BAB II

LANDASAN TEORI

2.1 Pembelajaran

Pembelajaran adalah adalah proses interaksi peserta didik dengan pendidik dan sumber belajar pada suatu lingkungan belajar. Dengan kata lain, pembelajaran adalah proses untuk membantu peserta didik agar dapat belajar dengan baik. Dalam *konteks* pendidikan, guru mengajar agar peserta didik dapat belajar dan menguasai isi pelajaran hingga mencapai sesuatu *objektif* yang ditentukan (aspek *kognitif*), juga dapat memengaruhi perubahan sikap (aspek afektif), serta keterampilan (aspek *psikomotor*) seorang peserta didik, namun proses pengajaran ini memberi kesan hanya sebagai pekerjaan satu pihak, yaitu pekerjaan pengajar saja.

2.2 Game Edukasi

2.2.1 Pengertian Game

Dalam kamus bahasa Indonesia "*Game*"diartikan sebagai permainan. Permainan adalah kegiatan yang *kompleks* yang didalamnya terdapat peraturan, play dan budaya. Sebuah permainan adalah sebuah *sistem* dimana pemain terlibat dalam konflik buatan, disini pemain berinteraksi dengan sistem dan konflik dalam permainan merupakan rekayasa atau buatan, dalam permainan terdapat peraturan yang bertujuan untuk membatasi perilaku pemain dan menentukan permainan.

Menurut Kurniawan Teguh Martono (2011 : 49) *Game* merupakan salah satu media yang digunakan untuk menyampaikan suatu pesan kepada orang umum dalam bentuk permaian yang dapat menghibur.

2.2.2 Pengertian Edukasi

Edukasi adalah proses yang dilakukan oleh seseoarang untuk menemukan jati dirinya, yang dilakukan dengan mengamati dan belajar yang kemudian melahirkan tindakan dan prilaku

2.2.3 Pengertian Game Edukasi

Education game adalah *game* yang khusus dirancang untuk mengajarkan *user* suatu pembelajaran tertentu, pengembangan konsep dan pemahaman dan membimbing mereka dalam melatih kemampuan mereka, serta memotivasi mereka untuk memainkannya, Hurd dan Jenuings (2009:6).

Menurut Kurniawan Teguh Martono (2011 : 50) *Game* edukasi adalah memadukan antara sisi belajar dan bermain.

Menurut Priyanto Hidayatullah, Aldi Daswanto dan Sulistyo Ponco Nugroho (2011) *Game* edukasi adalah jenis *game* yang baik jika dilibatkan dalam proses pendidikan.

2.3 Pengertian Mobile

Mobile dapat diartikan sebagai perpindahan yang mudah dari satu tempat ke tempat yang lain, misalnya telepon mobile berarti bahwa terminal telepon yang dapat berpindah dengan mudah dari satu tempat ke tempat lain tanpa terjadi pemutusan atau terputusnya komunikasi.

2.3.1 Karakteristik Perangkat Mobile

Perangkat mobile memiliki banyak jenis dalam hal ukuran, desain dan layout, tetapi mereka memiliki kesamaan karakteristik yang sangat berbeda dari sistem desktop, yaitu antara lain :

a) Ukuran yang kecil

Perangkat mobile memiliki ukuran yang kecil. Konsumen menginginkan perangkat yang terkecil untuk kenyamanan dan mobilitas mereka.

b) Memory yang terbatas

Perangkat mobile juga memiliki memory yang kecil, yaitu primary (RAM) dan secondary (disk). Pembatasan ini adalah salah satu faktor yang mempengaruhi penulisan program untuk berbagai jenis dari perangkat ini. Dengan pembatasan jumlah dari memory, pertimbangan-pertimbangan khusus harus diambil untuk memelihara pemakaian dari sumber daya yang mahal ini.

c) Daya proses yang terbatas

Sistem mobile tidaklah setangguh rekan mereka yaitu desktop. Ukuran, teknologi dan biaya adalah beberapa faktor yang mempengaruhi status dari sumber daya ini. Sepertiharddisk dan RAM, Anda dapat menemukan mereka dalam ukuran yang pas dengan sebuah kemasan kecil.

d) Mengkonsumsi daya yang rendah

Perangkat mobile menghabiskan sedikit daya dibandingkan dengan mesin desktop.

Perangkat ini harus menghemat daya karena mereka berjalan pada keadaan dimana daya yang disediakan dibatasi oleh baterai-baterai.

e) Kuat dan dapat diandalkan

Karena perangkat mobile selalu dibawa kemana saja, mereka harus cukup kuat untuk menghadapi benturan-benturan, gerakan, dan sesekali tetesan-tetesan air.

f) Konektivitas yang terbatas

Perangkat mobile memiliki bandwith rendah, beberapa dari mereka bahkan tidak tersambung. Kebanyakan dari mereka menggunakan koneksi wireless.

g) Masa hidup yang pendek

Perangkat-perangkat konsumen ini menyala dalam hitungan detik kebanyakan dari mereka selalu menyala. Coba ambil kasus sebuah handphone, mereka booting dalam hitungan detik dan kebanyakan orang tidak mematikan handphone mereka bahkan ketika malam hari. PDA akan menyala jika anda menekan tombol power mereka.

2.4 Perancangan Dengan Pendekatan Berorientasi Objek

Analisis dan Perancangan perangkat lunak dengan pendekatan objek atau dikenal dengan pendekatan berorientasi objek *(Object Oriented Approach)* yaitu pendekatan untuk pengembangan perangkat lunak yang menitik beratkan permasalahan pada abstraksi objek-objek yang ada di dunia nyata.

