Bab 7

Security

7.1 Tujuan

Pada akhir pembahasan bab ini, siswa diharapkan mampu :

- Memahami dasar security dan kriptografi
- Memahami proteksi domains dan permissions
- Bagaimana menambahkan permissions pada MIDlet Suite
- Bagaimana membuat MIDlet Suite menggunakan NetBeans Mobility Pack
- Bagaimana membuat message digest menggunakan SATSA
- Bagaimana melakukan enkripsi menggunakan symmetric keys

7.2 Dasar Security

Kriptografi

Kriptografi adalah cabang dari ilmu matematika yang memiliki banyak fungsi dalam pengamanan data. Kriptografi adalah proses mengambil message dan menggunakan beberapa fungsi untuk menggenerasi materi kriptografis (sebuah digest atau message terenkripsi).

Kriptografi adalah salah satu dari teknologi yang digunakan dalam layanan security seperti integrity, confidentiality, identity dan non repudiation.

Tipe Security Services

Sebelum kita memasuki bahasan dasar fungsi kriptografi, kita pelajari terlebih dahulu beberapa security services penting yang digunakan dalam sebuah aplikasi :

- **Authentification –** Adalah proses verifikasi indentitas dari pengguna pada akhir alur komunikasi.
- **Confidentiality –** Jika kita mengirimkan data sensitive melalui sebuah jaringan, kita ingin memastikan bahwa hanya penerima yang dituju yang dapat membacanya.
- **Integrity –** Kita ingin memastikan bahwa data yang kita terima tidak mengalami pengubahan, penambahan ataupun pemisahan.
- **Non-repudiation –** Service ini dapat menunjukkan bukti bahwa pengirim telah mengirimkan message, atau penerima telah menerima message.
- **Authorization –** Untuk memastikan bahwa user memiliki hak akses spesifik terhadap data penting maupun sumber data.

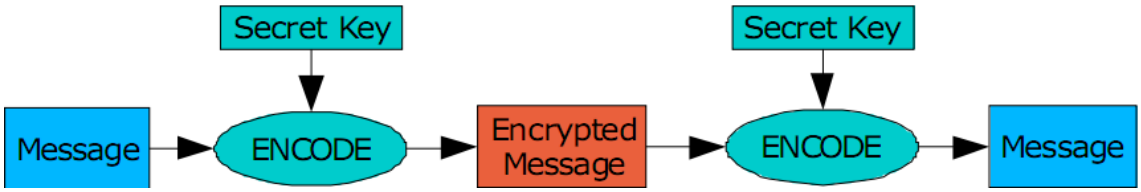
Message Digests

Sebuah message digest juga disebut sebagai digital fingerprint. Sebuah message digest adalah sebuah kesimpulan matematis dari sebuah message atau file. Hal ini untuk memastikan integritas dari message atau file. Sehingga dapat memberikan informasi bahwa sebuah message telah mengalami perubahan atau tidak. Mengubah satu karakter dari sebuah file atau message dapat menyebabkan perubahan drastis dari message digest. Digest terbuat melalui sebuah proses yang sangat menyulitkan untuk membuat dua file atau message yang berbeda dengan message digest yang sama.

Sebuah message digest berfungsi dalam satu alur fungsi. Sebuah message digest relatif mudah untuk diproses, namun sangat sulit jika dilakukan dengan cara sebaliknya. Dari sebuah message digest, sangat sulit untuk mengolah dan membuat sebuah message yang dapat menghasilkan message digest yang sama.

Kriptografi Symmetric Key

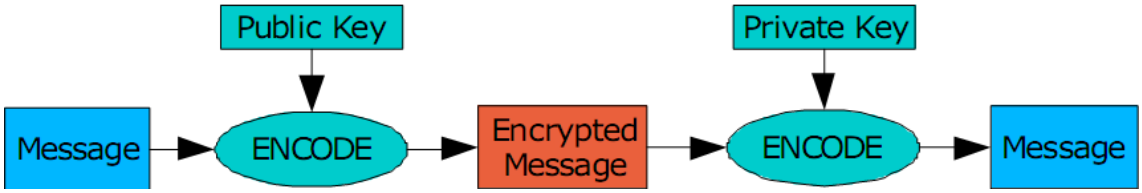
Dengan kriptografi symmetric key, sebuah message dapat terenkripsi dan terdekripsi menggunakan key yang sama. Baik pengirim maupun penerima message harus memiliki key yang sama supaya proses tersebut berjalan dengan sukses. Pengirim menggunakan key rahasia untuk mengenkripsi message, sedangkan penerima menggunakan kunci yang sama untuk mendekripsi message. Sekali message telah terenkripsi, message tersebut dapat dikirimkan melalui jaringan tanpa dipahami oleh penyadap.



Kriptografi Asymmetric Key

Permasalahan dari symmetric key adalah kedua pihak harus memiliki key yang sama. Key tersebut harus dikirimkan secara aman menuju penerima dari beberapa sebab sehingga key dapat dicuri dan digunakan untuk mendekripsi sebuah message.

Dengan menggunakan asymmetric keys, pengirim mengenkripsi message dengan menggunakan public key penerima. Kemudian penerima mendekripsi message tersebut menggunakan private key. Private key hanya dimiliki oleh penerima. Antara private dan public key merupakan komplemen matematis sehingga message yang terenkripsi menggunakan public key dapat terdekripsi menggunakan private key. Hal tersebut secara komputasi juga sulit untuk membuat private key ulang menggunakan public key.



Kriptografi Public Key

Algoritma public key menuntut penggunaan complementary key secara terpisah dalam proses enkripsi dan dekripsi. Hal ini menunjukkan kepastian bahwa akan memakan waktu yang sangat lama untuk mengetahui private key melalui pengolahan public key. Tuntutan ini membuat distribusi public key menjadi mudah tanpa mengkhawatirkan kerahasiaan dari private key. Algoritma public key yang amat popular adalah algortima RSA. Keamanan dari RSA telihat dari tingkat kesulitan faktorial numerik dalam cakupan yang besar.

Digital Signature

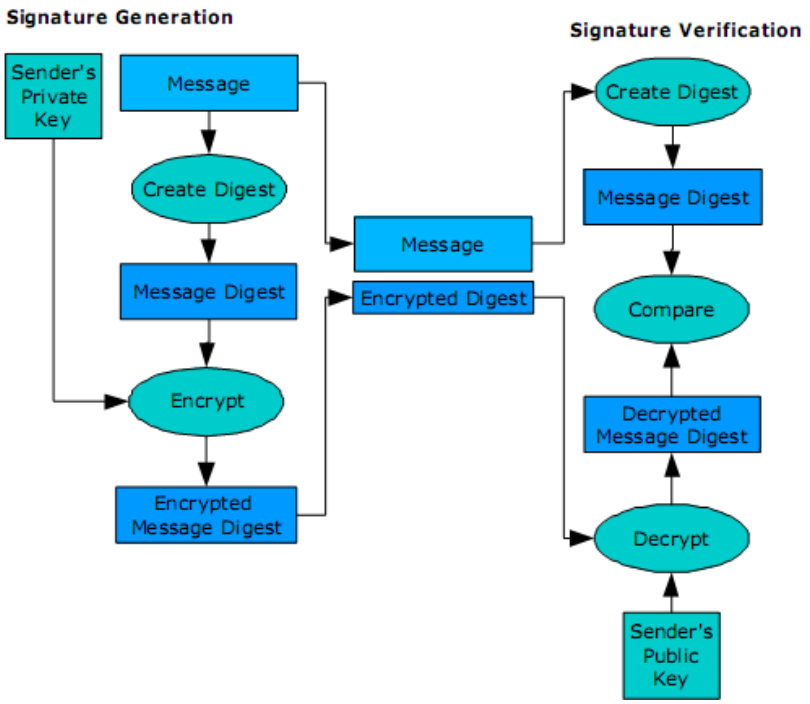
Sebuah digital signature mirip dengan message digest kecuali bahwa digest dihasilkan oleh private key dari sebagian personal atau entitas. Public key digunakan dalam verifikasi bahwa message yang ditandai berasal dari penanda.

Key Management

Salah satu permasalahan dari kriptografi public key adalah key management. Bagaimana anda mengetahui bahwa public key yang anda gunakan dalam verifikasi otentifikasi dari digital signature adalah public key asli yang dikirimkan oleh pengirim?

