BAB II

TINJAUAN PUSTAKA

II.1. Sepeda Motor

Sepeda motor adalah sebuah mesin yang terbuat dari ribuan komponen. Secara umum, pemilik dan pengguna sepeda motor berharap tidak ada kerusakan pada motor miliknya, namun permasalahan pada motor seringkali terjadi. Untuk mengatasi masalah yang mungkin terjadi pemilik dan pengguna motor setidaknya mampu mengetahui lebih kerusakan pada mesin motor sehingga dapat dilakukan penanganan dini. Didalam dunia sepeda motor, maka dikenal ada 3 (tiga) jenis mesin yang digunakan yaitu mesin 2 TAK, 4 TAK dan battere. Secara harfiah, sebenarnya yang disebut dengan TAK adalah langkah atau dalam bahasa Inggrisnya disebut dengan *STROKE*. Dengan kata lain, 2 TAK adalah mesin 2 langkah, sementara mesin 4 TAK adalah mesin 4 langkah. Kembali kepada langkah tersebut, maka langkah disini merupakan proses. Untuk memudahkan pengertian terhadap hal tersebut, maka dapat dijelaskan bahwa proses yang terjadi pada mesin 4 langkah adalah sebagai berikut: *INTAKE – COMPRESSION – POWER – EXHAUST* Sementara, proses ini dipersingkat pada mesin 2 tak yang memiliki ruang dibawah piston yang digunakan untuk pemampatan udara dan kompresi(*Anggraheni Rukmana ; 2011 : 2*).

II.2. Jenis mesin Injeksi

Jenis mesin Injeksi dengan Analogi system injection dapat diartikan sebagai suatu cara menyuntikkan bahan bakar langsung ke ruang bakar. Jumlah

bahan bakar serta debit udara yang disuntikkan dikontrol oleh sebuah rangkaian solid, dikenal dengan ECU (Electronic Control Unit) sebagai otaknya. Ada beberapa kelebihan dari mesin injection. Pertama, campuran bahan bakar dan udara terkontrol dengan tepat sehingga pembakaran jadi lebih baik. Yang kedua, mesin dengan sistem injeksi bekerja lebih halus sehingga tenaga lebih besar. Yang ketiga, sistem pengapian lebih sempurna dibandingkan dengan mesin karburator sehingga gas emisi yang dihasilkan rendah*(Novita Tri Handayani ; 2012 : 3).*

II.3. *Forward Chaining*

Forward Chaining adalah suatu metode pengambilan keputusan yang umum digunakan dalam sistem pakar. Proses pencarian dengan metode forward chaining berangkat dari kiri ke kanan, yaitu dari premis menuju kepada kesimpulan akhir, metode ini sering disebut datadriven yaitu pencarian dikendalikan oleh data yang diberikan.Forward Chaining juga disebut penalaran maju yaitu aturan – aturan diuji satu demi satu dalam urutan tertentu. Mesin inferensi akan mencocokan fakta atau statement dalam knowledge base dengan situasi yang dinyatakan dalam rule bagian IF. Jika fakta yang ada dalam Knowledge Base sudah sesuai dengan kaidah IF, maka rule itu distimulasi dan rule berikutnya diuji. Proses pengujian rule satu demi satu berlanjut sampai satu putaran lengkap melalui seluruh perangkat rule *(Level Perdana ; 2012 : 2).*

II.4. Sistem Pakar

Sistem pakar adalah sistem berbasis komputer yang menggunakan pengetahuan, fakta, dan teknik penalaran dalam memecahkan masalah yang

biasanya hanya dapat dipecahkan oleh seorang pakar dalam bidang tersebut (Martin dan Oxman, 1998). Pada dasarnya sistem pakar diterapkan untuk mendukung aktivitas pemecahan masalah. Beberapa aktivitas pemecahan yang dimaksud antara lain: pembuatan keputusan (decision making), pemaduan pengetahuan (knowledge fusing), pembuatan desain (designing), perencanaan (planning), prakiraan (forecasting), pengaturan (regulating), pengendalian (controlling), diagnosis (diagnosing), perumusan (prescribing), penjelasan (explaining), pemberian nasihat (advising) dan pelatihan (tutoring). Selain itu sistem pakar juga dapat berfungsi sebagai asisten yang pandai dari seorang pakar (Martin dan Oxman, 1998) *(Aryati Wuryandari, S.T ; 2013 : 74).*