2.4.1 Karakteristik Pendekatan Berorientasi Objek

Karakteristik pendekatan objek adalah sebagai berikut:

1. Pendekatan lebih pada data dan bukanya pada prosedur/fungsi.

2. Program besar dibagi pada sesuatu yang disebut objek-objek
3. Struktur data dirancang dan menjadi karakteristik dari objek-objek.
4. Fungsi-fungsi yang mengoperasikan data tergabung dalam suatu objek yang sama.
5. Data tersembunyi dan terlindung dari fungsi/prosedur yang ada diluar
6. Objek-objek dapat saling berkomunikasi dengan saling mengirim message (pesan) satu sama lain.
7. Pendekatan yang digunakan yaitu pendekatan bawah ke atas (*bottom up approach*)

2.4.2 Konsep Pendekatan Berorientasi Objek

Konsep pendekatan beriorientasi objek adalah sebagai berikut :

1. Object / Objek

Objek adalah orang, tempat, benda, kejadian atau konsep-konsep yang ada di dunia nyata dan penting bagi suatu *aplikasi*. Sebuah objek adalah Entitas yang memiliki Identitas, *State* (keadaan sesaat) dan Behavior (perilaku). State sebuah objek adalah kondisi objek tersebut yang dinyatakan dalam *Atribut atau property*. *Behavior* sebuah objek mendefinisikan bagaimana sebuah objek bertindak/bereaksi yang dinyatakan dalam *Operation.* Satu object dapat diturunkan menjadi object dalam bentuk lain, kemudian saling mengkait menyusun sesuatu yang lebih rumit.

2. Class / Kelas

Class adalah kumpulan atau himpunan objek-objek yang sejenis, memiliki kesamaan *atribut/property*, perilaku, serta relasi dengan objek lain yang

mirip. Notasi kelas digambarkan dengan kotak, dengan nama kelas didalamnya ditulis menggunakan huruf besar di awal kata. Bila sebuah kelas memiliki 2 suku kata atau lebih, maka penulisannya disatukan tanpa spasi dengan huruf awal tiap suku menggunakan huruf besar.Contohnya adalah Barang Elektronik dapat dikatakan sebagai sebuah Kelas apabila memiliki kesamaan dengan objek yang ada padanya misalnya Mesin Cuci, Televisi, Radio, Kulkas adalah objek-objek yang dapat dikelompokkan kedalam satu kelas yaitu Barang Elektronik rumah tangga.

3. Attribute / Atribut

Attribute adalah data yang dimiliki suatu objek atau *property* dari sebuah *Class* yang menggambarkan batas nilai yang mungkin ada pada obyek dari kelas. Sebuah kelas bisa memiliki nol atau lebih atribut. Notasi atribut digambarkan dengan kotak dibawah kotak *class*, dengan nama atribut didalamnya ditulis menggunakan huruf kecil. Jika sebuah atribut memiliki 2 atau lebih suku kata, maka semua suku kata ditulis disatukan tanpa spasi, awal suku kata pertama dengan huruf kecil dan awal suku kata berikutnya dengan huruf besar. Notasi atribut dapat ditambahkan informasi dengan tipe-tipe atribut tersebut. Penulisan tipe pada atribut dipisahkan dengan tanda titik dua (:), tipe yang ditambahkan berupa *String, Floating-Point number, Integer dan Boolean.*

4. Operation / Operasi

Operation adalah sesuatu yang bisa dilakukan oleh sebuah *class*. Notasi penulisannya sama dengan atribut. Bagian operation ini juga bisa diberikan

tambahan informasi, yaitu dengan menambahkan parameter yang akan dilakukan operation dalam tanda kurung. Contoh parameternya adalah *function*.

5. Inheritance / Pewarisan

Inheritance atau pewarisan memungkinkan dibuat *class* yang menyerupai *class* lain yang telah ada sebelumnya, tetapi masih memiliki beberapa sifat induknya. Misalkan dari sebuah mobil biasa, anda dapat membuat mobil balap serta mobil angkutan umum. Prosesnya adalah dengan mengubah sifat dari mobil biasa tersebut.

6. Polymorphisme / Kebanyakrupaan

Polymorphism adalah object yang memiliki fungsi sama dengan object dasar tetapi memiliki satu atau lebih sifat berbeda atau dengan kata lain *Polymorphisme* adalah pemisahan secara jelas diantara *subsistem* yang berbeda. Sebagai contoh misalkan sebuah kelas memiliki operasi '*OPEN*', operasi open ini bisa dipakai untuk membuka pintu, membuka buku, membuka baju dan lainnya. Meskipun '*OPEN*' memiliki tujuan yang sama, tapi apa yang dilakukannya berbeda.

7. Encapsulation / Pembungkusan

Encapsulation sering disebut dengan penyembunyian informasi (*Hidding*), suatu konsep berdasarkan fakta di dunia nyata yang menyatakan bahwa segala sesuatu tidak perlu diperlihatkan. Misalnya kita tidak perlu tahu apa yang dilakukan sistem ketika kita menekan remote untuk menghidupkan televisi.

8. Responsibilities / Tanggung Jawab

Responsibilities adalah model tambahan yang digambarkan pada bagian bawah suatu kelas setelah bagian operasi digunakan untuk menjelaskan pernyataan-pernyataan mengenai apa-apa yang bisa dilakukan oleh kelas tersebut.

