Współpraca z zewnętrznym oprogramowaniem: JDBC, JNI (Java Native Interface)

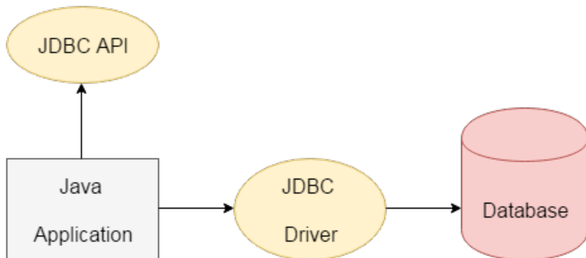
Robert A. Kłopotek
r.klopotek@uksw.edu.pl

Wydział Matematyczno-Przyrodniczy. Szkoła Nauk Ścisłych, UKSW

1.06.2017

Java JDBC

- Java JDBC (Java DataBase Connectivity) jest interfejsem Java API służącym do łączenia i wykonywania kwerend z bazą danych.
- JDBC API używa sterowników jdbc (JDBC driver) do łączenia się z bazą danych.
- Przed JDBC ODBC API był bazą danych API do łączenia i wykonywania kwerendy z bazą danych.
- ODBC API używa sterownika ODBC, który jest napisany w języku C (tzn. zależny od platformy i niezabezpieczony). Dlatego w Javie zdefiniowano własny interfejs API (JDBC API), który używa sterowników JDBC (napisanych w języku Java).
- Pakiety obsługujące JDBC to `java.sql` i `javax.sql`



Komponenty JDBC (1/2)

- **DriverManager**: klasa zarządzająca listą sterowników bazy danych. Odpowiada żądaniom połączenia z aplikacji Java z odpowiednim sterownikiem bazy danych przy użyciu podprotokołu sieci. Pierwszy sterownik rozpoznający pewien podprotokół w ramach JDBC będzie używany do nawiązania połączenia z bazą danych.
- **Driver**: interfejs obsługujący komunikację z serwerem bazy danych. Używa się go bardzo rzadko. Zamiast tego używamy obiektów **DriverManager**, które zarządzają obiektami tego typu.
- **Connection**: interfejs ze wszystkimi metodami kontaktowania się z bazą danych. Obiekt połączenia reprezentuje kontekst komunikacyjny, tzn. cała komunikacja z bazą danych odbywa się wyłącznie za pośrednictwem obiektu połączenia.

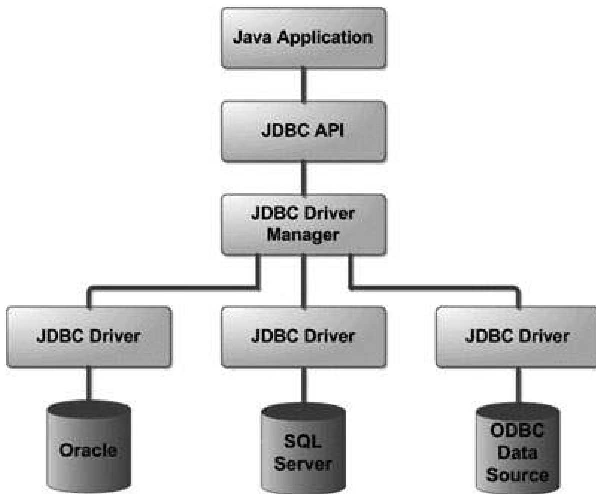
Komponenty JDBC (2/2)

- **Statement**: Używa się obiektów utworzonych z tego interfejsu, aby przesłać kwerendy SQL do bazy danych. Niektóre interfejsy pochodne akceptują parametry, aby móc wykonywać procedury składowane.
- **ResultSet**: Obiekty te przechowują dane pobrane z bazy danych po wykonaniu zapytania SQL przy użyciu obiektów **Statement**. Działają jako iterator, dzięki czemu można poruszać się w danych.
- **SQLException**: klasa wyjątków obsługująca wszelkie błędy występujące w aplikacji używającej baz danych.

