

Reflection API

Yohanes Nugroho

Overview

- Mengenal Implementasi bytecode Java
- Reflection API
- Kegunaan Reflection API
- Memanfaatkan Reflection API untuk sistem plugin
- Membandingkan Reflection API dan Lambda Expression

Implementasi bytecode Java

- Semua program Java akan dikompilasi menjadi bytecode *sebelum dieksekusi*
 - *Bytecode adalah bahasa mesin untuk JVM (Java Virtual Machine)*
 - *Umumnya bytecode ini akan disimpan dalam filesystem dalam bentuk file .class*
- *Tidak ada kewajiban bahwa bytecode yang akan dieksekusi harus berada dalam file (bisa dari sumber lain)*

Class Loader

- Class Loader adalah bagian dalam JVM yang bertugas meload kelas ke dalam memori
- Defaultnya (dengan Class Loader bawaan Java) kelas dapat diloat dari:
 - filesystem (yang paling umum),
 - file jar (jika file kelas dikompresi),
 - jaringan (misalnya dalam kasus applet)

Custom Class Loader

- Programmer dapat membuat Class Loader sendiri yang bisa meload kelas dari mana saja (jaringan, file, dll) bahkan menciptakan kelas 'on the fly' (saat runtime)
 - Class Loader ini disebut dengan Custom Class Loader
- Custom Class Loader dibuat dengan menurunkan kelas abstrak `java.lang.ClassLoader`
 - Contoh pembuatan `ClassLoader` ada di dokumentasi API Java (Lihat dokumentasi API kelas `ClassLoader`)
 - Umumnya programmer tidak perlu membuat class Loader sendiri

Reflection API

- Class Loader bisa menciptakan kelas baru pada saat runtime, namun kelas baru tersebut mungkin tidak bisa digunakan karena programmer tidak mengetahui apa saja method dan property yang ada pada kelas baru (jika kelas tersebut tidak diciptakan programmer yang sama)
- Untuk bisa melihat dan mengeksekusi isi sebuah kelas baik yang dikenal maupun tak dikenal saat kompilasi, Java memiliki Reflection API

kelas Class

- Java memiliki kelas yang bernama Class (dengan C besar) untuk mengakses isi sebuah kelas, nama lengkapnya adalah `java.lang.Class`
 - perlu diingat bahwa semua package `java.lang` tidak perlu diimport
- Kelas Class perlu diinstansiasi dengan cara khusus (tidak bisa dengan **new**)
 - Lihat bagian berikut

Instansiasi objek Class

- Sebuah objek Class pasti berhubungan dengan sebuah nama kelas yang sudah dikenal Java
 - Instansiasi dapat dilakukan dengan 3 cara

- Dengan nama kelas:

```
Class c =  
    Class.forName("java.lang.String");
```

- Dengan literal nama kelas:

```
Class c = String.class;
```

- Dengan objek yang sudah dimiliki sebelumnya:

```
String s = "hello";  
Class c = s.getClass();
```


Method pada kelas Class

- Kelas Class memiliki banyak method, beberapa yang penting adalah:
 - `newInstance()` yang membuat objek baru
 - `getClasses()` yang mengembalikan interface dan kelas parent
 - `getFields()` dan `getMethods()` yang mengembalikan daftar field (attribut) dan method yang dimiliki
 - jika nama method atau field sudah diketahui, maka ada juga method `getField()` dan `getMethod()` (tunggal, bukan jamak)

Instansiasi Objek dengan objek Class

- Jika kita sudah memiliki objek Class (dengan salah satu cara yang sudah diberikan):

```
Class c = String.class;  
String s = (String)c.newInstance();  
Class cv =  
    class.forName("java.lang.Vector");  
Vector v = (Vector)cv.newInstance();
```

- Akan berguna dalam pembuatan plugin (ekstensi program yang ditambahkan tanpa mengkompilasi program [akan dijelaskan di bagian lain])

Kelas Field dan kelas Method

- Java juga memiliki kelas yang bernama Field dan Method (keduanya diawali huruf besar)
- Kelas Field memungkinkan akses terhadap Field dalam sebuah kelas
 - mendapatkan nama, tipe, membaca dan mengeset nilai field
- Kelas Method memungkinkan akses terhadap sebuah method dalam sebuah kelas
 - mendapatkan nama, return value, dan parameter (singkatnya: signature) sebuah method
 - memanggil method

Contoh

- Mendapatkan daftar method kelas String:

```
Class c = String.class;
Methods m[] = c.getMethods();
for (Method e: m) {
    System.out.println(e.getName());
}
```

