BAB II
LANDASAN TEORI

2.1 Kajian Pustaka

2.1.1 Aplikasi Perbandingan Algoritma *Breadth First Search* dan *Depth First Search* pada permainan Tetravex Berbasis Android.

Penelitian mahasiswa Universitas Sumatera Utara yang dilakukan untuk pengembangan menganalisis dan mengimplementasikan algoritma *Breadth First Search* dan *Depth First search* dalam pencarian penyelesaian permainan *Tetravex* berbasis *Mobile Game* Android. Penelitian tersebut bertujuan untuk mempermudah *player* untuk menganalisis algoritma agar dapat menyelesaikan *game* dalam tahap tertentu. *Tetravex* merupakan *game puzzel* yang dimainkan oleh satu orang *player* (Wijaya, 2012).

2.1.2 Aplikasi *Mobile Game* Edukasi Matematika Berbasis Android.

Penelitian ini merupakan penelitian mahasiswa Institut Sains dan Teknologi AKPRIND Yogyakarta pada tahun 2013. Dalam penelitian tersebut di latar belakangi jika dalam bermain sebuah *game* tidak harus membuang waktu yang sia-sia, melainkan juga menghasilkan informasi dan pembelajaran untuk penggunanya. Sistem tersebut merupakan pengimplementasian teknik *design game* dalam pembuatan *game mobile,* sehingga aplikasi tersebut dapat memberikan pengetahuan dan kemampuan tentang matematika (Lestari, 2013).

2.1.3 Aplikasi *Mobile Game* Edukasi Matematika Berbasis Android.

Penelitian ini merupakan penelitian mahasiswa Universitas Negri Yogyakarta pada tahun 2015. Penelitian tersebut pengembangan *game* edukasi ini dilakukan dalam beberapa tahapan, yaitu mengumpulkan data, perancangan sistem, pembuatan sistem, pengujian sistem, dan implementasi sistem. Aplikasi ini dikembangkan karena jumlah *game* edukasi pada *playstore* masih kurang dibandingkan dengan *genre game* yang lain (Rifai, 2015).

2.2 Kerangka Pemikiran

Kerangka pemikiran pada Gambar 2.1 menjelaskan alur proses pembuatan laporan Tugas Akhir.

1. Latar Belakang Masalah

Game Playerunknown Battleground memerlukan data spesifikasi perlengkapan untuk mempermudah *user* atau pemain dalam memainkan *game* dan aplikasi ini di buat bertujuan untuk memberikan kemudahan bagi *user* atau pemain dalam memainkan *game*.

2. Rumusan Masalah

Bagaimana Membangun aplikasi tersebut untuk mempermudah *user* atau pemain dalam meliat informasi spesifikasi perlengkapannya ?

3. Pengumpulan Data Tertulis dan Tidak Tertulis

Mengumpulkan data yang diperlukan untuk penelitian, baik melalui *interview*, observasi maupun dukomentasi.

4. Penguasaan Dasar

Melakukan beberapa percobaan membuat aplikasi android dengan tujuan agar dapat lebih menguasai pemprograman android menggunakan android studio yang merupakan *tool* lengkap yang terdiri dari SDK (*Software Development Kit*), Android dan API (*Application Proggraming Interface*) sebagai alat untuk *design* tampilan, *coding*, dan emulasi.

5. Observasi Aplikasi

Mencari beberapa aplikasi atau tinjauan pustaka yang berkaitan dengan pemprograman android, karya ilmiah, buku yang dapat dijadikan referensi dalam membangun aplikasi spesifikasi *weapon equipment* dan *attachment* pada *game playerunknown battleground*.

6. Analisis dan perancangan Sistem Berorientasi Objek

Menganalisa dan merancang aplikasi yang akan di bangun, yang meliputi *design* aplikasi dan isi aplikasi yang akan di bangun.