Architektura JDBC

- Interfejs JDBC obsługuje zarówno modele dwuwarstwowych, jak i trójwarstwowych do przetwarzania baz danych, ale generalnie architektura JDBC składa się z dwóch warstw -
  - JDBC API: zapewnia połączenie z aplikacją do Managera JDBC .
  - JDBC Driver API: obsługuje połączenie JDBC Manager-to-Driver.
- Interfejs JDBC API używa menedżera sterowników i sterowników specyficznych dla bazy danych w celu zapewnienia przejrzystej łączności z heterogenicznymi bazami danych.
- Menedżer sterowników JDBC zapewnia, że do uzyskania dostępu do każdego źródła danych używany jest prawidłowy sterownik.
- Driver manager jest w stanie obsługiwać wiele równoczesnych sterowników podłączonych do wielu heterogenicznych baz danych.

Architektura JDBC



JDBC Driver

- Sterownik JDBC to składnik oprogramowania, który umożliwia aplikacji Java do współpracować z bazą danych.
- Są cztery typy sterowników JDBC:
  - Sterownik mostu JDBC-ODBC
  - Sterownik Native-API (częściowo w Javie)
  - Sterownik protokołu sieciowego (w pełni w Javie)
  - Cienki sterownik (w pełni w Javie)

JDBC-ODBC bridge driver

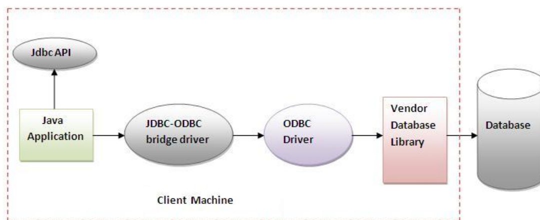
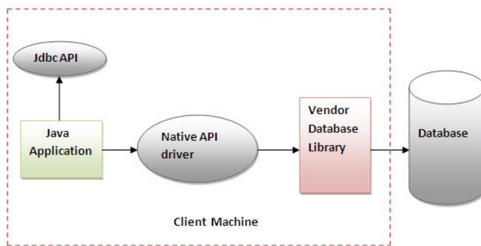


Figure- JDBC-ODBC Bridge Driver

- Zalety
  - łatwy w użyciu
  - można łatwo połączyć z dowolną bazą danych.
- Wady
  - Wydajność jest zmniejszona, ponieważ wywołanie metody JDBC jest konwertowane na wywołania funkcji ODBC.
  - Sterownik ODBC musi być zainstalowany na komputerze klienckim.

Native-API driver



- Zalety
  - Wydajność większa niż za pomocą sterownika mostu JDBC-ODBC
- Wady
  - Sterownik natywny musi być zainstalowany na każdym komputerze klienckim.
  - Biblioteka dostawcy musi być zainstalowana na komputerze klienckim.

Network Protocol driver

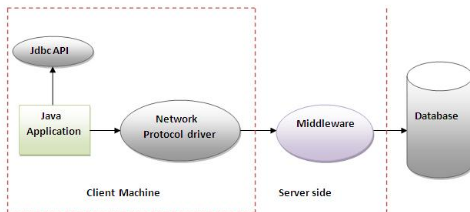


Figure- Network Protocol Driver

- Zalety
  - Żadna biblioteka strony klienta nie jest wymagana ze względu na serwer aplikacji, który może wykonywać wiele zadań, takich jak kontrola, równoważenie obciążenia, logowanie itp.
- Wady
  - Obsługa sieci jest wymagana na komputerze klienckim.
  - Wymaga oprogramowania specyficznego dla konkretnej bazy danych, który ma być wykonany w warstwie środkowym. Z tego powodu utrzymanie sterownika protokołu sieciowego staje się kosztowne.

Thin driver

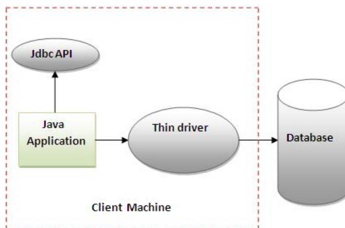


Figure- Thin Driver

- Zalety
  - Lepsza wydajność niż wszystkie inne sterowniki.
  - Żadne oprogramowanie nie jest wymagane po stronie klienta lub po stronie serwera.
- Wady
  - Sterowniki zależą od bazy danych.

