# Implementasi UML pada Program Java

[*Unified Modeling Language*](http://informatika.web.id/unified-modelling-language.htm) merupakan salah satu alat bantu yang dapat digunakan dalam **bahasa pemograman** yang berorientasi objek, saat ini UML akan mulai menjadi standar masa depan bagi industri pengembangan sistem/perangkat lunak yang berorientasi objek sebab pada dasarnya UML digunakan oleh banyak perusahaan raksasa seperti IBM, Microsoft, dan sebagainya.

## Definisi UML

1. *Unified Modeling Language* merupakan metode pengembangan perangkat lunak (sistem informasi) dengan menggunakan metode grafis serta merupakan bahasa untuk visualisasi, spesifikasi, konstruksi serta dokumentasi.
2. *Unified Modeling Language (UML)*adalah bahasa yang telah menjadi standard untuk visualisasi, menetapkan, membangun dan mendokumentasikan arti suatu sistem perangkat lunak.
3. *Unified Modeling Language (UML)*dapat didefinisikan sebagai sebuah bahasa yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem perangkat lunak.
4. *Unified Modeling Language (UML)* merupakan standard modeling language yang terdiri dari kumpulan-kumpulan diagram, dikembangkan untuk membantu para pengembang sistem dan *software*agar bisa menyelesaikan tugas-tugas seperti :
   * Spesifikasi
   * Visualisasi
   * Desain arsitektur
   * Konstruksi
   * Simulasi dan testing
   * Dokumentasi

Berdasarkan beberapa pendapat yang dikemukakan diatas dapat ditarik kesimpulan bahwa “U*nified Modeling Language (UML)*adalah sebuah bahasa yang berdasarkan grafik atau gambar untuk menvisualisasikan, menspesifikasikan, membangun dan pendokumentasian dari sebuah sistem pengembangan perangkat lunak berbasis Objek (*OOP)  (*[*Object Oriented programming*](http://informatika.web.id/object-oriented-programming.htm)*)”.*

## Langkah-langkah penggunaan Unified Modeling Language (UML)

Adapun langkah-langkah penggunaan *Unified Modeling Language (UML)* diantaranya sebagai berikut :

1. Buatlah daftar *business process* dari *level* tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Petakan *use case* untuk setiap *business process* untuk mendefinisikan dengan tepat fungsional yang harus disediakan oleh sistem, kemudian perhalus *use casediagram* dan lengkapi dengan *requirement, constraints* dan catatan-catatan lain.
3. Buatlah *deployment diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan*requirement* lain *non fungsional, security* dan sebagainya yang juga harus disediakan oleh sistem.
5. Berdasarkan use case diagram, mulailah membuat activity diagram.
6. Definisikan obyek-obyek level atas *package* atau *domain* dan buatlah *sequence* dan/atau *collaboration* utuk tiap alir pekerjaan, jika sebuah *use case* memiliki kemungkinan alir normal dan *error*, buat lagi satu diagram untuk masing-masing alir.
7. Buatlah rancangan *user interface model* yang menyediakan antarmuka bagi pengguna untuk menjalankan *skenario use case.*
8. Berdasarkan model-model yang sudah ada, buatlah *class diagram*. Setiap *package*atau *domain* dipecah menjadi *hirarki class* lengkap dengan *atribut* dan metodenya. Akan lebih baik jika untuk setiap *class*dibuat *unit test* untuk menguji *fungsionalitas class* dan interaksi dengan *class* lain.
9. Setelah *class diagram* dibuat, kita dapat melihat kemungkinan pengelompokkan *class* menjadi komponen-komponen karena itu buatlah *component diagram* pada tahap ini. Selain itu, definisikan *test integrasi* setiap komponen untuk meyakinkan ia dapat bereaksi dengan baik.
10. Perhalus *deployment diagram* yang sudah dibuat. Detailkan kemampuan dan *requirement* piranti lunak, sistem operasi, jaringan dan sebagainya. Petakan komponen ke dalam *node.*
11. Mulailah membangun sistem. Ada dua pendekatan yang tepat digunakan:
    * Pendekatan *use case* dengan *mengassign*setiap *use case* kepada tim pengembang tertentu untuk mengembangkan unit kode yang lengkap dengan *test.*
    * Pendekatan komponen yaitu *mengassign* setiap komponen kepada tim pengembang tertentu.
12. Lakukan **uji modul dan uji integrasi** serta perbaiki model beserta *code*nya*.* Model harus selalu sesuai dengan *code* yang aktual.
13. Perangkat lunak siap dirilis