7. Implementasi

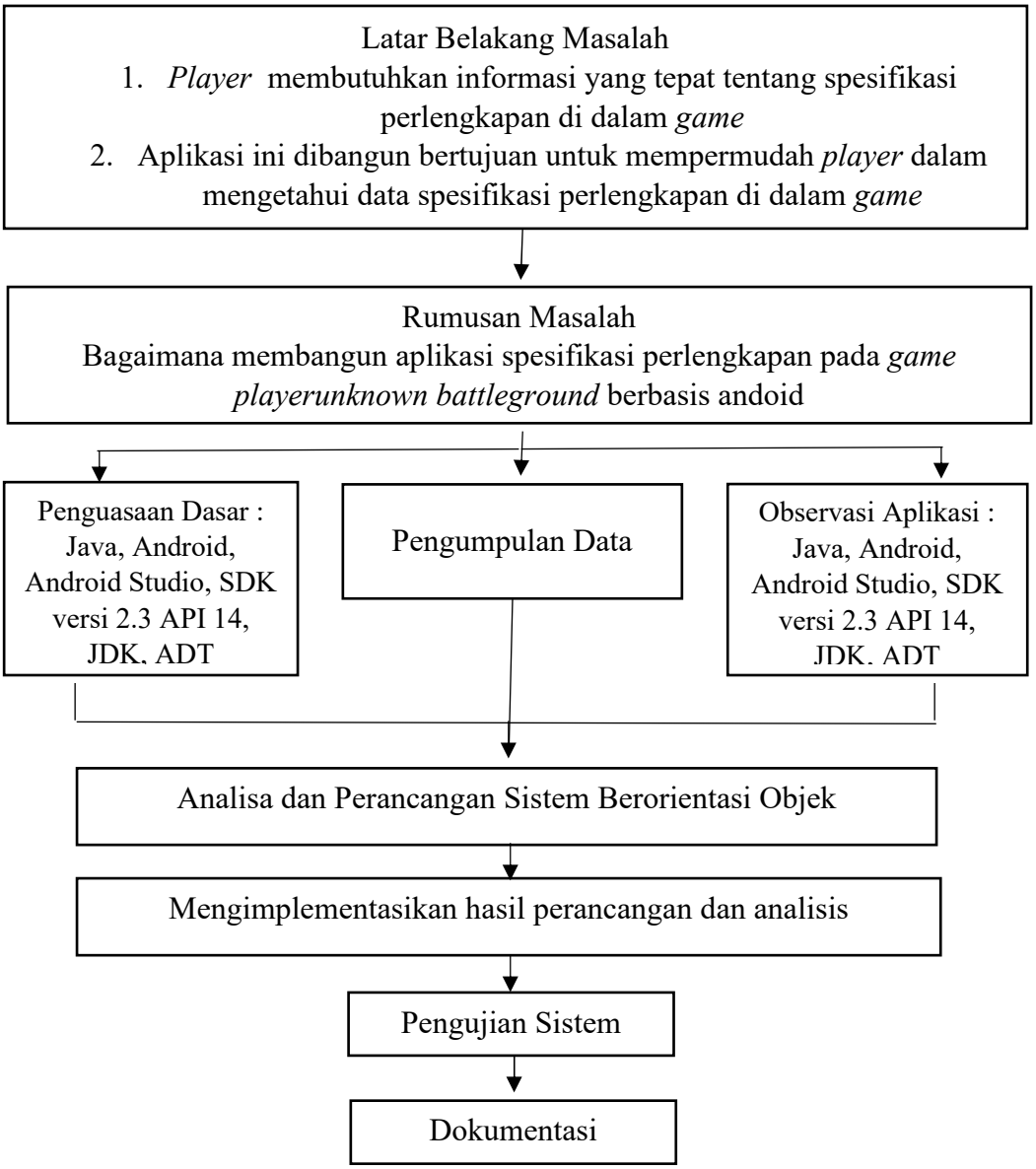
Membangun aplikasi spesifikasi *weapon equipment* dan *attachment* pada *game playerunknown battleground* sesuai dengan data-data yang didapatkan dari *game* tersebut.

8. Pengujian Sistem

Tahap ini sistem yang telah siap digunakan kemudian dilakukan pengujian untuk mengetahui jika ternyata masih ada kesalahan atau kekurangan pada sistem yang telah di buat.

9. Dokumentasi

Pada tahap ini akhir dimana sistem telah siap digunakan untuk mempermudah *user* atau pemain untuk mengetahui lebih dalam tentang informasi perlengkapan di dalam *game*



Gambar 2.1. Diagram Kerangka Pemikiran

2.3 Teori Pendukung

Penyusunan Tugas Akhir memerlukan suatu referensi pendukung yang digunakan sebagai landasan teori agar penelitian dapat berjalan dengan benar dan tidak meyimpang dari kaedah ilmu pengetahuan yang ada. Landasan teori diperoleh dari berbagai sumber dan literatur yang mempublikasikan pendapat beberapa

ilmuwan yang digunakan sebagai pendukung pembahasan masalah dalam penelitian Tugas Akhir. Berikut ini beberapa diantaranya:

2.3.1 Android

Android adalah sebuah sistem operasi untuk perangkat *mobile* yang menyertakan *middleware* (*virtual machine*) dan sejumlah aplikasi utama. Android merupakan modifikasi dari kernel Linux (Andry, 2011).

