kepada :

1. Tuhan Yang Maha Esa, karena atas rahmat-Nya kami mampu menyelesaikan tugas Pemodelan Berorientasi Objek ini dengan baik.
2. Bapak Wayan Firdaus Mahmudy, Ph.D. selaku dosen pengampu Pemodelan Berorientasi Objek yang telah membagi ilmunya kepada kami.
3. Rekan-rekan semua di kelas Pemodelan Berorientasi Objek yang telah memberi dukungan.

Dengan segala kerendahan hati, kami mengharapkan kritik dan saran yang bersifat membangun dari para pembaca. Kami berharap makalah ini bermanfaat bagi semua pihak.

Malang, 14 Desember 2013

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Autoshapes merupakan salah satu aplikasi pada aplikasi Microsoft Word yang dapat digunakan untuk membentuk suatu objek atau bentuk, seperti bujur sangkar, segitiga, lingkaran, dan lain-lain. Pengunaannya pun cukup mudah, tinggal mengklik Autoshapes pada menu pilihan dan memilih objek yang ingin digambar pada lembar kerja Microsoft Word. Objek atau bentuk yang digambar juga dapat diubah-ubah dan dimodifikasi dengan mudah, seperti merubah ukuran gambar *(change size)*, memberikan warna pada gambar, memindah posisi atau letak gambar *(move)*, dan memutar bentuk gambar *(rotate)*. Cara melakukan proses pengubahan dan modifikasi objek yang digambar pada lembar kerja Microsoft Word cukup mudah. Pada setiap objek yang digambar terdapat titik-titik pada setiap ujung objek. Untuk mengubah ukuran gambar tinggal mengklik dan menggeser titik pada objek yang ingin diubah ukurannya. Untuk memberi warna pada objek yang digambar tinggal mengklik 2 kali pada garis atau titik objek yang akan diberi warna. Ketika telah diklik 2 kali akan muncul menu-menu lain untuk memberi warna pada obejk. Untuk memindah objek tinggal mengklik 2 kali pada garis atau titik objek dan objek dapat dipindah-pindah letaknya. Untuk memodifikasi objek seperti memutar gambar caranya masih sama dengan mengklik 2 kali pada objek yang dipilih pada lembar kerja Microsoft Word. Ketika objek telah diklik 2 kali akan muncul beberpa titik pada ujung-ujung objek. Untuk memutar objek tinggal memutar titik pada objek yang bewarna berbeda, biasanya hijau dan terletak paling atas diantara titik-titik objek lainnya pada objek yang dipilih. Itulah penjelasan kemudahan pembuatan objek gambar pada Microsoft Word.

Dengan memanfaatkan menu Autoshapes pada Microsoft Word, dapat dibuat untuk penerapan pembelajaran pemodelan berorientasi objek sebagai media pembelajaran pembuatan aplikasi menggunakan pemodelan berorientasi objek. Proses pemilihan, pembuatan, pengubahan, dan modifikasi objek dapat digunakan

sebagai sarana pembelajaran pemodelan berorientasi objek. Bentuk sebagai class utama dapat diturunkan ke class-class yang ada dibawahnya menjadi bentuk-bentuk bujursangkar, segitiga, lingkaran, dan lain-lain. Proses penurunan ini dinamakan *inheritance* yang merupakan salah satu pembelajaran dalam belajar pemodelan berorientasi objek java. Pemodelan Autoshapes ini juga dapat direpresentasikan dalam bentuk diagram-diagram yang digunakan dalam pemodelan berorientasi objek, seperti use case, class diagram, state diagram, activity diagram, dan lain-lain. Pemahaman konsep Autoshapes pada Microsoft Word ini untuk diterapakan sebagai representasi pembelajaran pemodelan berorientasi objek menjadi salah satu alasan kami memilihnya.

1.2 Rumusan Masalah

1) Cara merepresentasikan konsep pemodelan berorientasi objek seperti *inheritance* dari aplikasi Autoshapes Microsoft Word.
2) Cara merepresentasikan pemodelan berorientasi objek dari aplikasi Autoshapes Microsoft Word dalam bentuk digram, meliputi use case, class diagram, state diagram, activity diagram, dan lain-lain.
3) Cara merepresentasikan Autoshapes Microsoft Word dalam bentuk aplikasi sebagai bentuk penerapan pemodelan berorientasi objek.

1.3 Batasan Masalah

1) Aplikasi Autoshapes ini menggunakan bahasa pemrograman berorientasi objek yaitu java.
2) Aplikasi yang dibuat adalah aplikasi berbasis desktop.
3) Pada aplikasi tidak terdapat menu inputan.
4) Aplikasi ini mampu melakukan pembuatan objek shape beserta manipulasinya.

1.3 Tujuan

1) Memberikan pemahaman konsep-konsep dalam pemodelan berorientasi objek seperti *inheritance*.

2) Memberikan pemahamsan mengenai diagram-diagram yang digunakan dalam pemodelan berorientasi objek.
3) Memberikan pemahaman penerapan pemodelan berorientasi objek dalam bentuk aplikasi.

1.4 Manfaat

1) Memberikan pemahaman lebih mengenai penggunaan konsep-konsep yang terdapat dalam pemodelan berorientasi objek, seperti *inheritance*, *polymorphism*, *class*, dan lain-lain.
2) Memberikan pemahaman lebih mengenai perbedaan setiap diagram yang diterapkan dan digunakan dalam pemodelan berorientasi objek.
3) Memberikan bentuk nyata dari penerapan pemodelan berorientasi objek dalam bentuk aplikasi Autoshapes pada menggunakan java.

1.5 Sistematika Penulisan Laporan

Adapun sistematika penulisan yang digunakan dalam menyusun Laporan Ujian Akhir Semester (UAS) Mata Kuliah Pemodelan Berorientasi Objek (PBO) ini adalah :

BAB I : PENDAHULUAN

Bab ini menguraikan latar belakang, rumusan masalah, tujuan, manfaat, dan sistematika penulisan laporan ujian akhir sekolah mata kuliah pemodelan berorientasi objek.

BAB II : DASAR TEORI

Bab ini menguraikan tentang tinjauan pustaka dan referensi yang mendasari proses perancangan dari pemodelan berorientasi objek aplikasi Autoshapes pada Microsoft Word.

BAB III : PERANCANGAN

Bab ini menguraikan proses perancangan pemodelan berorientasi objek aplikasi Autoshapes pada Microsoft Word dalam bentuk diagram-diagram beserta penjelasan pada tiap-tiap diagram.

BAB IV : IMPLEMENTASI

Bab ini menguraikan proses implementasi dari perancangan pemodelan berorientasi objek aplikasi Autoshapes pada Microsoft Word.

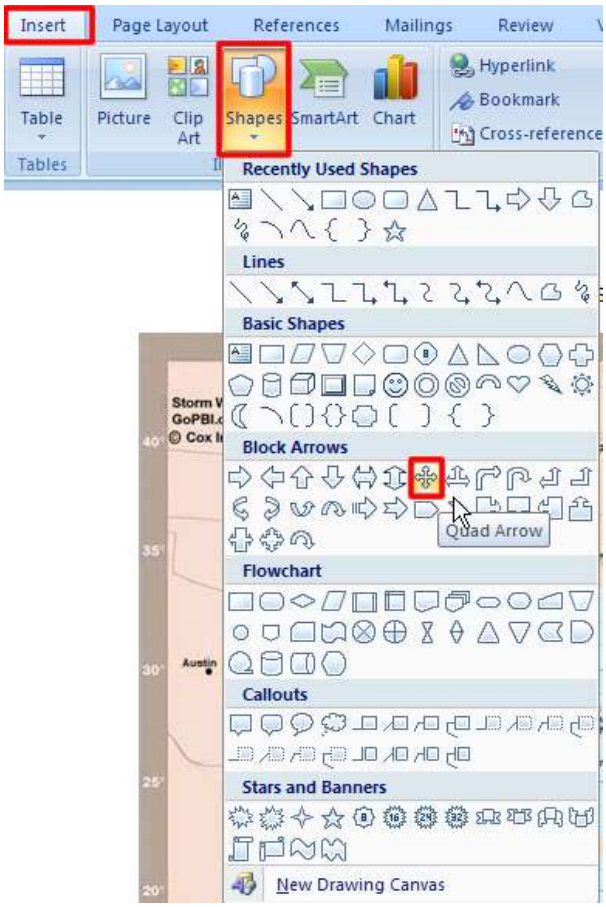
BAB V : PENUTUP

Bab ini menguraikan kesimpulan dan saran dari perancangan pemodelan berorientasi objek aplikasi Autoshapes pada Microsoft Word.

BAB II
DASAR TEORI

2.1 Autoshapes

Autoshape merupakan fitur yang ada pada aplikasi microsoft office word dan beberapa aplikasi yang ada dalam microsoft office suite, digunakan sebagai alternatif untuk menambahkan grafis pada lembar kerja, user bisa membuat gambar menggunakan autoshape. Suatu autoshape merupakan bentuk yang sudah ditetapkan atau bentuk-bentuk bebas seperti oval, kubus, simbol flow chart, atau bentuk bebas; yang bisa digunakan dengan cepat pada dokument kemudian mengaturnya; seperti ukuran, warna garis, dan warna isi. Contoh tampilan toolbar autoshape.



Gambar 2.1. Fitur autoshape pada aplikasi microsoft office word

2.2 Model dan Pemodelan Obyek

Dalam bukunya The Unified Modeling Language Reference Manual, James Rumbaugh dkk (1999) mengemukakan bahwa suatu model merupakan penyajian objek dalam media tertentu ke dalam media yang sama atau media yang berbeda. Model menjadi bagian penting dari objek yang akan dimodelkan sehingga mudah dimengerti. Sebagai contoh, model dari pesawat terbang mungkin akan digambarkan menggunakan aplikasi komputer seperti Blender atau bahkan ia dimodelkan dengan membuat miniaturnya. Salah satu manfaat pembuatan model adalah penghematan biaya pengujian objek yang akan dibangun. Jika seorang profesor langsung membuat pesawat terbang sesuai ide-idenya tanpa membuat modelnya terlebih dahulu, maka dipastikan perusahaan yang mengusungnya akan mengeluarkan lebih banyak biaya dari pada mereka yang memodelkan pesawat untuk ide proyek yang sama jika dalam tahap pengujian mengalami kegagalan. Contoh lainnya adalah, hampir semua mahasiswa jurusan ilmu komputer/informatika akan memperoleh *chapter* tentang algoritma untuk menyelesaikan suatu masalah, sebelum melakukan penulisan dalam bahasa pemrogramman, mahasiswa akan memodelkan algoritma tersebut ke dalam diagram alir.

Pemodelan berorientasi obyek (PBO) merupakan suatu cara memodelkan sistem dengan merepresentasikannya sebagai objek yang memiliki komponen objek berupa data/atribut dan perilaku. Pemodelan berorientasi objek menggambarkan sistem ke dalam bentuk yang lebih sederhana sehingga mudah dipahami dan dikembangkan. Dalam PBO terdapat prinsip abstraksi, encapsulasi, modularitas, hirarki, typing, konkurensi dan persistensi.

2.3 Kelas dan Obyek

Konsep kelas dan objek dapat dianalogikan dengan pembuatan kue. Loyang sebagai kelas dan kue sebagai objek sedangkan pembuatan kue merupakan instansiasi objek. Dengan kata lain, kelas merupakan komponen pembentuk objek sedangkan objek merupakan produk dari kelas setelah adanya instansiasi. (Nurudin Santoso, 2011) Konsep kelas dan objek merupakan salah

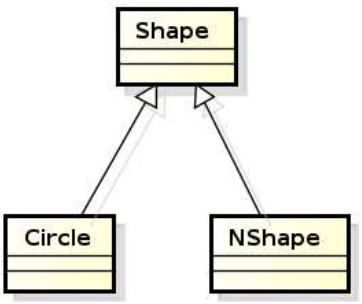
satu bagian terpenting dalam pengembangan beroprientasi objek sehingga developer harus berpikir dalam konsep objek.