Kegunaan Reflection

- Digunakan IDE untuk design GUI
 - IDE tidak perlu mengetahui sejak awal method apa saja yang ada di sebuah komponen sehingga komponen bisa ditambah kapan saja
- Digunakan Application Server (misalnya J2EE)
 - Suatu aplikasi yang menjalankan aplikasi lain dan memberikan pengaturan tambahan (dalam kasus J2EE: konkurensi, fitur terdistribusi)
- Digunakan untuk plugin
 - Editor seperti JEdit memakai plugin

Menggunakan Reflection untuk Plugin

- Reflection dapat digunakan untuk membuat Plugin
- Apa itu plugin? Plugin adalah kelas atau komponen yang bisa ditambahkan ke aplikasi tanpa perlu rekompilasi aplikasi
 - Contoh yang terkenal adalah plugin browser yang memungkinkan browser bisa membuka aneka macam file di dalam browser (flash, PDF) dengan bantuan plugin

Problem Description

- Kita memiliki sebuah aplikasi SpreadSheet
- Ada banyak format spreadsheet, misalnya:
 - Ms Excel, OpenOffice, CSV
- Kita ingin membuat plugin untuk membuka File pada sebuah Spreadsheet
 - Cukup dengan menambah kelas baru, maka aplikasi Spreadsheet yang kita miliki dapat membuka jenis file baru

Contoh Sistem Plugin [1]: deskripsi plugin

- Plugin diimplementasikan sebagai kelas
 - Untuk memudahkan pembuatan program: plugin pasti mengimplementasikan interface FileLoader
- Plugin diletakkan di sebuah direktori tertentu
 - untuk menambah format file yang didukung, plugin cukup dikompilasi (tanpa mengkompilasi seluruh aplikasi spreadsheet) disalin ke direktori tersebut, tanpa mengkompilasi ulang aplikasi

Contoh Sistem Plugin [2a]: Design Interface FileLoader

- Interface ini diperlukan agar Plugin pasti mengimplementasikan method untuk meload file
 - cara lain adalah dengan mewajibkan programmer plugin memakai nama method tertentu, tapi ini lebih rumit dan tidak bisa diperiksa *compiler*
- Isi interface (daftar method) boleh seperti apa saja (sesuai kebutuhan)

Contoh Sistem Plugin [2b]: Interface FileLoader

```
public interface FileLoader {  
    /*load file */  
    public void loadFile(String filename);  
    /*cek apakah file didukung plugin ini*/  
    public boolean isSupported(String  
        filename);  
    /*dapatkan jumlah baris*/  
    public int getRowCount();  
    /*dapatkan jumlah kolom*/  
    public int getColCount();  
    /*dapatkan isi sel tertentu*/  
    public String getCell(int row, int col);
```

}

Contoh Sistem Plugin [3]: Algoritma

- Algoritma untuk meload file kelas di sebuah direktori adalah:
 - [1] Dapatkan daftar file di direktori
 - [2] Panggil `Class.forName("nama kelas")` untuk mendapatkan objek Class untuk file tersebut
 - [3] Periksa apakah kelas mengimplementasikan `FileLoader`
 - [4] Instansiasi kelas
 - [5] Panggil method pada objek

Contoh Sistem Plugin [3-1]: List file

- Contoh berikut untuk mendapatkan daftar file di sebuah direktori (untuk mempersingkat, semua exception handling tidak ditampilkan):
- Contoh nama direktori adalah "plugin" (direktori ini harus ada di class path)

```
File f = new File("plugin");  
String names[] = f.list();
```

Contoh Sistem Plugin [3-2]: Meload Kelas

- Loading kelas dilakukan dengan `Class.forName()`, Class Loader Java akan otomatis mencari ke classpath

```
for (String name: names) {  
    Class c = Class.forName(name);  
    /*proses kelas */  
}
```

Contoh Sistem Plugin [3-3a]: Memeriksa Interface

- Kelas yang ada di direktori plugin belum tentu mengimplementasikan interface `FileLoader`, jadi kita perlu memeriksanya dulu
- Dapatkan daftar interface yang diimplementasikan kelas (ingat sebuah kelas bisa mengimplementasikan banyak interface) dengan `getInterfaces()`, dan periksa namanya

Contoh Sistem Plugin [3-3b]: Kode Memeriksa Interface

```
Class intf[] = c.getInterfaces();  
for (Class itf : intf) {  
    if (itf.getName().equals("FileLoader"))  
    {  
        /*proses kelas*/  
    } else {  
        /*abaikan kelas ini*/  
    }  
}
```