Pada awalnya sistem operasi ini dikembangkan oleh sebuah perusahaan bernama Android, Inc. Awal mula nama Android muncul, Android Inc. Adalah sebuah perusahaan *start-up* kecil yang berlokasi di Palo Alto, California, Amerika Serikat yang didirikan oleh Andy Rubin bersama Rich Miner, Nick Sears, dan Chris White. Pada bulan Juli 2005, perusahaan tersebut diakuisisi oleh Google dan para pendirinya bergabung ke Google. Andy Rubin sendiri kemudian diangkat menjadi Wakil Presiden divisi *Mobile* dari Google.

2.3.2 Perancangan Berorientasi Objek

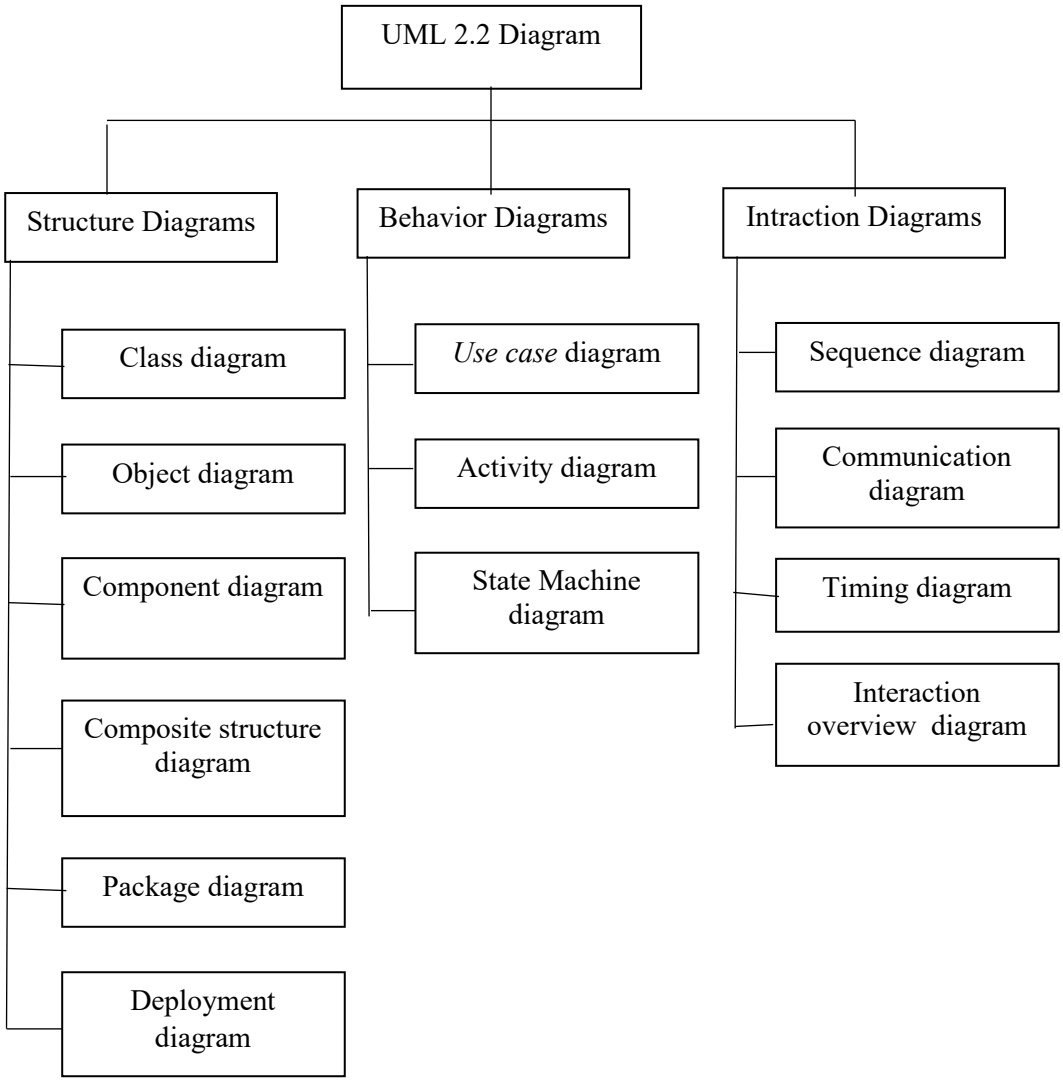
Teknologi objek menganalogikan sistem aplikasi seperti kehidupan nyata yang di dominasi oleh objek, di dalam membangun sistem berorientasi objek akan menjadi lebih baik apabila langkah awalnya didahului dengan proses analisis dan perancangan yang berorientasi objek. Tujuannya adalah mempermudah *programmer* di dalam mendesain program dalam bentuk objek-objek dan hubungan antar objek tersebut untuk kemudian dimodelkan dalam sistem nyata. Suatu perusahaan *software* yaitu Rational *Software*, telah membentuk konsarium dengan berbagai organisasi untuk meresmikan pemakaian *Unified Modelling Language* (UML) sebagai bahasa standar dalam *Object Oriented Analysist Design* (Nugroho, 2005).