2.5 UML

2.5.1 Pengertian UML

UML adalah bahasa untuk menspesifikasi, memvisualisasi, membangun dan mendokumentasikan *artifacts* (bagian dari informasi yang digunakan atau dihasilkan oleh proses pembuatan perangkat lunak, *artifact* tersebut dapat berupa model, deskripsi atau perangkat lunak) dari sistem perangkat lunak, seperti pada pemodelan bisnis dan sistem non perangkat lunak lainnya [HAN98]. Selain itu *UML* adalah bahasa pemodelan yang menggunakan konsep orientasi *object*. *UML* dibuat oleh Grady Booch, James Rumbaugh, dan Ivar Jacobson di bawah bendera Rational Software Corp [HAN98]. *UML* menyediakan notasi-notasi yang membantu memodelkan sistem dari berbagai perspektif. *UML* tidak hanya digunakan dalam pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan.

2.5.2 Gambaran Umum UML

Gambaran umum mengenai *UML* dapat dijelaskan berdasarkan kegunanaan dari UML itu sendiri, yaitu :

1. *Modeling Language*, *UML* sebagai bahasa untuk pemodelan sistem

UML merupakan bahasa pemodelan yang memiliki pembendaharaan kata dan cara untuk mempresentasikan secara fokus pada konseptual dan fisik dari suatu sistem. Dengan UML akan memberitahukan kita bagaimana untuk membuat dan membaca bentuk model yang baik, tetapi UML tidak dapat memberitahukan model apa yang akan dibangun dan kapan akan membangun model tersebut.

2. *Visualizing, UML* sebagai bahasa untuk menggambarkan sistem

   UML tidak hanya merupakan rangkaian *simbol* grafikal, cukup dengan simbol pada notasi *UML* merupakan penetapan semantik yang baik. Dengan cara ini, satu pengembang dapat menulis model *UML* dan pengembang lain atau perangkat yang sama lainnya dapat mengartikan bahwa model tersebut tidak ambigu. Hal ini akan mengurangi eror yang terjadi karena perbedaan bahasa dalam komunikasi model konseptual dengan model lainnya.

3. *Specifying, UML* sebagai dasar bahasa untuk menspesifikasikan sistem

   Maksudnya membangun model yang sesuai, tidak ambigu dan lengkap. Pada faktanya UML menunjukan semua spesifikasi keputusan analisis, desain dan implementasi yang penting yang harus dibuat pada saat pengembangan dan penyebaran dari sistem software insentif.

4. *Constructing, UML* sebagai bahasa untuk membangun sistem

   UML bukan bahasa, pemograman *visual*, tetapi model *UML* dapat dikoneksikan secara langsung pada bahasa pemprograman visual. Maksudnya membangun model yang dapat dimapping ke bahasa pemprograman seperti

java, C++ atau tabel pada *database relation* atau penyimpanan tetap pada *database* berorientasi objek.

5. *Documenting, UML* sebagai bahasa untuk pendokumentasian sistem

   UML menunjukan dokumentasi dari arsitektur sistem dan detail dari semuanya. *UML* hanya memberikan bahasa untuk memperlihatkan permintaan dan untuk tes. *UML* menyediakan bahasa untuk memodelkan aktifitas dari perencanaan *project dan manajemen pelepasan (release management).*

6. Area dan tujuan penggunaan UML

   UML (Unifed Modeling Language) digunakan paling efektif pada domain seperti:

   a. Sistem Informasi Perusahaan
   b. Sistem Perbankan dan perekonomian
   c. Bidang Telekomunikasi
   d. Bidang Transportasi
   e. Bidang Penerbangan
   f. Bidang Perdagangan
   g. Bidang Pelayanan Elektronik
   h. Bidang pengetahuan
   i. Bidang Pelayanan Berbasis *Web* Terdistribusi

UML tidak terbatas untuk pemodelan software saja. Pada faktanya *UML* banyak digunakan untuk memodelkan sistem *non-software* sperti :

a. Aliran kerja pada sistem perundangan
b. Struktur dan kelakuan dari Sistem Kepedulian Kesehatan Pasien

c. Desain hardware, dll.

Tujuan penggunaan *UML* adalah sebagai berikut :

a. Memodelkan suatu sistem (bukan hanya perangkat lunak) yang menggunakan konsep berorientasi objek.
b. Menciptakan suatu bahasa pemodelan yang dapat digunakan baik oleh manusia maupun mesin.

Keunggulan menggunakan UML dibandingkan menggunakan metodologi terstruktur:

a. *Uniformity*

Pengembangan cukup menggunakan 1 metodologi dari tahap analasis hingga perancangan. Memungkinkan merancanga komponen antarmuka secara terintegrasi bersama perancangan PL dan perancangan terstruktur data.

b. *Understandability*

Kode yang dihasilkan dapat diorganisasikan kedalam kelas-kelas yang berhubungan dengan masalah sesungguhnya sehingga lebih mudah untuk dipahami.

c. *Stability*

Kode program yang dihasilkan *relative* stabil sepanjang waktu, karena mendekati permasalahan yang sesungguhnya.

d. *Reusability*

Dengan metodologi berorientasi objek, dimungkinkan penggunaan ulang kode, sehingga pada akhirnya akan sangat mempercepat waktu pengembangan perangkat lunak (atau sistem informasi).