II.5. Database

Database merupakan kumpulan dari data yang saling berhubungan satu dengan yang lainnya, tersimpan diperangkat keras komputer dan digunakan diperangkat lunak untuk memanipulasinya (Jogiyanto HM : 1999:711). Database merupakan salah satu komponen yang sangat penting dalam sistem informasi, karena merupakan basis sistem dalam menyediakan informasi bagi para pemakai*(Indra Warman ; 2012 : 45).*

II.6. *Java*

Java memiliki cara kerja yang unik dibandingkan dengan bahasa perograman lainya yaitu bahasa perograman *java* bekerja mengunakan *interptreter* dan juga *compiler* dalam proses pembuatan program, *Interpreter java* dikenal sebagai perograman *bytecode* yaitu dengan cara kerja mengubah paket

class pada *java* dengan *extensi*. *Java* menjadi *class*, hal ini dikenal sebagai *class bytecode*, yaitunya *class* yang dihasilkan agar program dapat dijalankan pada semua jenis perangkat dan juga *platform*, sehingga program *java* cukup ditulis sekali namun mampu bekerja pada jenis lingkungan yang berbeda (*Indri Rahmayun* ; 2014 : 64).

Java menurut definisi dari Sun "Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer standalone ataupun pada lingkungan jaringan Java 2 adalah generasi kedua dari Java platform (generasi awalnya adalah JDK atau Java Development Kit). Java inilah yang berdiri diatas mesin interpreter yang diberi nama Java Virtual Machine(JVM). JVM inilah yang akan membaca bytecode dalam file .class dari suatu program sebagai representasi langsung program yang berisi bahasa mesin". Oleh karena itu bahasa java disebut juga sebagai bahasa pemrograman yang portable karena dapat dijalankan sebagai sistem operasi, asalkan pada sistem operasi tersebut terdapat JVM *(Utomo Budiyanto; 2011 : 27).*

II.7. Netbeans

NetBeans merupakan salah satu IDE yang dikembangkan dengan bahasa pemrograman java. NetBeans mempunyai lingkup pemrograman java terintergrasi dalam suatu perangkat lunak yang di dalamnya menyediankan pembangunan pemrogramanGUI, text editor, complier, dan interpreter.NetBeans adalah sebuah perangkat lunak open source sehingga dapat digunakan secara gratis untuk keperluan komersial maupun nonkomersial yang didukung oleh Sun Microsystem *(Atik Rusmayanti ; 2014 : 2-3).*

II.8. MySQL

MySQL Server 2000 adalah suatu Perangkat lunak Mangement system Relational Database (RDBMS) yang handal. Didesain untuk mendukung proses transaksi yang besar (seperti order entri yang online, inventori, akuntansi atau manufaktur). MySQL Server akan secara otomatis menginstal enam database utama, yaitu master, model, tempdb, pubs, Northwind, dan Msdb *(Anis nurhanaf ; 2013 : 3).*

MySQL (My Structure Query Language) adalah salah satu DataBase Management System (DBMS). MySQL berfungsi untuk mengelola database menggunakan bahasa SQL. MySQL bersifat open source sehingga kita bisa menggunakannya secara gratis. Pemrograman PHP juga sangat mendukung/support dengan database MySQL *(Harun Al-Rosyid; 2013 : 2).*

II.9. UML *(Unified Modelling Language)*

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, C# atau

VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C. Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*). Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: *metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock*, dsb. Masa itu terkenal dengan masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan. Dimulai pada bulan Oktober 1994 *Booch, Rumbaugh dan Jacobson*, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease *draft* pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5

yang dirilis bulan Maret 2003. *Booch, Rumbaugh* dan *Jacobson* menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek *(Yuni Sugiarti ; 2013 : 33).*

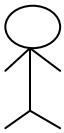
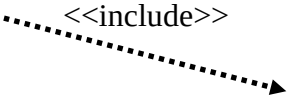
Dalam pembuatan skripsi ini penulis menggunakan diagram *Use Case* yang terdapat di dalam UML. Adapun maksud dari *Use Case* Diagram diterangkan dibawah ini.