2.3.3 UML (*Unified Model Language*)

Unified Modeling Language adalah standar internasional untuk *Object-Oriented notasi Analysis* dan *Design* (OOAD). *Unified Modeling Language* adalah spesialisasi bahasa yang dapat digunakan untuk *Object Modeling*, *Unified Modeling Language* dalam sebuah bahasa untuk menentukan visualisasi, konstruksi, dan mendokumentasikan *artifacts* dari sistem *software*, untuk memodelkan bisnis, dan

sistem *non-software* lainnya. *Unified Modeling Language* merupakan sistem arsitektur yang bekerja dengan satu bahasa yang konsisten untuk menentukan, visualisasi, konstruksi dan mendokumentasikan artifact yang terdapat dalam sistem (Nugroho, 2005).

Unified Modeling Language terdiri dari 13 macam diagram yang dikelompokkan dalam 3 kategori. Pembagian kategori dan macam-macam diagram dapat dilihat pada Gambar 2.2.

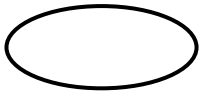
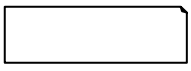


Gambar 2.2. Diagram UML

2.3.4 *Use Case* Diagram

Use Case atau diagram *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan di buat seperti Table 2.1. *Use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu (Rosa, 2016).

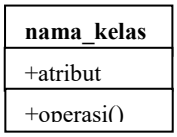
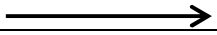
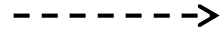
Tabel 2.1. Simbol-simbol pada Diagram *Use Case.*

No.	Simbol	Deskripsi
1.	 *Use case*	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan anta unit atau aktor, biasanya dinyatakan dengan menggunakan kata kerja di awal frase nama *Use case.*
2.	 Aktor/*Actor*	Orang, proses atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar 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 di awal frase nama aktor.
3.	*Association*	Komunikasi anatara aktor dan *Use case* yang berpartisispasi pada *Use case* atau *Use case* memiliki interaksi dengan aktor.
4.	*<<extend>>* 	Relasi *Use case* tambahan ke sebuah *Use case* dimana *Use case* yang ditambahkan dapat berdiri sendiri walau tanpa *Use case* tambahan itu, mirip dengan prinsip inheritance pada pemrograman berorientasi objek.
5.	 *<<include>>*	Relasi *Use case* tambahan ke sebuah *Use case* dimana *Use case* yang ditambahkan memerlukan *Use case* ini untuk menjalankan fungsina atau sebagai syarat dijalankan *Use case* ini.
6.	 *Generalization*	Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua buah *Use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya.
7	 *Note*	Memberikan informasi method apa yang ada didalamnya dan nama database.

2.3.5 *Class* Diagram

Diagram kelas atau *class* diagram pada Tabel 2.2 menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan di buat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas, sedangkan operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas (Rosa, 2016).

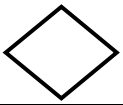
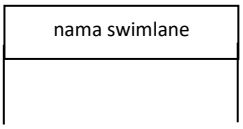
Tabel 2.2. Simbol-simbol pada *Class* Diagram

No.	Simbol	Deskripsi
1.	 Kelas	Kelas pada struktur sistem.
2.	 ***Interface***	Sama dengan konsep *interface* dalam pemrograman berorientasi objek.
3.	asosiasi / a*ssociation*	Relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplcity.*
4.	asosiasi berarah / *directed association* 	Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan *multiplicity.*
5.	generalisasi 	Relasi antar kelas dengan menggunakan makna generalisasi-spesialisasi (umum-khusus).
6.	kebergantungan / *dependency* 	Relasi antar kelas dengan makna kebergantungan antar kelas.
7.	agregasi / *aggregation* 	Relasi antar kelas dengan makna semua-bagian (*whole-part*)

2.3.6 *Activity* Diagram

Activity diagram menggambarkan rangkaian aliran aktifitas, digunakan untuk mendeskripsikan aktifitas yang di bentuk dalam suatu operasi, sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi (Rosa, 2016). Simbol-simbol yang digunakan pada *activity diagram* disajikan pada Tabel 2.3.

