JAVA VIRTUAL MACHINE PADA SISTEM OPERASI WINDOWS XP UNTUK MENGEKSEKUSI SISTEM PENJUALAN

Rika Rosnelly[1], Retantyo Wardoyo[2]
[1]STMIK Potensi Utama
Jl. KL. Yos Sudarso Km. 6,5 No. 3A Tj. Mulia Medan[1]
Email : rika@potensi-utama.ac.id[1]
[2]Universitas Gadjah Mada
FMIPA UGM Sekip Utara Bulaksumur Yogyakarta 55281[2]
Email: rw@ugm.ac.id[2]

Abstrak
Interpreter bytecode Java biasa disebut JVM (Java Virtual Machine) yang disediakan oleh Sun Microsystem untuk setiap platform. Dengan teknik ini program yang didistribusikan merupakan hasil kompilasi .class yang bentuknya sama untuk setiap platform. Kemudian nanti setelah .class ini hendak dijalankan maka barulah diinterpretasikan oleh masing-masing JVM yg ada pada setiap platform. Dalam makalah ini menjelaskan JVM menterjemahkan beberapa instruksi pada sistem penjualan menggunakan aplikasi NetBeans IDE 7.0 pada sistem operasi Windows XP dimana pada aplikasi tersebut terdapat Java Virtual Machine

***Kata kunci* :** *Java Virtual Machine , NetBeans IDE 7.0, Windows XP*

1. PENDAHULUAN

Java Virtual Machine (JVM) adalah sebuah spesifikasi untuk sebuah komputer abstrak. JVM terdiri dari sebuah kelas pemanggil dan sebuah interpreter Java yang mengeksekusi kode arsitektur netral. Kelas pemanggil memanggil file.class dari kedua program Java dan Java API untuk dieksekusi oleh interpreter Java. Interpreter Java mungkin sebuah perangkat lunak interpreter yang menterjemahkan satu kode byte pada satu waktu, atau mungkin sebuah just-intime (JIT) kompiler yang menurunkan *bytecode* arsitektur netral kedalam bahasa mesin untuk *host computer*.

Sebuah contoh mesin virtual Java mulai menjalankan aplikasi tersendiri dengan metode `main()` dari beberapa inisial class. Metode `main()` harus public, static, return `void`, dan menerima parameter : sebuah `String` array. Setiap kelas dengan seperti sebuah metode `main()`dapat digunakan sebagai titik awal untuk aplikasi Java.

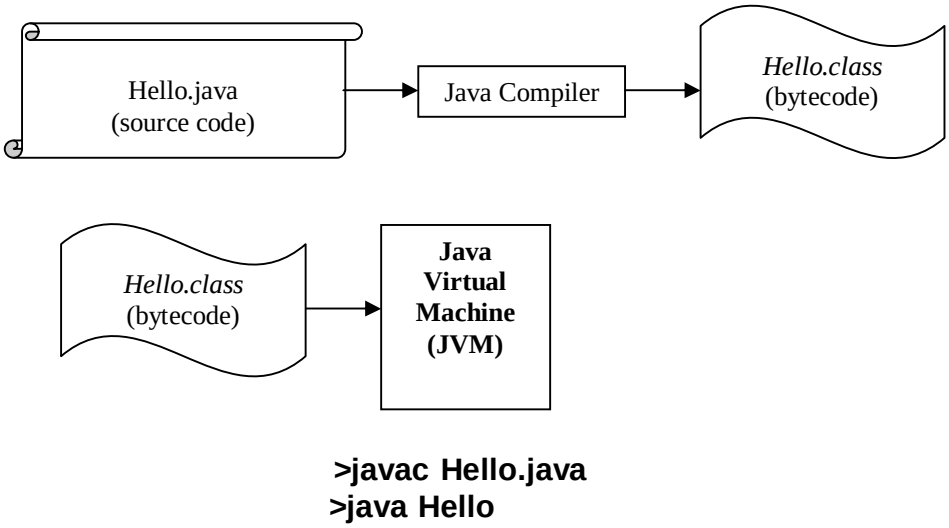
Dalam makalah ini menjelaskan JVM menterjemahkan beberapa instruksi pada sistem penjualan menggunakan aplikasi *NetBeans IDE 7.0* pada sistem operasi *Windows XP* dimana pada aplikasi tersebut terdapat *Java Virtual Machine.*

2. DASAR TEORI

Java adalah nama salah satu bahasa pemrograman komputer yang berorientasi objek, diciptakan oleh satu tim dari perusahaan Sun Microsystem, perusahaan workstation UNIX (Sparc) yang cukup terkenal. JAVA diciptakan berdasarkan bahasa C++, dengan tujuan *platform independent* (dapat dijalankan pada berbagai jenis hardware tanpa kompilasi ulang), dengan slogan *Write Once Run Anywhere* (WORA). Dibanding bahasa C++, JAVA pada hakikatnya lebih sederhana dan memakai objek secara murni.

