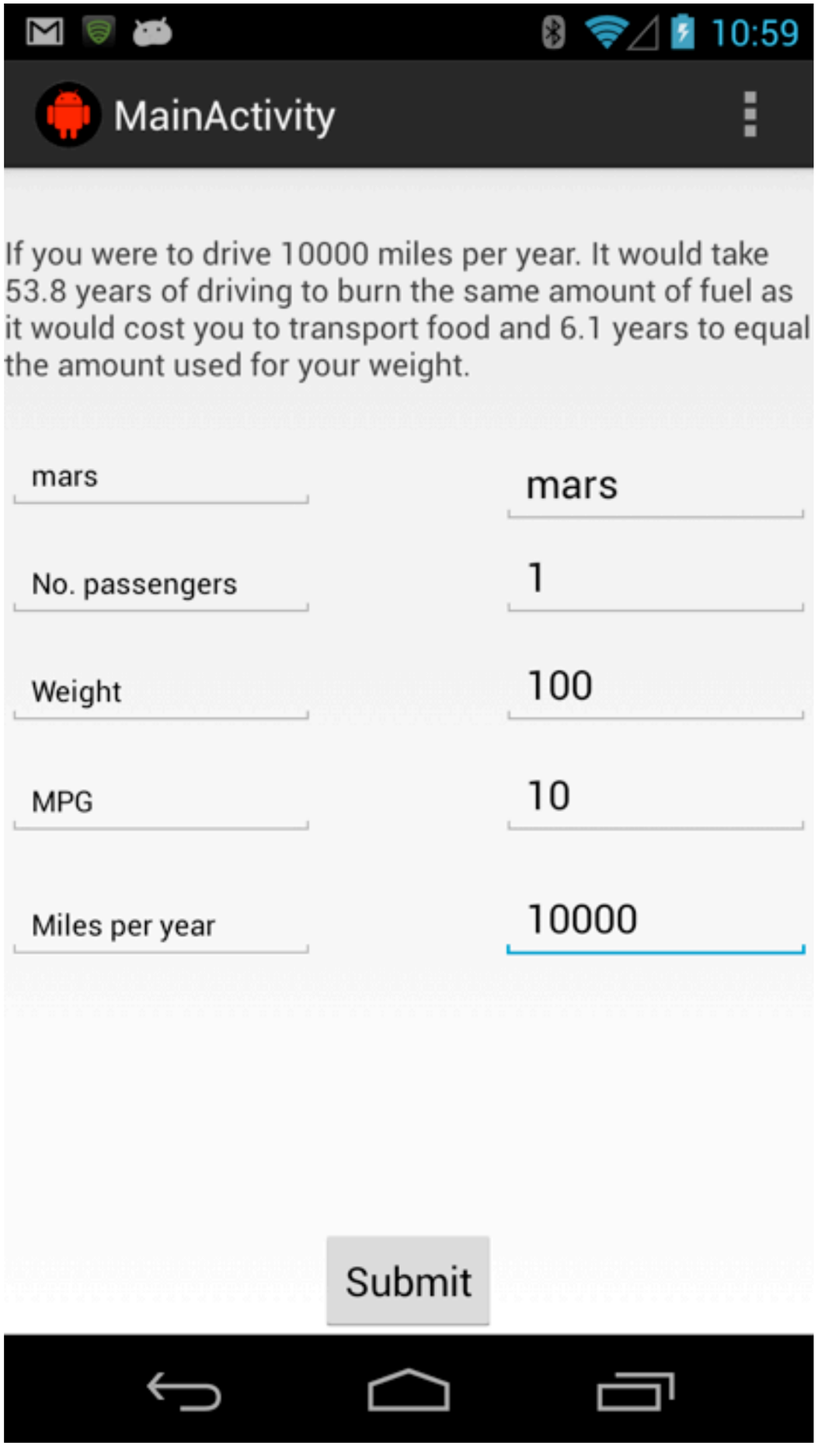
CS 580 Client-Server Programming
Fall Semester, 2012
Doc 22 Mars, JDBC & SSL
Nov 26, 2012