Tabel 2.3. Simbol-simbol *Activity* Diagram

No.	Simbol	Deskripsi
1.	status awal 	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal.
2.	aktivitas 	Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja.
3.	percabangan / *decision* 	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu.
4.	penggabungan / *join*	Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu.
5.	status akhir 	Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir.
6.	Swimlane 	Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi.

2.3.7 *Sequence* Diagram

Diagram *sequence* menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan di terima antar objek. Membuat diagram *sequence* juga dibutuhkan untuk melihat skenario yang ada pada *use case* (Rosa, 2016). *Sequence* diagram disajikan pada Tabel 2.4.

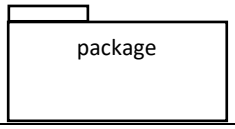
Tabel 2.4. Simbol-simbol *Sequence* Diagram

No.	Simbol	Deskripsi
1.	nama objek : nama kelas objek	Menyatakan objek yang berinteraksi pesan.
2.	garis hidup / *lifeline*	Menyatakan kehidupan suatu objek.
3.	waktu aktif	Menyatakan objek dalam keadaan aktif dan berinteraksi.
4.	pesan tipe create <<create>>	Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat.
5.	pesan tipe call 1 : nama_metode()	Menyatakan suatu objek memanggil operasi/metode yang ada pada objek lain atau dirinya sendiri.
6.	pesan tipe send 1 : masukan	Menyatakan suatu objek mengirimkan data/masukan/informasi ke objek lainnya, arah panah mengarah pada objek yang dikirimi.
7.	pesan tipe return 1 : keluaran	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian.

2.3.8 *Component* Diagram

Diagram komponen atau *component* diagram pada Tabel 2.5 di buat untuk menunjukkan organisasi dan ketergantungan di antara kumpulan komponen dalam sebuah sistem. Diagram komponen fokus pada komponen sistem yang dibutuhkan dan ada di dalam sistem. Komponen dasar yang biasanya ada dalam suatu sistem adalah komponen *user interface* yang menangani tampilan, komponen *bussiness processing* yang menangani fungsi-fungsi proses bisnis, komponen data yang menangani manipulasi data, komponen *security* yang menangani keamanan sistem (Rosa, 2016).

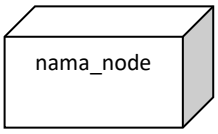
Tabel 2.5. Simbol-simbol *Component* Diagram

No.	Simbol	Deskripsi
1.	*Package* 	*Package* merupakan sebuah bungkusan dari satu atau lebih komponen.
2.	komponen 	Komponen sistem.
3.	kebergantungan / *dependency* 	Kebergantungan antar komponen, arah panah mengarah pada komponen yang dipakai.
4.	antar muka / *interface* 	Sama dengan konsep interface pada pemrograman berorientasi objek, yaitu sebagai antarmuka komponen agar tidak mengakses langsung komponen.
5.	*link*	Relasi antar komponen.

2.3.9 *Deployment* Diagram

Deployment diagram menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi yang disajikan pada Tabel 2.6. *Deployment* diagram juga dapat digunakan untuk memodelkan hal-hal berikut. Sistem tambahan atau *embedded system* yang menggambarkan rancangan *device, node, dan hardware*. sistem *clientserver*, atau sistem terdistribusi murni rekayasa ulang aplikasi (Rosa, 2016).

Tabel 2.6. Simbol-simbol *Deployment* Diagram

No.	Simbol	Deskripsi
1.	 *Constraint*	*Constraint* adalah mekanisme perpanjangan yang memungkinkan untuk menyempurnakan semantik elemen model UML.
2.	 *Node*	Biasanya mengacu pada perangkat keras (*hardware*), perangkat lunak yang tidak dibuat sendiri (*software*), jika di dalam *node* disertakan komponen untuk mengkonsistenkan rancangan maka komponen yang diikutsertakan harus sesuai dengan komponen yang telah didefinisikan sebelumnya pada diagram komponen.
3.	kebergantungan / *dependency*	Kebergantungan antar *node*, arah panah mengarah pada *node* yang dipakai.
4.	*link*	Relasi antar *node.*