2.4 Relasi Antar Kelas

Dibawah ini merukana beberapa relasi antar kelas

2.4.1 Generalisasi

Generalisasi merupakan relasi “is a” (merupakan/adalah). Generalisasi sering disamakan dengan inheritance yang sebenarnya keduanya memiliki perbedaan dalam implementasinya. Generalisasi menggambarkan relasi suatu kelas sedangkan inheritance lebih pada implementasinya dari generalisasi dalam pemgrogramman berorientasi objek. Contoh dari relasi generalisasi adalah

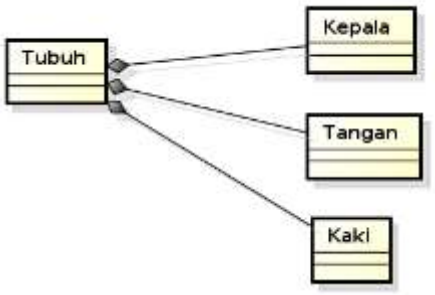


Gambar 2.2. Relasi generalisasi

Circle merupakan bangun datar (shape) sedangkan shape merupakan bagian umum dari circle dan nshape.

2.4.2 Komposisi

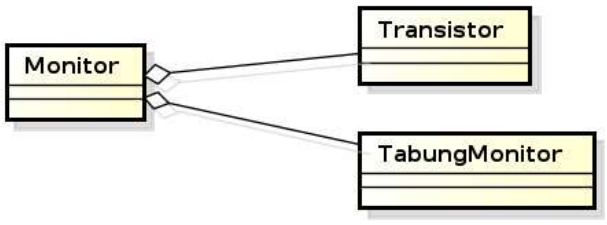
Komposisi merupakan bentuk relasi “has a” yang “kuat”. Kelas-kelas yang terhubung oleh komposisi memiliki ketergantungan yang kuat. Suatu kelas menjadi pemilik kelas lainnya. Sebagai contoh, hubungan antara tubuh manusia (kelas keseluruhan) dengan tangan, kaki, dan kepala (kelas bagian). Kelas bagian tergantung pada kelas keseluruhan. Ilustrasi relasi komposisi bisa dilihat pada gambar 3.



Gambar 2.3. Relasi komposisi

2.4.3 Agregasi

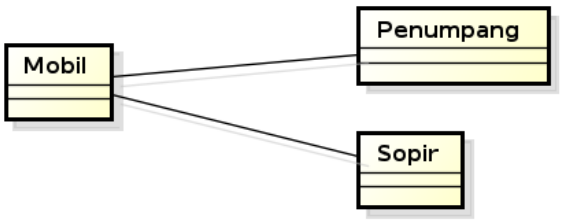
Agregasi merupakan relasi “has a” begitu juga dengan relasi asosiasi dan komposisi, ketiganya merepresentasikan kolaborasi antar kelas. Namun ketiganya berbeda, agregasi merepresentasikan hubungan antar kelas sehingga keseluruhan kelas memiliki peran yang lebih besar atau lebih penting. Meski demikian, kelas-kelas bagian tetap bisa berdiri sendiri tanpa keseluruhan kelas. Relasi ini memiliki hirarki ”part of” dan “consist” yaitu kelas penampung “berisi” kelas lain yang menyusunnya atau kelas-kelas penyusun merupakan bagian dari kelas penampungnya (keseluruhan kelas). Sebagai ilustrasi, hubungan antara monitor komputer dengan tabung monitor dan transistor adalah hubungan agregasi. Layar monitor sebagai keseluruhan kelas berisi tabung dan transistor sebagai kelas-kelas bagian. Keberadaan monitor tergantung pada komponen penyusunnya. Monitor tidak akan berfungsi sebagai monitor jika transistor sipisahkan dari relasinya. Namun transistor dapat berdiri sendiri dan fungsinya sebagai penguat atau sirkuit switching tidak hilang, begitu juga dengan tabung monitor. Agregasi merupakan bentuk yang lebih sederhana dari komposisi. Jika digambarkan dalam bentuk diagram, relasi agregasinya seperti pada gambar 4.



Gambar 2.4. Relasi agregasi

2.4.4 Asosiasi

Sama dengan relasi agregasi, yaitu merupakan relasi “has a” namun berbeda dalam implementasinya. Pada asosiasi, biasanya kita melihat kelas-kelas yang membentuk kelas keseluruhan. Asosiasi bisa dikatakan relasi yang lemah karena kelas-kelas bagian bersifat *independent*. Dalam relasi asosiasi, tidak ada kelas yang menjadi pemilik kelas lainnya. Setiap kelas merupakan pemilik dirinya sendiri. Suatu kelas hanya memanfaatkan fungsi/peran yang dimiliki oleh kelas lainnya. Jika pada agregasi kita melihatnya kelas keseluruhan, pada relasi asosiasi kita bisa melihat secara keseluruhan dan bagian-bagiannya. Sebagai contoh hubungan antara penumpang dan sopir (sebagai kelas bagian) dengan mobil (ketika berelasi dengan penumpang dan sopir, menjadi kelas keseluruhan). Penumpang sewaktu-waktu bisa lepas dari relasinya dengan mobil yaitu ketika dia tutun dari mobil karena sudah tidak membutuhkan peran dari mobil, begitu juga dengan sopir; namun ketiganya tetap dapat berdiri sendiri. Ilustrasi relasi asosiasi dapat dilihat pada gambar diagram dibawah ini



Gambar 2.5. Relasi asosiasi

2.5 Enkapsulasi

Konsep enkapsulasi juga dikenal dengan sebutan *information hiding*. Konsep enkapsulasi merupakan salah saru prinsip dasar dalam desain software. Konsep enkapsulasi sering dijadikan salah satu acuan untuk membedakan modul-module software yang dirancang dengan baik atau buruk. Parameternya adalah sejauh mana modul menyembunyikan data enviousinternal dan rincian aktifitas lainnya dari modul lain. Sebuah module yang dirancang dengan baik menyembunyikan semua data internal dan rincian aktifitas yang ada didalamnya, dengan rapi memisahkan antara API dan aktifitas didalam modul. Modul

berkomunikasi dengan modul lainnya hanya melaui API dan tidak mengetahui bagian dalam modul satu sama lain [Jos-01].

2.6 Polimorphism

Polimorphism merupakan istilah yang sering digunakan dalam pemodelan suatu sistem yang berbasis objek. Polimorphism mengijinkan developer untuk menggunakan nama yang sama untuk beberapa hal berbeda dan compiler secara otomatis mengetahui bagian yang harus digunakan. Hal ini menyediakan kemampuan dari method untuk pekerjaan yang berbeda berdasarkan objek tertentu. Berdasarkan istilah, polimorphism terdiri dari dua kata yaitu poli dan morphism; poli berarti banyak, morphism berarti bentuk. Sehingga, polimorphism merupakan sesuatu yang bisa berubah ke dalam bentuk yang banyak (memiliki banyak bentuk). Salah satu contoh polimorphism adalah overriding dan overloading. [Deb-13]

2.7 Interface

Inheritance merupakan bagian dari model pngembangan sistem/software berorientasi objek yang mengijikan seolah-olah developer menggunakan multiple inhritance. Interface merupakan mekanisme yang memungkinkan suatu kelas dapat berbagi perilaku/method signature (constant) agar bisa digunakan oleh kelas lain.

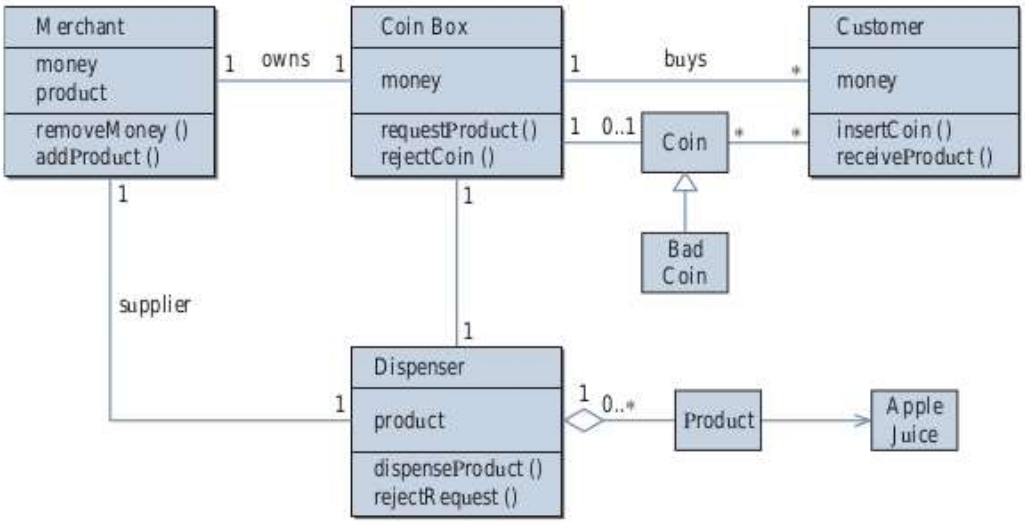
2.8 Unified Modeling Language (UML)

Unified modeling language (UML) menyediakan notasi dalam bentuk gambar untuk memodelkan sistem yang dikembangkan menggunakan object-oriented software engineering (OOSE). OOSE memfokuskan pengidentifikasian pada elemen-elemem masalah yang menghasilkan atau menggunakan informasi dan mendeskripsikan hubungannya yang ada diantara element-element tersebut. Dalam OOSE, object didefinisan untuk merepresentasikan element-element ini selama proses desain dan analisis. Diagram-diagram UML mengijinkan pengembang software untuk mengindikasi hubungan diantara objek yang

digunakan untuk mendefinisikan sistem. UML sangat berguna jika developer berencana mengimplementasikan sistem dalam bahasa berorientasi objek seperti java. Dibawah ini adalam diagram-diagram dalam UML.

2.8.1 Diagram Kelas

Diagram kelas merupakan salah satu diagram UML yang paling penting, digunakan oleh software engineer. Diagram kelas digunakan untuk membuat model logic dari sistem berorientasi computer. Suatu diagram kelas menunjukkan struktur kelas, isi, hubungan statis diantara kelas-kelas yang digunakan untuk memodelkan sistem. Hubungan ini juga dikenal dengan istilah asosiasi dan digambarkan sebagai garis menghubungakan node yang berhubungan. Setiap node dalam diagram kelas diberi label dengan nama kelasnya. Node kelas bisa juga berisi daftar data atribut dan prototipe method. Keberadaan atribut atau method bisa ditandai dengan awalan + (public) atau - (private) pada namanya. Dibawa ini contoh diagram kelas.