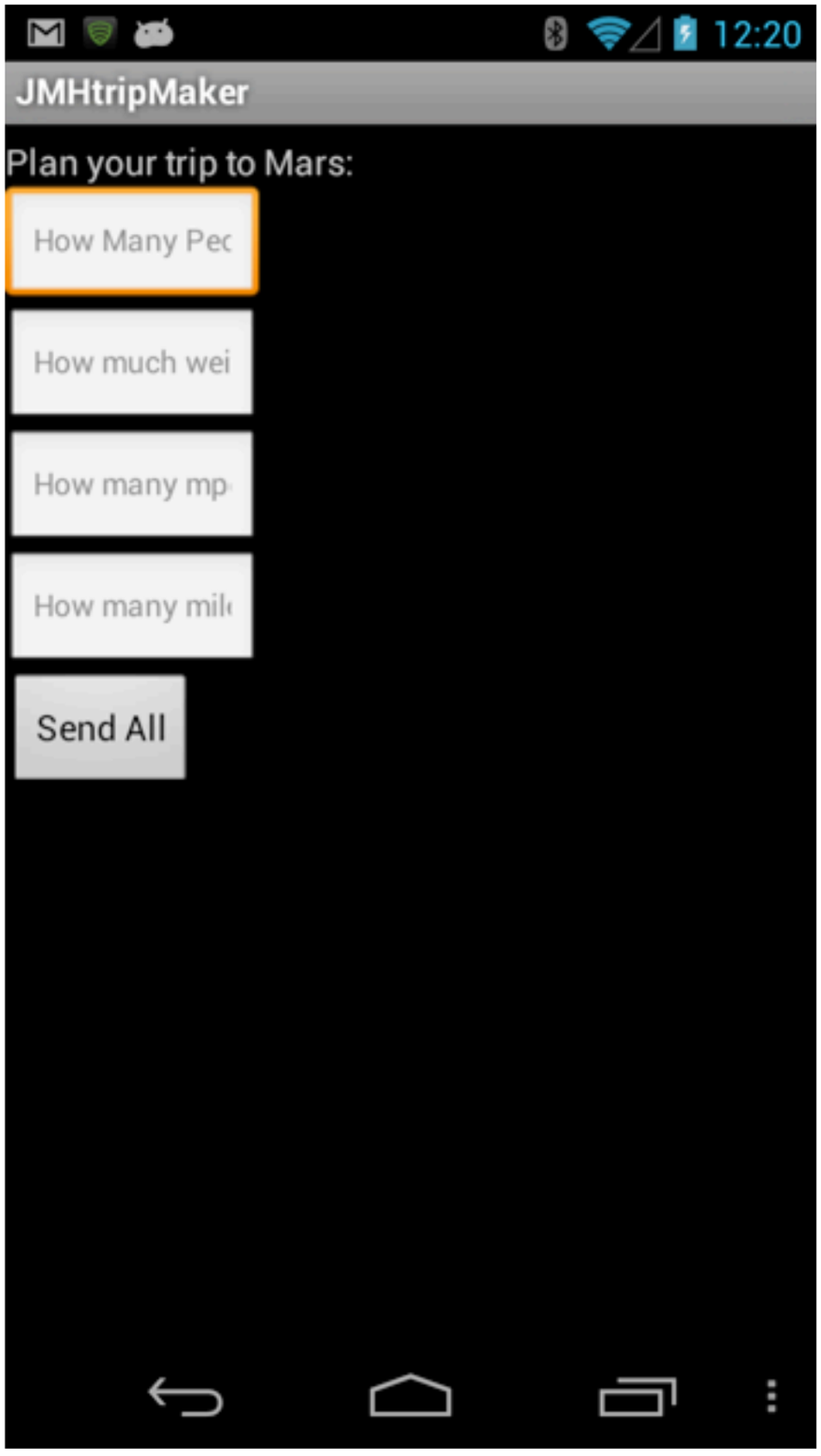
References

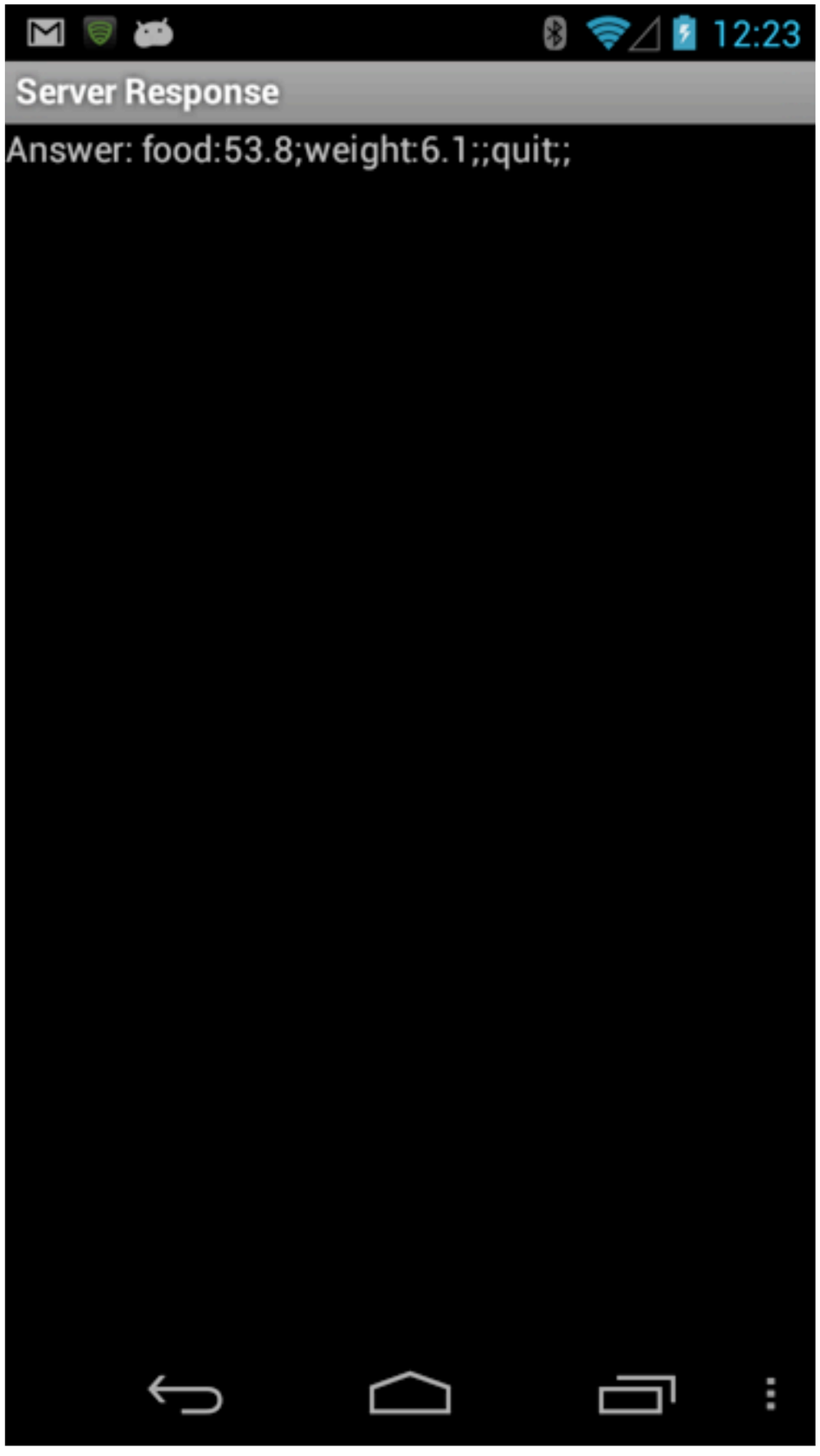
http://java.sun.com/javase/6/docs/technotes/guides/jdbc/index.html Sun's on-line JDBC Tutorial & Documentation