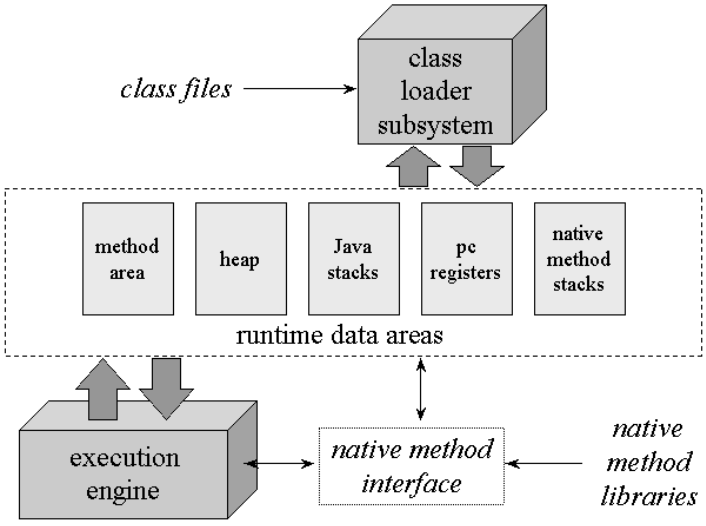
Java file kelas. adalah pengkodean program yang padat untuk stack berbasis virtual mesin. Hal ini dimaksudkan untuk digunakan dalam lingkungan jaringan, yang mengharuskan mesin independent dan meminimalkan konsumsi bandwidth jaringan. Namun, seperti di semua mesin virtual ditafsirkan, kinerja tidak cocok dengan kode yang dihasilkan untuk mesin target. Untuk memperbaiki masalah ini, banyak implementasi dari Java Virtual Machine (JVM) menggunakan "just-in-time" (JIT) kompiler, di mana Java bytecode dijabarkan ke dalam kode mesin [1].

Adapun kompilasi dengan Java dan eksekusi dengan Java ditunjukkan pada Gambar 1 [3].



Gambar 1. Kompilasi dengan javac dan eksekusi dengan Java [Sumber : Dr. Tsang]

1. Arsitektur Java Virtual Machine
   Dalam spesifikasi mesin virtual Java, behaviour mesin virtual digambarkan dari sisi subsistem, area memori, tipe data, dan instruksi.
   Arsitektur *Java Virtual Machine* menunjukkan diagram blok mesin virtual Java yang mencakup subsistem utama dan daerah memori yang diuraikan dalam spesifikasi, masing-masing mesin virtual Java memiliki subsistem class loader yaitu mekanisme untuk memuat tipe (kelas dan interface) diberikan nama-nama yang memenuhi syarat. Setiap mesin virtual Java juga memiliki mesin eksekusi yaitu mekanisme yang bertanggung jawab untuk melaksanakan instruksi yang terdapat dalam *methods of loaded classes.* Adapun arsitektur *Java Virtual Machine* ditunjukkan pada Gambar 2.



Gambar 2. Arsitektur internal *Java Virtual Machine*

Ketika Java Virtual Machine menjalankan sebuah program, butuh memori untuk menyimpan banyak hal, termasuk *bytecode* dan banyak informasi lain ekstrak dari *loaded class files,* objek program instantiate, parameter methods, nilai return, variabel lokal, dan antara hasil perhitungan. Java virtual machine mengatur memori yang dibutuhkan untuk mengeksekusi program ke beberapa *runtime* area data.
Meskipun terdapat runtime area data yang sama dalam beberapa bentuk di setiap implementasi JVM, spesifikasi cukup abstrak. Beberapa runtime area data membagi antara application's threads dan lainnya