Digital Certificates adalah messages yang dibuat oleh Certification Authority (CA) yang menyertakan keabsahan entitas dari public key. Pada dasarnya, Digital Certificates adalah container dari Public Keys. Untuk mendapatkan sertifikat dari CA, sebuah entitas menyertakan dokumentasi yang membuktikan eksistensi dari identitas. Setelah melalui proses verifikasi identifikasi, CA kemudian menandai public key dari identitas yang menggunakan private key dari CA. Namun kita telah menciptakan problem yang lain. Bagaimana kita memverifikasi bahwa public key dari CA adalah asli? Anda akan mengetahui bahwa kita hanya membuat rangkaian dari keabsahan.

Solusinya adalah penandaan certificate secara pribadi. CA menandai public key menggunakan private key yang sesuai. Kemudian certificate yang telah ditandai dan mengandung public key dari CA akan didistribusikan secara bebas. Hal ini dikenal sebagai root certificate. Certificate yang ditandai secara pribadi dapat dibuat dengan mudah oleh siapapun. Aplikasi seperti web browser dan email umumnya disertai dengan root certificates dari Certificate Authorities yang diterima secara luas.



7.3 J2ME Security

Protection Domains – Sebuah protection domains mendefinisikan rangkaian permissions yang disertakan pada MIDlet Suite. MIDP 2.0 menjelaskan bahwa paling tidak terdapat dua buah protection domains : untrusted dan trusted domains. Untrusted domains adalah pembatasan dimana akses terhadap protected API pada kondisi default tidak diijinkan. Seorang user secara eksplisit harus mengatur tipe akses MIDlet Suite terhadap API. Untrusted MIDlets (berjalan di untrusted domains) tidak memerlukan user permission dalam mengakses protected API.

Untrusted dan trusted domain menyediakan akses yang tak terbatas pada Record Management, MIDlet life cycle, LDCUI, Game dan Multimedia API. Bagaimanapun, API untuk koneksi HTTP dan HTTPS menuntut kejelasan permissions dari user jika MIDlet Suite berjalan pada untrusted domain. Sebuah protection domain adalah rangkaian dari ”Allowed” dan ”User” permissions yang diberikan kepada MIDlet Suite.

Permissions

Terdapat dua tipe mode interaksi permissions, mode Allowed dan User. Pada mode Allowed, user tidak diminta melakukan pengaturan permission saat MIDlet mengakses sebuah API yang terproteksi. Sebuah aplikasi secara otomatis memberikan hak akses terhadap resource dan interaksi dari user tidak diperlukan.

Dalam mode User, device menanyakan apakah user menginginkan untuk mencabut atau memberikan hak akses MIDlet terhadap resource. Frekuensi dari pertanyaan bergantung pada mode interaksi yang dipilih oleh user.

Mode Interaksi User

Sebuah user permission memiliki salah satu dari 3 mode interaksi berikut :

Blanket – User memberikan permission pada MIDlet Suite untuk mengakses resource atau API secara permanen. User tidak akan lagi diminta melakukan pengaturan setiap MIDlet Suite berjalan. Permission yang ada akan tetap eksis hingga MIDlet Suite dihapus dari device atau user merubah permission tersebut.

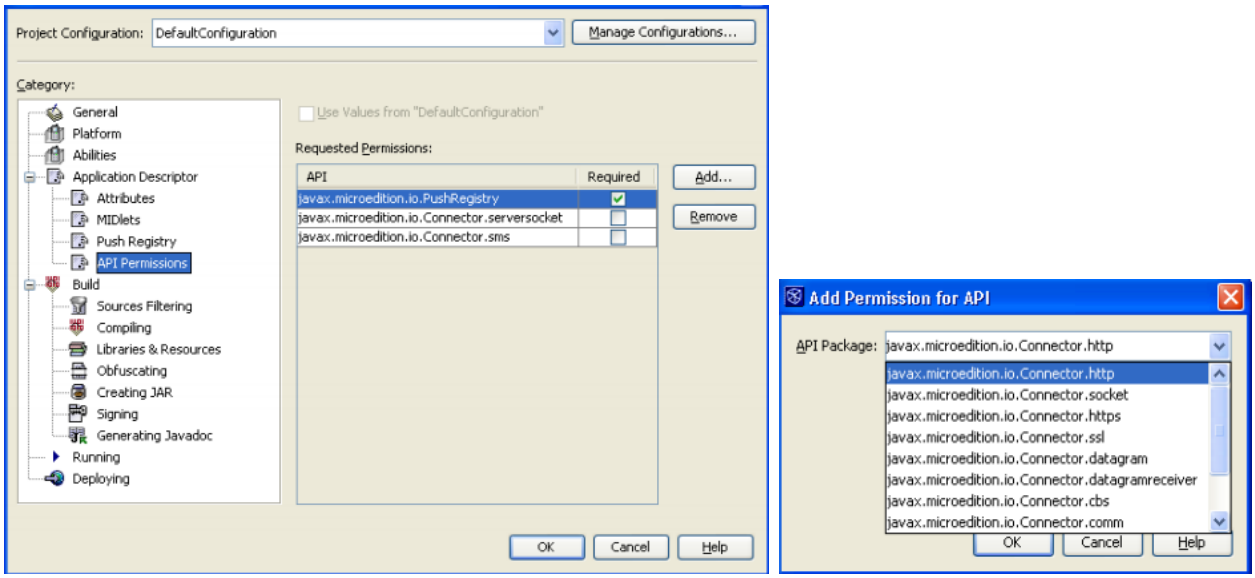
Membuat sebuah permission adalah salah satu dari cara pengamanan akses terhadap restricted APIs. Dalam MIDP, nama dari permission menggunakan nama dari package dari API tersebut sebagai prefix dan bersifat case sensitive. Jika permission tersebut ditujukan kepada sebuah class, maka penamaan permission harus mengandung nama class dan package.

Sebuah MIDlet dapat menuntut adanya permission dengan mendeklarasikan MIDlet-Permissions ataupun atribut MIDlet-Permissions-Opt pada application descriptor. Jika MIDlet Suite menyertakan atribut MIDlet-Permissions, atribut permission tersebut harus diberi hak akses terhadap protection domain. Jika hak akses tidak diberikan, maka proses instalasi akan dibatalkan.

Multiple permissions dituliskan menggunakan tanda koma (,) sebagai pemisah.

```
MIDlet-Permissions: javax.microedition.io.Connector.http
MIDlet-Permissions-Opt: javax.wireless.messaging.sms.receive,
javax.wireless.messaging.sms.send
```

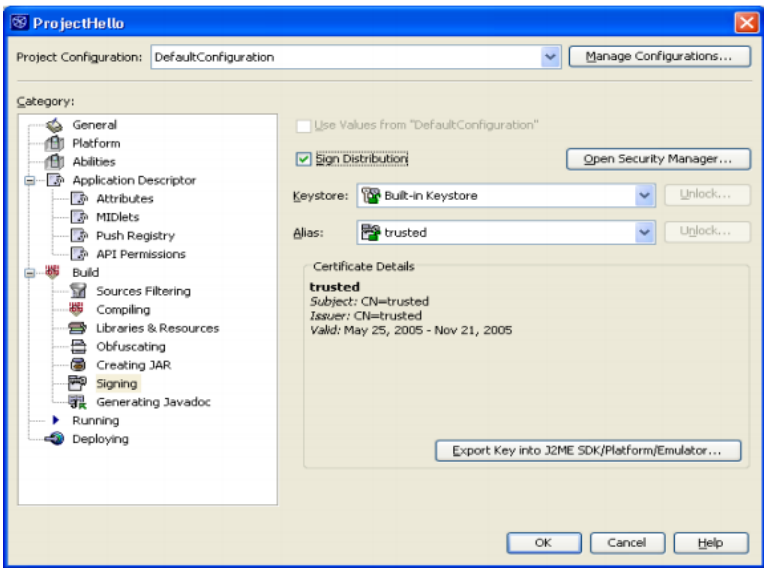
Membuat permissions pada MIDlet Suite menggunakan NetBeans Mobility Pack :



Trusted MIDlets – Sebuah MIDlet dapat diputuskan sebagai trusted application jika authentifikasi dan integritas dari file JAR dapat terverifikasi oleh device dan terbatas pada sebuah protection domain. Proses verifikasi dilakukan oleh device menggunakan certificates.

Menandai MIDlet pada NetBeans Mobility Pack :

Untuk memberi tanda pada MIDlet Suite menggunakan NetBeans Mobility Pack, buka project properties (klik kanan project name pada projects tab dan pilih Properties). Periksa bagian ”Sign Distribution” :



7.4 Menggunakan Security dan Trust Services API (SATSA)

Security and Trust Services API (SATSA) terdefinisi dalam Java Spesification Request (JSR) 177. SATSA adalah sebuah pilihan package yang menyediakan APIs untuk fungsi – fungsi security seperti manajemen digital signatures, pembuatan message digest dan digital signatures, berhubungan dengan Java Cards dan operasi kriptografi lainnya.