1. *Use Case Diagram*

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case diagram* dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use*

case juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain*(Yuni Sugiarti ; 2013 : 41).*

Table II.2. *Use Case* Diagram

Simbol	Deskripsi
Use Case 	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antara unit dan aktor, biasanya dinyatakan dengan menggunakan kata kerja diawali frase nama use case
Aktor  Nama aktor	Orang ,proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang ; biasanya dinyatakan menggunakan kata benda diawali frase nama aktor
Asosiasi / association	Komunikasi antara aktor dan use case yang berpartisipasi pada use case memiliki interaksi dengan aktor
Extend  <<extend>>	Relasi use case tambahan kesebuah use case dimana use case ditambahan dapat berdiri sendiri walaupun tanpa use case tambahan itu ; mirip dengan prinsip inheritens pada pemograman berorientasi objek, biasanya use case tambahan memiliki nama depan yang sama dengan use case yang ditambahkan, arah panah menunjukan pada use case yang dituju
Include <<include>> 	Relasi use case tambahan kesebuah use case dimana use case yang ditambahkan memerlukan use case ini untuk menjalankan fungsinya atau sebagai syarat dijalankan use case ini. Ada dua sudut pandang yang cukup besar mengenai include di use case, include berarti use case tambahan yang dijalankan.

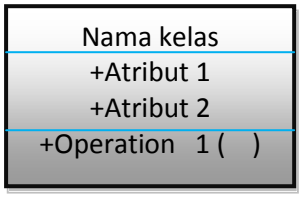
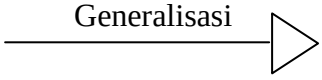
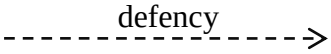
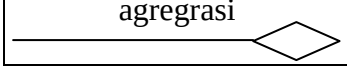
Sumber : (Yuni Sugiarti ; 2013 : 42)