Gambar 2.6. Diagram kelas

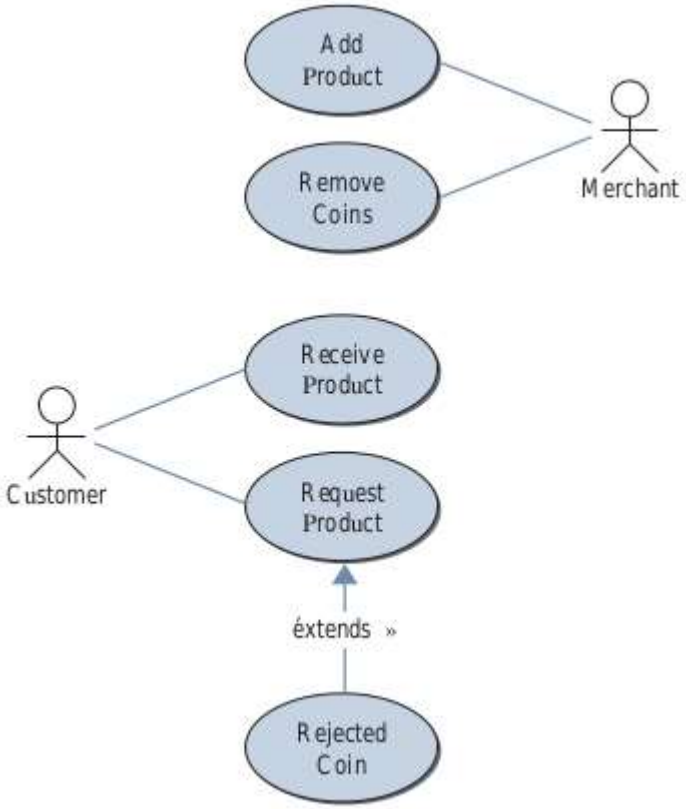
Garis asosiasi menandakan bahwa ada hubungan antara kedua kelas. Beberapa asosiasi bisa diberi label berupa tulisan yang menandakan jenis hubungan antar kelas. Masing-masing bagian akhir dari asosiasi di beri label nomor, tanda bintang (*), atau range untuk mendeskripsikan keberagaman link

(misal 1..* menandakan kebergaman dengan range dari 1 ke banyak). Hubungan "bagian-keseluruhan" (dikenal dengan istilah agrregasi dalam UML) ditandai dengan notasi wajik terbuka (kosong) pada bagian akhir link. Hubungan inheritance (generalisasi dalam UML) ditandai dengan segitiga terbuka (kosong) mengarah pada kelas super yang sesuai.

2.8.2 Diagram Use Case

Diagram use case digunakan untuk model requirement fungsional sistem. Diagram ini menunjukkan interaksi antara user dengan sistem. Diagram use case digambarkan untuk tidak tergantung pada desain user antar muka yang akan digunakan di sistem nantinya. Use case menyimpulkan beberapa skenario untuk tugas atau tujuan user. Suatu skenario merupakan instansiasi dari use case untuk aktor tertentu, pada waktu yang spesifik, dengan data yang spesifik. Setiap skenario akan dideskripsikan dengan tulisan dan digambarkan dalam bentuk grafis dengan diagram sequence. Use case diagram mendampingi software engineer untuk mengembangkan *test cases*.

User merupakan aktor yang direpresentasikan dalam diagram use case dengan label *orang-orangan*. Node use case berlabel tugas atau tujuan user. User dihubungkan pada node yang sesuai dengan galis. Link bisa diberi label berupa tulisan <<extends>> untuk menunjukkan secara eksplisit interaksi yang bersifat pilihan atau penggunaan penanganan pengecualian. Tulisan <<uses>> bisa digunakan untuk memberikan label pada link use case yang sedang digunakan sebagai subsistem pada use case yang bersangkutan. Masing-masing lintasan pada diagram use case merepresentasikan usecase yang terpisah. Dibawah ini contoh gambar diagram use case.



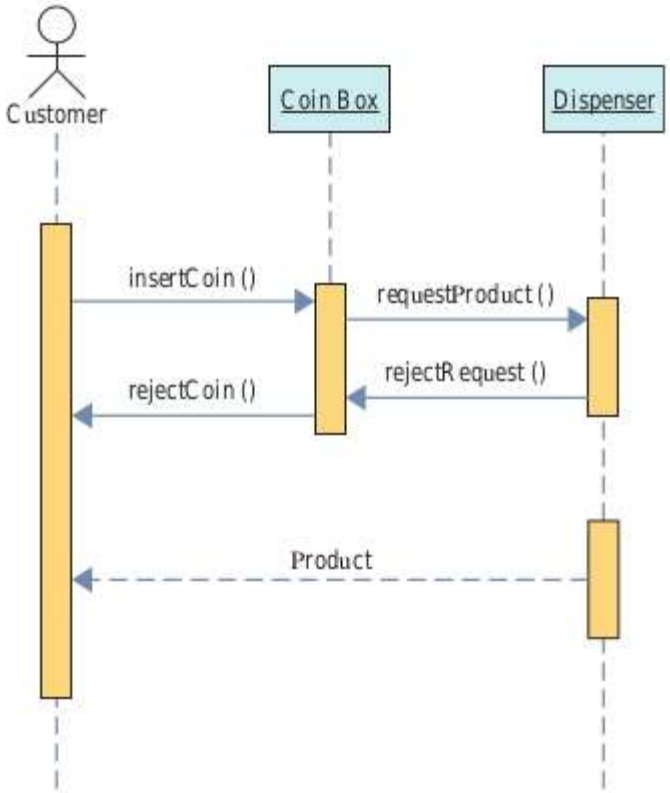
Gambar 2.7. Diagram use case

2.8.3 Diagram Sequence

Diagram sequence memodelkan tingkah laku sistem untuk use case dengan menunjukkan interaksi-interaksi kelas yang diperperlukan. Siaram sequence menggambarkan aliran kerja dari use case secara grafis. Diagram sequence menunjukkan urutan temporal dari pertukaran pesan diantara kumpulan objek sbagaimana mereka berkomunikasi untuk mencapai tugas spesifik. Khususnya, mereka menunjukkan bagaimana aktor berinteraksi dengan sistem agar pekerjaan selesai (misal, pesan apa yang dikirimkan dan kapan dikirimkan). Skenario/peristiwa yang dimodelkan dalam diagram sequence merupakan skenario eksternal ditandai dengan *anchor*.

Aktor dan object disusun secara horizontal dari bagian atas diagram. Dimensi vertikal merepresentasikan waktu. Garis vertikal disebut juga *lifeline* dipasangkan pada masing-masing aktor atau objek. *Lifetime* menjadi kotak aktivasi untuk menunjukkan periode aktivasi dari objek atau aktor. Pesan

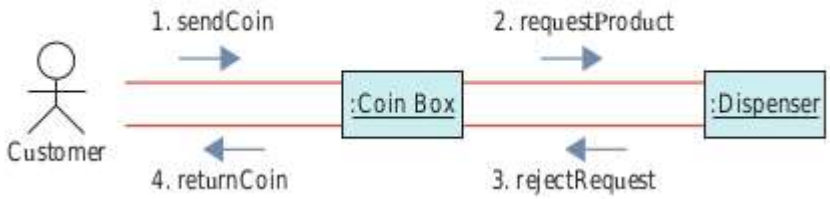
direpresentasikan menggunakan panah berlabel pesan. Label pesan bisa berisi daftar argumen dan tipe kembalian. Panah dengan garis putus-putus bisa digunakan untuk menandakan alur objek. Jika objek *life* berakhir selama eksekusi use case, tanda X diletakkan pada bagian bawah *lifeline*-nya. Dibawah ini merupakan contoh diagram sequence.



Gambar 2.8. Diagram sequence

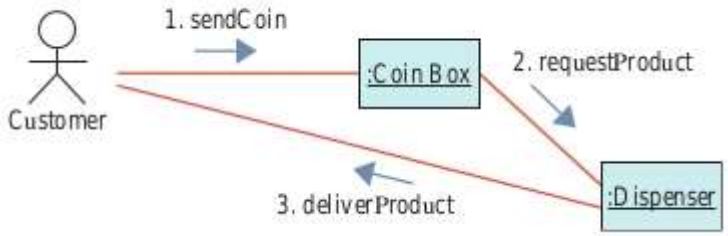
2.8.4 Diagram Kolaborasi

Diagram kolaborasi menunjukkan pesan yang melewati struktur sistem. Fokus dari diagram ini adalah peran yang ada pada objek-objek sebagaimana mereka berinteraksi untuk menyadari fungsi sistem. Mereka bisa digunakan untuk merepresentasikan porsi pola desain dan berguna untuk proses validasi diagram kelas. Dibawah ini contoh diagram kolaborasi.



Gambar 2.9. Diagram kolaborasi pengembalian koin

Suatu diagram kolaborasi merupakan grafik terarah dengan objek dan aktor sebagai puncaknya. Link direksional digunakan untuk menandakan komunikasi antar objek. Link ini diberi label menggunkan pesan yang sesuai. Masing-masing pesan diawali dengan nomor berurutan untuk menandakan waktu urutan yang dibutuhkan untuk melengkapi fungsi sistem. Seperti pada gambar dibarah ini, tidak setiap diagram kolaborasi bisa digambarkan seara horocontal atau vertikal.



Gambar 2.10. Diagram kolaborasi product delivery

BAB III
PERANCANGAN

Bab ini membahas analisis kebutuhan perancangan dan penerapan pemodelan berorientasi objek aplikasi autoshapes. Perancangan yang dilakukan yaitu meliputi yaitu penuyusan diagram kelas, proses analisa kebutuhan perangkat lunak, serta diagram sequence. Pada Tahap analisa kebutuhan perangkat lunak terdiri dari identifikasi aktor, daftar kebutuhan sistem, dan use case diagram.

3.1 Analisis Kebutuhan Perangkat Lunak

Analisa kebutuhan perangkat lunak ini diawali dengan identifikasi aktor yang terlibat dalam sistem, penjabaran kebutuhan sistem dan memodelkannya ke dalam *use case* diagram. Analisa kebutuhan ini ditujukan untuk menggambarkan kebutuhan-kebutuhan yang harus disediakan oleh sistem agar dapat memenuhi kebutuhan pengguna.

3.1.1 Identifikasi Aktor

Pada tahap ini mempunyai tujuan untuk melakukan identifikasi aktor yang akan berinteraksi dengan sistem. Aktor menggambarkan pengguna dari aplikasi (user). Aktor membantu memberikan suatu gambaran jelas tentang apa yang harus dikerjakan oleh software. Pada aplikasi autoshapes ini hanya terdapat satu aktor yaitu user. Di sini tidak diperlukan admin karena admin tidak melakukan interaksi dengan sistem. Jadi di aplikasi ini hanya user yang berinteraksi secara penuh dengan sistem atau aplikasi. User di sini mempunyai peran dalam melakukan aktifitas seperti membuat objek bangun datar serta melakukan menipulasi terhadap objek. Dan untuk lebih jelasnya mengenai peran aktor dapat dilihat pada table 3.1.

Tabel 3.1 Tabel identifikasi aktor

Aktor	Deskripsi Aktor
User	Merupakan aktor yang berperan dalam : ✓ Membuat objek bangun datar. ✓ Melakukan editing atau manipulasi objek. ✓ Melakukan penambahan objek ✓ Mengahapus objek yang telah dibuat.

3.1.2 Analisis Kebutuhan Fungsional

Daftar kebutuhan ini terdiri dari sebuah kolom yang menguraikan kebutuhan yang harus disediakan oleh sistem, dan pada kolom yang lain akan menunjukkan nama use case berisi nama use case yang menunjukkan fungsionalitas masing-masing kebutuhan tersebut. Daftar kebutuhan sistem secara keseluruhan ditunjukkan pada Tabel 3.2.