Contoh berikut ini menunjukkan cara pembuatan message digest dan enkripsi sebuah message menggunakan symmetric keys :

Membuat Message Digest :

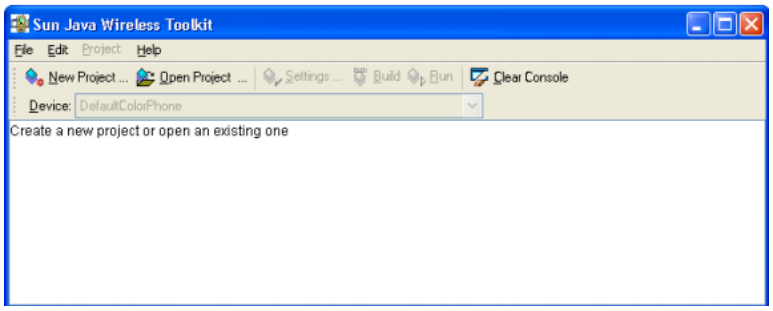
```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.security.*;
public class DigestMidlet extends MIDlet {
  public void startApp() {
    String message = "I LOVE JENI!";
    System.out.println("Generating digest for message: " + message);
    byte[] digest = generateDigest(message.getBytes());
    System.out.println("SHA-1 Digest:");
    for (int i=0; i<digest.length; i++)
      System.out.print(digest[i] + " ");
    System.out.println();
  }
```

```
  public void pauseApp() {}
  public void destroyApp(boolean unconditional) {}
  public byte[] generateDigest(byte[] message) {
    String algorithm = "SHA-1";
    int digestLength = 20;
    byte[] digest = new byte[digestLength];
    try {
      MessageDigest md;
      md = MessageDigest.getInstance(algorithm);
      md.update(message, 0, message.length);
      md.digest(digest, 0, digestLength);
    } catch (Exception e) {
      System.out.println("Exception: " + e.getMessage());
    }
    return digest;
  }
}
```

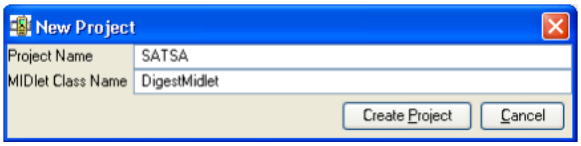
Sun Java Wireless Toolkit 2.3 menyediakan dukungan JSR 177 (atau SATSA) :

Proses Build dan Run file DigestMidlet :

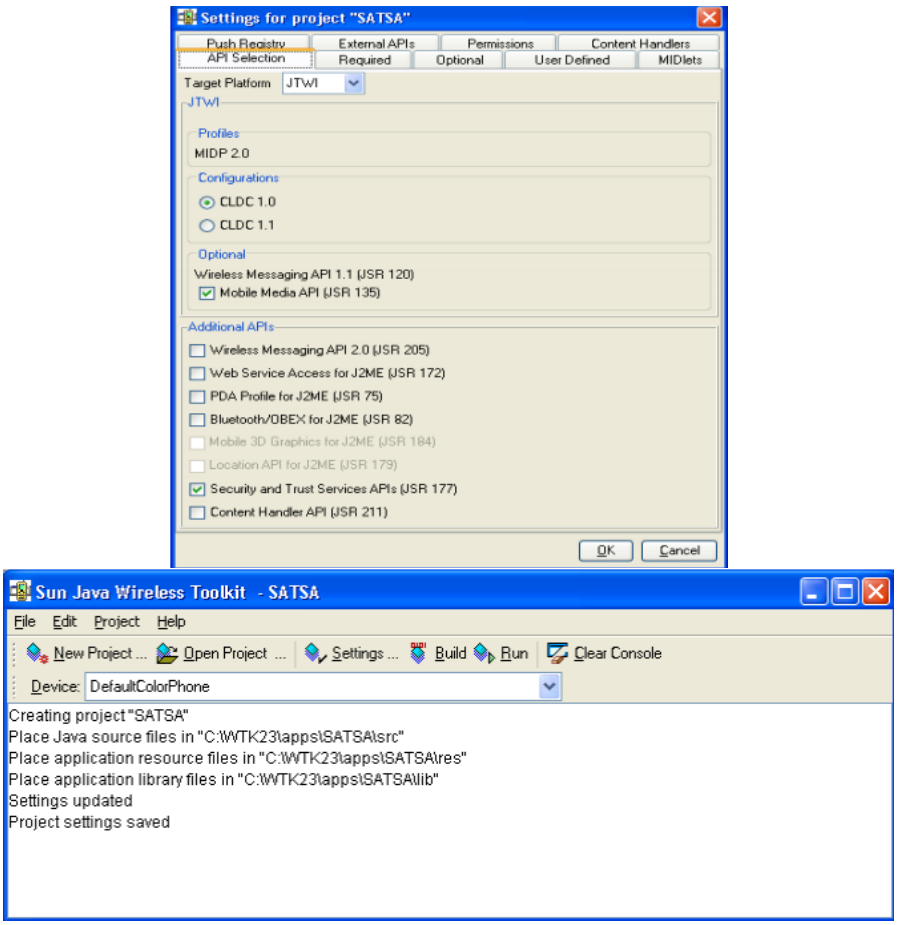
1. Buka aplikasi Ktoolbar dari Wireless Toolkit :



2. Pilih “New Project” kemudian tentukan nama project dan class MIDlet :



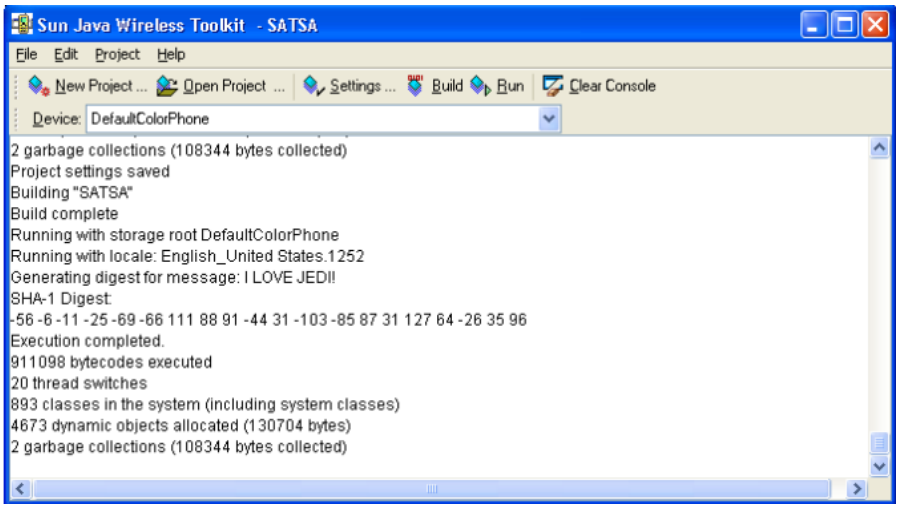
3. Pilih “JWTI” sebagai Target Platform. Tandai pilihan “Security and Trust Services APIs (JSR 177) dan klik “OK”.



4. Buat file dengan nama DigestMidlet.java pada direktori : WTK/apps/SATSA/src (WTK adalah direktori instalasi dari Sun Java Wireless Toolkit, secara default adalah C:\WTK23, dan SATSA adalah nama project).

5. Klik ”Build”

6. Klik ”Run” untuk menjalankan MIDlet pada emulator. Jika anda menjalankan MIDlet pada emulator, anda tidak akan mendapatkan output grafis apapun. Output yang akan dihasilkan adalah berupa console pada WTK Tollbar.