Połączenie z bazą danych

- Aby używać standardowego pakietu JDBC, który umożliwia operacje SQL SELECT, INSERT, UPDATE i DELETE, należy dodać następujący import do kodu źródłowego

```
import java.sql.* ;  // Dla standartowych programów JDBC
import java.math.* ; // dla wsparcia typów BigDecimal i BigIntege
```

- Aby połączyć się z baza danych wystarczy 5 kroków
  - Zarejestrowanie klasy sterownika
  - Stworzenie połączenia
  - Stworzenie zapytana SQL
  - Wykonanie kwerendy (opcjonalnie pobranie wyniku)
  - Zamknięcie połączenia

Zarejestrowanie klasy sterownika

- Trzeba zarejestrować sterownik w programie przed jego użyciem.
- Rejestracja sterownika jest procesem, w którym plik klasy sterownika (np. Oracle, MySQL) jest załadowany do pamięci, dzięki czemu może być wykorzystany jako implementacji interfejsów JDBC.
- Wystarczy zrobić rejestrację tylko raz w programie.
- Sterownik można zarejestrować na jeden z dwóch sposobów.
  - Poprzez metodę Class.forName()
  - Poprzez metodę DriverManager.registerDriver()

Rejestrowanie - Class.forName()

- Najczęstszym podejściem do zarejestrowania sterownika jest użycie metody Class.forName() do dynamicznego ładowania pliku klasy sterownika do pamięci, co automatycznie rejestruje ją.
- Metoda ta jest preferowana, ponieważ umożliwia konfigurowanie i przenoszenie sterowników.
- Przykład:

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

Rejestrowanie - DriverManager.registerDriver()

- Drugim podejściem, którego można użyć do zarejestrowania sterownika, jest użycie statycznej metody DriverManager.registerDriver ().
- Jeśli używamy JVM niezgodnego z JDK, należy użyć metody registerDriver(), takiej jak ta dostarczona przez firmę Microsoft.
- Przykład:

```
try {
   Driver myDriver = new oracle.jdbc.driver.OracleDriver();
   DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

Tworzenia połączenia z bazą danych

- Po załadowaniu sterownika można nawiązać połączenie przy użyciu metody DriverManager.getConnection()
- Trzy przeciążone metody:
  - `getConnection(String url)`
  - `getConnection(String url, Properties prop)`
  - `getConnection(String url, String user, String password)`
- Przykład:

```
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

Popularne Connection String

Typ Bazy danych	DriverClassName	Connection String
MySQL	com.mysql.jdbc.Driver	"jdbc:mysql://<hostname>:<portNumber>/<databaseName>"
PostgreSql	org.postgresql.Driver	"jdbc:postgresql://<hostname>:<portNumber>/<databaseName>"
SQLServer	com.microsoft.sqlserver.jdbc.SQLServerDriver	"jdbc:sqlserver://<hostname>:<portNumber>;<databaseName>"
Oracle	oracle.jdbc.driver.OracleDriver	"jdbc:oracle:thin:@<hostname>:<portNumber>:<databaseName>"
DB2	COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver	"jdbc:db2://<hostname>:<portNumber>/<databaseName>"
"DB2(on As/400)"	com.ibm.as400.access.AS400JDBCDriver	"jdbc:as400://<hostname>:<portNumber>/<databaseName>;"

Tworzenia i wykonywanie zapytań

- Do tworzenia zapytań SQL służy metoda createStatement() interfejsu Connection. Obiekt klasy Statement jest odpowiedzialny za wykonywanie kwerend z bazą danych.

```
public Statement createStatement()throws SQLException
```

- Do wykonywania kwerend w bazie danych służy metoda executeQuery () interfejsu Statement. Metoda zwraca obiekt ResultSet, który może być użyty do pobrania wszystkich rekordów tabeli.

```
public ResultSet executeQuery(String sql)throws SQLException
```

- Przykład:

```
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("SELECT id_osoby, imie FROM osoby");
while(rs.next()){
  System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

Przykład - ile miast