## Ruang Lingkup UML

Dalam kerangka spesifikasi, U*nified Modeling Language (UML)* menyediakan model-model yang tepat, tidak mendua arti *(ambigu)* serta lengkap.

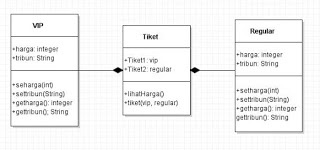
Secara khusus,*Unified Modeling Language (UML)*menspesifikasikan langkah-langkah penting dalam [pengambilan keputusan](http://informatika.web.id/pengambilan-keputusan.htm) analisis, perancangan serta implementasi dalam sistem yang sangat bernuansa perangkat lunak *(software intensive  system).*

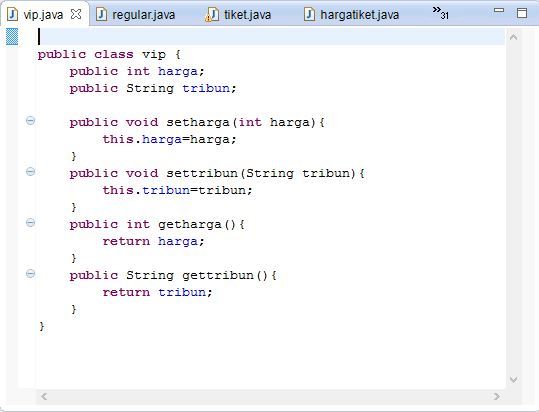
Dalam hal ini, U*nified Modeling Language (UML)*bukanlah merupakan bahasa pemprograman tetapi model-model yang tercipta berhubungan langsung dengan berbagai macam bahasa pemprograman, sehingga adalah mungkin melakukan pemetaan *(mapping)* langsung dari model-model yang dibuat dengan U*nified Modeling Language (UML)*dengan bahasa-bahasa pemprograman berorientasi obyek, seperti *Java, Borland Delphi, Visual Basic, C++,* dan lain-lain.

Pemetaan *(mapping) Unified Modeling Language (UML)*bersifat dua arah yaitu :

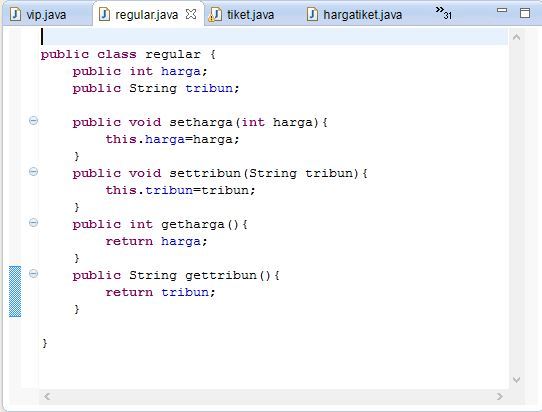
1. Generasi  kode bahasa pemprograman tertentu dari U*nified Modeling Language (UML) forward engineering.*
2. Generasi kode belum sesuai dengan kebutuhan dan harapan pengguna, pengembang dapat melakukan langkah balik bersifat *iterative* dari implementasi ke U*nified Modeling Language (UML)*hingga didapat sistem/peranti lunak yang sesuai dengan harapan pengguna dan pengembang.