Tabel 3.2. Tabel analisis kebutuhan fungsional

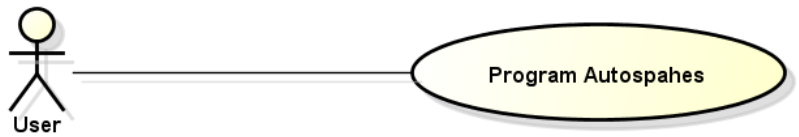
Kebutuhan	**Aktor**	**Nama *Use case***
Sistem menyediakan fasilitas untuk memilih objek shape.	user	Pilih Shape
Sistem menyediakan fasilitas untuk membuat atau menggambar shape	user	Draw Shape
Sistem menyediakan fasilitas untuk merezise shape	user	Resize Shape
Sistem menyediakan fasilitas untuk pewarnaan shape	user	Color Warna
Sistem menyediakan fasilitas untuk pemindahan posisi shape.	user	Move Shape
Sistem menyediakan fasilitas	user	Rotate Shape

untuk merotasi shape.		
Sistem menyediakan fasilitas menghapus shape.	user	Remove Shape

3.1.3 Diagram Use Case

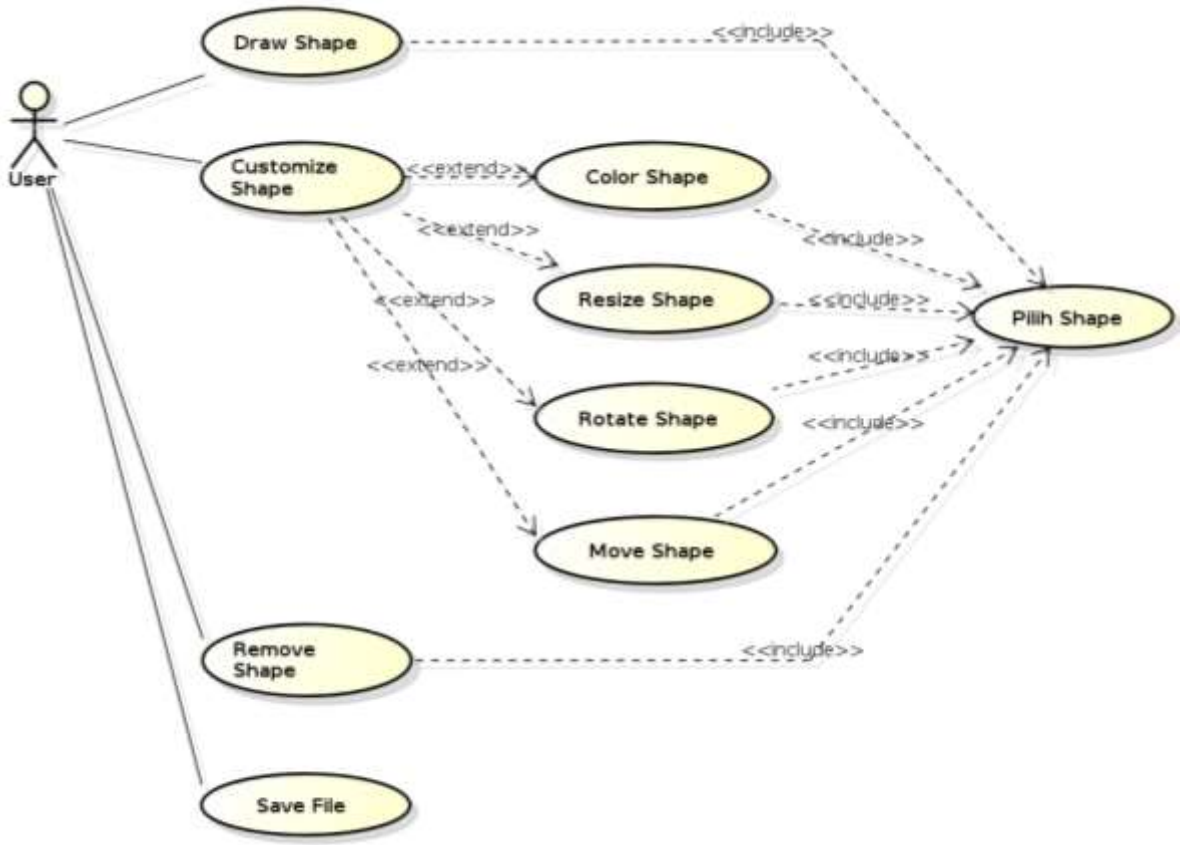
Use case dan aktor menggambarkan ruang lingkup sistem yang sedang dibangun. *Use case* meliputi semua hal yang ada pada sistem, sedangkan aktor meliputi semua hal yang ada di luar sistem. Aktor termasuk seseorang atau apa saja yang berhubungan dengan sistem yang dibangun.

Use case diagram menggambarkan interaksi antara aktor dengan proses atau sistem yang dibuat. Dalam pemodelan dengan menggunakan UML, semua perilaku dimodelkan sebagai *use case* yang mungkin dispesifikasikan mandiri dari realisasinya. *Use case* mendeskripsikan kumpulan urutan (sequence) di mana tiap urutan menjelaskan interaksi sistem dengan sesuatu di luar sistem (sering dinamakan dengan aktor). *Use case* menampilkan spesifikasi fungsional yang diharapkan dari sistem/perangkat lunak yang kelak kita kembangkan. Perancangan global *Use case* untuk Aplikasi Autoshapes akan dijelaskan pada Gambar 4.2.



Gambar 3.1 Global *Use case* Aplikasi Autoshapes

Penjabaran setiap *use case* diagram yaitu menjelaskan secara detail mengenai fungsionalitas keseluruhan dari sistem aplikasi autoshapes. Fungsionalitas dari beberapa aktor yang terdapat pada global *Use case* dapat digambarkan secara detail mengenai aktivitas-aktivitas yang dilakukan oleh aktor. Penjabaran *use case* diagram sistem aplikasi autoshapes akan dijelaskan pada gambar 3.2.



Gambar 3.2. Diagram use case aplikasi autoshape

3.1.3.1 Skenario Use case

Masing-masing *use case* yang terdapat pada diagram *use case*, dijabarkan dalam skenario *use case* secara lebih detail. Pada skenario *use case*, akan diberikan uraian nama *use case*, aktor yang berhubungan dengan *use case* tersebut, tujuan dari *use case*, deskripsi global tentang *use case*, kondisi awal yang harus dipenuhi dan kondisi akhir yang diharapkan setelah berjalannnya fungsional *use case*. Pada skenario *use case* juga akan diberikan ulasan yang berkaitan dengan tanggapan dari sistem terhadap aksi yang diberikan oleh aktor. Skenario *use case* juga terdapat kejadian alternatif yang merupakan jalannya sistem jika terdapat kondisi tertentu.

1. Use case Pilih Shape

   Pada *use case* Pilih Shape, akan dijelaskan secara detail tentang mekanisme memilih shape yang dilakukan oleh user. Skenario *use case* Pilih Shape dijelaskan pada Tabel 3.3.

Tabel 3.3 *Use case* Pilih Shape

Use case	***Pilih Shape***
Aktor	User
Tujuan	Memilih bentuk shape (bangun datar) yang akan dibuat.
Deskripsi	*Use case* ini menjelaskan tentang proses pemilihan bentuk shape yang akan dibuat oleh user
Kondisi Awal	User mengarahkan kursor pada menu pilihan shape
Kondisi Akhir	Kursor user telah siap untuk menggambar bentuk shape yang telah dipilih.
Skenario : Pilih Shape	
Aksi dari Aktor	**Tanggapan dari Sistem**
1. User memilih bentuk shape	2. Sistem menampilkan daftar pilihan bentuk shape yang berbeda – beda.
3. User mengklik salah satu bentuk shape.	4. Sistem telah siap untuk menggambar bentuk shape sesuai yang telah dipilih oleh user.

2. Use case Draw Shape

   Pada *use case* Draw Shape, akan dijelaskan secara detail tentang mekanisme membuat atau menggambar sebuah shape yang dilakukan oleh user. Skenario *use case* Draw Shape dijelaskan pada Tabel 3.4.

Tabel 3.4. *Use case* Draw Shape

Use case	***Draw Shape***
Aktor	User
Tujuan	Membentuk atu menggambar Shape pada kotak gambar sesuai bentuk yang telah dipilih
Deskripsi	*Use case* ini menjelaskan tentang proses pembuatan atau menggambar shape menggunakan kursor oleh user.
Kondisi Awal	User telah memilih bentuk shape sebelum membuatnya.
Kondisi Akhir	Shape telah berhasil dibuat atau digambar sesuai bentuk yang telah dipilih user
Skenario : Draw Shape	
Aksi dari Aktor	**Tanggapan dari Sistem**
1. User menggerakkan kursor pada kotak gambar.	2. Sistem menampilkan bentuk shape dengan luas sesuai pergerakan kursor.

3. Use case Customize Shape

   Pada *use case* Customize Shape, akan dijelaskan secara detail tentang mekanisme kustomisasi atau manipulasi shape yang dilakukan oleh user. Skenario *use case* Customize Shape dijelaskan pada Tabel 3.5.

Tabel 3.5. *Use case* Customize Shape

Use case	***Customize Shape***
Aktor	User
Tujuan	Melakukan kustomisasi atau manipulasi

	Shape yang telah dibuat pada kotak gambar
Deskripsi	*Use case* ini menjelaskan tentang proses kustomisasi atau manipulasi Shape yang telah dibuat pada kotak gambar.
Kondisi Awal	User telah membentuk shape pada kotak gambar.
Kondisi Akhir	Shape yang diedit oleh user mengalami perubahan dari kondisi semula.
Skenario : Customize Shape	
Aksi dari Aktor	**Tanggapan dari Sistem**
1. User mengklik objek yang akan dikostumisasi	2. Sistem akan menandai objek shape yang akan dikostumisasi atau yang telah diklik.
3. User melakukan kustomisasi terhadap objek shape.	4. Sistem akan merubah objek shape sesuai yang diperintahkan user.

4. Use case Color Shape

   Pada *use case* Color Shape, akan dijelaskan secara detail tentang mekanisme pemberian maupun pergantian warna pada objek shape yang dilakukan oleh user. Skenario *use case* Color Shape dijelaskan pada Tabel 3.6.

Tabel 3.6 *Use case* Color Shape

Use case	***Color Shape***
Aktor	User
Tujuan	Melakukan pemberian maupun pergantian warna pada objek shape yang telah dibuat

Deskripsi	*Use case* ini menjelaskan tentang proses pemberian maupun pergantian warna pada objek shape yang telah dibuat
Kondisi Awal	User telah membentuk shape pada kotak gambar dan memilih warna yang akan diberikan pada objek shape.
Kondisi Akhir	Shape yang diberi warna akan berubah warnanya sesuai dengan warna yang dipilih oleh user.
Skenario : Color Shape	
Aksi dari Aktor	**Tanggapan dari Sistem**
1. User memilih dengan mengklik warna yang akan diberikan pada objek shape.	2. Sistem akan menyimpan sementara warna yang telah dipilih oleh user.
3. User mengklik objek shape yang akan diberi warna.	4. Sistem akan merubah warna objek shape sesuai warna yang dipilih oleh user.

5. Use case Resize Shape

   Pada *use case* Resize Shape, akan dijelaskan secara detail tentang mekanisme perubahan ukuran (resize) objek shape yang dilakukan oleh user. Skenario *use case* Resize Shape dijelaskan pada Tabel 3.7.

Tabel 3.7. *Use case* Resize Shape

Use case	***Resize Shape***
Aktor	User

Tujuan	Melakukan perubahan ukuran pada objek shape yang dipilih oleh user
Deskripsi	*Use case* ini menjelaskan tentang proses perubahan ukuran pada objek shape yang dipilih oleh user.
Kondisi Awal	User telah membentuk shape pada kotak gambar.
Kondisi Akhir	Shape mengalami perubahan ukuran dari kondisi awal.
Skenario : Resize Shape	
Aksi dari Aktor	**Tanggapan dari Sistem**
1. User mengklik atau memilih objek yang akan dirubah ukurannya.	2. Sistem akan menandai objek yang telah dipilih oleh user.
3. User menyeret(drag) objek yang dipilih untuk dirubah ukurannya.	4. Sistem akan mengganti ukuran objek shape sesuai yang user perintahkan.