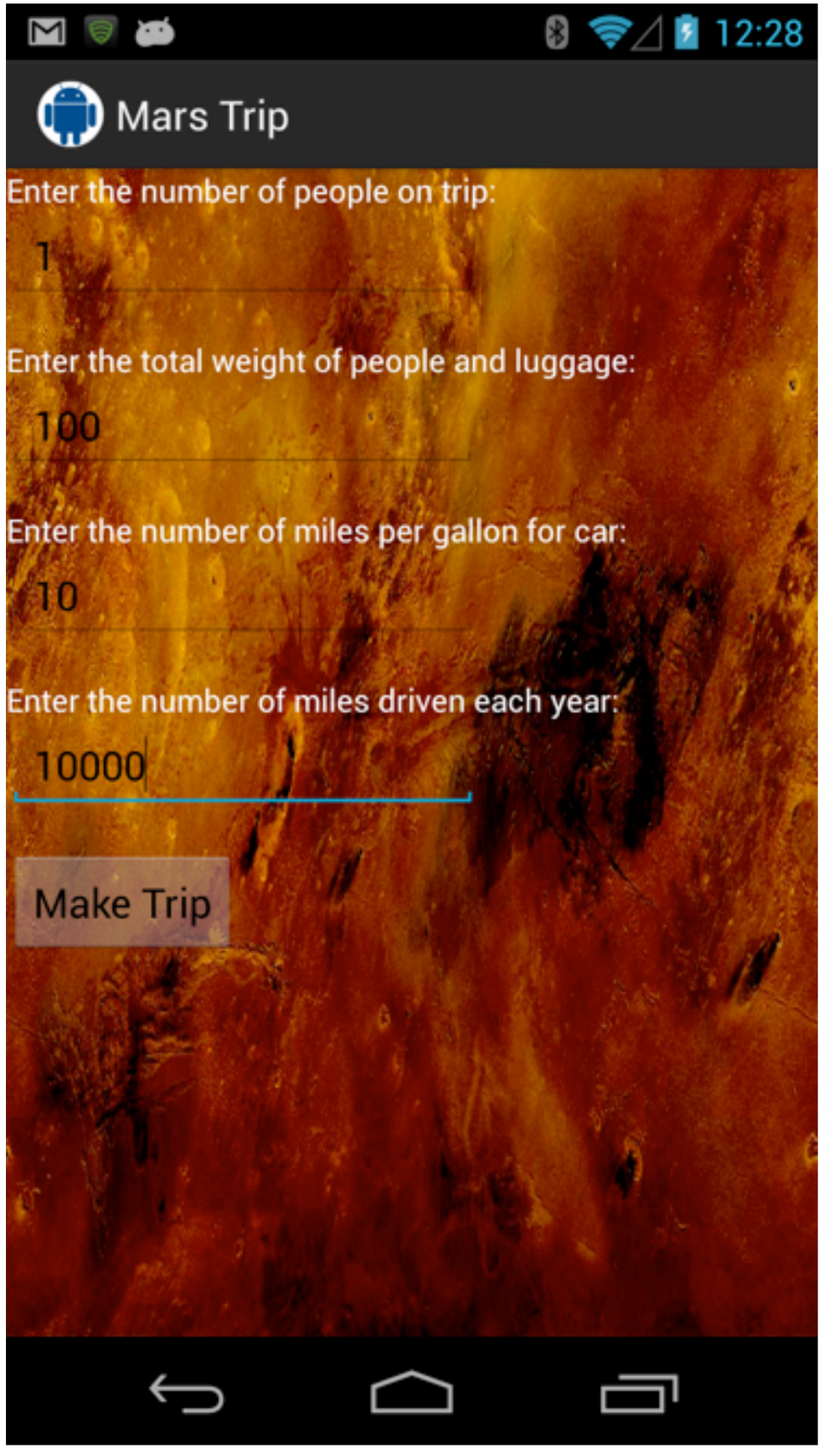
SqliteJDBC Docs, http://www.zentus.com/sqlitejdbc/