Contoh Sistem Plugin [3-4]: Instansiasi Kelas

- Jika sudah yakin bahwa kelas mengimplementasikan FileLoader, maka kelas bisa diinstansiasi

```
FileLoader fl =  
    (FileLoader) c.newInstance();
```

- Umumnya untuk menyimpan daftar plugin digunakan Vector:

```
Vector<FileLoader> flplugins = new  
    Vector<FileLoader>();  
  
flplugins.add(fl);
```


Contoh Sistem Plugin [3-5]: Memanggil method

- Kelas yang mengimplementasikan FileLoader yang sudah diinstansiasi bisa dipakai langsung:

```
fl.loadFile("hello.csv");
```

- Untuk plugin file loader, salah satu penggunaannya adalah menanyakan satu persatu plugin apakah mendukung file tertentu lalu meminta plugin yang mendukung untuk meloadnya

Contoh Sistem Plugin [3-5]: Contoh method LoadFile

```
boolean loadFile(String name) {  
    for (FileLoader fl : flplugins) {  
        if (fl.isSupported(name)) {  
            fl.loadFile(name);  
            for(int i = 0; i<fl.getRowCount(); i++){  
                for(int j = 0; j<fl.getColCount(); j++){  
                    addCell(i, j, fl.getCell(i,j));  
                }  
            }  
            return true; /*ok, ada plugin yg sesuai*/  
        }  
    }  
    return false; /*gagal, file tidak didukung*/  
}
```

Contoh penggunaan plugin

- Beberapa contoh plugin yang bisa dibuat:
 - plugin untuk menyimpan file
 - plugin untuk memproses data tertentu (misal plugin untuk mendefinisikan fungsi baru untuk spreadsheet)

Passing Kode Sebagai Argumen

[1]

- Kadang ada bagian program yang cocok dijadikan parameter agar kita tidak perlu membuat beberapa kode, atau membuat banyak "if"
- Contoh kasus:
 - Sorting: apakah menaik atau menurun, bagaimana perbandingan dilakukan
 - Filter: bagaimana filtering dilakukan

Passing Kode Sebagai Argumen

[1]

- Berbagai bahasa mendukung passing kode sebagai argumen/parameter sebuah fungsi
 - Lambda Expression (LISP)
 - Ekspresi yang dipassing
 - Function Pointer (C/C++)
 - Alamat ke kode yang dipassing
- Dalam Objek, ada dua cara passing kode:
 - Berdasarkan nama
 - Berdasarkan objek

Teori Formal

- Teori formal mengenai passing kode dapat ditemui di
 - Pelajaran Otomata: Mesin Turing
 - Pengkodean transisi state sebagai isi pita mesin turing
 - Lambda Calculus/Metoda Formal
 - Lambda Function
- Kuliah ini hanya mengajarkan sisi praktis dari sudut pandang pemrograman

Lambda Expression LISP

- Lambda Expression merupakan ekspresi tanpa nama yang memungkinkan kita memberikan fungsi sebagai parameter
 - Selain LISP, beberapa bahasa fungsional lain seperti Scheme juga mendukung lambda expression
- Di LISP semua adalah list, bahkan kode program juga dipandang sebagai list
 - Fungsi dipass sebagai list

Function Pointer di C

- Function pointer berguna untuk memberikan alamat fungsi (yang prototipenya diketahui) ke suatu fungsi lain
 - Hal ini memungkinkan mempassing fungsi (sesuatu yang sifatnya statik) untuk dipanggil (mirip dengan konsep lambda di LISP)
- Beberapa bahasa prosedural selain C (misalnya Pascal yang baru) juga mendukung function pointer

Passing Kode di bahasa OOP

- Function pointer kadang masih bisa dipakai (tergantung bahasa, C++ bisa, Java tidak)
- Cara yang umum digunakan adalah passing menggunakan “Interface” (atau kelas abstrak di C++)
 - Kode yang dipassing harus mengimplementasikan Interface tertentu
- Cara passing ini didukung oleh semua bahasa yang mendukung OOP

Passing Kode dengan Nama

- Java dan beberapa bahasa lain yang mendukung konsep Reflection (misalnya C# dan Python) memungkinkan passing kode berdasarkan nama (string nama)
 - Bahasa-bahasa ini memiliki kesamaan yaitu memiliki fase kompilasi-interpretasi
- Dari nama, kelas bisa diinspeksi, objek bisa diciptakan dan dipanggil

Penutup

- Passing kode merupakan teknik yang umum digunakan dalam program besar
- Java mendukung konsep passing kode dengan nama dan objek