6. Use case Rotate Shape

   Pada *use case* Rotate Shape, akan dijelaskan secara detail tentang mekanisme merotasi shape yang dilakukan oleh user. Skenario *use case* Rotate Shape dijelaskan pada Tabel 3.8.

Tabel 3.8. *Use case* Rotate Shape

Use case	***Rotate Shape***
Aktor	User
Tujuan	Melakukan rotasi terhadap shape yang dilakukan oleh user
Deskripsi	*Use case* ini menjelaskan tentang proses merotasi terhadap shape yang dilakukan oleh user
Kondisi Awal	User telah membentuk shape pada kotak gambar.
Kondisi Akhir	Posisi objek shape akan berotasi sekian derajat dari posisi semula.
Skenario : Rotate Shape.	
Aksi dari Aktor	**Tanggapan dari Sistem**
1. User memilih dengan mengklik shape.	2. Sistem akan menandai objek yang telah dipilih oleh user.
3. User memutar shape sekian derajat sesuai keinginan.	4. Sistem merotasi shape sekian derajat dari posisi semula sesuai perintah user.

7. Use case Move Shape

   Pada *use case* Move Shape, akan dijelaskan secara detail tentang mekanisme memindahkan objek shape yang dilakukan oleh user. Skenario *use case* Move Shape dijelaskan pada Tabel 3.9.

Tabel 3.10 *Use case* Move Shape

Use case	***Move Shape***
Aktor	User
Tujuan	Melakukan pemindahan posisi terhadap shape yang dilakukan oleh user.
Deskripsi	*Use case* ini menjelaskan tentang proses pemindahan posisi terhadap shape yang dilakukan oleh user
Kondisi Awal	User telah membentuk shape pada kotak gambar.
Kondisi Akhir	Posisi objek shape akan berpindah dari posisi semula.
Skenario : Tambah Shape.	
Aksi dari Aktor	**Tanggapan dari Sistem**
1. User memilih dengan mengklik shape.	2. Sistem akan menandai objek yang telah dipilih oleh user.
3. User menggeser shape dengan jarak tertentu sesuai dengan keinginan.	4. Sistem memindahkan shape dengan jarak tertentu dari posisi semula sesuai perintah user.

8. Use case Remove Shape

   Pada *use case* Remove Shape, akan dijelaskan secara detail tentang mekanisme penghapusan shape yang dilakukan oleh user. Skenario *use case* Remove Shape dijelaskan pada Tabel 3.11.

Tabel 3.11. *Use case* Remove Shape

Use case	***Remove Shape***
Aktor	User
Tujuan	Melakukan penghapusan shape yang dilakukan oleh user.
Deskripsi	*Use case* ini menjelaskan tentang proses penghapusan shape yang dilakukan oleh user.
Kondisi Awal	User telah membentuk shape pada kotak gambar.
Kondisi Akhir	Shape yang dihapus oleh user akan hilang.
Skenario : Remove Shape.	
Aksi dari Aktor	**Tanggapan dari Sistem**
1. User memilih dengan mengklik shape yang akan dihapus.	2. Sistem menandai objek shape yang dipilih oleh user.
3. User menekan menu hapus.	4. Sistem menghapus atau menghilangkan objek shape yang dipilih.

9. Use case Save File

   Pada *use case* Save File, akan dijelaskan secara detail tentang mekanisme penyimpanan file yang dilakukan oleh user. Skenario *use case* Save File dijelaskan pada Tabel 3.12.

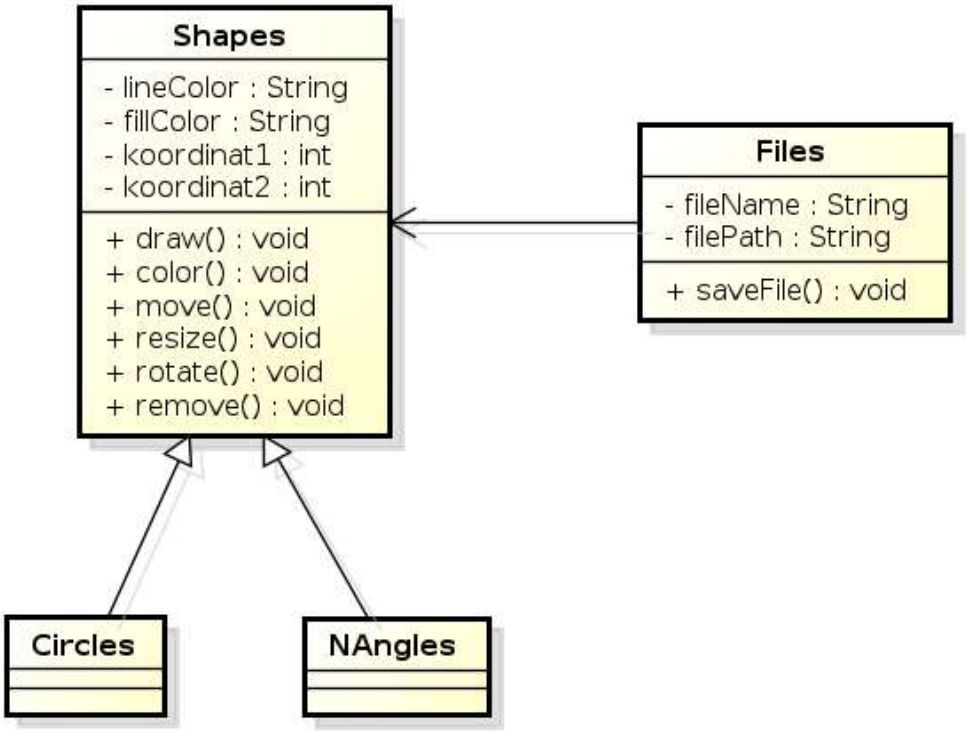
Tabel 3.12 *Use case* Save File

Use case	***Save File***
Aktor	User
Tujuan	Melakukan melakukan penyimpanan file

	yang telah dibuat.
Deskripsi	*Use case* ini menjelaskan tentang proses penyimpanan file yang dilakukan oleh user.
Kondisi Awal	User telah membentuk shape pada kotak gambar.
Kondisi Akhir	Shape dibuat oleh user telah tersimpan dalam sistem.
Skenario : Save File.	
Aksi dari Aktor	**Tanggapan dari Sistem**
1. User memilih menu save pada aplikasi.	2. Sistem menyimpan objek yang telah dibuat ole user.

3.2 Penyusuann Diagram Kelas

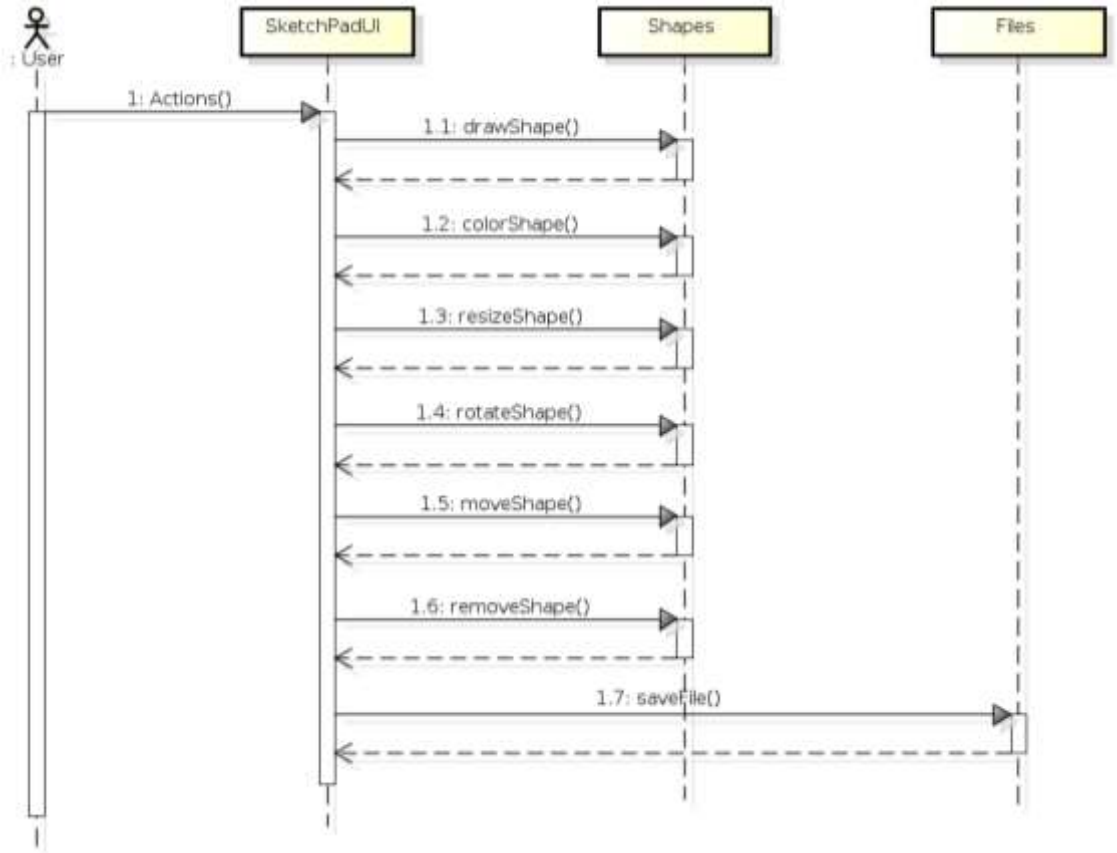
Pada penyusunan diagram kelas ini terdapat beberapa kelas yang diidentifikasi. Dimana tiap – tiap kelas tersebut merepresentasikan bentuk kelas pada kode aplikasi yang nantinya akan dibuat oleh programmer. Tiap kelas mempunyai atribut serta method yang sesuai dengan kelasnya. Pada kasus aplikasi autoshape ini terjadi mekanisme inheritance dimana terdapat kelas induk serta anak. Kelas induk disini adalah kelas Shapes yang mewariskan atribut serta methodnya kepada kelas anak yaitu circle dan NAngle sehingga atribut serta method dari kelas Shapes juga dimiliki oleh kelas circle dan kelas NAngle. Selain itu juga terdapat kelas Files yang berfungsi dalam menyimpan file yang telah dibuat oleh user. Berikut ini adalah detail dari kelas diagram aplikasi autoshapes.



Gambar 3.3. Diagram kelas aplikasi autoshapes

3.3 Penyusunan Diagram Sequence

Diagram sequence adalah suatu diagram yang digunakan untuk menunjukkan kolaborasi dari obyek-obyek dan mendefinisikan urutan pesan atau event antara obyek tersebut berdasarkan waktu. Interaksi antar objek dalam diagram ini digambarkan dalam dua dimensi yakni dimensi vertical dan dimensi horisontal. Dimensi vertical mrepresentasikan waktu proses sedangkan dimensi horisontal menunjukan classifier roles (peran pengklasifikasi) yang merepresentasikan objek-objek individu dalam kolaborasi (kerjasama). Berikut ini adalah diagram sequence pada aplikasi autoshapes.



Gambar 3.4. Diagram sequence aplikasi autoshapes

BAB IV
IMPLEMENTASI

Pada tahapan implementasi ini merupakan penerapan dari perancangan yang telah dibahas pada bab sebelumnya. Tujuan implementasi ini sendiri agar perancangan aplikasi yang telah dibuat sebelumnya dapat diwujudkan secara nyata dalam bentuk produk aplikasi.