7. Bangunan Dasar *UML*

Untuk memahami *UML* diperlukan pemahaman konseptual. Metodologi *UML* menggunakan 3 bangunan dasar untuk mendeskripsikan sistem.perangkat lunak yang akan dikembangkan, yaitu:

1. *Things* (sesuatu), ada 4 macam *Things* dalam *UML* yaitu:
   a. Structural Things

      Merupakan bagian yang *relatif statis* dalam model *UML*. Bagian yang *relatf statis* dapat berupa elemen-elemen yang besifat fisik maupun konseptual. Secara keseluruhan ada 7 macam *Structural Things*:

      1) *Classes* / Kelas adalah himpunan dari objek-objek yang berbagi atribut serta operasi yang sama. Kelas mengimplementasikan satu atau lebih antarmuka. Secara grafis, kelas digambarkan dengan empat persegi panjang yang memuat nama, atribut serta operasi yang dimilikinya.
      2) *Interfaces* / Antarmuka adalah kumpulan dari operasi-operasi yang menspesifikasikan layanan/servis suatu kelas atau komponen/objek. Antarmuka ini mendeskripsikan perilaku yang tampak dari luar suatu elemen. Secara grafis, antarmuka digambarkan dengan lingkaran kecil dan nama yang didahului dengan garis tegak.
      3) *Collaborations* / Kolaborasi adalah sesuatu yang mendefinisikan interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dan sinergis. Secara grafis, kolaborasi digambarkan elips bergaris putus-putus yang memuat nama kolaborasi itu.

4) *Use-Cases* adalah deskripsi dari urutan aksi-aksi yang ditampilkan sistem dengan hasil yang terukur bagi suatu Actor. Use-Case digunakan untuk menstrukturkan perilaku pada suatu model. Secara grafis use case digambarkan dengan elips tegas yang berisi namanya.
5) *Activity Class* / Kelas Aktif adalah kelas (biasa) dimana objek-objek yang dimilikinya memiliki 1 atau lebih proses dan lebih jauh menginisialisasi suatuu aktifitas kendali. Secara grafis, kelas aktif digambarkan sebagai kelas biasa tapi memiliki batas yang lebih tebal, yang memuat nama, atribut serta operasi yang dimilikinya.
6) *Components* / Komponen adalah bagian fisik dan bagian yang dapat digantikan dalam suatu sistem, misalnya berkas *ActiveX*, *COM*, DLL yang keberadaanya dibutuhkan oleh sistem yang akan dikembangkan. Komponen ini merepresentasikan konsep-konsep *reusable component*. Secara grafis, komponen digambarkan dengan empat persegi panjang seperti kelas tetapi ditambahkan *Tab*.
7) *Nodes* / Simpul adalah elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi; secara umum menggunakan kapasitas memori dan kemampuan pemrosesan. Secara grafis, simpul digambarkan sebagai kubus yang berisi namanya.

b. *Behavioral Things*

Merupakan bagian yang dinamis pada model UML, biasanya merupakan kata kerja dari model UML yang mencerminkan perilaku sepanjang ruang dan waktu. Ada 2 macam Behavior Things, yaitu:

1) *Interaction*s / interaksi adalah suatu perilaku yang mencakup himpunan pesan-pesan (*message*) atau kumpulan objek & operasinya yang diperlukan untuk menyelesaikan suatu fungsi tertentu. Sebuah interaksi terdiri dari beberapa unsur, yaitu: Pesan, Perilaku dan *Link*, Secara grafis, pesan digambarkan dengan tanda panah tegas dan memuat nama operasinya.
2) *State Machine*, adalah perilaku yang menspesifikasikan urutan kedudukan suatu objek atau interaksi-interaksi sepanjang waktu untuk menanggapi *event-event* yang terjadi. Penggambaran *state* memiliki beberapa unsur, yaitu: *State* itu sendiri, Transisi (perubahan dari suatu state ke state lain), *Event* (suatu keadaan yang memicu sebuah transisi), Aktifitas (tanggapan terhadap transisi). Secara grafis *State* digambarkan sebagai empat persegi panjang dengan sudut tumpul, yang memuat namanya serta subsistem didalamnya jika ada.

c. *Grouping Things*

Merupakan bagian pengorganisasian (*Packages*) dalam UML. Digunakan untuk menggambarkan paket-paket untuk menyederhanakan model-model UML yang rumit. Paket-paket ini kemudian dapat di dekomposisi. Paket berguna untuk pengelompokkan sesuatu misalnya model-model serta subsistem-subsistem.

d. *Annotational Things*

Merupakan bagian yang memperjelas model UML (*Notes*), seperti komentar-komentar yang menjelaskan fungsi secara rinci serta ciri-ciri tiap elemen dalam model UML.

2. *Relationship* (hubungan)

Relationship adalah hubungan-hubungan yang terjadi antara elemen dalam UML. Hubungan-hubungan ini Penting sekali dalam UML. Model-model UML harus dibuat menggunakan relationship. Ada 4 macam relationship dalam UML yang dapat digunakan untuk menggambarkan model-model UML yang representative:

a. *Dependency* (kebergantungan)

Dependency adalah hubungan dimana perubahan yang terjadi dalam suatu eleman independent (mandiri) akan mempengaruhi elemen yang bergantung padanya. Secara grafis, dependency digambarkan dengan tanda panah terputus-putus.

b. *Association* (asosiasi)

Asosiasi adalah sesuatu yang menghubungkan antara suatu objek dengan objek lainnya yang menggambarkan bagaimana hubungan antar objek tersebut dalam bentuk agregasi. Secara grafis, asosiasi digambarkan dengan garis tegas tanpa tanda panah.

c. *Generalizations* (generalisasi)

Generalisasi adalah hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari obyek yang ada diatasnya yaitu objek induk

(ancestor). Arah dari atas ke bawah atau dari ancestor ke descendent dinamakan *Spesialisasi*. Arah sebaliknya yaitu bawah ke atas dinamakan *Generalisasi*. Secara grafis *Generalisasi* digambarkan dengan garis dan panah yang berkepala sigitiga kosong dan mengarah ke objek induk.

d. Realizations (realisasi)

Realisasi adalah operasi yang benar-benar dilakukan oleh suatu objek. Secara grafis, realisasi digambarkan dengan tanda panah bergaris putus-putus dengan kepala panah kosong.