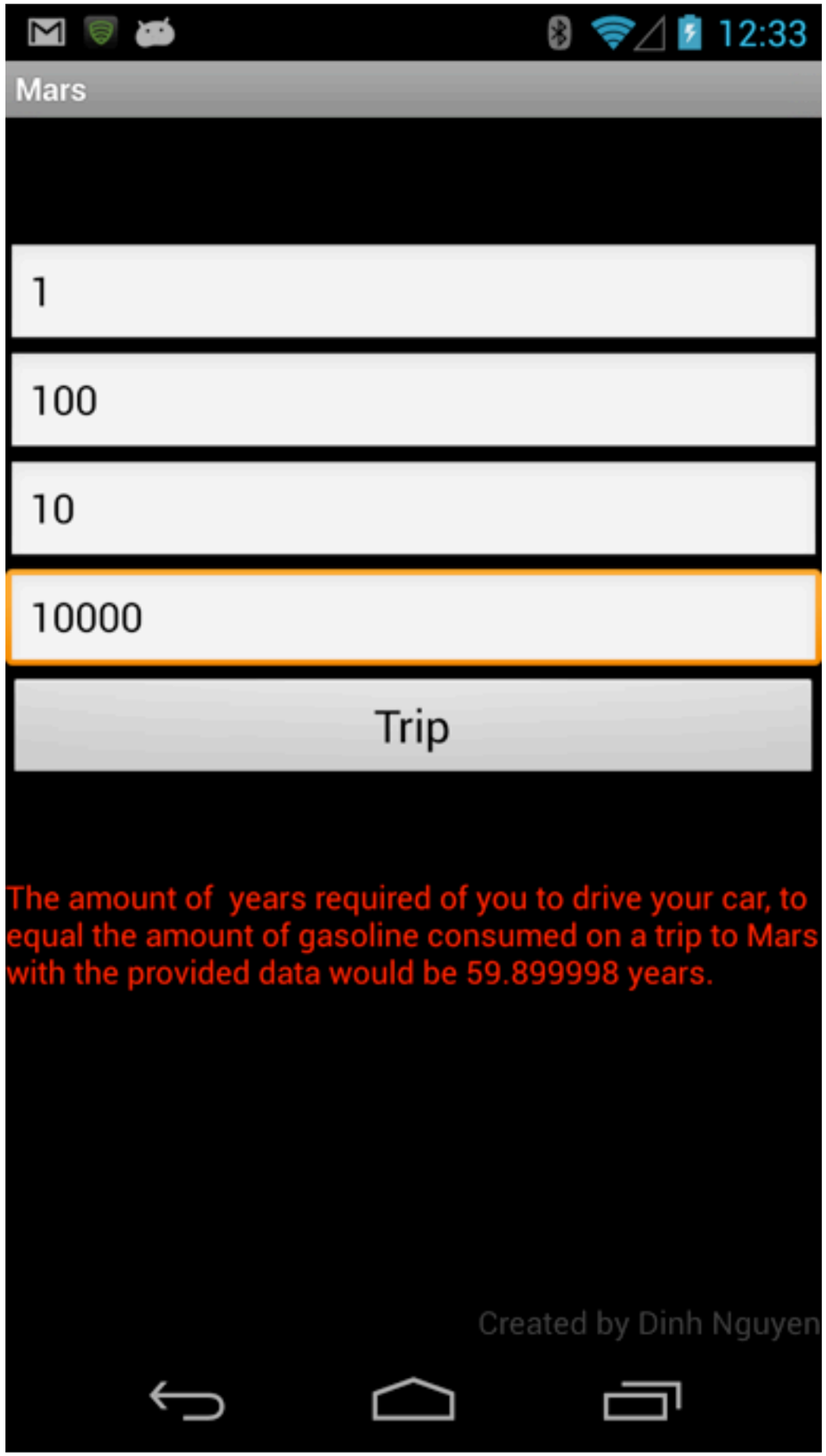
Mars Clients

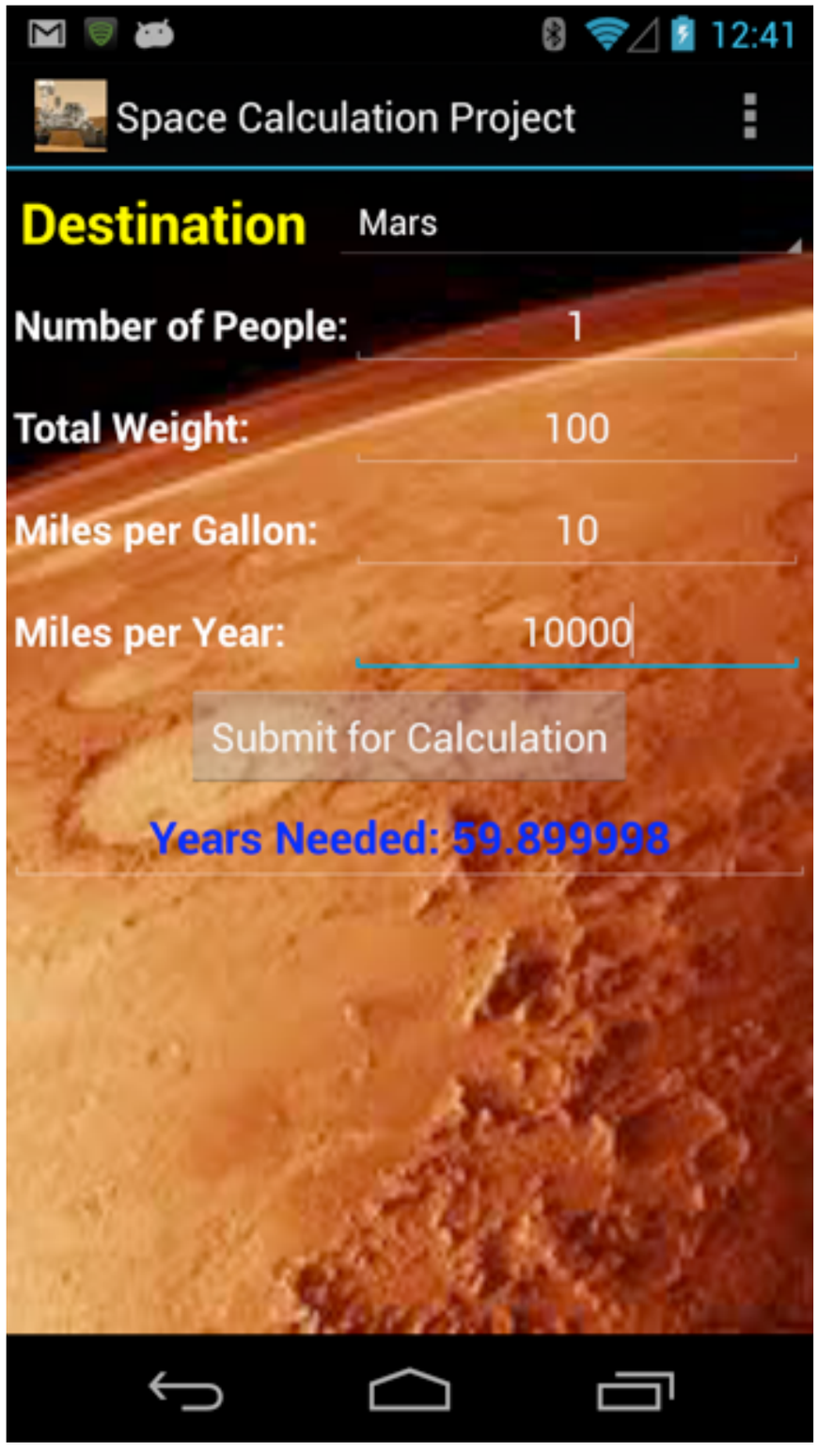

10:59
MainActivity
If you were to drive 10000 miles per year. It would take 53.8 years of driving to burn the same amount of fuel as it would cost you to transport food and 