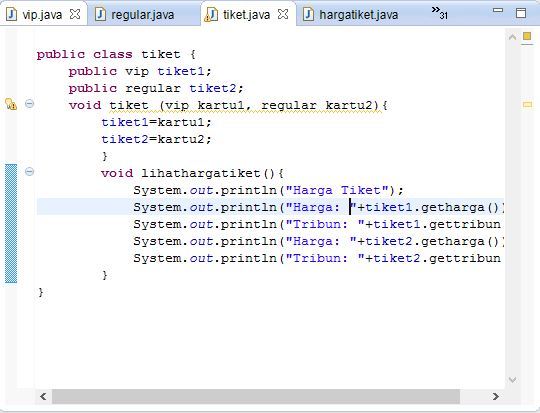
Berikut adalah implementasi UML pada harga tiket regular dan VIP:

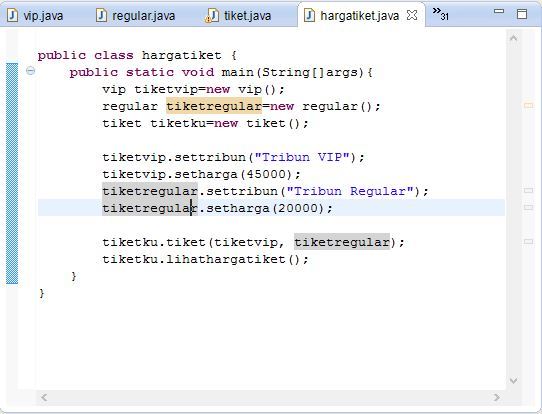




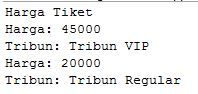
REPORT THIS AD







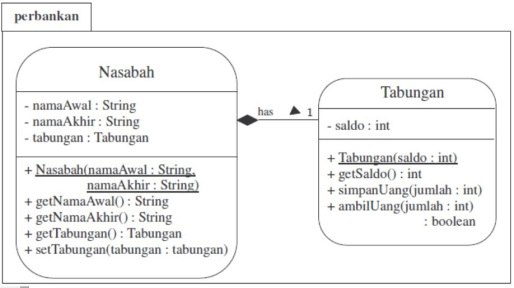
Berikut adalah hasil dari program tersebut:



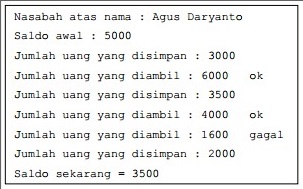
**IMPLEMENTASI UML DENGAN CLASS DIAGRAM**

**Identifikasi Kasus Program Perbankan**

Pada kasus ini kita akan mengimplementasikan UML Class diagram dalam program package perbankan. Program yang akan kita buat untuk menangani kasus dalam transaksi perbankan seperti transaksi debet dan kredit. Namun, kita buat yang lebih sederhana seperti simpan dan ambil uang.



Nasabah yang bernama Agus Daryanto akan melakukan beberapa transaksi simpan dan ambil uang. Dalam proses ambil dan simpan uang akan ada informasi berhasil (ok) atau gagal. Kira-kira jika dijalankan program akan menampilkan seperti di bawah ini:



Khusus untuk proses pengambilan uang, bila uang yang diambil lebih dari saldo maka transaksi akan dibatalkan (gagal).

**Analisis Kasus Program Perbankan**

Berdasarkan ouput diatas, kita dapat lihat saat program dijalankan, akan tampil saldo awal, saldo akhir, dan saldo sekarang. Jumlah saldo awal Pak Agus 5000. Kemudian pada kasus ini Pak Agus akan melakukan penyimpanan dan pengambilan uang. Pada saat penyimpanan saldo akan bertambah, dan sebaliknya pada pengambilan uang.  Karena saldo tidak boleh kurang dari nol, maka transaksi akan gagal saat ingin melakukan pengambilan uang lebih dari saldo. Program akan berhenti dengan menampilkan saldo Pak Agus saat ini.

**Implementasi Kasus Program Perbankan**

Seperti biasa, pertama kali yang dilakukan adalah membuat java project, package, dan class java. Selanjutnya tambahkan main() di dalam kelas. Di dalam method main kita akan bekerja untuk menuliskan source kode programnya. Source program yang akan kita buat bisa mengikuti seperti di UML diagram. Pertama kita buat Class Tabungan.java yang berisi saldo. Sebelumnya saya jelaskan arti tanda (-) pada UML Class diagram berarti modifier private dan tanda (+) berarti modifier public. Setelah itu, kita buat constructor saldo dan getter saldo.