4.1 Lingkungan Implementasi

Pada implementasi aplikasi autoshape ini diperlukan perangkat keras dan perangkat lunak dalam membuat produk aplikasi sesuai yang diinginkan. Maka dari itu diperlukan pembahasan spesifikasi perangkat keras (hardware) serta perangkat lunak (software)

4.1.1 Hardware Requirement

Pada aplikasi autoshape dibutuhkan perangkat keras (hardware) dalam membangunya. Adapun spesifikasi dari hardware yang direkomendasikan tersebut yaitu sebagai berikut:

Processor : Intel Core (Tm)2 duo CPU T6400 @ 2Ghz.
Memory : 1 GB RAM
Display Mode : 1280 x 800 (32 Bit) (60 Hz).
Harddisk : 500 GB

4.1.2 Software

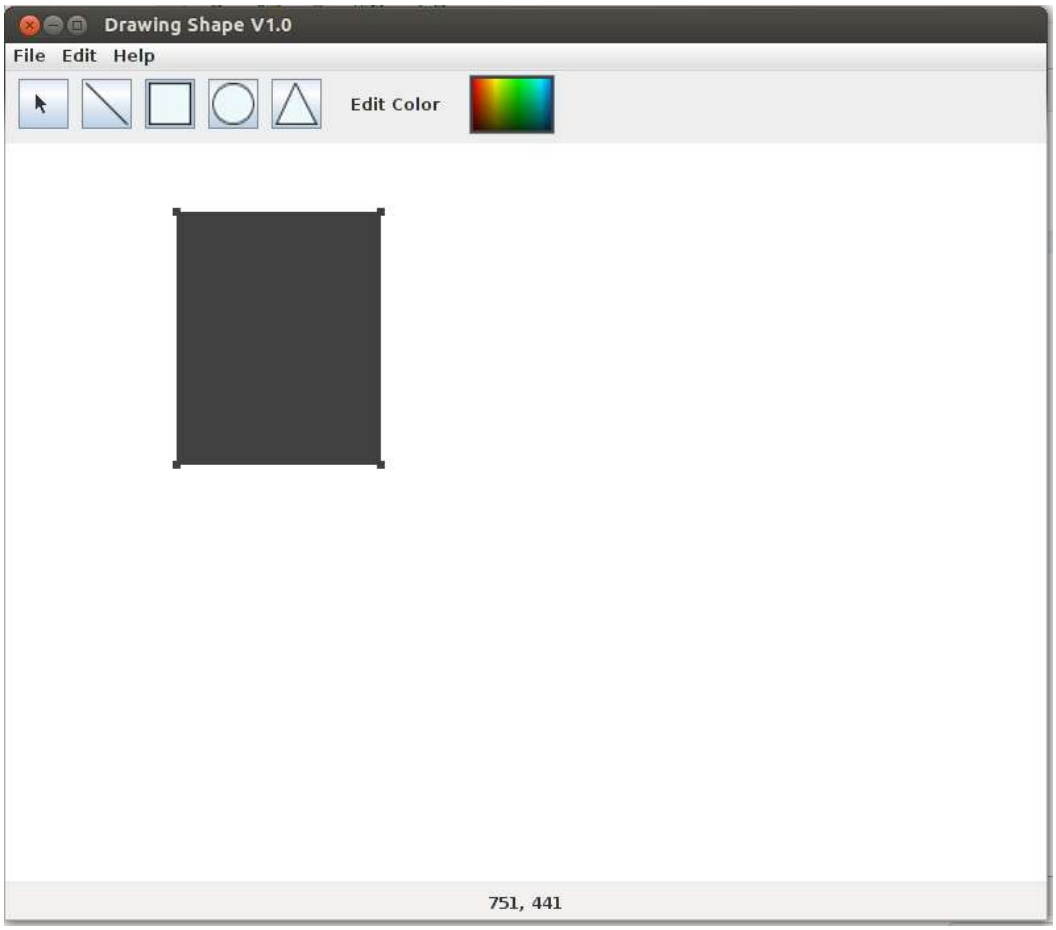
Selain hardware dibutuhkan pula software dalam membuat serta mengembangkan aplikasi autoshape ini. Software yang direkomendasikan tersebut yaitu antara lain:

OS : Multi OS
IDE : Netbeans 7.3.1
Bahasa Pemrograman : Java

Versi JVM : OpenJDK 1.7 atau JDK 7

4.2 Implementasi Aplikasi

Pada bagian ini akan dijelaskah mengenai aplikasi yang dibuat berdasarkan perancangan pada bab sebelumnya. Aplikasi Autoshape ini dikembangkan dengan menggunakan user interface yang friendly dan mudah dimengerti untuk pemula maupun yang sudah berpengalaman. Terdapat beberapa menu yang berfungsi untuk menggambar objek shape dan juga menu untuk memanipulasi (pewarnaan) objek shape tersebut. Berikut ini merupakan user interface dari aplikasi autoshape yang dibuat berdasarkan perancangan pada bab sebelumnya:



Gambar 4.1. Antar muka aplikasi autoshapes

4.2.1 Penjelasan User Interface

Pada user interface aplikasi autoshape terdapat beberapa menu yang berfungsi untuk gambar serta manipulasi objek. Pada aplikasi autoshape ini dibagi menjadi 2 menu utama yaitu.

1. Menu Gambar.

   Menu ini pada user interface ditunjukkan dengan simbol shape antara lain persegi, lingkaran, serta segitiga. Dengan mengklik salah satu diantara symbol atau menu tersebut dengan menggunakan kursor, user dapat menggambar pada kanvas (lembar kerja) sesuai dengan shape yang user pilih sebelumya.

2. Menu Manipulasi (Pewarnaan)

   Pada menu ini berfungsi untuk memanipulasi objek shape yang telah tergambar pada kanvas. Manipulasi itu sendiri berkaitan dengan pewarnaan garis serta bidang objek shape. Terdapat banyak pilihan warna yang user dapat pilih sesuai dengan keinginan dengan cara mengklik menu edit color. Dimana pada menu edit color tersebut terdapat banyak pilihan warna sesuai standard pewarnaan RGB.

Selain kedua menu tersebut terdapat pula bagian yang tak kalah pentingnya yaitu **kanvas (lembar kerja).** Dimana kanvas pada aplikasi autoshape ini ditunjukkan dengan area kosong berwarna putih dibawah daftar menu. Kanvas ini berfungsi sebagai bidang atau tempat untuk menggambar shape oleh user dengan menggunakan kursor.

Pada kanvas ini selain user dapat melakukan aktifitas menggambar aplikasi ini juga menyediakan fungsi – fungsi yang dapat dilakukan terhadap objek shape yang tergambar pada kanvas. Adapun fungsi – fungsi tersebut yaitu antara lain sebagai berikut:

1. **Resize Shape**

   User dapat melakukan perubahan ukuran pada shape dengan menahan klik pada objek shape kemudian menggeser kursor ke dalam maupun ke luar.

Dengan user melakukan hal tersebut maka shape akan mengalami perubahan ukuran menjadi lebih kecil ataupun menjadi lebih besar.

2. **Rotate Shape**

   User dapat melakukan rotasi terhadapa shape dengan cara menahan klik pada objek shape kemudian melakukan gerakan memutar dengan kursor dengan derajat sesuai yang user inginkan.

3. **Move Shape**

   User dapat melakukan aktifitas pemindahan shape dengan cara menahan klik pada objek shape kemudian menggeser atau memindahkan objek tersebut ke posisi yang user inginkan.

4. **Remove Shape**

   User dapat menghapus objek shape dengan cara mengklik objek shape yang akan dihapus kemudian dengan memencet tombol delete pada keyboard maka objek shape tersebut akan otomatis terhapus atau hilang dari kanvas (lembar kerja)

Selain fungsi – fungsi tersebut aplikasi autoshape ini menyediakan fasilitas untuk melakukan **save file objek autoshape** yang telah tergambar pada kanvas (lembar kerja). Dimana file yang disimpan nantinya dalam format image (gambar).

4.2.2 Source Code Program

GUI.java

```
/*
 * @version     1.0 10/8/99
 * @author      Julie Zelenski
 * @source
http://www.stanford.edu/class/cs193j/assignments/hw2/
 * @editor      1. Nick Parlante 2/2002
 *              2. EkoSubha, Huda, and Adam at Jan 11th, 2014
 */

package drawingshape;

import java.awt.BorderLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
```

```
import javax.swing.JFrame;

public class GUI {

    static public void main(String[] args) {
        JFrame frame = new JFrame("Drawing Shape V1.0");

        MenuBar menuBar = new MenuBar();
        ToolBar toolBar = new ToolBar();
        StatusBar statusBar = new StatusBar();
        DrawingCanvas canvas = new DrawingCanvas(toolBar,
statusBar, 790, 560);

        frame.getContentPane().setLayout(new BorderLayout(4,4));
        frame.getContentPane().add(toolBar, BorderLayout.NORTH);
        frame.getContentPane().add(canvas, BorderLayout.CENTER);
        frame.getContentPane().add(statusBar, BorderLayout.SOUTH);
        frame.setJMenuBar(menuBar.createMenuBar(canvas));

        frame.pack();           // resize window to fit components
at preferred size
        frame.setVisible(true); // bring window on-screen

        frame.addWindowListener(
                new WindowAdapter() {
                    @Override
                    public void windowClosing(WindowEvent e) {
                        System.exit(0);
                    }
                });
        frame.setLocationRelativeTo(null);  //centered window

    }

}
```

MenuBar.java

```
/**
 * The MenuBar class set up some menu.
 * this class is implemented from createMenuBar of JavaDraw class
 *
 * @version     1.0 January 11th, 2014
 * @author      Julie Zelenski
 * @source
http://www.stanford.edu/class/cs193j/assignments/hw2/
 * @editor      EkoSubha at Jan 13rd, 2014
 *
 */
package drawingshape;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
```

```
import javax.swing.KeyStroke;

public class MenuBar {

    public MenuBar() {
    }

    public JMenuBar createMenuBar(final DrawingCanvas canvas) {
        JMenuBar menuBar = new JMenuBar();
        JMenuItem submenu;
        int menuMask =
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask();

        JMenu menu = new JMenu("File");
        // submenu save
        menu.add(submenu = new JMenuItem("Save"));

submenu.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
menuMask));
        submenu.addActionListener( new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
canvas.saveToFile();}});

        menu.add(submenu = new JMenuItem("Quit"));

submenu.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q,
menuMask));
        submenu.addActionListener( new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
System.exit(0);}});

        menuBar.add(menu);

        // edit menu
        menu = new JMenu("Edit");

        // submenu delete
        menu.add(submenu = new JMenuItem("Delete"));

submenu.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_BACK_SPA
CE, 0));
        submenu.addActionListener( new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
canvas.delete();}});

        menu.add(submenu = new JMenuItem("Clear all"));
        submenu.addActionListener( new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
canvas.clearAll();}});
        menuBar.add(menu);

        // edit menu
        menu = new JMenu("Help");
```

```
        // submenu delete
        menu.add(submenu = new JMenuItem("About"));
        submenu.addActionListener( new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
canvas.clearAll();}});
        menuBar.add(menu);

        return menuBar;
    }
}
```