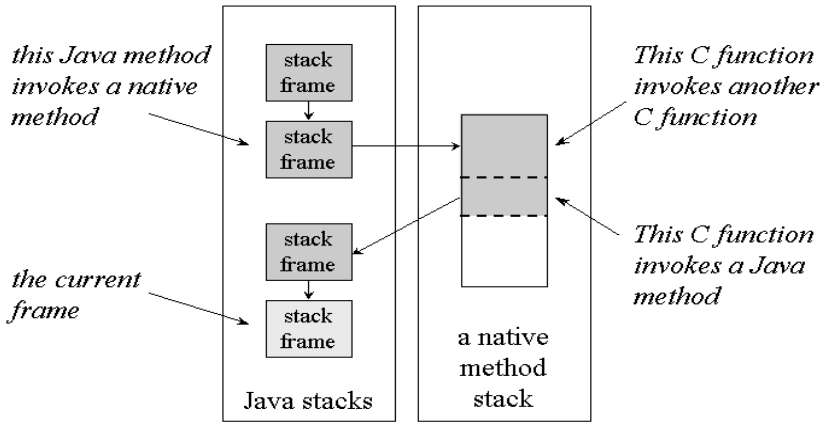
yang unik ke individual threads. Setiap contoh dari JVM memiliki satu area *method* dan satu *heap*. Daerah ini dibagi oleh semua thread yang sedang berjalan dalam JVM. Ketika mesin virtual memuat file class, kemudian menguraikan informasi tentang tipe dari data biner yang terdapat dalam file class. Ketika program berjalan, mesin virtual tempat semua obkjekprogram instantiate ke *heap*.
Java stack terdiri dari stack frames (frame). Sebuah frame berisi tumpukan dari satu pemanggilan metode Java. Ketika thread memanggil sebuah method, mesin virtual Java mendorong frmae baru ke thread Java stack. Ketika method sudah lengkap mesin virtual muncul dan membuang frame untuk method tersebut.

2. Subsistem Class Loader
   Java Virtual Machine terdiri dari 2 jenis class loader : *bootstrap class loader* dan *user-defined class loaders*. Bootstrap class loader merupakan bagian dari implementasi mesin virtual, dan class loader yang didefenisikan user merupakan bagian dari aplikasi Java. Kelas dimuat oleh class loader yang berbeda ditempatkan ke dalam ruang terpisah dalam JVM.
   Subsistem class loader melibatkan banyak bagian lain dari mesin virtual Java dan beberapa kelas dari perpustakaan java.lang. Sebagai contoh, class loader yang didefenisikan user berada pada regular Java yang mana class turun dari `java.lang.ClassLoader`. Method dari kelas `ClassLoader` memungkinkan aplikasi Java untuk mengakses mesin virtual class loading mesin.
3. Method Area
   Didalam JVM misalnya, informasi tentang memuat tipe disimpan di dalam logical area dari memori yang disebut method area. Ketika JVM memuat sebuah tipe, menggunakan class loader untuk mencari file kelas yang tepat. Class loader membaca pada file kelas—linear aliran data biner—dan meneruskannya ke mesin virtual. Memori untuk kelas (statis) variabel dideklarasikan di kelas juga diambil dari method area. Jumlah multi-byte dalam file kelas tersebut disimpan dalam big-endian (byte terpenting dahulu) order. Mesin virtual akan mencari melalui dan menggunakan jenis informasi yang disimpan di daerah metode seperti menjalankan aplikasi itu di hosting. Sebagai contoh bagaimana mesin virtual Java menggunakan informasi disimpan di method area, perhatikan class dibawah ini:

```
// On CD-ROM in file jvm/ex2/Lava.java
class Lava {
   private int speed = 5; // 5 kilometers per hour
   void flow() {
   }
}
// On CD-ROM in file jvm/ex2/Volcano.java
class Volcano {
   public static void main(String[] args) {
      Lava lava = new Lava();
      lava.flow();
   }
}
```

   Paragraf berikut menggambarkan bagaimana sebuah implementasi bisa mengeksekusi instruksi pertama dalam bytecode untuk metode main () dari aplikasi Volcano. Implementasi yang berbeda dari mesin virtual Java dapat beroperasi dengan cara yang sangat berbeda. Uraian berikut ini menggambarkan salah satu cara - tapi bukan satu-satunya cara - mesin virtual Java dapat mengeksekusi instruksi pertama dari metode Volcano's main ().
   Untuk menjalankan aplikasi Volcano, Anda beri nama "Volcano" ke mesin virtual Java. Diberi nama Volcano, mesin virtual menemukan dan membaca di dalam file Volcano.class. Ekstraksi defenisi dari kelas Volcano dari data biner dalam file kelas dan tempat-tempat informasi diimpor ke dalam method area. Mesin virtual kemudian memanggil metode main (), dengan menafsirkan bytecode disimpan di method area. Sebagai mesin virtual mengeksekusi main (), ia menjaga pointer ke constant pool (struktur data di daerah metode) untuk kelas saat ini (kelas Volcano).
4. Heap
   Mesin virtual Java memiliki instruksi yang mengalokasikan memori pada tumpukan untuk objek baru, namun tidak memiliki instruksi untuk membebaskan memori. Sama seperti Anda tidak dapat secara eksplisit gratis obyek dalam kode sumber Java, Anda tidak bisa secara eksplisit gratis obyek di bytecode Java. Mesin virtual sendiri bertanggung jawab untuk memutuskan apakah dan kapan memori ditempati oleh benda-benda yang tidak lagi direferensikan oleh aplikasi yang berjalan. Biasanya, implementasi mesin virtual Java menggunakan garbage collector untuk mengelola heap.