6.1 years to equal the amount used for your weight.
mars
mars
No. passengers
1
Weight
100
MPG
10
Miles per year
10000
Submit

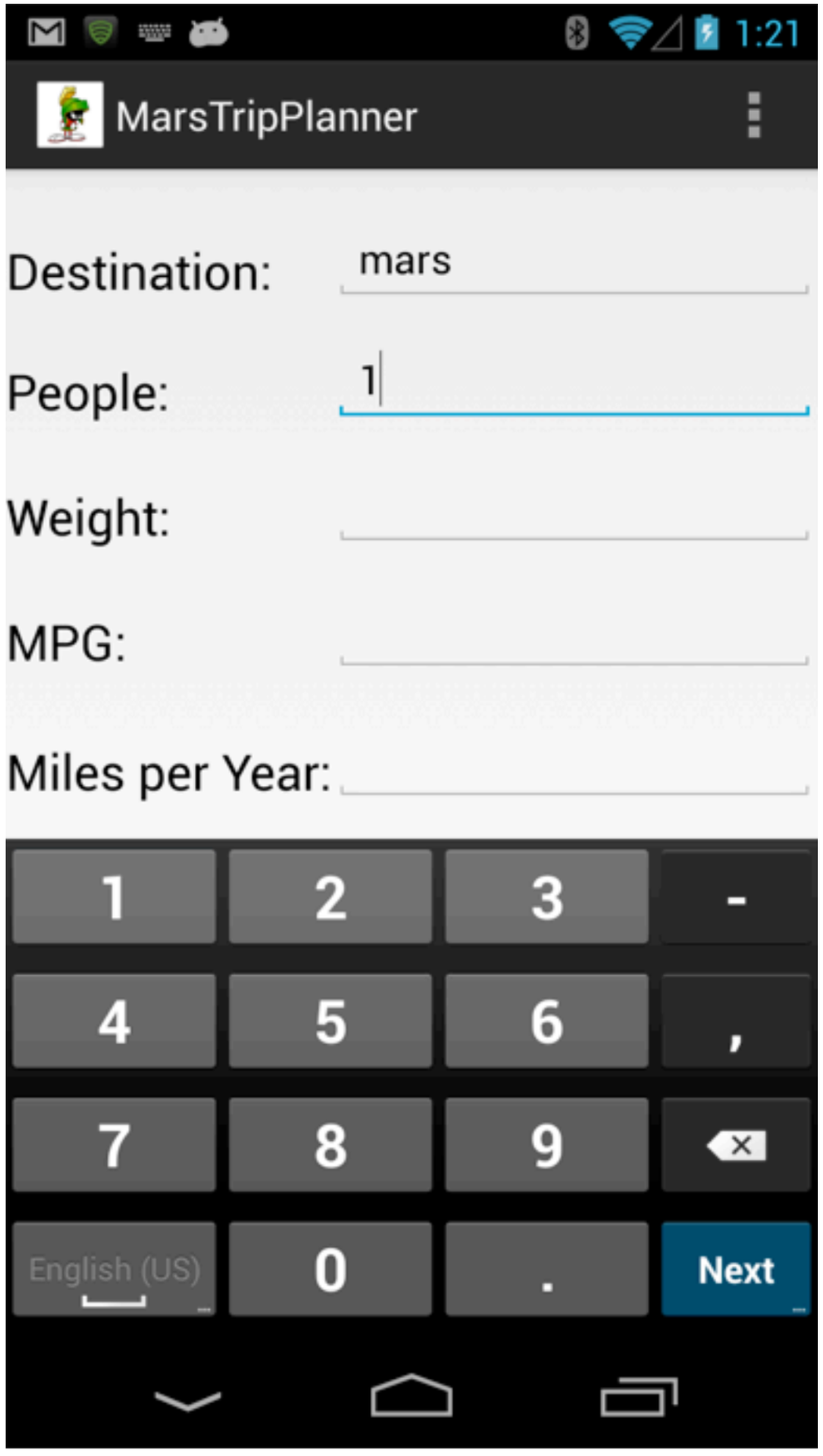

12:20
JMHtripMaker
Plan your trip to Mars:
How Many Pec
How much wei
How many mp
How many mil
Send All