```
import java.sql.*;

public class PokazWoj {
  public static void main(String[] args){
    try{
      Class.forName("com.mysql.jdbc.Driver");
      Connection con=DriverManager.getConnection(
            "jdbc:mysql://localhost/bazy_danych?" +
            "user=user&password=pass");
      Statement stmt=con.createStatement();
      ResultSet rs=stmt.executeQuery(
            "SELECT␣w.kod_woj,␣count(id_miasta)␣AS␣ile_miast␣" +
            "FROM␣woj␣w␣" +
            "LEFT␣JOIN␣miasta␣m␣ON␣m.kod_woj=w.kod_woj␣" +
            "GROUP␣BY␣w.kod_woj;");
      while(rs.next())
        System.out.println(rs.getString(1)+"␣"+rs.getInt(2));
      con.close();
    }catch(Exception e){ System.out.println(e);}
  }
}
```

Interakcja z bazą danych

- Po uzyskaniu połączenia możemy współdziałać z bazą danych. Interfejs JDBC **Statement**, **CallableStatement** i **PreparedStatement** definiuje metody i właściwości umożliwiające wysyłanie poleceń `T-SQL` lub `PL/SQL` i odbieranie danych z bazy danych.
- **Statement** - Używane do ogólnego dostępu do bazy danych. Przydatne podczas korzystania ze statycznych instrukcji SQL w czasie wykonywania. Interfejs **Statement** nie może akceptować parametrów.
- **PreparedStatement** - do wielokrotnie wykorzystywanych instrukcji SQL. Interfejs **PreparedStatement** akceptuje parametry wejściowe w czasie wykonywania.
- **CallableStatement** - używany, aby uzyskać dostęp do procedur przechowywanych w bazie danych. Interfejs **CallableStatement** może również akceptować parametry wejściowe w czasie wykonywania.

Obiekt klasy Statement

- Zanim będzie można użyć obiektu Statement do wykonania instrukcji SQL, należy go utworzyć, korzystając z metody createStatement() obiektu Connection
- Metody
  - `boolean execute (String SQL)` - SQL DDL
  - `int executeUpdate (String SQL)` - INSERT, UPDATE lub DELETE
  - `ResultSet executeQuery (String SQL)` - SELECT

Obiekt klasy PreparedStatement

- Interfejs PreparedStatement rozszerza interfejs Statement, co zapewnia dodatkową funkcjonalność z kilkoma zaletami w porównaniu ze zwykłym obiektem Statement.
- Wszystkie parametry w JDBC są reprezentowane przez symbol "?", znany jako marker parametrów. Trzeba podać wartości dla każdego parametru przed wykonaniem kwerendy SQL
- Metody setXXX () łączą wartości z parametrami, gdzie XXX reprezentuje typ danych typu Java wartości, która ma zostać powiązana z parametrem wejściowym. Jeśli zapomnisz dostarczyć wartości, otrzymasz SQLException.
- Każdy znacznik parametrów jest określany przez jego pozycję porządkową. Pierwszy znacznik przedstawia pozycję 1, następną pozycję 2, i tak dalej. Ta metoda różni się od indeksów tablic Java, która zaczynają się od 0
- Przykład:

```
PreparedStatement ps = con.prepareStatement(
            "SELECT * FROM osoby WHERE id_osoby=?");
ps.setInt(1,id_osoby);
```

Obiekt klasy CallableStatement

- Istnieją trzy typy parametrów: IN, OUT i INOUT. Obiekt PreparedStatement używa tylko parametru IN. Obiekt CallableStatement może używać wszystkich trzech.
- Przed wykonaniem instrukcji należy powiązać wartości z wszystkimi parametrami, jeśli nie to otrzymamy SQLException.
- Jeśli masz parametry IN, postępuj zgodnie z tymi samymi zasadami i technikami, które dotyczą obiektu PreparedStatement: użyj metody setXXX (), która odpowiada typowi danych języka Java.
- Podczas korzystania z parametrów OUT i INOUT należy użyć dodatkowej metody interfejsu CallableStatement, registerOutParameter (). Metoda RegisterOutParameter () wiąże typ danych JDBC, z typem danych, od którego oczekiwana jest wartość zwracana procedury składowanej.
- Aby powiązać typ OUT nalezy uzyc funckji setXXX()

Klasa ResultSet

- Obiekt ResultSet utrzymuje kursor wskazujący bieżący wiersz w zestawie wynikowym. Termin "zestaw wyników" odnosi się do danych wierszy i kolumn zawartych w obiekcie ResultSet.
- Są trzy typy metod interfejsu ResultSet
  - metody do nawigowania
  - metody Get - służą do odczytu danych
  - metody Update - aktualizacja rekordów w bazie danych!
- JDBC udostępnia następujące metody połączeń do tworzenia kwerend z odpowiednim ResultSet
  - `createStatement(int RSType, int RSConcurrency)`
  - `prepareStatement(String SQL, int RSType, int RSConcurrency)`
  - `prepareCall(String sql, int RSType, int RSConcurrency)`

Typy ResultSet

- Możliwe wartości RSType
  - `ResultSet.TYPE_FORWARD_ONLY` - (domyślna) Kursor może poruszać się tylko do przodu.
  - `ResultSet.TYPE_SCROLL_INSENSITIVE` - Kursor może przewijać do przodu i do tyłu, a zestaw wyników nie jest wrażliwy na zmiany wprowadzone przez inne osoby do bazy danych, które wystąpiły po utworzeniu zestawu wyników.
  - `ResultSet.TYPE_SCROLL_SENSITIVE` - Kursor może przewijać do przodu i do tyłu, a zestaw wyników jest wrażliwy na zmiany wprowadzone przez inne do bazy danych, które wystąpiły po utworzeniu zestawu wyników.
- Możliwe wartości RSConcurrency
  - `ResultSet.CONCUR_READ_ONLY` - (domyślna) zbiór wynikowy jest tylko do odczytu
  - `ResultSet.CONCUR_UPDATABLE` - zbiór wynikowy można zmieniać i aktualizacje nanosić do bazy danych!

Metody aktualizacji ResultSet

- `public void updateXXX(int columnIndex, XXX s)` - Ustawia wartość kolumny dla danego wiersza, na który wskazuje kursor
- `public void updateXXX(String columnName, XXX s)` - Ustawia wartość kolumny dla danego wiersza, na który wskazuje kursor
- `public void updateRow()` - Aktualizuje bieżący wiersz, aktualizując odpowiedni wiersz w bazie danych.
- `public void deleteRow()` - Usuwa bieżący wiersz z bazy danych
- `public void refreshRow()` - Odświeża dane w zestawie wyników w celu odzwierciedlenia ostatnich zmian w bazie danych.
- `public void cancelRowUpdates()` - Anuluje wszystkie aktualizacje wprowadzone w bieżącym wierszu.
- `public void insertRow()` - Wstawia wiersz do bazy danych. Ta metoda może być wywołana tylko wtedy, gdy kursor wskazuje na wiersz do wstawiania.

Przykład - ResultSet wstawianie (1/2)

```
import java.sql.*;
public class Rsmd{
 public static void main(String args[]){
  try{
   Class.forName("com.mysql.jdbc.Driver");
   Connection con=DriverManager.getConnection(
         "jdbc:mysql://localhost/bazy_danych?" +
         "user=user&password=pass");
   Statement stmt = con.createStatement(
              ResultSet.TYPE_SCROLL_INSENSITIVE,
              ResultSet.CONCUR_UPDATABLE);
   String sql_query="SELECT␣*␣FROM␣osoby";
   ResultSet rs=stmt.executeQuery(sql_query);
   printRs(rs);
   rs.moveToInsertRow(); // przestawienie kursora
   rs.updateInt("id_osoby",10);
   rs.updateInt("id_miasta",4);
   rs.updateString("imie","Jan");
   rs.updateString("nazwisko","Kowalski");
   rs.insertRow(); // wstawienie wiersza do bazy danych
   printRs(rs);
   con.clo
```

Przykład - ResultSet wstawianie (2/2)

```
public class Rsmd{
 ...
 private static void printRs(ResultSet rs) throws SQLException{
  rs.beforeFirst();
  while(rs.next()){
   int id_osoby  = rs.getInt("id_osoby");
   int id_miasta = rs.getInt("id_miasta");
   String imie = rs.getString("imie");
   String nazwisko = rs.getString("nazwisko");

   System.out.print(id_osoby+ "\t" + id_miasta
                    +"\t" + imie + "\t"+nazwisko);
   System.out.println();
  }
  System.out.println();
 }
}
```

Przykład - ResultSet wstawianie (wydruk)

```
1 1 Maciej Stodolski
2 2 Jacek Korytkowski
3 3 Mis Nieznany
4 4 Krol Neptun
5 2 Juz Niepracujacy