3. Diagrams (diagram)

Suatu sistem yang kompleks harus dapat dipandang dari sudut yang berbeda-beda sehingga didapat pemahaman secara menyeluruh. Untuk keperluan tersebut UML menyediakan 9 jenis diagrams yang dapat dikelompokkan berdasarkan sifatnya; Statis atau Dinamis. Kesembilan diagram ini tidak mutlak digunakan atau dibuat sesuai kebutuhan. Pada pemodelan UML dimungkinkan menggunakan diagram lain sejauh diagram tersebut dapat membantu pemahaman mendalam tentang suatu sistem atau perangkat lunak. Ke-9 diagram tersebut adalah sebagai berikut:

a. *Class* Diagrams (Statis)

Diagram ini memperlihatkan himpunan kelas-kelas, antarmuka-antarmuka, kolaborasi-kolaborasi, dan relasi-relasi. Diagram ini umum ditemui pada pemodelan sistem berorientasi objek. Meski sifatnya statis, sering pula memuat kelas-kelas aktif.

1. *Class Diagram* menggambarkan interaksi antar kelas dalam sistem tersebut.
2. Pembuatan *Class* sama dengan pembuatan Objek-objek

b. *Object Diagrams (Statis)*

Diagram ini memperlihatkan objek-objek serta relasi-relasi antar objek. Diagram objek memperlihatkan instantiasi statis dari segala sesuatu yang dijumpai dari pada diagram kelas.

c. *Use-Case Diagrams (Statis)*

Dagram ini memperlihatkan himpunan *Use-Case* dan *Actor-Actor* (jenis khusus dari kelas). Diagram ini penting untuk mengorganisasi dan memodelkan perilaku dari suatu sistem yang dibutuhkan serta diharapkan pengguna.

1. Use-Case diagram menggambarkan hubungan use-case dengan actor.
2. Use-case merepresentasikan fungsi, kebutuhan dari perpektif user
3. Actor adalah orang atau sistem yang menerima atau memberikan informasi dari sistem ini

d. *Sequence Diagrams (Dinamis)*

Diagram sequence (urutan) adalah diagram interaksi yang menekankan pada pengiriman pesan (*message*) dalam suatu waktu tertentu.

1. Sequence Diagram mengambarkan alur kerja dari fungsi-fungsi dalam sistem dengan use-case dimana didalamnya terdapat actor
2. Actor adalah orang atau sistem yang menerima atau memberikan informasi dari sistem ini. Dalam gambar diatas Actor-nya adalah Joe.

3. Diagram ini sangat memperhatikan waktu/ terurut berdasarkan kejadian (sequence)

e. *Collaboration Diagrams (Dinamis)*

Diagram kolaborasi adalah diagram interaksi yang menekankan organisasi structural dari objek-objek yang menerima serta mengirim pesan (*message*).

1. Informasi yang disampaikan sama dengan sequencial diagram namun beda dalam pengambaran dan kegunaan saja.
2. Dalam diagram ini digambarkan hubungan antar objek dan actor dengan tidak memperhatikan waktu/urutan

f. *State Chart Diagram (Dinamis)*

Diagram ini memperlihatkan *state-state* pada sistem; memuat state, transisi, event, serta aktifitas. Diagram ini terutama penting untuk memperlihatkan sifat dinamis dari antarmuka, kelas, kolaborasi, dan terutama penting pada pemodelan sistem-sistem reaktif.

1. State Chart Diagram memberikan berbagai cara/jalan kepada model untuk berbagai kejadian yang mungkin terjadi dalam sebuah objek
2. Diagram ini digunakan untuk menggambarkan berbagai prilaku objek yang sifatnya dinamis dalam sebuah sistem

g. *Activity Diagrams (Dinamis)*

Diagram ini adalah tipe khusus dari diagram state yang memperlihatkan aliran dari suatu aktifitas ke aktifitas lainnya dalam suatu sistem. Diagram ini terutama penting dalam pemodelan fungsi-fungsi dalam suatu sistem dan memberi tekanan pada aliran kendali antar objek.

1. Memberikan gambaran ilustrasi alur dari setiap fungsi yang ada dalam sistem
2. Aliran dimulai dari suatu titik hingga ke titik akhir yang disepakati.

h. *Component Diagrams (Statis)*

Diagram ini memperlihatkan organisasi serta kebergantunngan pada komponen-komponen yang telah ada sebelumnya. Diagram ini berhubungan dengan diagram kelas dimana komponen secara tipikal dipetakan kedalam satu atau lebih kelas-kelas, antarmuka-antarmuka, serta kolaborasi-kolaborasi.

1. Menggambarkan model secara fisik sebagai sebuah *software* komponen yang ada dalam sebuah sistem
2. Komponen-komponen tersebut nantinya diarahkan pada suatu bahasa pemrograman tertentu

i. *Deployment Diagrams (Statis)*

Diagram ini memperlihatkan konfigurasi saat aplikasi dijalankan (*run-time*), memuat simpul-simpul atau node beserta komponen-komponen yang ada didalamnya. *Deployment Diagrams* berhubungan erat dengan diagram komponen dimana *Deployment Diagrams* memuat satu atau lebih komponen-komponen. Diagram ini sangat berguna saat aplikasi kita berlaku sebagai aplikasi yang dijalankan pada banyak mesin (*distributed computing*).