ToolBar.java

```
/**
 * @version     1.0 10/12/99
 * @author      Julie Zelenski
 * @source
http://www.stanford.edu/class/cs193j/assignments/hw2/
 * @editor      EkoSubha, Huda, and Adam at Jan 11th, 2014
 */

package drawingshape;

import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Vector;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JColorChooser;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JToggleButton;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;



public class ToolBar extends JPanel {
    protected JToggleButton[] toolButtons;
    protected JButton colorButton;
    protected Vector listeners;

    protected static final String[] toolNames = {"select32",
"line32", "rect32", "oval32", "triangle32"};

    public static final int SELECT = 0, RECT = 1, OVAL = 2, LINE =
3, SCRIBBLE = 4;

    /**
     * Constructor for the ToolBar. Creates the tool buttons,
arranges
```

```
     * within the panel in proper layout, attachs necessary
listeners, etc.
     * Records references to the needed buttons so we can later
set/get the
     * tool and color choices.
     */
    public ToolBar() {
        setLayout(new FlowLayout(FlowLayout.LEFT, 10, 3));
        Insets noMargin = new Insets(0,0,0,0);

        // Create a row of toggle buttons, each with a tool image,
put into a button group so
        // turning on one turns off the others.
        ButtonGroup group = new ButtonGroup();
        toolButtons = new JToggleButton[toolNames.length];

        for (int i = 0; i < toolNames.length; i++) {
            ImageIcon shapeIcon =
createImageIcon("Images/"+toolNames[i] + ".png");
            toolButtons[i] = new JToggleButton(shapeIcon, i==0);
            group.add(toolButtons[i]);
            add(toolButtons[i]);
            toolButtons[i].setMargin(noMargin);
        }

        add(new JLabel("   Edit Color   "));

        ImageIcon colorIcon =
createImageIcon("Images/editColor.png");
        //ImageIcon colorIcon =
createImageIcon("Images/Bucket.GIF");
        colorButton = new JButton(colorIcon);
        //colorButton = new JButton(new
ImageIcon("Images/Bucket.GIF"));
        colorButton.setBackground(Color.darkGray);
        colorButton.setMargin(noMargin);
        add(colorButton);
        colorButton.addActionListener( new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Color chosenColor;
                if ((chosenColor = JColorChooser.showDialog(null,
"Choose shape fill color", getCurrentColor())) != null)
                    setCurrentColor(chosenColor);
            }});
        listeners = new Vector();
    }

    /**
     * Returns current tool selected, using the public enumerated
ToolBar
     * constants SELECT, RECT, OVAL, etc. to identify the
different tools.
     * If there is currently no selected button, (shouldn't
happen), -1
     * is returned.
     */
```

```
    public int getCurrentTool() {
        for (int i = 0; i < toolButtons.length; i++)
            if (toolButtons[i].isSelected()) return i;
        return -1;
    }

    /**
     * Changes current tool, by selecting a new button and
deselecting
     * previous selection. toolNum parameter should be one of the
     * public enumerated ToolBar constants SELECT, RECT, OVAL,
etc.
     * If invalid, an array out of bounds exception will be
thrown.
     */
    public void setCurrentTool(int toolNum) {
        toolButtons[toolNum].setSelected(true);
    }

    /**
     * Returns current color showing on the toolbar's color
button.
     */
    public Color getCurrentColor() {
        return colorButton.getBackground();
    }

    /**
     * Sets current color showing on the toolbar's color button.
     * Anytime the current color is changed, a ChangeEvent is
     * sent to all registered listeners.
     */
    public void setCurrentColor(Color color) {
        colorButton.setBackground(color);
        fireStateChanged();
    }

    /**
     * Adds a ChangeListener to the toolbar. Whenever the
     * color is changed on the toolbar, the listener will be
     * notified with a change event.
     */
    public void addChangeListener(ChangeListener l) {
        listeners.addElement(l);
    }

    /**
     * Removes a ChangeListener from the toolbar.
     */
    public void removeChangeListener(ChangeListener l) {
        listeners.removeElement(l);
    }

    /**
     * Notify all listeners registered for notification when the
     * toolbar change.
     */
```

```
    protected void fireStateChanged() {
        ChangeEvent changeEvent = new ChangeEvent(this);
        for (int i = 0; i < listeners.size(); i++)

((ChangeListener)listeners.elementAt(i)).stateChanged(changeEvent)
;
    }

    /** Returns an ImageIcon, or null if the path was invalid. */
    protected ImageIcon createImageIcon(String path) {
        java.net.URL imgURL = getClass().getResource(path);
        if (imgURL != null) {
            return new ImageIcon(imgURL);
        } else {
            System.err.println("Couldn't find file: " + path);
            return null;
        }
    }

}
```

DrawingCanvas.java

```
/*
 *
 * @version     1.0 10/13/99
 * @author      Julie Zelenski
 * @source
http://www.stanford.edu/class/cs193j/assignments/hw2/
 * @editor      EkoSubha, Huda, and Adam at Jan 11th, 2014
 */

package drawingshape;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.util.ArrayList;
import java.util.Iterator;
import javax.swing.JFileChooser;
import javax.swing.JPanel;

public class DrawingCanvas extends JPanel {
    protected ArrayList allShapes;  // list of all shapes on
canvas
    protected Rect selectedShape;   // currently selected shape
(can be null at times)
    protected ToolBar toolbar;      // reference to toolbar to
message for tool&color settings
    protected StatusBar statusBar;
    protected MenuBar menuBar;
    protected Rectangle rectangle;
```

```
    /**
     * Constructor for creating a new empty DrawingCanvas. We set
up
     * our size and background colors, instantiate an empty vector
of shapes,
     * and install a listener for mouse events using our inner
class
     * CanvasMouseHandler
     */
    public DrawingCanvas(ToolBar tb, StatusBar st, int width, int
height) {
        setPreferredSize(new Dimension(width, height));
        setBackground(Color.white);
        toolbar = tb;
        statusBar = st;
        allShapes = new ArrayList();
        selectedShape = null;

        CanvasMouseHandler handler = new CanvasMouseHandler();
        addMouseListener(handler);
        addMouseMotionListener(handler);
    }

    /**
      * All components are responsible for drawing themselves in
      * response to repaint() requests.  The standard method a
component
      * overrides is paint(Graphics g), but for Swing components,
the default
      * paint() handler calls paintBorder(), paintComponent() and
paintChildren()
      * For a Swing component, you override paintComponent and do
your
      * drawing in that method.  For the drawing canvas, we want
to
      * clear the background, then iterate through our shapes
asking each
      * to draw. The Graphics object is clipped to the region to
update
      * and we use to that avoid needlessly redrawing shapes
outside the
      * update region.
      */
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Rectangle clipRect = g.getClipBounds();
        Iterator iter = allShapes.iterator();

        while (iter.hasNext()) {
            ((Rect)iter.next()).draw(g, clipRect);
        }
    }

    /**
      * Changes the currently selected shape. There is at most
```

```
      * one shape selected at a time on the canvas. It is possible
      * for the selected shape to be null. Messages the shape to
      * change its selected state which will in turn refresh the
      * shape with the knobs active.
      */
    protected void setSelectedShape(Rect shapeToSelect) {
        if (selectedShape != shapeToSelect) {   // if change in
selection
            if (selectedShape != null)          // deselect
previous selection
                selectedShape.setSelected(false);
            selectedShape = shapeToSelect;      // set selection
to new shape
            if (selectedShape != null) {
                shapeToSelect.setSelected(true);
            }
        }
    }

    /**
      * A hit-test routine which finds the topmost shape
underneath a
      * given point.We search Vector of shapes in back-to-front
order
      * since shapes created later are added to end and drawn
last, thus
      * appearing to be "on top" of the earlier ones.  When a
click comes
      * in, we want to select the top-most shape.
      */
    protected Rect shapeContainingPoint(Point pt) {
        for (int i = allShapes.size()-1; i >= 0; i--)
        {
            Rect r = (Rect)allShapes.get(i);
            if (r.inside(pt)) return r;
        }
        return null;
    }

    /**
     * The inner class CanvasMouseHandler is the object that
handles the
     * mouse actions (press, drag, release) over the canvas. Since
there is
     * a bit of state to drag during the various operations (which
shape,
     * where we started from, etc.) it is convenient to
encapsulate all that
     * state with this little convenience object and register it
as the
     * handler for mouse events on the canvas.
     */
    protected class CanvasMouseHandler extends MouseAdapter
implements MouseMotionListener {
        Point dragAnchor;   // variables using to track state
during drag operations
        int dragStatus;
```

```
        /** When the mouse is pressed we need to figure out what
          * action to take.  If the tool mode is arrow, the click
might
          * be a select, move or reisze. If the tool mode is one
of the
          * shapes, the click initiates creation of a new shape.
          */
        @Override
        public void mousePressed(MouseEvent event) {
            Rect clicked;// = null;
            Point curPt = event.getPoint();

            if (toolbar.getCurrentTool() == ToolBar.SELECT) {
                // first, determine if click was on resize knob of
selected shape
                if (selectedShape != null && (dragAnchor =
selectedShape.getAnchorForResize(curPt)) != null) {
                    dragStatus = DRAG_RESIZE;   // drag will
resize this shape
                } else if ((clicked = shapeContainingPoint(curPt))
!= null) { // if not, check if any shape was clicked
                    setSelectedShape(clicked);
                    dragStatus = DRAG_MOVE;     // drag will move
this shape
                    dragAnchor = curPt;
                } else {                        // else this was a
click in empty area, deselect selected shape,
                    setSelectedShape(null);
                    dragStatus = DRAG_NONE;     // drag does
nothing in this case
                }
            } else {
                Rect newShape = new Rect(curPt,
DrawingCanvas.this);   // create rect here
                allShapes.add(newShape);
                setSelectedShape(newShape);
                dragStatus = DRAG_CREATE;       // drag will
create (resize) this shape
                dragAnchor = curPt;
            }
        }

        /** As the mouse is dragged, our listener will receive
periodic
          * updates as mouseDragged events. When we get an update
position,
          * we update the move/resize event that is in progress.
          */
        @Override
        public void mouseDragged(MouseEvent event) {
            Point curPt = event.getPoint();

            switch (dragStatus) {
                case DRAG_MOVE:
                    selectedShape.translate(curPt.x -
dragAnchor.x, curPt.y - dragAnchor.y);
```

```
                    dragAnchor = curPt; // update for next dragged
event
                    break;
                case DRAG_CREATE: case DRAG_RESIZE:
                    selectedShape.resize(dragAnchor, curPt);
                    break;
            }
        }

        @Override
        public void mouseMoved(MouseEvent e) {
            statusBar.getStatusBar().setText(String.format("%d,
%d", e.getX(),e.getY()));
        }

        static final int DRAG_NONE = 0, DRAG_CREATE = 1,
DRAG_RESIZE = 2, DRAG_MOVE = 3;
    }

    /** A little helper routine that will be useful for the load &
save
     * operations.  It brings up the standard JFileChooser dialog
and
     * allows the user to specify a file to open or save. The
return
     * value is the full path to the chosen file or null if no
file was
     * selected.
     */
    protected String filenameChosenByUser(boolean forOpen) {
        JFileChooser fc = new
JFileChooser(System.getProperty("user.dir") +
java.io.File.separator + "Documents");
        int result = (forOpen? (fc.showOpenDialog(this)) :
fc.showSaveDialog(this));
        java.io.File chosenFile = fc.getSelectedFile();
        if (result == JFileChooser.APPROVE_OPTION && chosenFile !=
null)
            return chosenFile.getPath();
        return null; // return null if no file chosen or dialog
cancelled
    }

    /** These are the unimplemented menu commands.
     * The menus are already set up to send the correct messages
to the
     * canvas, but the method bodies themselves are currently
completely
     * empty. It will be your job to fill them in!
     */
    public void delete() {}
    public void clearAll() {}
    public void saveToFile() {}

}
```