1 1 Maciej Stodolski
2 2 Jacek Korytkowski
3 3 Mis Nieznany
4 4 Krol Neptun
5 2 Juz Niepracujacy
10 4 Jan Kowalski
```

Metadane JDBC

- Interfejs ResultSetMetaData
  - `public int getColumnCount()`
  - `public String getColumnName(int index)`
  - `public String getColumnTypeName(int index)`
  - `public String getTableName(int index)`
- Interfejs DatabaseMetaData
  - `public String getDriverName()`
  - `public String getDriverVersion()`
  - `public String getUserName()`
  - `public String getDatabaseProductName()`
  - `public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)`

Przykład - ResultSetMetaData

```
import java.sql.*;
public class Rsmd{
 public static void main(String args[]){
  try{
   Class.forName("com.mysql.jdbc.Driver");
   Connection con=DriverManager.getConnection(
         "jdbc:mysql://localhost/bazy_danych?" +
         "user=user&password=pass");
   PreparedStatement ps=con.prepareStatement(
              "SELECT␣imie,␣nazwisko␣FROM␣osoby");
   ResultSet rs=ps.executeQuery();
   ResultSetMetaData rsmd=rs.getMetaData();
   System.out.println("Ilosc␣kolumn:␣"+rsmd.getColumnCount());
   System.out.println("Nazwa␣pierwszej␣kolumny:␣"
               +rsmd.getColumnName(1));
   System.out.println("Typ␣pierwszej␣kolumny:␣"
               +rsmd.getColumnTypeName(1));
   System.out.println("Mapowanie␣typu␣pierwszej␣kolumny:␣"
               +rsmd.getColumnClassName(1));
   con.close();
  }catch(Exception e){ System.out.println(e);}
 }
}
```

Przykład - ResultSetMetaData (wydruk)

```
Ilosc kolumn: 2
Nazwa pierwszej kolumny: imie
Typ pierwszej kolumny: VARCHAR
Mapowanie typu pierwszej kolumny: java.lang.String
```

Przykład - DatabaseMetaData

```
import java.sql.*;
public class Dbmd{
 public static void main(String args[]){
  try{
   Class.forName("com.mysql.jdbc.Driver");

   Connection con=DriverManager.getConnection(
         "jdbc:mysql://localhost/bazy_danych?" +
         "user=user&password=pass");
   DatabaseMetaData dbmd=con.getMetaData();
   System.out.println("Driver␣Name:␣"+dbmd.getDriverName());
   System.out.println("Driver␣Version:␣"+dbmd.getDriverVersion());
   System.out.println("UserName:␣"+dbmd.getUserName());
   System.out.println("Database␣Product␣Name:␣"
                     +dbmd.getDatabaseProductName());
   System.out.println("Database␣Product␣Version:␣"
                     +dbmd.getDatabaseProductVersion());
   con.close();
  }catch(Exception e){ System.out.println(e);}
 }
}
```

Przykład - DatabaseMetaData (wydruk)

```
Driver Name: MySQL-AB JDBC Driver
Driver Version: mysql-connector-java-5.0.5
( $Date: 2007-03-01 00:01:06 +0100 (Thu, 01 Mar 2007) $
, $Revision: 6329 $ )
UserName: user@localhost
Database Product Name: MySQL
Database Product Version: 5.7.17-log
```

Java Native Interface

- JNI jest mechanizmem, który pozwala
  - programowi Java na wywołanie funkcji w programie C lub C ++.
  - programowi C lub C ++ na wywołania metody w programie Java
- Powody używania metod natywnych
  - Uzyskanie dostępu do funkcji systemu, które można łatwiej obsługiwać w języku C lub C ++.
  - Potrzebny dostęp do starszego kodu, który został dobrze przetestowany.
  - Potrzebujesz wydajności oferowanej przez C a której Java (jeszcze) nie ma
- NAJWAŻNIEJSZE: kod natywny nie jest przenośny!

Jak napisać aplikację JNI?

- Kod Java
  - Zadeklaruj metodę używając modyfikatora **native**, która nie ma ciała `private native void sayHello()`
  - Upewnij się, że biblioteka współdzielona, która ma zostać utworzona później, jest ładowana przed wywołaniem metody natywnej `System.loadLibrary("hello");`
  - Zwykle biblioteka jest ładowana w bloku statycznym w klasie, która wywołuje metodę natywną.
  - Skompiluj klasę `javac HelloJNI.java`
- Stwórz nagłówek C zawierający prototypy funkcji dla natywnych metod `javah HelloJNI`
- Napisz implementację C natywnych metod wykorzystujących zniekształcone nazwy i dodatkowe parametry.
- Skompiluj kod C i utwórz bibliotekę współdzieloną