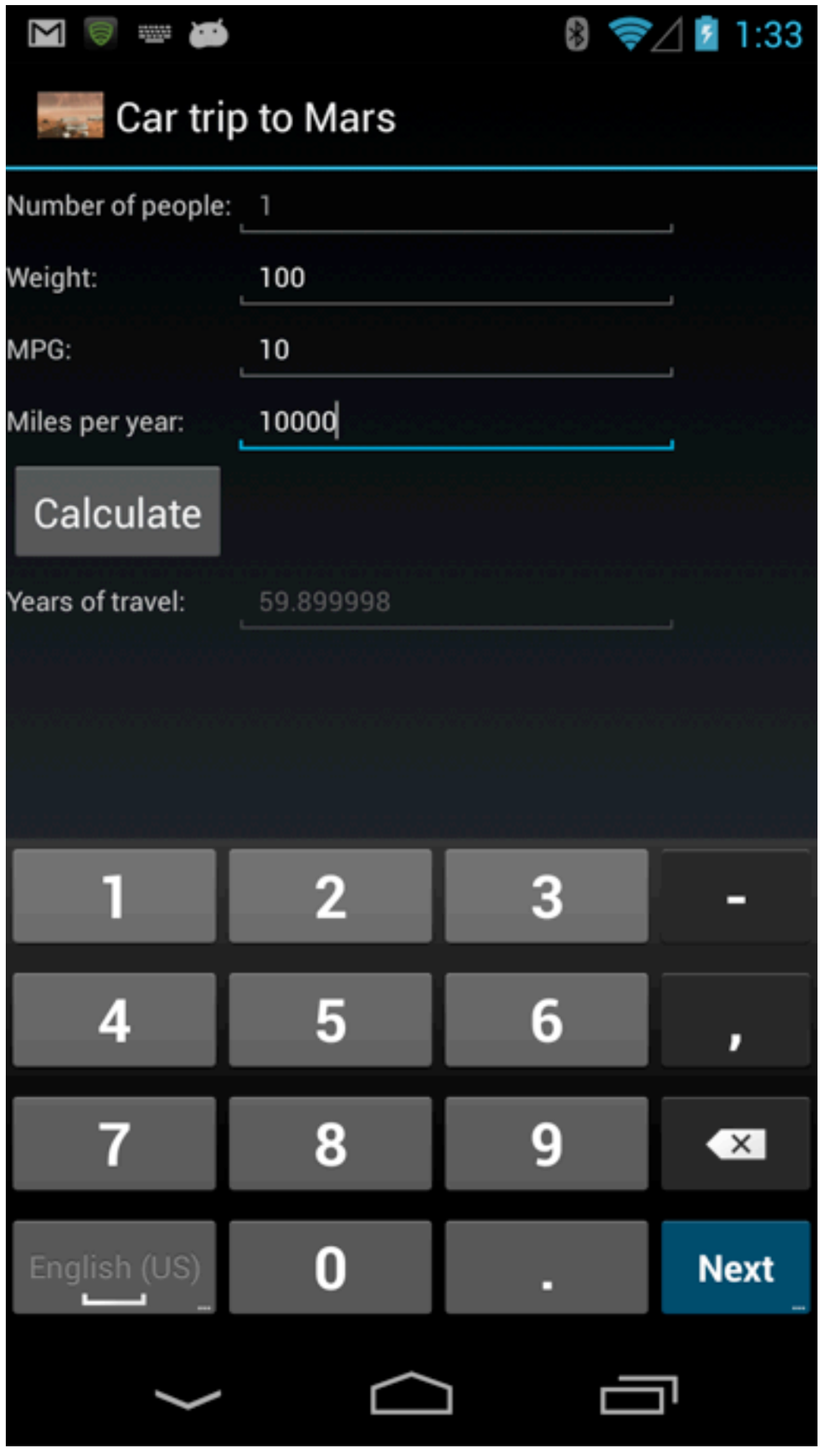

12:23
Server Response
Answer: food:53.8;weight:6.1;;quit;;