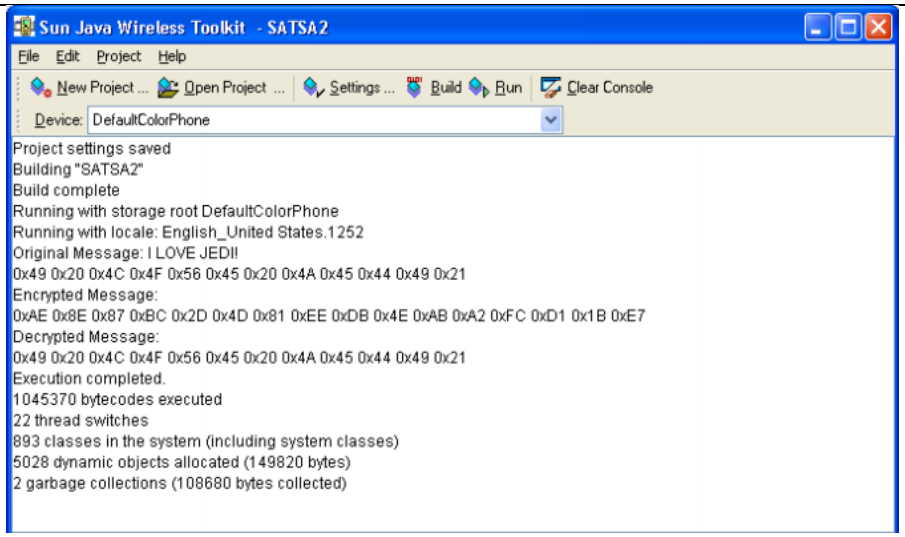


Enkripsi dan Dekripsi Messages menggunakan symmetric keys :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.microedition.securityservice.*;

public class SymmetricCipherMidlet extends MIDlet {
  private static final byte[] key = {
    (byte) 0xab, (byte) 0xcd, (byte) 0xef, (byte) 0x88,
    (byte) 0x12, (byte) 0x34, (byte) 0x56, (byte) 0x78
  };
  String message = "I LOVE JEDI!";

  public SymmetricCipherMidlet() {
    try {
      System.out.println("Original Message: " + message);
      printBytes(message.getBytes());
      byte[] encryptedMessage = encrypt("DES/ECB/PKCS5Padding",
        message.getBytes(), key, "DES");
      System.out.println("Encrypted Message:");
      printBytes(encryptedMessage);
      byte[] decryptedMessage = decrypt("DES/ECB/PKCS5Padding",
        encryptedMessage, key, "DES");
      System.out.println("Decrypted Message:");
      printBytes(decryptedMessage);
    } catch (Exception e) {
      System.out.println("Exception: " + e.getMessage());
      e.printStackTrace();
    }
  }

  private void printBytes(byte[] bytes) {
    for (int i = 0; i<bytes.length; i++)
      System.out.print(toHex(bytes[i]) + " ");
    System.out.println();
```

```
  }

  private String toHex(byte b) {
    char d1 = toHexDigit((byte) ((b>>4) & 0x0F));
    char d2 = toHexDigit((byte) (b & 0x0F));
    return ("0x"+d1+d2);
  }

  private char toHexDigit(byte x) {
    if (x > 9)
      return ((char) ('A' + (x-10)));
    else
      return ((char) ('0' + x));
  }

  private byte[] encrypt(String algorithm, byte[] message, byte[] keybytes,
  String keyAlgo) throws NoSuchAlgorithmException, NoSuchPaddingException,
  InvalidKeyException, ShortBufferException, IllegalBlockSizeException,
  BadPaddingException {
    Cipher cipher = Cipher.getInstance(algorithm);
    Key key = new SecretKeySpec(keybytes, 0, keybytes.length, keyAlgo);
    int blockSize = 16;
    int cipherLength = ((message.length / blockSize)
      + ((message.length % blockSize) > 0 ? 1 : 0)) * blockSize;
    cipher.init(Cipher.ENCRYPT_MODE, key);
    byte[] cipherText = new byte[cipherLength];
    cipher.doFinal(message, 0, message.length, cipherText, 0);
    return(cipherText);
  }

  public byte[] decrypt(String algorithm, byte[] cipherText, byte[] keybytes,
  String keyAlgo) throws NoSuchAlgorithmException, NoSuchPaddingException,
  InvalidKeyException, ShortBufferException, IllegalBlockSizeException,
  BadPaddingException {
    Cipher cipher = Cipher.getInstance(algorithm);
    Key key = new SecretKeySpec(keybytes, 0, keybytes.length, keyAlgo);
    cipher.init(Cipher.DECRYPT_MODE, key);
    byte[] decrypted = new byte[message.length()];
```

```
    cipher.doFinal(cipherText, 0, cipherText.length, decrypted, 0);
    return(decrypted);
  }

  public void startApp() {}

  public void pauseApp() {}

  public void destroyApp(boolean unconditional) {}
}
```



Bab 8

Web Services

8.1 Tujuan

Diakhir pembahasan, diharapkan siswa dapat :

- Mengetahui bagaimana memodelkan data menggunakan XML
- Mengetahui beberapa Java API yang digunakan dalam pemrosesan XML
- Mendeskripsikan apa yang bisa dilaksanakan oleh web service
- Mengetahui beberapa key standard dalam web service
- Mengetahui bagaiamana membuat web service mobile sebagai client

8.2 Pengenalan terhadap XML

XML (eXtensible Markup Language) adalah sebuah meta-language untuk mendeskripsikan data. XML merupakan sebuah cara merepresentasikan data tanpa tergantung kepada system. Ia juga dapat digunakan sebagai extension markup languages. XML adalah berbasis text, sehingga ia dapat dengan mudah dipindahkan dari satu sistem komputer ke sistem yang lain. Dengan XML, data direpresentasikan dalam sebuah dokumen yang terstruktur dan ia juga telah menjadi sebuah standard industri.

Element

Sebuah dokumen XML adalah sebuah dokumen yang mudah dibaca dan terdiri dari XML tag atau element. Sama halnya dengan HTML, XML tag didefinisikan dengan kurung siku <>. Sebuah dokumen XML memiliki struktur seperti entities didalam sebuah tree.

Anda dapat menggunakan tag sesuai dengan yang Anda inginkan, selama semua aplikasi yang menggunakan dokumen tersebut menggunakan tag dengan nama yang sama. Tag dapat memiliki attributes. Dalam contoh dibawah ini, "task" pertama memiliki "id" dengan attribut "1", sedangkan "task" yang kedua memiliki "id" dengan attribute "2".

```
<tasks>
  <task id="1">
    <name>connect to database</name>
    <duration>2</duration>
    <assignedTo>alex</assignedTo>
    <progress>50</progress>
  </task>
  <task id="2">
    <name>list table rows</name>
```

```
    <duration>4</duration>
    <assignedTo>alex</assignedTo>
    <progress>100</progress>
  </task>
</tasks>
```

Attributes

Tag dapat juga terdiri dari attribute-attribute. Didalam contoh, tag "task" memiliki attribute dengan nama "id". Sebuah attribut diikuti dengan tanda sama dengan (=) dan diikuti dengan value atau nilainya. Pada saat mendesai sebuah struktur XML, permasalahan yang selalu muncul adalah apakah sebuah data element harus menjadi attribute dari sebuah element atau menjadi sebuah sub-element. Seperti contoh, XML diatas juga dapat kita tulis lagi sebagai berikut:

```
<tasks>
  <task>
      <id>1</id>
      <name>connect to database</name>
   ...
</tasks>
```

Tidak ada aturan yang pasti, struktur element seperti apa yang harus kita anut. Akan tetapi dalam beberapa situasi, aturan-aturan dibawah ini harus dipenuhi:

- Data akan memiliki beberapa sub-struktur, pada kasus dimana Anda harus menggunakan sebuah sub-element karena ia tidak boleh dimodelkan sebagai attribut.
- Data akan terdiri dari beberapa baris apabila attribut ingin dibuat sesederhana mungkin – sebuah string yang pendek tetapi mudah untuk dibaca dan digunakan.
- Data element dimungkinkan untuk muncul kembali.
- Data akan sering berubah.

XML Schema

XML tag harus bersifat extensible, dimana seorang desainer system dimungkinkan untuk menuliskan sendiri XML tag-nya dalam pendeskripsian sebuah content. Anda dapat menciptakan tag-tag yang berbeda untuk setiap format dokumen yang Anda inginkan didalam aplikasi atau sistem Anda.

Tag didefinisikan menggunakan XML schema language. Sebuah schema mendefinisikan struktur dari dokumen XML. Sebuah skema juga digunakan membatasi content dari sebuah dokumen XML kedalam sebuah element, attributes, dan values tertentu. Sebuah Document Type Definition schema adalah bagian dari spesifikasi

XML. Kita akan memanggil schema yang ditulis dalam bahasa ini disebut sebagai DTD. DTD ini juga mendefinisikan tag atau attribute mana yang sangat diperlukan dan mana yang bersifat optional.