```
gcc -Wl,--add-stdcall-alias -I"%JAVA_HOME%\include"
-I"%JAVA_HOME%\include\win32" -shared -o hello.dll HelloJNI.c
```

Przykład - HelloJNI.java

```java
public class HelloJNI {
   static {
      System.loadLibrary("hello");
   }
   private native void sayHello();

   public static void main(String[] args) {
      new HelloJNI().sayHello();
   }
}
```

Przykład - HelloJNI.h wygenerowany

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class HelloJNI */

#ifndef _Included_HelloJNI
#define _Included_HelloJNI
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     HelloJNI
 * Method:    sayHello
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_HelloJNI_sayHello
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

Przykład - HelloJNI.c implementacja

```c
#include <jni.h>
#include <stdio.h>
#include "HelloJNI.h"

JNIEXPORT void JNICALL
Java_HelloJNI_sayHello(JNIEnv *env, jobject thisObj) {
   printf("Hello World!\n");
   return;
}
```

Typy w JNI

- JNI definiuje wiele typów JNI w systemie natywnym, które odpowiadają typom Java
- Prymitywy jint, jbyte, jshort, jlong, jfloat, jdouble, jchar, jboolean odpowiadają typom int, byte, short, long, float, double, char i boolean
- Typy referencji
  - jobject dla java.lang.Object
  - jclass dla java.lang.Class.
  - jstring dla java.lang.String.
  - jthrowable dla java.lang.Throwable.
  - jarray dla tablic Java. Jest 8 tablic z typami prymitywnymi i 1 dla obiektów: intArray, jbyteArray, jshortArray, jlongArray, jfloatArray, jdoubleArray, jcharArray, jbooleanArray i jobjectArray
- Funkcje natywne odbierają argumenty w powyższych typach JNI i zwracają wartość typu JNI

Programy natywne

- Programy natywne musza dokonywać konwersji z typów JNI na swój własny i podobnie zwracając wynik
- Typowe działanie funkcji natywnej
  - Odbierz argumenty w typie JNI (przekazany przez program Java).
  - Aby uzyskać odniesienie do typu JNI, przekonwertuj lub skopiuj argumenty do lokalnych typów macierzystych, np. Jstring do łańcucha znaków C, jintArray do int[], itd. Prymitywne typy JNI takie jak jint i jdouble nie wymagają konwersji i mogą być obsługiwane bezpośrednio.
  - Wykonaj operacje używając lokalnych typów.
  - Utwórz zwracany obiekt w typie JNI i skopiuj wynik do zwracanego obiektu.
  - Powrót z funkcji.
- Najbardziej mylące i wymagające zadanie w programowaniu JNI polega na konwersji (lub przekształceniu) pomiędzy typami odniesienia JNI (np. jstring, jobject, jintArray, jobjectArray) i typami natywnymi (C-string, int []).
- Interfejs JNI zapewnia jednak wiele funkcji do konwersji.

Przykład - TestJNIPrimitive.java

```
public class TestJNIPrimitive {
   static {
      System.loadLibrary("TestJNI");
   }
   private native double average(int n1, int n2);

   public static void main(String args[]) {
      System.out.println("W Javie, średnia= "
         + new TestJNIPrimitive().average(3, 2));
   }
}
```

Przykład - TestJNIPrimitive.h wygenerowany

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class TestJNIPrimitive */

#ifndef _Included_TestJNIPrimitive
#define _Included_TestJNIPrimitive
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     TestJNIPrimitive
 * Method:    average
 * Signature: (II)D
 */
JNIEXPORT jdouble JNICALL Java_TestJNIPrimitive_average
  (JNIEnv *, jobject, jint, jint);

#ifdef __cplusplus
}
#endif
#endif
```