Selanjutnya buat obyek simpan uang dan ambil uang, seperti yang sudah dijelaskan di atas. Kode program keseluruhannya menjadi seperti di bawah ini

|  |
| --- |
| Public class Tabungan {  private int saldo; |
|  |
| public Tabungan(int saldo) { |
| this.saldo = saldo; |
| } |
| public int getSaldo() { |
| return saldo; |
| } |
| public int simpanUang(int jumlah) { |
| return saldo = saldo + jumlah; |
| } |
| public boolean ambilUang(int jumlah) { |
| if (saldo – jumlah < 0) { |
| return false; |
| } else { |
| saldo -= jumlah; |
| return true; |
| } |
| } |
| } |

Setelah membuat Class tabungan lalu, buatla Class Nasabah.java yang terdiri dari modifier (private) nama awal, nama akhir, dan tabungan. Silahkan tambahkan constructor nama awal dan nama akhir, serta getter keduanya. Lalu tambahkan getter dan setter tabungan. Kode programnya akan seperti di bawah ini:

|  |  |
| --- | --- |
|  | Public class Nasabah{  private String namaAwal; |
|  | private String namaAkhir; |
|  | private Tabungan tabungan; |
|  |  |
|  | public Nasabah(String namaAwal, String namaAkhir) { |
|  | this.namaAwal = namaAwal; |
|  | this.namaAkhir = namaAkhir; |
|  | } |
|  | public String getNamaAwal() { |
|  | return namaAwal; |
|  | } |
|  | public String getNamaAkhir() { |
|  | return namaAkhir; |
|  | } |
|  | public Tabungan getTabungan() { |
|  | return tabungan; |
|  | } |
|  | public void setTabungan(Tabungan tabungan) { |
|  | this.tabungan = tabungan; |
|  | } |
|  | } |

Setelah itu kita akan buat kode program dengan keluaran seperti gambar pada penjelasan awal. Kode programnya akan seperti di bawah ini:

|  |  |
| --- | --- |
|  | public class UjiTabunganNasabah{  public static void main(String[] args) { |
|  | int tmp; |
|  | boolean status; |
|  | Nasabah nasabah = new Nasabah("Agus","Daryanto"); |
|  | System.out.println("Nasabah atas nama "+ nasabah.getNamaAwal()+nasabah.getNamaAkhir()); |
|  | nasabah.setTabungan(new Tabungan (5000)); |
|  | tmp = nasabah.getTabungan().getSaldo(); |
|  | System.out.println("Saldo : "+tmp); |
|  | nasabah.getTabungan().simpanUang(3000); |
|  | System.out.println("Jumlah yang disimpan 3000"); |
|  | status=nasabah.getTabungan().ambilUang(6000); |
|  | System.out.println("Jumlah yang diambil 6000"); |
|  | if(status) |
|  | System.out.println(" OK"); |
|  | else |
|  | System.out.println(" Gagal"); |
|  | nasabah.getTabungan().simpanUang(3500); |
|  | System.out.println("Jumlah yang disimpan 3500"); |
|  | status=nasabah.getTabungan().ambilUang(4000); |
|  | System.out.println("Jumlah yang diambil 4000"); |
|  | if(status) |
|  | System.out.println(" OK"); |
|  | else |
|  | System.out.println(" Gagal"); |
|  | status=nasabah.getTabungan().ambilUang(1600); |
|  | System.out.println("Jumlah yang diambil 1600"); |
|  | if(status) |
|  | System.out.println(" OK"); |
|  | else |
|  | System.out.println(" Gagal"); |
|  | nasabah.getTabungan().simpanUang(2000); |
|  | System.out.println("Jumlah yang disimpan 2000"); |
|  | tmp=nasabah.getTabungan().getSaldo(); |
|  | System.out.println("Saldo sekarang = "+tmp); |
|  | } |
|  | } |