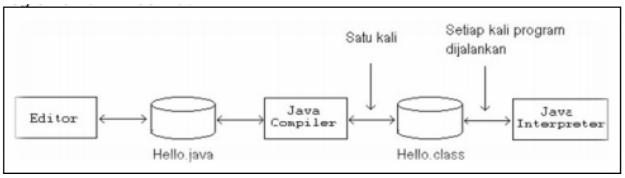
1. Diagram Deployment menggambarkan bentuk layout secara fisik bentuk jaringan dan posisi komponen-komponen dari sistem
2. Pendekatan yang digunakan dalah pendekatan terhadap hasil implementasi/ program

2.6 Java

2.6.1 Pengertian Java

Java adalah bahasa pemprograman yang disusun oleh james Gosling yang dibantu oleh rekan-rekannya seperti Patrick Naugton, Chris Warth, Ed Frank dan Mike Sheridan disuatu perusahan perangkat lunak yang bernama *Sun Microsystem,* pada tahun 1991. Bahasa pemprograman ini mula-mula diinisialisasi dengan nama ”Oak”, namun pada tahun 1995 diganti dengan nama “Java”.

Pada saat yang sama, *World Wide Web* dan *Internet* berkembang sangat cepat. *Gosling* menyadari bahwa *Java* dapat digunakan untuk pemrograman *Internet*. Dengan keluarnya versi 1.2, *platform Java* akhirnya dipilahpilah menjadi beberapa edisi, yaitu: *The Standard Edition (Java SE), Enterprise Edition (Java EE), Micro Edition (Java ME)* dan *JavaCard API.*



Gambar 2.1 Aliran proses kompilasi dan eksekusi sebuah program java

Tabel 2.1 *Platform Java*

Java SE – Java Standard Edition	Aplikasi Desktop
Java EE – Java Enterprise Edition	Aplikasi enterprise dengan fokus pada pengembangan sisi web server, termasuk *servlet, JSP, EJB*

	dan XML
Java ME – Java Micro Edition	Perangkat Mobile
JavaCard	*Smart Cards*

2.6.2 Arsitektur Java

Secara arsitektur, java tidak berubah sedikitpun semenjak awal mula bahasa dirilis. Kompiler Java (yang disebut dengan javac atau java compiler) akan mentransformasikan kode-kode dalam bahasa ke dalam suatu *bytecode*. *Bytecode* adalah sekumpulan perintah hasil kompilasi yang kemudian dapat dieksekusi melalui sebuah mesin komputer abstrak, yang disebut dengan *JVM* (*Java Virtual Machine*).

2.6.3 Pengertian J2ME

J2ME merupakan sebuah kombinasi yang terbentuk antara sekumpulan *interface java* yang sering disebut dengan *Java API* (*Application Programming Interface*) dengan *JVM* (*Java Virtula Machine*) yang didesain khusus untuk alat, yaitu *JVM* dengan ruang yang terbatas.

Menurut Samuel Mahatma Putra at all (2010 : 14) *Sun Microsystems* mendefinisikan J2ME sebagai "*Java run-time environment*" yang sangat optimal dalam penargetan berbagai macam produk konsumen, termasuk *pager*, telepon selular, *screenphones*, digital *set-top boxes* dan sistem navigasi pada mobil. J2ME diperkenalkan pada juni 1999 pada konfrensi *JavaOne Developer*. J2ME membawa fungsionalitas dari *Java Language* ke perangkat yang lebih kecil, sehingga memungkinkan mobile device untuk berbagi aplikasi. Dengan J2ME,

Sun telah beradaptasi untuk platform *java* pada produknya yang didasari pada perangkat penghitung yang lebih kecil.

Menurut Shalahudin (2008 : p2) J2ME digunakan untuk menjalankan dan mengembangkan aplikasi – aplikasi Java pada perangkat seperti telepon genggam, PDA (*Personal Digital Assistance*) Palm, dan pocket PC. Karena adanya J2ME, yang memungkinkan bagi para pengembang untuk bisa membuat aplikasi *wireless* yang multi platform, yang dapat diimplementasikan pada berbagai merek telepon genggam, yang mendukung aplikasi *Java*.

J2ME pada dasarnya terdiri dari tiga buah bagian, yaitu :

1. Konfigurasi

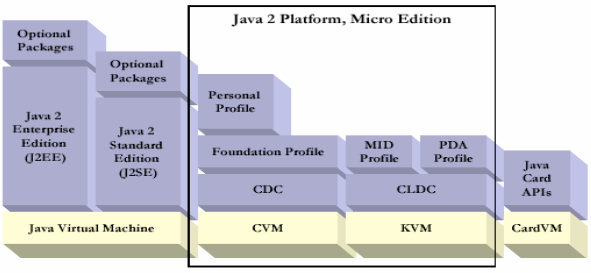
   Konfigurasi merupakan bagian yang berisi *JVM* dan beberapa *Library* kelas lainnya. Terdapat dua buah konfigurasi yang disediakan oleh *Sun Microsystem*, yaitu *CLDC* (*Connected Limited Device Configuration*) dan *CDC* (*Connected Device Configuration*).