2.4 JAVA

Java merupakan bahasa berorientasi objek (OOP) yaitu cara ampuh dalam pengorganisasian dan pengembangan perangkat lunak. Pada (OOP), program komputer sebagai kelompok objek yang saling berinteraksi. Deskripsi ringkas (OOP) adalah mengorganisasikan program sebagai kumpulan komponen, disebut objek. Objek-objek ini ada secara independen, mempunyai aturan-aturan berkomunikasi dengan objek lain dan untuk memerintahkan objek lain guna meminta informasi tertentu atau meminta objek lain mengerjakan sesuatu. *Java* adalah nama sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer yang berdiri sendiri (standalone) ataupun pada lingkungan jaringan (Shalahuddin, 2010).

Java termasuk bahasa *Multithreading*. *Thread* adalah untuk menyatakan program komputer melakukan lebih dari satu tugas di satu waktu yang sama. *Java* menyediakan kelas untuk menulis program *multithreaded*, program mempunyai lebih dari satu *thread* eksekusi pada saat yang sama sehingga memungkinkan program menangani beberapa tugas secara efisien.

Program *Java* melakukan *garbage collection* yang berarti program tidak perlu menghapus sendiri objek-objek yang tidak digunakan lagi. Fasilitas ini mengurangi beban pengelolaan memori oleh pemrogram dan mengurangi atau mengeliminasi sumber kesalahan terbesar yang terdapat di bahasa yang memungkinkan alokasi dinamis.

2.5 Pengertian Jaringan

Jaringan komputer adalah suatu sistem yang terdiri atas sebuah komputer-komputer yang di desain untuk bisa berbagi sumber daya (*printer*, CPU), berkomunikasi (surel, pesan instan), dan bisa mengakses informasi (peramban *web*). jaringan komputer ialah suatu sistem yang terdiri atas sebuah komputer-komputer yang di desain untuk bisa berbagi sumber daya (*printer*, CPU), berkomunikasi (surel, pesan instan), dan bisa mengakses informasi (Yudianto, 2007).

a. *Access* ke informasi yang berada di tempat yang jauh. Salah satu bidang *access* informasi jarak jauh yang sudah ada adalah *access* ke institusi keuangan. Contohnya membayar tagihan, mengelola rekening bank secara elekronis.
b. Komunikasi orang ke orang. Bidang komunikasi orang ke orang yang dapat dilakukan saat ini yaitu surat kabar *online*, *real time email* dan *video conference* yang memungkinkan hubungan orang ke orang dapat dilakukan melalui jarak jauh.

2.6 Perangkat Lunak Pendukung

2.6.1 Android Studio

Android Studio adalah lingkungan pengembangan terpadu *Integrated Development Environment* (IDE) untuk pengembangan aplikasi android, berdasarkan *IntelliJ IDEA* (Andry, 2011). Selain merupakan editor kode *IntelliJ* dan alat pengembang yang berdaya guna, Android Studio menawarkan fitur lebih banyak untuk meningkatkan produktivitas anda saat membuat aplikasi android, misalnya:

a. Sistem versi berbasis *Gradle* yang fleksibel.
b. Emulator yang cepat dan kaya fitur.
c. Lingkungan yang menyatu untuk pengembangan bagi semua perangkat Android.
d. Instant *Run* untuk mendorong perubahan ke aplikasi yang berjalan tanpa membuat APK baru.
e. *Template* kode dan integrasi *GitHub* untuk membuat fitur aplikasi yang sama dan mengimpor kode contoh.
f. Alat pengujian dan kerangka kerja yang ekstensif.
g. Alat Lint untuk meningkatkan kinerja, kegunaan, kompatibilitas versi, dan masalah-masalah lain.
h. Dukungan *C++* dan *NDK*.
i. Dukungan bawaan untuk *Google Cloud Platform*, mempermudah pengintegrasian *Google Cloud Messaging* dan *App Engine.*