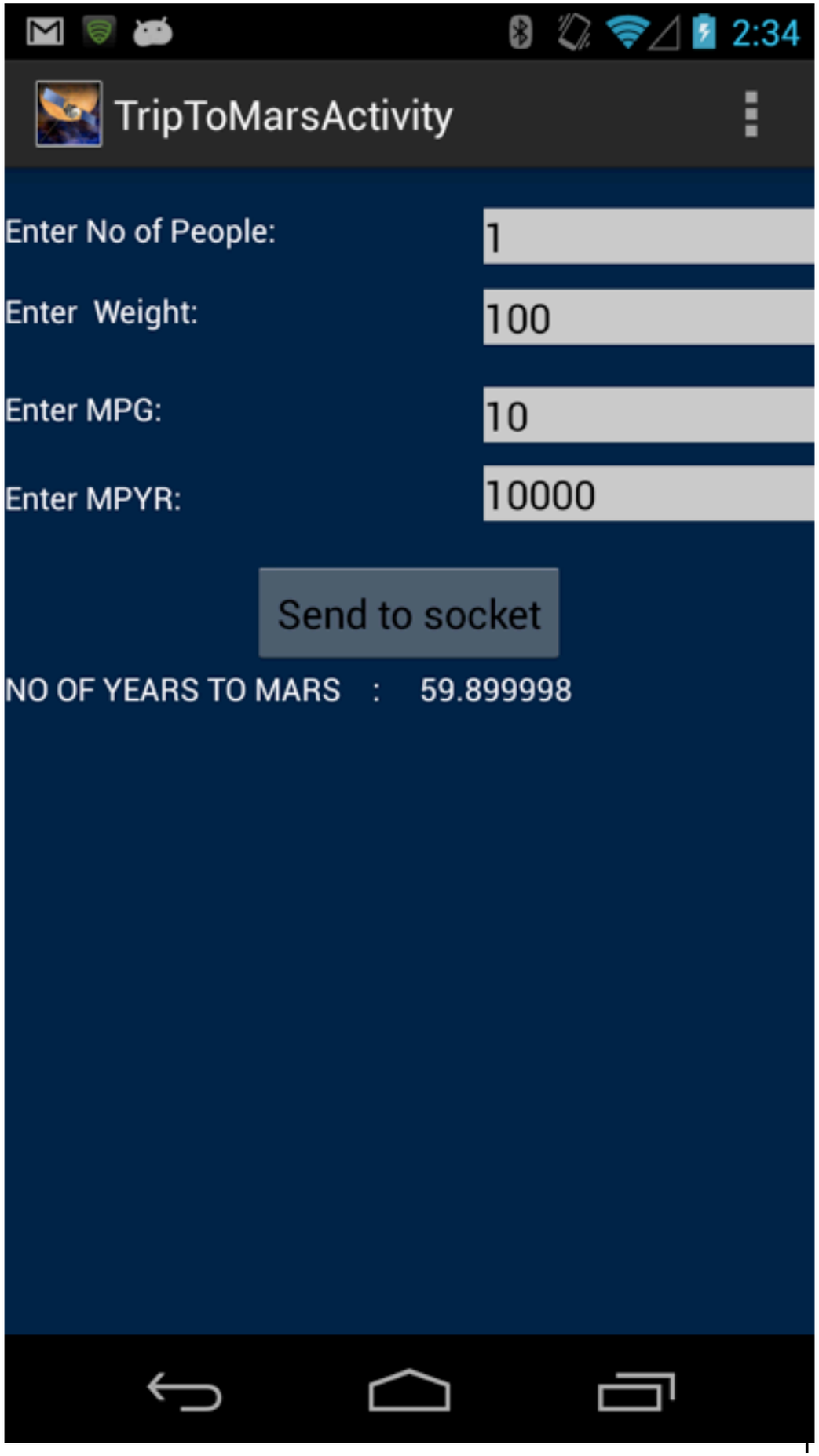

12:28
Mars Trip
Enter the number of people on trip:
1
Enter the total weight of people and luggage:
100
Enter the number of miles per gallon for car:
10
Enter the number of miles driven each year:
10000
Make Trip


12:33
Mars
1
100
10
10000
Trip
The amount of years required of you to drive your car, to equal the amount of gasoline consumed on a trip to Mars with the provided data would be 59.899998 years.
Created by Dinh Nguyen


12:41
Space Calculation Project
Destination
Mars
Number of People:
1
Total Weight:
100
Miles per Gallon:
10
Miles per Year:
10000
Submit for Calculation
Years Needed: 59.899998


1:21
MarsTripPlanner
Destination:
mars
People:
1
Weight:
MPG:
Miles per Year:
1
2
3
-
4
5
6
,
7
8
9
English (US)
0
.
Next


1:33
Car trip to Mars
Number of people:
1
Weight:
100
MPG:
10
Miles per year:
10000
Calculate
Years of travel:
59.899998
1
2
3
-
4
5
6
,
7
8
9
English (US)
0
.
Next


2:34
TripToMarsActivity
Enter No of People:
1
Enter Weight:
100
Enter MPG:
10
Enter MPYR:
10000
Send to socket
NO OF YEARS TO MARS : 59.899998

```
}
public String parse(StringBuffer d){
	String a = d.toString();

	if(a.contains("food") && a.contains("weight")){
		String b = (a.substring(5, a.indexOf("w")-1));
		String c = (a.substring(a.indexOf("t")+2, a.length()-2));
		return ("Food:"+b + " "+"Weight:" + c);
	}
	else

		return a;

}
public StringBuffer serverResponse(String ServerName, int port, byte[] test){
```