```
<!ELEMENT tasks (task)*>
<!ELEMENT task (name, duration, assignedTo, progress) >
<!ATTLIST task
  id CDATA #REQUIRED
>
<!ELEMENT task (name, duration, assignedTo, progress) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT duration (#PCDATA) >
<!ELEMENT assignedTo (#PCDATA) >
<!ELEMENT progress (#PCDATA) >
```

Namespace

Ada beberapa kasus dimana tag atau element Anda memiliki nama yang sama. Misalnya, ada beberapa element yang mempunya nama yang sama yaitu ”name”. Untuk mengatasi hal tersebut, sebuah namespace digunakan. Spesifikasi dari namespace akan digunakan oleh penulis dokumentasi untuk mengetahui schema atau DTD mana yang digunakan pada element tertentu. Namespace dapat diaplikasikan pada attribut dan juga pada elements.

8.3 Java APIs bagi XML

JAXP

JAXP atau Java API for XML Processing (JAXP) adalah sebuah fleksible API yang mendukung Anda untuk mendukung sembarang XML-complient parser didalam aplikasi Java Anda. Ia memiliki sebuah layer plugability, dimana Anda dapat menambahkan sembarang implementasi dari SAX atau DOM APIs.

JAXP juga telah mendukung namespaces, sehingga ia akan mendukung Anda untuk bekerja dengan schema XML dalam mengatasi permasalahan penamaan.

DOM API

Document Object Model (DOM) adalah sebuah struktur tree, dimana tiap node akan terdiri dari sebuah komponen dari struktur XML. DOM API mendukung Anda untuk membuat atau menghilangkan sebuah node didalam tree. Ia juga dapat digunakan untuk mengganti content dari node dan mengubah hierarki node. Oleh karena, sebuah tree me-representasikan keseluruhan XML data, DOM API membutuhkan banyak memori didalam runtime-nya.

SAX API

Simple API for XML (SAX) adalah sebuah event-based XML parser API. Ia akan membaca XML dokumen dari awal hingga akhir. Setiap saat bertemu dengan sebuah construction syntax, ia akan memberikan tanda (notify) untuk menjalankan program tersebut. Sebuah SAX parser akan memberikan tanda kepada aplikasi dengan jalan memanggil method dari interface ContentHandler.

Sebagai contoh, pada saat parser mencapai simbol kurang dari (<), ia akan memanggil method startElement. Pada saat bertemu dengan tag terakhir (sebuah slash yang diikuti dengan tanda lebih besar dari), hal ini disebut method endElement.

SAX sangatlah cepat dan efisien. Ia membutuhkan memori yang lebih sedikit daripada DOM, karena ia tidak mengharuskan untuk membuat sebuah struktur tree internal dari data-data XML. SAX hanya akan mengirimkan data setiap kali aplikasi ingin membaca data tersebut.

XLST API

Extensible Stylesheet Language Transformation (XSLT) standard mendefinisikan mekanisme untuk pengalamatan (addressing) data-data XML dan untuk menspesifikkan transformasi data.

Extensible Stylesheet Language (XSL) memiliki tiga sub-komponen:

XSL-FO – Formatting Objects standard. XSL-FO adalah sebuah standard yang menyediakan mekanisme untuk mendeskripsikan aspek-aspek dari sebuah object misalnya ukuran huruf dan layout halaman. Sub komponen ini tidak tercover didalam JAXP.

XSLT – adalah sebuah transformation language dimana Anda diharapkan dapat mendefisinikan sendiri transformasi dari sebuah XML ke format yang lain seperti HTML.

Xpath – adalah sebuah language spesification, dimana Anda diharapkan dapat menspesifikasikan sendiri bagian-bagian dari struktur XML yang direference setiap saat. Xpath adalah sebuah mekanisme pengalamatan yang mendukung Anda untuk mendefinisikan sebuah path kepada element.

JAX-RPC

Java API for XML-based RPC (JAX-RPC) adalah sebuah teknologi untuk membangun web services dan client web service yang dapat mengaktifkan remote procedure calls (RPC). RPC model mendukung client untuk mengeksekusi procedure pada remote system.

Didalam JAX-RPC, remote procedure calls dan hasilnya direpresentasikan pada protocol berbasis XML. SOAP adalah salah satu protokol yang menggunakan JAX-RPC. JAX-RPC akan meng-convert API pemanggilan dan beraksi dari dan untuk SOAP/XML messages. Hal ini akan menyebabkan SOAP menjadi lebih mudah dan menghilangkan kekompleksan.

8.4 Web Services

Web Services Messaging

W3C mendefinisikan web service sebgai "sebuah software aplikasi yang dapat teridentifikasi oleh URI dan memiliki interface yang didefiniskan, dideskripsikan, dan dimengerti oleh XML dan juga mendukung interaksi langsung dengan software aplikasi yang lain dengan menggunakan message berbasis XML melalui protokol internet".

Web service adalah sebuah sofware aplikasi yang tidak terpengaruh oleh platform, ia akan menyediakan method-method yang dapat diakses oleh network. Ia juga akan menggunakan XML untuk pertukaran data, khususnya pada dua entities bisnis yang berbeda.

Beberapa karakteristik dari web service adalah:

- Message-based
- Standards-based
- Programming language independent
- Platform-neutral

Beberapa key standard didalam web service adalah: XML, SOAP, WSDL and UDDI.

SOAP (Simple Object Access Protocol) adalah sebuah XML-based mark-up language untuk pergantian pesan diantara aplikasi-aplikasi. SOAP berguna seperti sebuah amplop yang digunakan untuk pertukaran data object didalam network. SOAP mendefinisikan empat aspek didalam komunikasi: Message envelope, Encoding, RPC call convention, dan bagaimana menyatukan sebuah message didalam protokol transport.

Sebuah SOAP message terdiri dari SOAP Envelop dan bisa terdiri dari attachments atau tidak memiliki attachment. SOAP envelop tersusun dari SOAP header dan SOAP body, sedangkan SOAP attachment membolehkan non-XML data untuk dimasukkan kedalam SOAP message, di-encoded, dan diletakkan kedalam SOAP message dengan menggunakan MIME-multipart.

Web Services Description

WSDL (Web Services Description Language) adalah sebuah XML-based language untuk mendeskripsikan XML. Ia menyediakan service yang mendeskripsikan service request dengan menggunakan protokol-protokol yang berbeda dan juga encoding. Ia akan memfasilitasi komunikasi antar aplikasi. WSDL akan mendeskripsikan apa yang akan dilakukan oleh web service, bagaimana menemukannya dan bagaimana untuk mengoperasikannya.

Spesifikasi WSDL mendefinisikan tujuh tipe element:

- Types : element untuk mendefinisikan tipe data. Mereka akan mendefinisikan tipe data (seperti string atau integer) dari element didalam sebuah message.
- Message : abstract, pendefinisian tipe data yang akan dikomunikasikan.
- Operation : sebuah deskripsi abstract dari sebuah action yang didukung oleh service.
- Port Type : sebuah koleksi abstract dari operations yang didukung oleh lebih dari satu endpoints.

- Binding : mendefinisikan penyatuan dari tipe port (koleksi dari operasi-operasi) menjadi sebuah protokol transport dan data format (ex. SOAP 1.1 pada HTTP). Ini adalah sebuah protokol konkret dan sebuah spesifikasi data format didalam tipe port tertentu.
- Port : mendefinisikan sebuah komunikasi endpoint sebagai kombinasi dari binding dan alamat network. Bagi protokol HTTP, ini adalah sebuah bentuk dari URL sedangkan bagi protokol SMTP, ini adalah sebuah form dari email address.
- Service : satu set port yang terkorelasi atau suatu endpoints.

WSDL mendefinisikan service sebagai sebuah koleksi dari endpoints network. Sebuah definisi abstrak dari endpoints dan messages adalah ia bersifat terpisah dari pembangunan network atau penyatuan data format. Pembagian ini menyebabkan penggunaan kembali abstract description dari data yang akan dipertukarkan (message exchange) dan abstract collection dari operasi (ports).

Protokol konkret dan spesfikasi data format bagi tipe port tertentu menentukan binding yang dapat digunakan kembali(reusable). Sebuah port adalah sebuah network address yang dikombinasikan reusable binding; sebuah service adalah koleksi dari port-port.

Web Service Discovery

UDDI (Universal Description, Discover and Integration) adalah sebuah service registry bagi pengalokasian web service. UDDI mengkombinasikan SOAP dan WSDL untuk pembentukan sebuah registry API bagi pendaftaran dan pengenalan service. Ia menyediakan sebuah area umum dimana sebuah organisasi dapat mengiklankan keberadaan mereka dan service yang mereka berikan (web service).

UDDI adalah sebuah framework yang mendefinisikan sebuah XML-based registry dimana sebuah organisasi dapat meng-upload informasi mengenai service yang mereka berikan. XML-based registry berisi nama-nama dari organisasi tsb, beserta service dan deskripsi dari service yang mereka berikan.

8.5 J2ME Web Services API (WSA)

Web Services API(JSR 172) menyediakan fungsi-fungsi tambahan yang mendukung web service. API ini berisi fungsi-fungsi dasar yang digunakan dalam web service client seperti remote web invocation dan XML parsing. WSA hanya merupakan subset dari J2SE API.

WSA hanya mendukung pemakaian dari web service. Hal ini berarti, aplikasi J2ME dengan menggunakan WSA hanya dapat menjadi konsumen dari service dan bukan merupakan producer service. UDDI juga tidak disupport oleh JSR 172. JAXP subset disupport oleh WSA spesification yang didukung oleh SAX. Ia tidak berisi dukungan bagi DOM dan XSLT.

Dokumen parsing XML menggunakan SAX

Untuk mendapatkan instance dari SAXParser, pertama-tama kita harus mendapatkan instance dari SAXParserFactory:

```
// Dapatkan instance dari SAX parser factory
SAXParserFactory factory = SAXParserFactory.newInstance();
```

Kemudian, kita akan mendapatkan instance dari SAX Parser:

```
SAXParser parser = factory.newSAXParser();
```

Pada akhirnya, kita akan membuat sebuah source input dan menggunakan event handler untuk mem-parser. Untuk mempermudah contoh, kita akan membuat sebuah stream dari sebuah String. Biasanya, hal ini bisa terlaksana dengan menggunakan resource dari file atau network. Contoh yang kita buat juga tidak memiliki sebuah GUI- ia hanya mencetak sebuah outcome dari sebuah parsing menjadi sebuah standard output.

```
ByteArrayInputStream stream =
  new ByteArrayInputStream(sampleXML.getBytes());
InputSource inputSource = new InputSource(stream);
SaxEventHandler handler = new SaxEventHandler();
parser.parse(inputSource, handler);
```

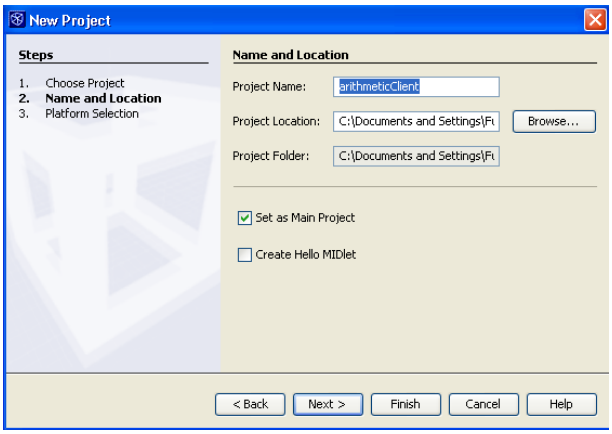
Kode berikut ini adalah kode untuk Event Handler:

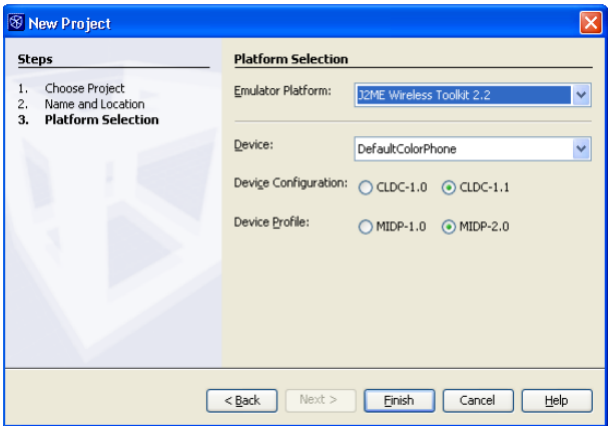
```
import java.util.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class SaxEventHandler extends DefaultHandler {
  private boolean finished;
  private Stack qNameStack = new Stack();
  private Vector tasks = new Vector();
  private static final String TASKS_ELEMENT = "tasks";
  private static final String TASK_ELEMENT = "task";
  private static final String NAME_ELEMENT = "name";
  private static final String ID_ATTRIBUTE= "id";

  public void startDocument() throws SAXException {
    finished = false;
    qNameStack.removeAllElements();
    tasks.removeAllElements();
  }

  public void endDocument() throws SAXException {
```

```
    finished = true;
    // Akhir dari dokumen, sekarang lakukan proses untuk memparsing object
    for (int i=0; i<tasks.size(); i++) {
      Task task = (Task) tasks.elementAt(i);
      System.out.println("Task: " + task.name);
    }
  }

  public void startElement(String uri, String localName, String qName,
  Attributes attributes) throws SAXException {
    if (TASK_ELEMENT.equals(qName)) {
      // Dapatkan id attribute dari sebuah task
      String id = attributes.getValue(ID_ATTRIBUTE);
      Task task = new Task(id);
      tasks.addElement(task);
    }
    qNameStack.push(qName);
  }

  public void characters(char[] ch, int start, int length)
  throws SAXException {
    String name = new String(ch, start, length);
    String qName = (String) qNameStack.peek();
    if (NAME_ELEMENT.equals(qName))
      if (tasks.size() > 0) {
        Task task = (Task) tasks.lastElement();
        task.name = name;
      }
  }

  public void endElement(String uri, String localName, String qName)
  throws SAXException {
    qNameStack.pop();
  }
}
```

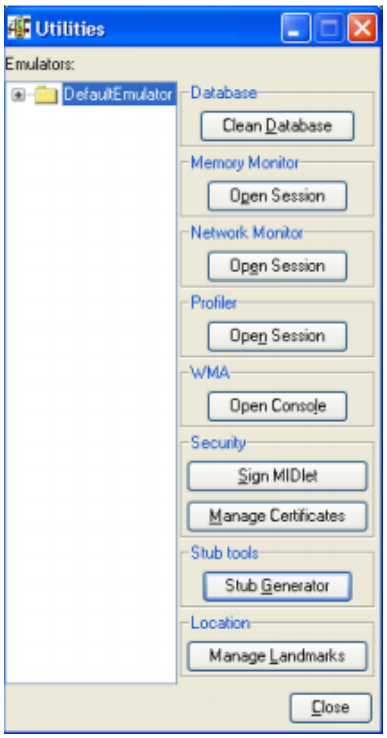
8.6 Membuat sebuah Mobile Web Service Client

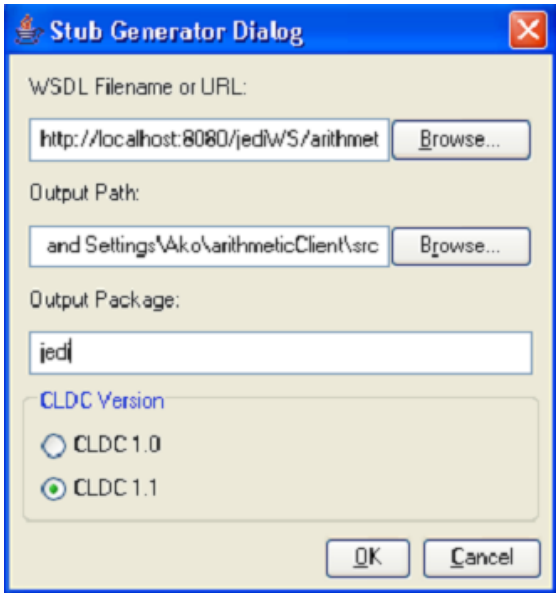
Kita akan membuat secara sederhana aplikasi mobile dan kita akan menamakannya "arithmeticClient".





Kemudian, kita akan menggunakan stub generator (Tools->Java Platform Manager -> Wireless Toolkit -> Tools and Extensions -> Open Utilities -> Stub Generator):





Sebelum kita dapat membuat sebuah web service client, kita harus memiliki sebuah file WSDL atau lokasi URL dimana terdapat service yang dapat kita pergunakan. Seperti yang telah disebutkan dalam bagian sebelumnya, JSR 172 tidak mendukung UDDI atau automatic discovery bagi service.

Masukkan lokasi WSDL dari sebuah web service. Path dari outputnya harus menjadi path dari direktori sumber project Anda (PROJECT_PATH/src). Sebuah stub generator tidak menerima package tanpa penamaan. Oleh karena itu, Anda harus meng-inputkan sebuah nama package. Kemudian, kita akan membuat sebuah Midlet yang akan menggunakan stub yang telah kita buat untuk mengakses sebuah web service:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import jedi.*;
public class WSClient extends MIDlet {
  private ArithmeticSEI_Stub stub;
  public void startApp() {
    System.out.println("Creating stub...");
    stub = new ArithmeticSEI_Stub();
    System.out.println("Invoking operation...");
    WebServiceClient client = new WebServiceClient();
    Thread thread = new Thread(client);
    thread.start();
  }

  public void pauseApp() {}

  public void destroyApp(boolean unconditional) {}

  class WebServiceClient implements Runnable {
    public void run() {
      try {
        int reply = stub.multiply(4, 5);
        System.out.println("Reply: " + reply);
      } catch (java.rmi.RemoteException rex) {
        System.out.println("Remote Exception: " + rex.getMessage());
      }
    }
  }
}
```

Bab 9

Optimisasi

9.1 Tujuan

Setelah menyelesaikan bab ini, pelajar diharapkan menguasai :

- Mengetahui teknik yang berbeda dalam optimisasi aplikasi mobile

9.2 Optimisasi

Sebelum benar-benar melakukan setiap optimisasi pada program Anda, Anda seharusnya perlu memastikan bahwa package software anda memiliki kualitas yang baik. Anda perlu meletakkan optimisasi dalam agenda Anda. Beberapa teknik yang dibahas pada bab ini seharusnya dapat membantu dalam menghindari beberapa kesalahan pemrograman.