5. Program counter
   Setiap thread program yang berjalan telah mendaftarkan pc sendiri, atau program counter, yang diciptakan pada saat thread dimulai. Sebagai thread mengeksekusi sebuah metode Java, register pc berisi alamat dari instruksi saat ini sedang dijalankan oleh thread tersebut. Sebuah "alamat " bisa menjadi penunjuk asli atau offset dari awal bytecode metode's. Jika thread adalah melaksanakan metode asli, nilai register pc tidak terdefinisi.
6. Java Stack
   Ketika sebuah thread baru diluncurkan, mesin virtual Java membuat Java stack baru untuk thread. Seperti disebutkan sebelumnya, Java stack disimpan disebuah thread's state pada frame terpisah. Mesin virtual Java hanya melakukan dua operasi langsung di Java Stacks: itu mendorong dan pop frame.
7. Native Method Stacks
   Jika antarmuka implementasi native method menggunakan suatu model C-linkage, maka metode tumpukan asli adalah C stack. Ketika sebuah program C memanggil fungsi C, tumpukan beroperasi dengan cara tertentu. Argumen untuk fungsi didorong ke dalam stack pada urutan tertentu. Nilai kembali dilewatkan kembali ke fungsi pemanggilan dengan cara tertentu.
   Sebuah metode antarmuka asli kemungkinan akan (sekali lagi, terserah kepada para desainer untuk memutuskan) dapat menelepon kembali ke dalam mesin virtual Java dan memanggil sebuah metode Java. Dalam hal ini, benang daun native method stack dan masuk ke Java stack. [2]. Adapun stack untuk thread yang memanggil Java dan native method ditunjukkan pada Gambar 3.



Gambar 3. stack untuk thread yang memanggil Java dan native method

3. PEMBAHASAN

Bahasa Java dapat disebut sebagai bahasa pemrograman yang portable karena dapat dijalankan pada berbagai sistem operasi, asalkan pada sistem operasi tersebut terdapat *Java Virtual Machine.* Pembahasan disini menjelaskan JVM mengeksekusi beberapa instruksi dari sistem penjualan menggunakan Java yang dijalankan pada aplikasi NetBeans IDE 7.0 dimana pada aplikasi tersebut sudah terdapat Java Virtual Machine.

Pada dasarnya aplikasi penjualan berbasis java ini merupakan aplikasi Database yang terdiri dari beberapa layer sesuai dengan DOA pattern yaitu :

a. Domain layer : layer domain (tempat class domain yang ada dalam sistem penjualan)
b. DAO layer : layer Data Akses Object (DAO) yang terdiri dari DAO interface dan DAO Implementation berisi metode manipulasi data/table (insert, update, delete, load, dll)
c. UI layer : layer interface sebagai tempat mendesain user interface dalam kode diatas supaya lebih sederhana ada pada testerCustomer yang merupakan main class.