2. Profil

   Adapun profil yang disediakan *Sun Microsystem*, yaitu *MIDP* (*Mobile Information Device Profile*).

3. Paket-paket operasional

   Merupakan paket-paket tambahan yang dibutuhkan oleh aplikasi sehingga pada saat proses *deployment* paket-paket tersebut perlu didistribusikan juga sebagai bagian dari aplikasi bersangkutan.



Gambar 2.2 Java 2 Micro Edition (J2ME)

2.6.4 CLDC

Menurut Budi Raharjo, Imam Heryanto dan Arif Haryono (2010 : 6) CLDC (Connected Limited Device Configuration) adalah sebuah konfigurasi yang terdapat di dalam J2ME untuk alat-alat yang memiliki keterbatasan ruang memori atau RAM (kurang dari 512 KB) dan pada umumnya dioperasikan dengan menggunakan baterai, serta memiliki bandwith yang kecil. Spesikasi CLDC pada J2ME adalah spesifikasi minimal dari *package*, kelas dan sebagian fungsi dari *Java Virtual Machine* yang dikurangi agar dapat diimplementasikan dengan keterbatasan sumber daya pada alat-alat tersebut.

2.6.5 MIDP

Mobile Information Device Profile (MIDP) berada di atas dari CLDC. Anda tidak bisa menulis aplikasi mobile hanya dengan menggunakan CLDC API. Anda harus tetap memanfaatkan MIDP yang mendefinisikan UI. Spesifikasi MIDP, kebanyakan seperti CLDC dan API lainnya sudah digambarkan melalui Java Community Process (JCP). JCP melibatkan sebuah kelompok ahli berasal dari lebih dari 50 perusahaan, yang terdiri atas pabrik perangkat mobile, pengembang software. MIDP terus berkembang, dengan versi-versi masa depan yang telah lulus dari proses ketat JCP.

Spesifikasi MIDP menggambarkan suatu perangkat MID yang memiliki karakteristik ini sebagai batas minimum:

1. Tampilan:

· Ukuran Layar: 96x54

· kedalaman tampilan: 1-bit

· Ketajaman pixel: sekitar 1:1

2. Masukan:

· Satu atau lebih mekanisme user-input: satu keybboard, dua keyboard, atau touch screen

3. Memory:

· 256 kilobytes of non-volatile memory untuk implementasi MIDP.

· 8 kilobytes of non-volatile memory for application-created persistent data

· 128 kilobytes of volatile memory for the Java runtime (e.g., the Java heap)

4. Jaringan:

· dua jalur, wireless, bandwidth terbatas

5. Sound:

· Kemampuan untuk memainkan nada-nada

MIDP menggambarkan model aplikasi, UI API, penyimpanan dan jaringan yang kuat, permainan dan media API, kebijakan keamanan, penyebaran aplikasi dan ketetapan over-theair.

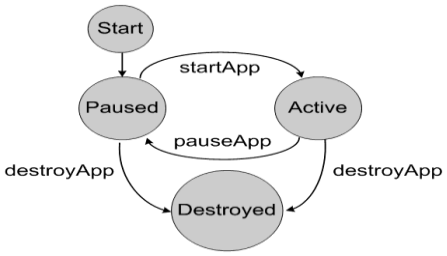
2.6.6 MIDlet

Menurut Budi Raharjo, Imam Heryanto dan Arif Haryono (2010:8) MIDlet adalah sebutan aplikasi-aplikasi yang dibuat dalam handphone dengan menggunakan profil MIDP. MIDlet berupa sebuah kelas abstrak yang merupakan subkelas dari bentuk dasar aplikasi sehingga antarmuka antara aplikasi J2ME dan aplikasi manajemen pada perangkat dapat terbentuk.

2.6.7 Siklus MIDlet

Kehidupan *MIDlet* dimulai ketika di-*instantiate* oleh *AMS*. *MIDlet* pada awalnya masuk status *"Pause"* setelah perintah baru dibuat. *AMS* memanggil

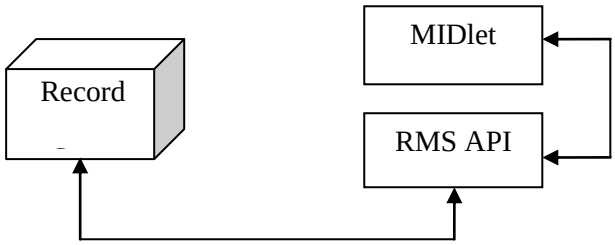
constructor public tanpa argumen dari *MIDlet*. Jika sebuah *exception* terjadi dalam *constructor*, *MIDlet* memasuki status *"Destroyed"* dan membuangnya segera. *MIDlet* masuk ke dalam status *"Active"* atas pemanggilan *method startUp()* oleh *AMS*. *MIDlet* masuk ke dalam status *"Destroyed"* ketika *AMS* memanggil *method destroyApp()*. Status ini juga kembali diakses ketika *method notifyDestroyed()* kembali dengan sukses kepada aplikasi. Dengan catatan bahwa *MIDlet* hanya bisa memasuki status *"Destroyed"* sekali dalam masa hidupnya.



Gambar 2.3 Daur Hidup MIDlet

2.7 RMS (*Record Managemen System*)

MIDlet tidak menggunakan file sistem untuk menyimpan data, tetapi menyimpan semua informasi dalam sebuah memori non-volatile (memori tetap) yang disebut dengan Record Management System (RMS). RMS (Record Management Sistem) adalah kumpulan record, dan record disimpan sebagai array dari byte dalam sebuah record store. RMS memiliki orientasi record basis data yang sederhana sehingga MIDlet dapat menyimpan informasi dan mengaksesnya. MIDlet yang berbeda dapat mengakses RMS (*Record Management System)* yang sama. Keterkaitan MIDlet, RMS (*Record Management System)* dan record store dapat dilihat pada gamabar di bawah ini.