Przykład - TestJNIPrimitive.c implementacja

```
#include <jni.h>
#include <stdio.h>
#include "TestJNIPrimitive.h"

JNIEXPORT jdouble JNICALL Java_TestJNIPrimitive_average
          (JNIEnv *env, jobject thisObj, jint n1, jint n2) {
   jdouble result;
   printf("W␣C,␣liczby␣to␣%d␣i␣%d\n", n1, n2);
   result = ((jdouble)n1 + n2) / 2.0;
   return result;
}

gcc -Wl,--add-stdcall-alias -I"%JAVA_HOME%\include"
-I"%JAVA_HOME%\include\win32" -shared -o TestJNI.dll TestJNIPrimiti
```

Przekazywanie typów złożonych (1/2)

- JNI zdefiniował typ jstring do reprezentowania obiektu Java String. Ostatni argument (typu JNI typu jstring) to ciąg znaków Java przekazany do programu C. Typ zwrotny jest również jstring.
- Przekazywanie obiektu String jest trudniejsze, bo ciąg znaków w C jest reprezentowany przez tablicę char*
- Środowisko JNI (dostępne za pośrednictwem argumentu JNIEnv *) udostępnia funkcje konwersji:
  - `const char* GetStringUTFChars(JNIEnv*, jstring, jboolean*)`
  - `jstring NewStringUTF(JNIEnv*, char*)`

Przekazywanie typów złożonych (2/2)

- Podobne funkcje istnieją dla konwertowania tablic
  - `jint* GetIntArrayElements(JNIEnv *env, jintArray a, jboolean *iscopy)`
  - `jsize  GetArrayLength(JNIEnv *env, jintArray a)`
  - `jintArray NewIntArray(JNIEnv *env, jsize len)`
  - `SetIntArrayRegion(JNIEnv *env, jintArray a, jsize start, jsize len, const jint *buf)`

Przykład - TestJNIPrimitiveArray.java

```
public class TestJNIPrimitiveArray {
 static {
  System.loadLibrary("TestJNIa");
 }
 private native double[] sumAndAverage(int[] numbers);

 public static void main(String args[]) {
  int[] numbers = {22, 33, 33};
  double[] results = new TestJNIPrimitiveArray().sumAndAverage(numb
  System.out.println("W␣Java,␣suma=␣" + results[0]);
  System.out.println("W␣Java,␣srednia=␣" + results[1]);
 }
}
```

Przykład - TestJNIPrimitiveArray.h wygenerowany

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class TestJNIPrimitiveArray */

#ifndef _Included_TestJNIPrimitiveArray
#define _Included_TestJNIPrimitiveArray
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     TestJNIPrimitiveArray
 * Method:    sumAndAverage
 * Signature: ([I)[D
 */
JNIEXPORT jdoubleArray JNICALL
Java_TestJNIPrimitiveArray_sumAndAverage
  (JNIEnv *, jobject, jintArray);

#ifdef __cplusplus
}
#endif
#endif
```

Przykład - TestJNIPrimitiveArray.c implementacja (1/2)

```c
#include <jni.h>
#include <stdio.h>
#include "TestJNIPrimitiveArray.h"

JNIEXPORT jdoubleArray JNICALL
          Java_TestJNIPrimitiveArray_sumAndAverage
          (JNIEnv *env, jobject thisObj, jintArray inJNIArray) {
 jint *inCArray =
             (*env)->GetIntArrayElements(env, inJNIArray, NULL);
 if (NULL == inCArray) return NULL;
 jsize length = (*env)->GetArrayLength(env, inJNIArray);

 jint sum = 0;
 int i;
 for (i = 0; i < length; i++) {
  sum += inCArray[i];
 }
 jdouble average = (jdouble)sum / length;
 (*env)->ReleaseIntArrayElements(env, inJNIArray, inCArray, 0);
 ...
}
```

Przykład - TestJNIPrimitiveArray.c implementacja (1/2)

```
JNIEXPORT jdoubleArray JNICALL
          Java_TestJNIPrimitiveArray_sumAndAverage
          (JNIEnv *env, jobject thisObj, jintArray inJNIArray) {
 ...
 jdouble outCArray[] = {sum, average};

 jdoubleArray outJNIArray = (*env)->NewDoubleArray(env, 2);
 if (NULL == outJNIArray) return NULL;

 (*env)->SetDoubleArrayRegion(env, outJNIArray, 0 , 2, outCArray);
 return outJNIArray;
}
```

Pytania?