Tujuan dari pembagian aplikasi ini menjadi beberapa layer *(separation of consern)* adalah untuk memudahkan dalam mengelola kode program sehingga lebih modular. Berikut potongan coding dari sistem penjualan sebagai berikut :

```
package sim.penjualan.entity;
public class Customer {
private String kdCustomer;
private String namaCustomer;
private String alamat;
private String noTelp;
```

```
   public String getKdCustomer() {
      return kdCustomer;
   }
   public void setKdCustomer(String kdCustomer) {
      this.kdCustomer = kdCustomer;
   }
// setter dan getter lain untuk tiap instan variabel
}
package sim.penjualan.dao;
import java.util.List;
import sim.penjualan.entity.Customer;
public interface customerDao {
   public void insert(Customer customer);
   public void delete(String id);
   public void update(String oldId,Customer customer);
   public Customer load(String id);
   public List<Customer> getAll();
}
package sim.penjualan.Impl;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import sim.penjualan.dao.customerDao;
import sim.penjualan.entity.Customer;
public class customerDaoImpl implements customerDao {
 private Connection conn = null;
   public customerDaoImpl(Connection conn) {
      this.conn = conn;
   }
// Implementasi dari interface customerDAO yaitu metode-metode untuk manipulasi database
Public class TesterCustomer {
.....
.....
Public static void main(String [] arg){
.............
      Customer cus=new Customer();  // ciptakan objek Customer
// Ciptakan objek DAO (customerDaoImpl) untuk akses table customer
CustomerDao dao=new customerDaoImpl(Objek dari Class connection);
      dao.insert(cus);// gunakan ref dao dan panggil metode insert dengan parameter objek customer
.....
}
}
```

Program java berjalan dengan menggunakan Compiler dan Interpreter, compiler bekerja untuk merubah kode sumber menjadi byte code, byte code yang dihasilkan akan sama walaupun dibuat pada sistem yang berbeda.
Byte code yang dihasilkan ini akan kemudian akan dieksekusi dengan menggunakan interpreter java *(java virtual machine) javac* yang akan menghasilkan *native code* yang spesifik untuk tiap mesin.
Tiap pemanggilan javac akan menghasilkan instans dari java virtual machine yang bertugas untuk melakukan managemen eksekusi program.
Semua *resource* yang digunakan oleh program akan diolah oleh JVM dalam memori. Semua objek akan diproses pada *heap* beserta instance variable, sedangkan metode-metode beserta dengan variabel-variabel lokal akan diproses pada bagian lain dari memori yang disebut stack. Pada program diatas saat melakukan eksekusi class testerCustomer dengan javac.

```
      Customer cus=new Customer();
// ciptakan objek Customer yang akan di proses di dalam heap
```

```
// kemudian variable cus yang mereferensi ke objek customer akan di simpan dalam stack
// dalam hal ini class loader juga bekerja untuk me-load class Customer yang akan di pakai
//Ciptakan objek DAO(customerDaoImpl) untuk akses table customer
     CustomerDao dao=new customerDaoImpl(Objek dari Class connection);
      dao.insert(cus);//
//variabel –variabel lokal yang ada di metode insert  juga akan di simpan di dalam stack
```

Semua variabel-variabel lokal yang ada dalam suatu metode akan langsung di hapus oleh *garbage collector* saat metode itu selesai di eksekusi, sedangkan objek yang tercipta hanya boleh di hapus oleh *garbage collector* jika objek tersebut tidak dipakai lagi, dengan kata lain tidak ada lagi variabel yang mereferensikan ke objek tersebut. Misalnya dalam kasus ini objek *customer* akan dihapus jika kita *set cus=null*, karena sementara ini hanya *variable cus* yang mereferensikan ke objek *Customer*.

4. SIMPULAN

Konsep mesin virtual sangat baik , namun cukup sulit untuk diimplementasikan. Dengan konsep WORA *(Write Once Run Anywhere)* maka hasil kompilasi bahasa Java yang berupa *bytecode* dapat dijalankan pada platform yang berbeda.

Dari hasil pembahasan diatas menjelaskan JVM mengeksekusi beberapa instruksi dimana program java berjalan dengan menggunakan compiler dan interpreter, compiler bekerja untuk merubah kode sumber menjadi byte code, byte code yang dihasilkan akan sama walaupun dibuat pada sistem yang berbeda.

Byte code yang dihasilkan ini akan kemudian akan dieksekusi dengan menggunakan interpreter java *(java virtual machine) javac* yang akan menghasilkan *native code* yang spesifik untuk tiap mesin. Tiap pemanggilan javac akan menghasilkan instans dari java virtual machine yang bertugas untuk melakukan managemen eksekusi program, sehingga dapat diambil kesimpulan bahwa dengan menggunakan JVM dapat mengenali file kelas Java dan bisa mematuhi semantik dari kode Java yang berisi file kelas sehingga dapat mengeksekusi sistem penjualan.

DAFTAR PUSTAKA

[1]. Jones Joel, *Annotating Java Class Files with Virtual Registers for Performance*, 2000
[2]. Odersky Martin, Spoon Lex, Venners Bill, *Programming in Scala*, Artima, 2008
[3]. Tsang, Dr., Wai Wan, *CSIS0911B Computer Concepts and Programming (Java), http://www.csis.hku.hk/~c0911b*