9.3 Eksekusi program

9.3.1 Gunakan StringBuffer sebagai pengganti String.

Anda perlu ingat bahwa pada Java, object String bersifat absolut atau abadi. Menggunakan method String menciptakan suatu object String terpisah. Perangkaian String yang sederhana menciptakan suatu object String ganda (kecuali jika String itu bersifat konstan dan kompiler cukup pandai untuk menggabungkan mereka pada proses compile berlangsung). Menggunakan StringBuffer tidak hanya mengoptimalkan runtime program Anda (lebih sedikit menimbulkan object runtime), itu juga mengoptimalkan pemakaian memori (lebih sedikit object String dibuat).

String	StringBuffer

`String a, b, c;`
`...`
`String message =`
`  "a=" + a + "\n"`
`  + "b=" + b + "\n"`
`  + "c=" + c + "\n";`
	`String a, b, c;`
	`...`
	`StringBuffer message = new StringBuffer(255);`
	`message.append("a="); message.append(a);`
	`message.append("\n"); message.append("b=");`
	`message.append(b);  message.append("\n");`
	`message.append("c="); message.append(c);`
	`message.append("\n");`

9.3.2 Gunakan clipping area dalam menggambar

Menggunakan Graphics.setClip() akan mengurangi waktu eksekusi karena Anda hanya akan menggambar nomor-nomor yang optimal dari pixel-pixel di layar. Ingat, bahwa menggambar grafik pada layar meminta banyak

terminologi pada waktu eksekusi. Mengurangi banyaknya pixel-pixel untuk digambar akan sangat mempengaruhi kinerja runtime program Anda.

```
Graphics g;
int x1, y1, x2, y2;
...
g.setClip(x1, y1, x2, y2);
g.drawString("JEDI", x, y, Graphics.TOP | Graphics.HCENTER);
// Operasi menggambar yang lainnya...
```

9.3.3 Hindari modifier yang sama

Menggunakan modifier yang sama mengambil sesuatu tanpa diduga pada kecepatan eksekusi program Anda karena hal tersebut menimbulkan beberapa ukuran tambahan sehingga itu tidak akan diakses secara bersamaan.

9.3.4 Lewatkan parameter sesedikit mungkin

Ketika memanggil suatu method, penerjemah akan mendorong semua parameter ke atas tumpukan eksekusi. Melewatkan banyak parameter akan mempengaruhi kecepatan eksekusi dan pemakaian Heap Memory.

9.3.5 Mengurangi pemanggilan method

Memanggil method menghabiskan Heap Memory dan waktu eksekusi. Lihat subbab sebelumnya.

9.3.6 Menunda semua inisialisasi

Untuk mempercepat awal permulaan aplikasi, tunda semua inisialisasi yang sangat besar sampai mereka dibutuhkan. Jangan meletakkan inisialisasi dalam konstruktor MIDlet atau method startApp. Mempercepat waktu load sebuah aplikasi akan menambah penggunaan aplikasi Anda. Kebanyakan user akan meninggalkan aplikasi ketika aplikasi tersebut membutuhkan waktu yang lama untuk start up. Ingat bahwa waktu load aplikasi anda secara langsung mempengaruhi kesan pertama pengguna aplikasi Anda.

9.3.7 Gunakan array sebagai pengganti collection

Mengakses Array lebih cepat daripada menggunakan vektor.

9.3.8 Menggunakan variabel lokal

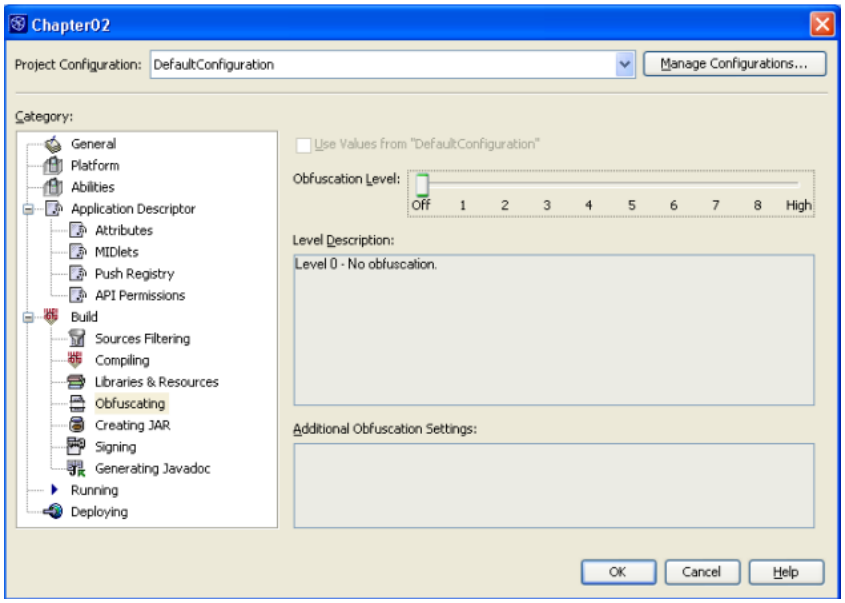
Hal tersebut lebih cepat mengakses variabel lokal daripada mengakses variabel instance.

9.4 Ukuran JAR

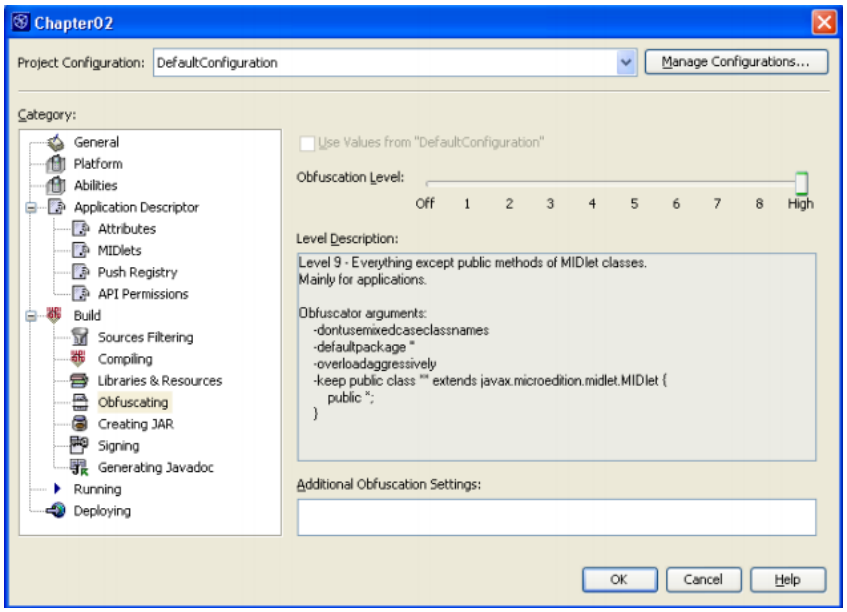
9.4.1 Gunakan Obfuscator

Tujuan utama obfuscator adalah untuk mengacak file class yang dikompile sehingga sulit untuk di decompile. Tetapi proses obfuscator juga mengurangi ukuran sebuah aplikasi. Salah satu method yang digunakan oleh obfuscator adalah memberi nama baru pada class menjadi sebuah nama. Karena obfuscator melakukan hal ini berdasar kepada modifier dari method-method. Jika method memiliki private atau protected modifier, lalu itu dapat diasumsikan aman ketika method ini tidak akan digunakan oleh package lainnya dan oleh karena itu dapat diberi nama baru kembali.