2.6.2 Struktur *Project*

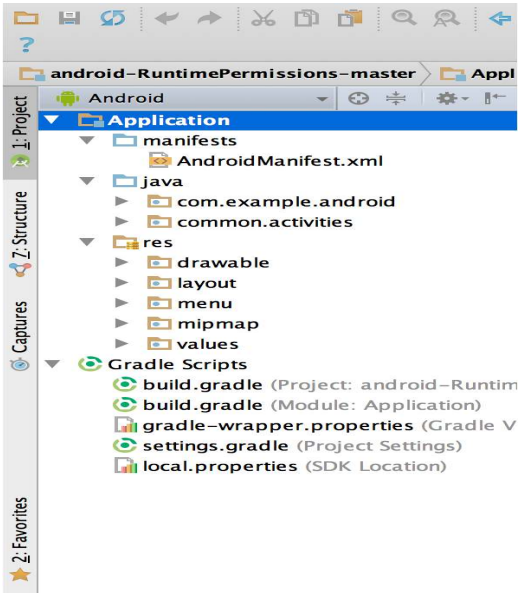
Setiap proyek di Android Studio berisi satu atau beberapa modul dengan *file* kode sumber dan *file* sumber daya. Jenis-jenis modul mencakup:

a. Modul aplikasi Android.
b. Modul Pustaka.
c. Modul Google *App Engine*.

Secara *default*, Android Studio akan menampilkan *file* proyek dalam tampilan proyek Android, seperti yang ditampilkan dalam gambar 2.3. Tampilan disusun berdasarkan modul untuk memberikan akses cepat ke *file* sumber utama proyek. Semua *file* versi terlihat di bagian atas, di bawah *Gradle Scripts* dan masing-masing modul aplikasi berisi *folder* berikut:

a. *manifests*: Berisi *file* AndroidManifest.xml.
b. *java*: Berisi *file* kode sumber *Java*, termasuk kode pengujian JUnit.
c. *res*: Berisi semua sumber daya bukan kode, seperti tata letak XML, string UI, dan gambar bitmap.

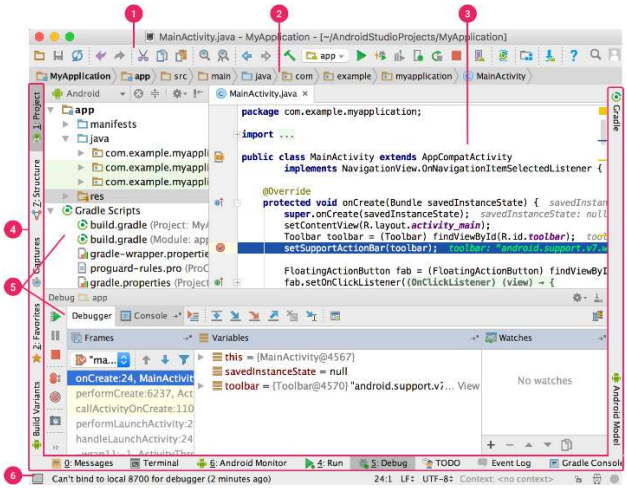
Struktur proyek Android pada *disk* berbeda dari representasi rata ini, untuk melihat struktur *file* sebenarnya dari proyek ini, pilih *project* dari *menu* tarik turun *project*.



Gambar 2.3. *File Project* di tampilan Android Studio.

2.6.3 Antarmuka Pengguna

Jendela utama android studio terdiri dari beberapa bidang logika yang didefinisikan pada Gambar 2.4.



Gamber 2.4. Jendela Utama Android Studio

1. Bilah alat memungkinkan untuk melakukan berbagai jenis tindakan, termasuk menjalankan aplikasi dan meluncurkan alat Android.
2. Bilah navigasi membantu bernavigasi di antara proyek dan membuka *file* untuk di edit. Bilah ini memberikan tampilan struktur yang terlihat lebih ringkas dalam jendela *project*.
3. Jendela *editor* adalah tempat membuat dan memodifikasi kode. Bergantung pada jenis *file* saat ini, *editor* dapat berubah. Misalnya, ketika melihat *file* tata letak, *editor* menampilkan *Layout Editor*.
4. Bilah jendela alat muncul di luar jendela IDE dan berisi tombol yang memungkinkan meluaskan atau menciutkan jendela alat individual.
5. Jendela alat memberiakses ke tugas tertentu seperti pengelolaan proyek, penelusuran, kontrol versi, dan banyak lagi bisa meluaskan dan juga menciutkannya.
6. Bilah status menampilkan status proyek dan IDE itu sendiri, serta setiap peringatan atau pesan.

2.6.4 *Software Develpment Kit* (SDK)

Software Development Kit (SDK) adalah *tools* API (*Application Programming Interface*) yang diperlukan untuk mulai mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman *Java*. Android merupakan *subset* perangkat lunak untuk ponsel yang meliputi sistem operasi, *middleware* dan aplikasi kunci yang di *release* oleh Google. Saat ini disediakan Android SDK (*Software Development Kit*) sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman *Java*. Sebagai platform aplikasi netral, Android memberikan anda kesempatan untuk membuat aplikasi yang kita butuhkan yang bukan merupakan aplikasi bawaan *Handphone* atau *Smartphone* (Nazarudin, 2011).