Gambar 2.4 Keterkaitan MIDlet dan Record Store

Mekanisme penyimpanan data pada MIDP telah tersedia secara persisten (tetap) dalam memori *device* yang bersangkutan. Data tersebut berupa kumpulan-kumpulan *record*. *Record* yang telah ditempatkan di dalam ruang penyimpanan dapat diambil kembali untuk kemudian digunakan sesuai kebutuhan. Ruang penyimpanan yang tetap (*persisten storage*) bersifat *non-volatile* (data tidak akan hilang walaupun program direstart atau *device* dimatikan) untuk menyimpan data dalam sebuah aplikasi.

2.8 Emulator Ponsel Java

Untuk menjalankan *MIDlet,* programmer tidak perlu memiliki dan mencobanya pada ponsel. Cukup dengan emulator dari ponsel yang dapat berjalan pada PC. Dalam hal ini, emulator platform yang dapat digunakan adalah *Sun Java ™ Wirelees Toolkit 2.5.2 For CLDC.*

Terdapat empat buah *emulator*. *Emulator* itu sendiri digunakan untuk mensimulasikan *device* di dalam komputer sehingga *aplikasi* yang kita buat dapat dijalankan dan diuji coba tanpa harus *di-install* ke dalam *device* bersangkutan terlebih dahulu. Berikut ini daftar *emulator* yang disediakan:

Tabel 2.2 Daftar Emulator

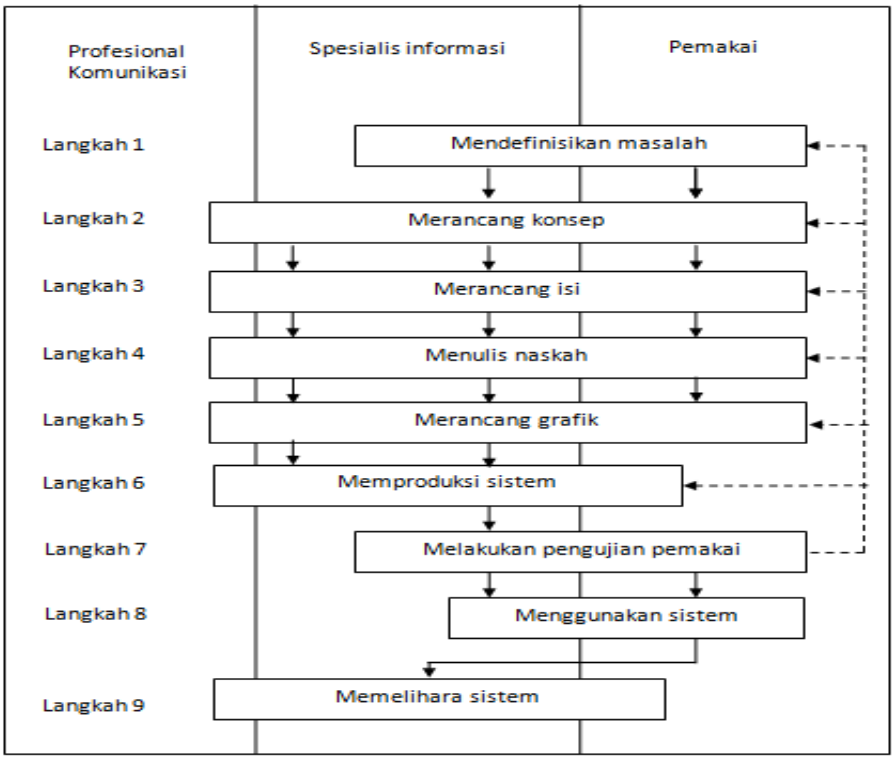
Nama Emulator	Ukuran Layar	Ukuran Canvas
DefaultColorPhone	240x320	240x289
DefaultGrayPhone	120x208	180x177
MediaControlSkin	120x208	180x177
QwertyDevice	636x235	540x204

2.9 Pengertian Netbeans

NetBeans adalah Integrated Development Environment(IDE)berbasiskan Java dari Sun Microsystems yang berjalan di atasSwing. Swing sebuah teknologi Java untuk pengembangan aplikasi Desktop yang dapat bejalan di berbagai macam platforms seperti Windows, Linux, Mac OS X and Solaris.

2.10 Metodologi Yang Diapakai Untuk Game Edukasi

Terdapat beberapa metode yang dipakai dalam pembuatan game edukasi, untuk lebih jelasnya dapat digambarkan seperti di bawah ini :



Gambar 2.5 Gambar Metode Game Edukasi

2.11 Pengguna Game Edukasi

Game edukasi ini dapat digunakan untuk semua kalangan anak sekolah dasar pada umumnya dan untuk anak kelas 4 pada khususnya.

2.12 Kurikulum Pemebelajaran Bahasa Inggris

Adapun kurikulum pembelajaran bahasa Inggris adalah sebagai berikut :

1. Membaca teks sederhana.
2. Memperkenalkan orang lain.
3. Mengenali nama benda.
4. Mempelajari benda yang bisa dihitung dan tidak bisa dihitung.
5. Mempelajari lawan kata.
6. Mempelajari penulisan jam.
7. Menyususn paragraph sederhana.