Netbeans Mobility Pack datang dengan satu obfuscator. Dia tidak diaktifkan sebagai default. Buka tab property dan klik pada cabang "Obfuscating" :



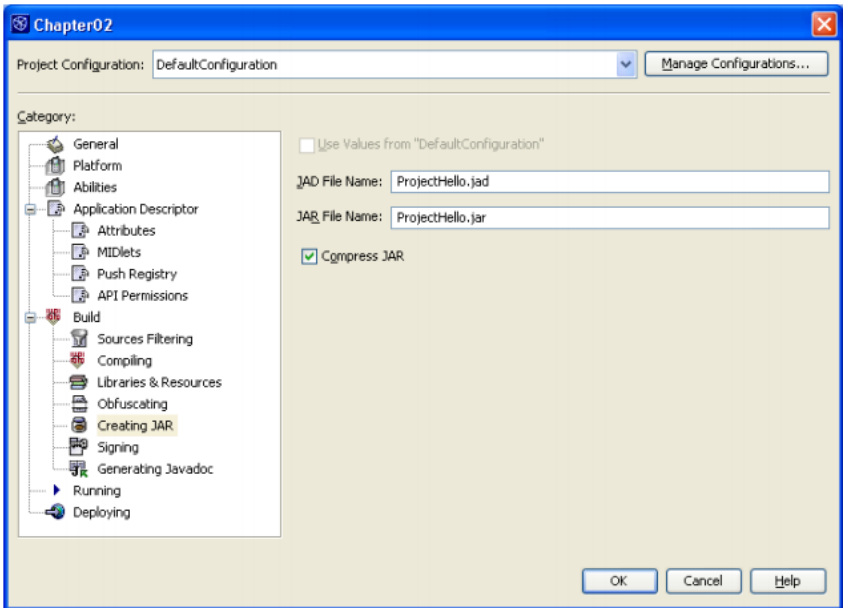
Ada sepuluh tingkat obfuscation, dari tanpa obfuscation sampai ke obfuscation yang paling agresif.



9.4.2 Memadatkan file JAR Anda

Pastikan bahwa sebelum mendistribusikan aplikasi Anda, Anda memampatkan file akhir JAR untuk distribusi. Sebuah file JAR adalah sebuah arsip ZIP, dan suatu arsip ZIP mempunyai beberapa tingkat tekanan (termasuk tanpa tekanan). NetBeans Mobility Pack tidak mendukung tingkat tekanan.

Untuk mengatur pilihan tekanan JAR, buka halaman properti dari aplikasi dan pilih cabang "Creating JAR". Centang radio box "Compress JAR" untuk memampatkan file JAR proyek Anda. Jangan lupa untuk membangun kembali proyek Anda.



9.4.3 Hindari membuat class yang tidak perlu

Ini akan tampak berlawanan untuk prinsip berorientasi object, tapi apakah Anda mengetahui bahwa suatu class kosong yang sederhana seperti ini :

```
public class EmptyClass {
  public EmptyClass(){}
}
```

akan dikompile menjadi file class dengan ukuran file sebesar 250kb.

Anda dapat mencoba mengkompile class kosong ini dan buktikan sendiri. Netbeans Mobility Pack menyimpan package file JAR di dalam folder distribusi dibawah folder proyek. Anda dapat merubah nama file .jar menjadi file .zip dan buka dengan program ZIP favorit Anda untuk melihat ukuran dari file class yang Anda kompile.

9.4.4 Hindari membuat interfaces

Teknik ini berkaitan dengan teknik sebelumnya. Memiliki banyak class dan interfaces akan menambahkan lebih ukuran file (kilobytes) dalam aplikasi Anda.

9.4.5 Hindari inner dan anonymous class

Sama seperti diatas. Inner class adalah semua class yang sama. Anonymous class mungkin tidak memiliki nama, tetapi mereka mengambil ruang yang sama untuk definisi class.

9.4.6 Gunakan satu Listener untuk object yang ganda

Ini akan mengurangi banyaknya class dalam aplikasi Anda. Buatlah MIDlet Anda mengimplementasikan CommandListener interface sehingga membantu anda memangkas package Anda oleh satu class (Dimana mengurangi 250 + byte).

9.4.7 Gunakan package default (package tanpa nama)

Didalam permintaan kita untuk package berukuran kecil, memendekkan (atau tidak menggunakan) nama package tersebut mendukung pengurangan byte.

9.4.8 Batasi penggunaan dari initializer static

Menggunakan inisialisasi static seperti ini :

```
int[] tones = { 64, 63, 65, 76, 45, 56, 44, 88 };
```

Akan dikompile oleh kompiler Java menjadi pernyataan berikut :

```
tones[0] = 64; tones[1] = 63; tones[2] = 65; tones[3] = 76;
tones[4] = 45; tones[5] = 56; tones[6] = 44; tones[7] = 88;
```

Contoh ini menggambarkan hanya delapan anggota array. Bayangkan jika inisialisasi ratusan nilai menggunakan statemen terpisah. Hal tersebut akan menjadikan overhead pada ukuran aplikasi Anda. Sebagai salah satu alternatif, Anda dapat menggunakan method getResourceAsStream() untuk mendapatkan nilai dari sebuah file atau menggunakan single string untuk menyimpan nilai array Anda.

9.4.9 Menggabungkan gambar ke dalam satu file

Memampatkan gambar lebih baik ketika di-kelompokkan menjadi satu file gambar. Karena memampatkan format gambar (contohnya PNG) adalah lebih spesifik untuk gambar daripada memampatkan method pengarsipan JAR. Ada teknik-teknik untuk mendapatkan gambar yang spesifik dari sebuah gambar yang besar yaitu dengan memotongnya.

9.4.10 Bereksperimen dengan memampatkan gambar

method tekanan(compressing) tidak diciptakan sama. Beberapa mungkin memampatkan lebih baik pada beberapa jenis gambar tetapi kadang memiliki rasio yang rendah dalam memampatkan jenis gambar yang lain. Pilih sebuah format gambar yang dapat meningkatkan rasio pemampatan gambar Anda. Terkadang, rasio pemampatan juga dipengaruhi oleh software pengolah gambar yang anda gunakan.

Cobalah bereksperimen dengan berbagai macam jenis software pengolah gambar untuk mendapatkan ukuran gambar yang lebih baik.

9.4.11 Gunakan class yang belum diinstal

Gunakan semua class yang bisa diterapkan yang tersedia pada platform yang anda gunakan. Buatlah class Anda sendiri yang tidak akan menambah ukuran aplikasi Anda, tetapi juga mengurangi stabilitas aplikasi Anda.

9.5 Jaringan

9.5.1 Gunakan thread yang terpisah

Gunakan thread yang terpisah untuk jaringan Anda yang berfungsi untuk menghindari screen lockups.

9.5.2 Memampatkan data jaringan

Menggunakan data yang dimampatkan untuk mengurangi lalu lintas jaringan dari aplikasi Anda. Hal ini akan membutuhkan client dan server Anda untuk menggunakan protokol dan method pemampatan yang sama.
Memampatkan XML akan memberikan rasio yang lebih baik karena data XML terwakili dalam suatu format teks.

9.5.3 Mengurangi lalu lintas jaringan

Karena komunikasi jaringan semakin lambat dan mahal, cobalah sebisa mungkin untuk memasukkan beberapa perintah kedalam satu permintaan jaringan. Ini akan mengurangi overhead yang dikenakan oleh protokol jaringan.

9.6 Penggunaan Memori

9.6.1 Gunakan struktur data ringkas

Gunakan struktur data memory yang sering digunakan. Array jarang bisa diwakili dengan cara lain tanpa mengkonsumsi jumlah yang sama dari memory. Ada tradeoff ketika mengoptimalkan untuk ukuran dan kecepatan. Menggunakan struktur data komplek akan mempengaruhi kecepatan eksekusi program Anda.

9.6.2 Membebaskan object yang tidak terpakai untuk garbage collection

Membebaskan object yang tak terpakai untuk garbage collection layar, koneksi jaringan, rekaman RMS. Menentukan variabel untuk menunjuk kepada object yang tak terpakai menjadi null dan akan memberi isyarat kepada garbage collector bahwa object ini aman untuk tidak di-load dari memory.

9.6.3 Jangan sering menggunakan layar on-the-fly

Tidak sering menggunakan object Screen (seperti Help dan about screen)on-the-fly akan banyak bebaskan banyak kebutuhan Anda yang menumpuk pada memory. Meski Anda harus membayar harga yaitu

loading yang lambat untuk screen tertentu. Layar ini akan diduga menimbun pada heap memory sementara mereka tidak digunakan untuk membantu dalam penghematan memory.

```
public void commandAction(Command c, Displayable d) {
  if (c == helpCommand)
    display.setCurrent(new HelpForm());
}
```