2.6.5 *Java Develpment Kit* (JDK)

Java Development Kit (JDK) adalah sekumpulan perangkat lunak yang dapat digunakan untuk mengembangkan perangkat lunak yang berbasis *Java*, Sedangkan JRE adalah sebuah implementasi dari *Java Virtual Machine* yang benar-benar digunakan untuk menjalankan program *java*, biasanya setiap JDK berisi satu atau lebih JRE dan berbagai alat pengembangan lain seperti sumber kompiler *java*, *bundling*, *debuggers*, *development libraries* dan lain sebagainya. Perbedaan JDK dengan SDK (*Software Development Kit*) yaitu JDK adalah sebuah SDK tetapi sebuah SDK tidak harus menjadi sebuah JDK (Nazarudin, 2011)

2.6.6 *Android Development Tools (ADT)*

Android Development Tools adalah *plugin* yang di desain untuk IDE Android Studio yang memberikan kemudahan dalam menggembangkan aplikasi Android. Dengan adanya ADT untuk *eclipse* akan memudahkan *developer* dalam membuat aplikasi *project* Android, membuat aplikasi GUI, dan menambahkan komponen-komponen yang lainnya, begitu juga dapat melakukan *running* aplikasi menggunakan Android SDK melalui Android Studio. Dengan ADT juga dapat membuat *package* Android (.apk) yang digunakan untuk distribusi aplikasi Android yang di rancang (Nazarudin, 2011).

2.7 Pengujian Sistem

Pengujian perangkat lunak ini menggunakan metode pengujian *free view*. Pengujian *free view* berfokus pada penilaian deskriptif.

Software Testing and QA Theory and Practice (*Chapter 17: Software Quality*) *University of Waterloo* (Thipathy, 2008), pengujian *free view* adalah pengujian yang sifatnya deskriptif di mana *software* yang di uji melalui lima sudut pandang atau kategori yang berbeda melalui penilaian dari *expertise* atau orang yang berpengalaman dari masing-masing sudut pandang. Lima sudut pandang tersebut adalah sebagai berikut:

2.7.1 *Transcendental View*

Kualitas menurut pandangan ini alah sesuatu yang dapat dikenali melalui pengalaman tapi tidak dapat selalu digambarkan. Objek atau *software* yang bagus itu menonjol dan dapat dengan mudah dikenali.

2.7.2 *User View*

Kualitas menyangkut sejauh mana produk memenuhi kebutuhan dan harapan pengguna dan apakah produk cocok untuk digunakan. Pendapat ini bersifat sangat *personal.* Sebuah produk berkualitas baik jika memukakan sebagian besar pengguna. Hal ini berguna untuk mengidentifikasikan fitur dari produk yang pengguna anggap penting. Pandangan ini dapat mencakup banyak unsur subjek, seperti kegunaan, keandalan dan keefisienan.

2.7.3 *Manufacturing View*

Pandangan ini berkaitan dengan faktor dalam industri manufaktur, apakah produk memenuhi persyaratan atau tidak. Setiap penyimpangan dari persyaratan yang di nilai mengurangi kualitas produk. Konsep proses memainkan peran kunci. Produk yang di buat harus orisinil sehingga biaya berkurang, misal biaya pembangunan dan biaya pemeliharaan.

Kesesuaian dengan persyaratan dan spesifikasi menyebabkan keseragaman dalam produk tapi beberapa berpendapat bahwa keseragaman tersebut tidak

menjamin kualitas. Kualitas produk dapat secara bertahap ditingkatkan dengan memperbaiki proses.

2.7.4 *Product View*

Jika sebuah produk diproduksi dengan sifat internal yang baik, maka produk akan memiliki sifat *external* atau *ouput* yang baik dan dapat di eksplorasi hubungan antara sifat *internal* dan kualitas *eksternal*.

2.7.5 *Value-based View*

Valuq-based View merupakan penggabungan dari dua konsep yaitu keunggulan dan kelayakan. Kualitas adalah ukuran dari keunggulan dan nilai adalah ukuran layak.

Berapa banyak pengguna bersedia membayar untuk tingkat kualitas tertentu. Kualitas tidak berarti, jika produk memenuhi nilai ekonimi. Pandangan berbasis nilai antara biaya dan kualitas