Statusbar.java

```
/**
 * The StatusBar class used to show status of mouse.
 *
 * @version     1.0 January 11th, 2014
 * @author      Eko Subha
 * @source
http://www.stanford.edu/class/cs193j/assignments/hw2/
 *
 */

package drawingshape;

//import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class StatusBar extends JPanel {
    private JLabel statusBar;

    public StatusBar() {
        this.setPreferredSize(new Dimension(500, 25));
        this.getPreferredSize();
        this.setBackground(new Color(242,241,240));

        statusBar = new JLabel();
        this.add(statusBar);
    }

    public JLabel getStatusBar() {
        return statusBar;
    }

    public void setStatusBar(JLabel statusBar) {
        this.statusBar = statusBar;
    }
}
```

Rect.java

```
/*
 *---------------------------------------------------------------
80 columns ---|
 * The RectShape class defines a simple rectangular shape object.
 * It tracks its bounding box, selected state, and the canvas it
is being
 * drawn in. It has some basic methods to select, move, and resize
the
 * rectangle. It has methods that draw the shape in the selected
or unselected
 * states and updates the canvas whenever the state or rect of the
rectangle
 * change. The code that is there works properly, but you will
need to extend
 * and change the code to support additional features.
 *
```

```
 * @version      1.0 10/13/99
 * @author       Julie Zelenski
 */

package drawingshape;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.Rectangle;

public class Rect {
    protected Rectangle rect;
    protected boolean isSelected;
    public DrawingCanvas canvas;

    protected static final int KNOB_SIZE = 6;
    protected static final int NONE = -1, NW = 0, SW = 1, SE = 2 ,
NE = 3;

    /** The constructor that creates a new zero width and height
rectangle
     * at the given position in the canvas.
     */
    public Rect(Point start, DrawingCanvas dcanvas) {
        canvas = dcanvas;
        rect = new Rectangle(start);
    }

    /** The "primitive" for all resizing/moving/creating
operations that
      * affect the rect bounding box. The current implementation
just resets
      * the rect variable and triggers a re-draw of the union of
the old &
      * new rectangles. This will redraw the shape in new size and
place and
      * also "erase" if rect are now smaller than before.  It is a
good
      * design to have all changes to a critical variable
bottleneck through
      * one method so that you can be sure that all the updating
that goes
      * with it only needs to be implemented in this one place. If
any of your
      * subclasses have additional work to do when the rect
change, this is
      * the method to override.  Make sure that any methods that
change the
      * rect call this method instead of directly manipulating the
variable.
      */
    protected void setRect(Rectangle newBounds) {
        Rectangle oldRect = rect;
        rect = newBounds;
        updateCanvas(oldRect.union(rect));
    }
```

```
    /** The resize operation is called when first creating a rect,
as well as
      * when later resizing by dragging one of its knobs.  The two
parameters
      * are the points that define the new bounding box.  The
anchor point
      * is the location of the mouse-down event during a creation
operation
      * or the opposite corner of the knob being dragged during a
resize
      * operation. The end is the current location of the mouse.
If you
      * create the smallest rectangle which encloses these two
points, you
      * will have the new bounding box.  Use the setBounds()
primitive which
      * is the bottleneck we are using for all geometry changes,
it handles
      * updating and redrawing.
      */
    public void resize(Point anchor, Point end) {
        Rectangle newRect = new Rectangle(anchor);
        newRect.add(end);           // creates smallest rectange
which includes both anchor & end
        setRect(newRect);     // reset rect & redraw affected
areas
    }


    /** The translate operation is called when moving a shape by
dragging in
      * the canvas.  The two parameters are the delta-x and delta-
y to move
      * by.  Note that either or both can be negative. Create a
new rectangle
      * from our rect and translate and then go through the
setBounds()
      * primitive to change it.
      */
    public void translate(int dx, int dy) {
        Rectangle newRect = new Rectangle(rect);
        newRect.translate(dx, dy);
        setRect(newRect);
    }

    /** Used to change the selected state of the shape which will
require
      * updating the affected area of the canvas to add/remove
knobs.
      */
    public void setSelected(boolean newState) {
        isSelected = newState;
        updateCanvas(rect, true);  // need to erase/add knobs
including extent of extended rect
    }
```

```
    /** The updateCanvas() methods are used when the state has
changed
      * in such a way that it needs to be refreshed in the canvas
to properly
      * reflect the new settings. The shape should take
responsibility for
      * messaging the canvas to properly update itself. The
appropriate AWT/JFC
      * way to re-draw a component is to send it the repaint()
method with the
      * rectangle that needs refreshing.  This will cause an
update()  event to
      * be sent to the component which in turn will call paint(),
where the
      * real drawing implementation goes.  See the paint() method
in
      * DrawingCanvas to see how it is implemented.
      */
    protected void updateCanvas(Rectangle areaOfChange, boolean
enlargeForKnobs) {
        Rectangle toRedraw = new Rectangle(areaOfChange);
        if (enlargeForKnobs)
            toRedraw.grow(KNOB_SIZE/2, KNOB_SIZE/2);
        canvas.repaint(toRedraw);
    }

    protected void updateCanvas(Rectangle areaOfChange) {
        updateCanvas(areaOfChange, isSelected);
    }

    /** When the DrawingCanvas needs a shape to draw itself, it
sends a draw
      * message, passing the graphics context and the current
region being
      * redrawn. If the shape intersects with that region, it must
draw itself
      * doing whatever it takes to properly represent itself in
the canvas
      * (colors, location, size, knobs, etc.) by messaging the
Graphics object.
      */
    public void draw(Graphics g, Rectangle clipRect) {
        if (!rect.intersects(clipRect))
            return;

        g.setColor(Color.darkGray);
        g.fillRect(rect.x, rect.y, rect.width, rect.height);
        if (isSelected) { // if selected, draw the resizing knobs
along the 4 corners
            Rectangle[] knobs = getKnobRects();
            for (int i = 0; i < knobs.length; i++)
                g.fillRect(knobs[i].x, knobs[i].y, knobs[i].width,
knobs[i].height);
        }
    }

    /** When the DrawingCanvas needs to determine which shape is
```

```
under
      * the mouse, it asks the shape to determine if a point is
"inside".
      * This method should returns true if the given point is
inside the
      * region for this shape. For a rectangle, any point within
the
      * bounding box is inside the shape.
      */
    public boolean inside(Point pt) {
        return rect.contains(pt);
    }

    /** When needed, we create the array of knob rectangles on
demand. This
      * does mean we create and discard the array and rectangles
repeatedly.
      * These are small objects, so perhaps it is not a big deal,
but
      * a valid alternative would be to store the array of knobs
as an
      * instance variable of the Shape and and update the knobs as
the rect
      * change. This means a little more memory overhead for each
Shape
      * (since it is always storing the knobs, even when not being
used) and
      * having that redundant data opens up the possibility of
bugs from
      * getting out of synch (rect move but knobs didn't, etc.)
but you may
      * find that a more appealing way to go.  Either way is fine
with us.
      * Note this method provides a nice unified place for one
override from
      * a shape subclass to substitute fewer or different knobs.
      */
    protected Rectangle[] getKnobRects() {
        Rectangle[] knobs = new Rectangle[4];
        knobs[NW] = new Rectangle(rect.x - KNOB_SIZE/2, rect.y -
KNOB_SIZE/2, KNOB_SIZE, KNOB_SIZE);
        knobs[SW] = new Rectangle(rect.x - KNOB_SIZE/2, rect.y +
rect.height - KNOB_SIZE/2, KNOB_SIZE, KNOB_SIZE);
        knobs[SE] = new Rectangle(rect.x + rect.width -
KNOB_SIZE/2, rect.y + rect.height - KNOB_SIZE/2, KNOB_SIZE,
KNOB_SIZE);
        knobs[NE] = new Rectangle(rect.x + rect.width -
KNOB_SIZE/2, rect.y - KNOB_SIZE/2, KNOB_SIZE, KNOB_SIZE);
        return knobs;
    }

    /** Helper method to determine if a point is within one of the
resize
      * corner knobs.  If not selected, we have no resize knobs,
so it can't
      * have been a click on one.  Otherwise, we calculate the
knob rects and
```

```
      * then check whether the point falls in one of them.  The
return value
      * is one of NW, NE, SW, SE constants depending on which knob
is found,
      * or NONE if the click doesn't fall within any knob.
      */
    protected int getKnobContainingPoint(Point pt) {
        if (!isSelected)      // if we aren't selected, the knobs
aren't showing and thus there are no knobs to check
            return NONE;

        Rectangle[] knobs = getKnobRects();
        for (int i = 0; i < knobs.length; i++)
            if (knobs[i].contains(pt))
                return i;
        return NONE;
    }

    /** Method used by DrawingCanvas to determine if a mouse click
is starting
     * a resize event. In order for it to be a resize, the click
must have
     * been within one of the knob rects (checked by the helper
method
     * getKnobContainingPoint) and if so, we return the "anchor"
ie the knob
     * opposite this corner that will remain fixed as the user
drags the
     * resizing knob of the other corner around. During the drag
actions of a
     * resize, that fixed anchor point and the current mouse point
will be
     * passed to the resize method, which will reset the rect in
response
     * to the movement. If the mouseLocation wasn't a click in a
knob and
     * thus not the beginning of a resize event, null is returned.
     */
    public Point getAnchorForResize(Point mouseLocation) {
        int whichKnob = getKnobContainingPoint(mouseLocation);

        if (whichKnob == NONE) // no resize knob is at this
location
            return null;
        switch (whichKnob) {
            case NW: return new Point(rect.x + rect.width, rect.y
+ rect.height);
            case NE: return new Point(rect.x, rect.y +
rect.height);
            case SW: return new Point(rect.x + rect.width,
rect.y);
            case SE: return new Point(rect.x, rect.y);
        }
        return null;
    }

    @Override
```

```
    public Object clone() {return this;}
}
```

BAB V
PENUTUP

5.1 Kesimpulan

Dari hasil perancangan serta implementasi aplikasi autoshape ini didapatkan suatu kesimpulan yaitu antara lain :

1. Dalam membangun aplikasi autoshape ini diperlukan suatu perancangan yang benar mulai dari use case diagram, class diagram, serta sequence diagram. Hal itu dilakukan agar user dapat memahami aplikasi autoshape yang dibuat oleh developer.
2. Pengembangan aplikasi autoshape ini pada dasarnya merupakan implementasi pemodelan berorientasi objek dari fitur autoshape pada Microsoft Word yang telah ada sebelumnya.
3. Dalam pembuatan suatu sistem atau program diperlukan kerangka pemodelan yang jelas agar dihasilkan program yang sesuai dengan keiinginan pengguna.

5.2 Saran

Beberapa saran untuk aplikasi autoshape ini yaitu antara lain :

1. Aplikasi autoshape ini diharapkan selanjutnya agar dapat dijadikan media pembelajaran bagi anak – anak dalam mengenalkan objek shape yang mempunyai beberapa tipe bentuk serta warna.
2. Aplikasi autoshape ini untuk selanjutnya agar dapat dikembangkan tidak hanya terbatas menggambar objek shape 2 dimensi namun dapat juga menggambar objek shape 3 dimensi.

DAFTAR PUSTAKA

[Deb-13] Debasish Jana. *Java and Object-Oriented Programming Paradigm*. Prentice-Hall of India, India, 2009. ISBN: 0-120-32775-6.

[Jos-01] Joshua Bloch. *Effective JavaTM Programming Language Guide*, First Edition. Addison-Wesley, Boston, 2001. ISBN: 0-201-31005-8.

[Tho-09] C. Thomas Wu. *An Introduction to Object-Oriented Programming with JavaTM*, Fifth Edition. Mc Graw-Hill, New York, 2009. ISBN: 0-07-352330-5.

LAMPIRAN