2. *Class Diagram*

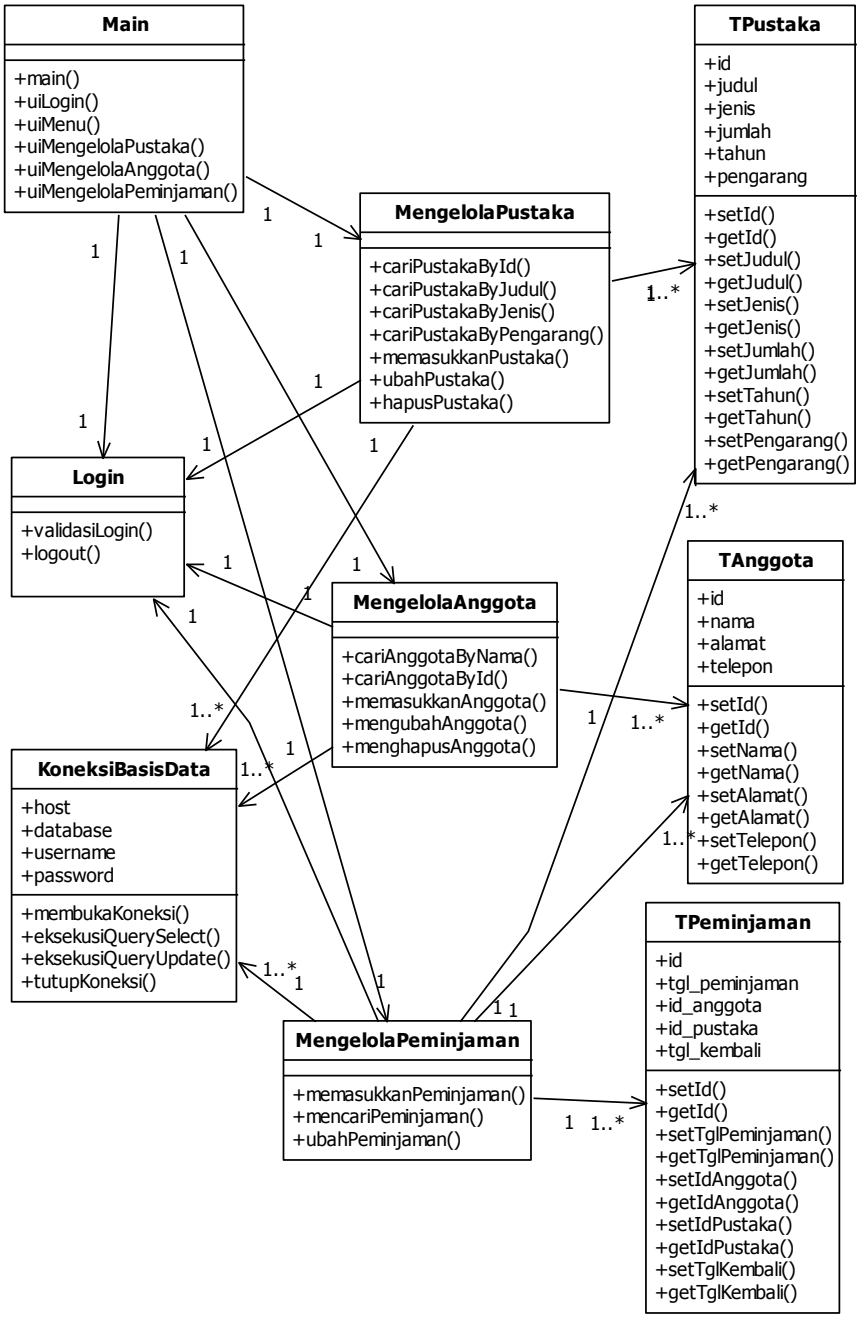
Diagram kelas atau *class* diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas

memiliki apa yang disebut atribut dan metode atau operasi. Berikut adalah simbol-simbol pada diagram kelas :

Table II.3. *Class* Diagram

Simbol	Deskripsi
Package 	Package merupakan sebuah bungkusan dari satu atau lebih kelas
Operasi 	Kelas pada struktur sistem
Anataramuka / interface  Interface	Sama dengan konsep interface dalam pemrograman berorientasi objek
Asosiasi 	Relasi antara kelas dengan makna umum, asosiasi biasanya juga disertai dengan multiplicity
Asosiasi berarah/directed asosiasi	Relasi antara kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan multiplictiy
Generalisasi 	Relasi antara dengan makna generalisasi sepesialisasi (umum-khusus)
Kebergantungan / defency 	Relasi antara kelas dengan makna kebergabungan antara kelas
agregrasi 	Relasi antara kelas dengan makna semua bagian (whole part)

Sumber : (Yuni Sugiarti ; 2013 : 59)



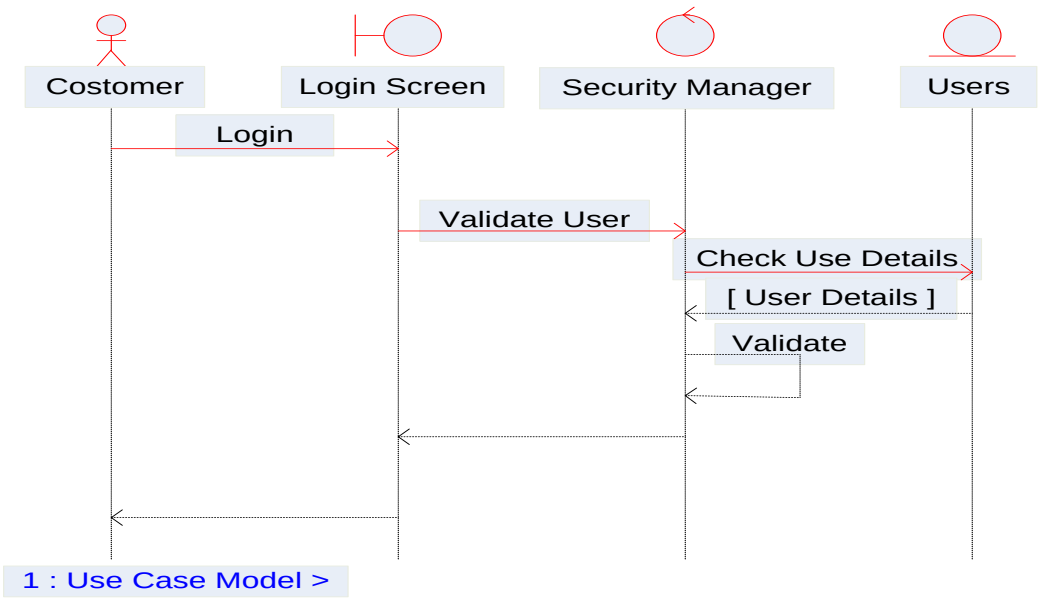
Gambar II.4. Contoh Class Diagram

Sumber : (Yuni Sugiarti ; 2013 : 63)

3. *Sequence Diagram*

Diagram *Sequence* menggambarkan kelakuan/prilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.



Gambar II.5. Contoh *Sequence* Diagram

Sumber : (Yuni Sugiarti ; 2013 : 63)

4. *Activity Diagram*

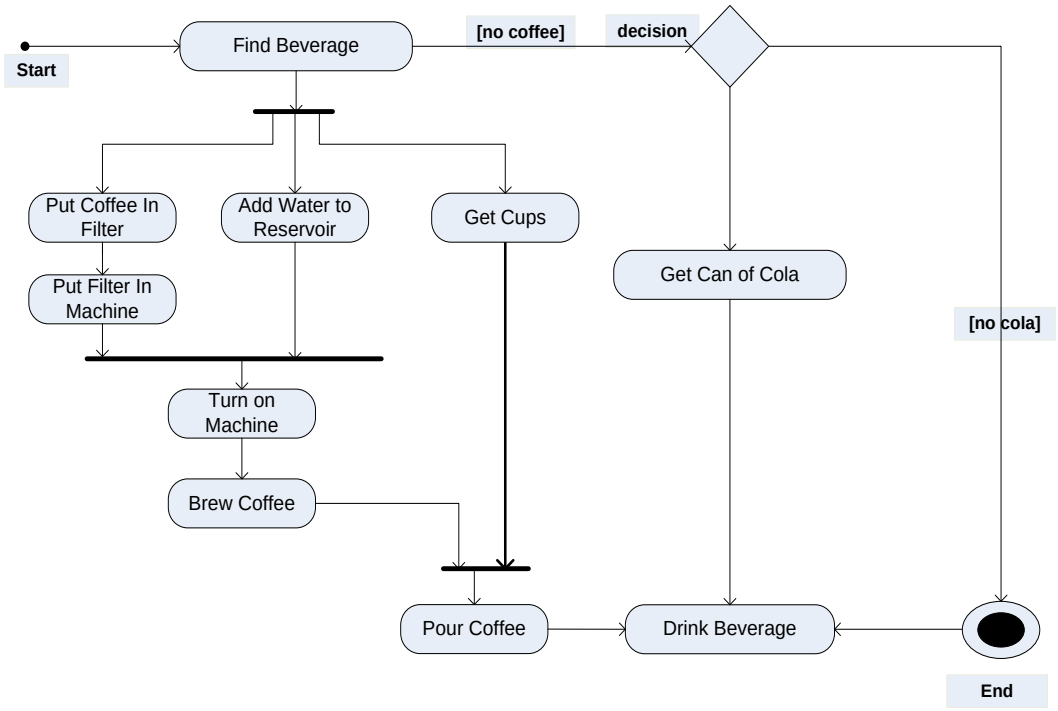
Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

Activity diagram dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.



Gambar II.6. Activity Diagram

Sumber : (Yuni Sugiarti ; 2013 : 76)