JDBC

Java – Connecting To Database

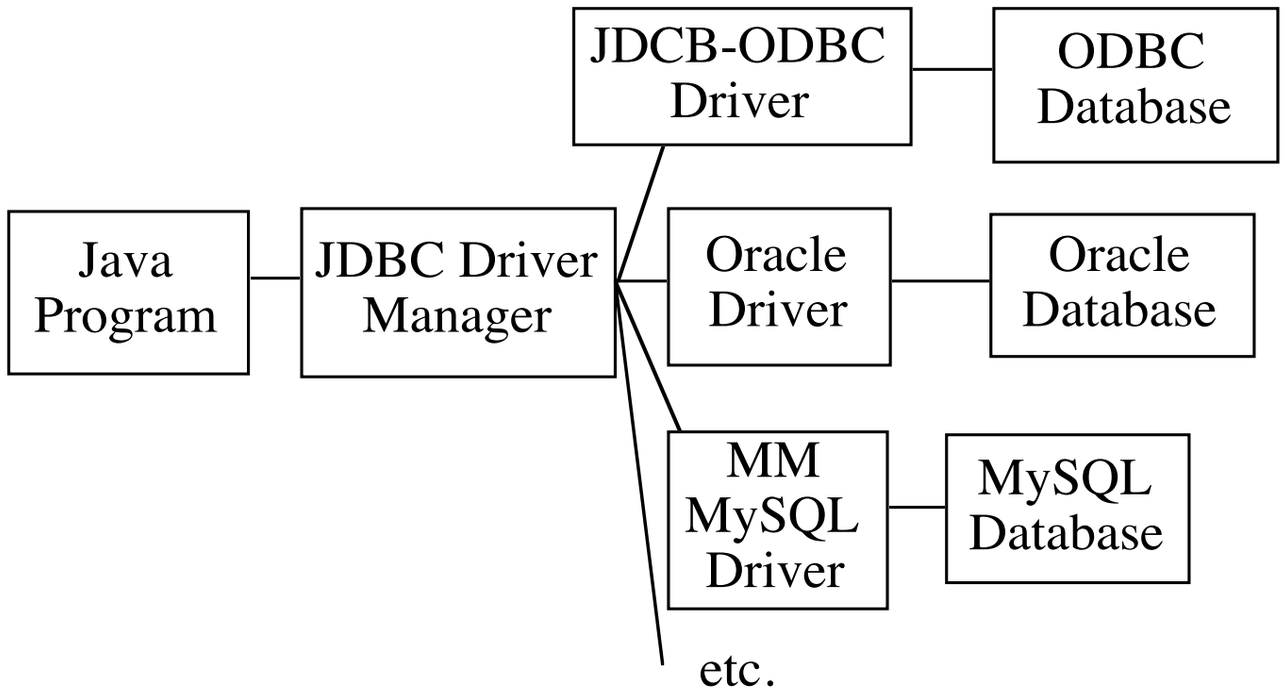
```
import java.sql.*;
public class Test {
  public static void main(String[] args) throws Exception {
    Class.forName("org.sqlite.JDBC");
    Connection conn = DriverManager.getConnection("jdbc:sqlite:test.db");
    Statement stat = conn.createStatement();
    stat.executeUpdate("drop table if exists people;");
    stat.executeUpdate("create table people (name, occupation);");
    PreparedStatement prep = conn.prepareStatement( "insert into people values (?, ?);");

    prep.setString(1, "Gandhi");
    prep.setString(2, "politics");
    prep.addBatch();
    conn.setAutoCommit(false);
    prep.executeBatch();
    conn.setAutoCommit(true);

    ResultSet rs = stat.executeQuery("select * from people;");
    while (rs.next()) {
      System.out.println("name = " + rs.getString("name"));
      System.out.println("job = " + rs.getString("occupation"));
    }
    rs.close();
    conn.close();
  }
}
```


This is the old way (Pre Java 6) of connecting.

JDBC



Drivers must be in your classpath

JDBC Drivers

Java supports four types of JDBC drivers

JDBC-ODBC bridge plus ODBC driver

Java code access ODBC native binary drivers

ODBC driver accesses databases

ODBC drivers must be installed on each client

Native-API partly-Java driver

Java code accesses database specific native binary drivers

JDBC-Net pure Java driver

Java code accesses database via DBMS-independent net protocol

Native-protocol pure Java driver

Java code accesses database via DBMS-specific net protocol

JDBC URL Structure

jdbc:<subprotocol>:<subname>

<subprotocol>

Name of the driver or database connectivity mechanism

<subname>

Depends on the <subprotocol>, can vary with vender

PostgreSQL

jdbc:postgresql:database

jdbc:postgresql://host/database

jdbc:postgresql://host:port/database

Sqlite

jdbc:sqlite:filename

MySQL

jdbc:mysql://[host][,failoverhost...][:port]/[database]
[?propertyName1][=propertyValue1][&propertyName2]
[=propertyValue2]...

Loading Driver

In your code

```
Class.forName("com.mysql.jdbc.Driver");
```

Command line

```
java –Djdbc.drivers=org.postgresql.Driver
yourProgramName
```

Loading Driver - Java 6, JDBC 4

Auto discovery

```
String dbUrl = "jdbc:postgresql://bismarck.sdsu.edu/test";
String user = "whitney";
String password = "mysecret";
Connection bismarck = DriverManager.getConnection( dbUrl, user, password);
Statement getTables = bismarck.createStatement();
ResultSet tableList = getTables.executeQuery("SELECT * FROM names");
while (tableList.next() )
    System.out.println("Last Name: " + tableList.getString(1) + '\t' +
            "First Name: " + tableList.getString( "first_name"));
bismarck.close();
```


Java 6 introduces auto discovery. We don't have to call Class.forName(). This requires Java 6 and JDBC 4 compliant drivers.

DriverManager.getConnection

Three forms:

getConnection(URL, Properties)
getConnection(URL, userName, Password)
getConnection(URLWithUsernamePassword)

Form 1

```
static String ARS_URL = "jdbc:oracle:@PutDatabaseNameHere";

DriverManager.getConnection(ARS_URL, "whitney","secret");
```

Form 2

```
DriverManager.getConnection(
    "jdbc:oracle:whitney/secret@PutDatabaseNameHere");
```

Form 3

```
java.util.Properties info = new java.util.Properties();
info.addProperty ("user", "whitney");
info.addProperty ("password","secret");

DriverManager getConnection (ARS_URL ,info );
```

java.sql verses javax.sql

java.sql
DriverManager

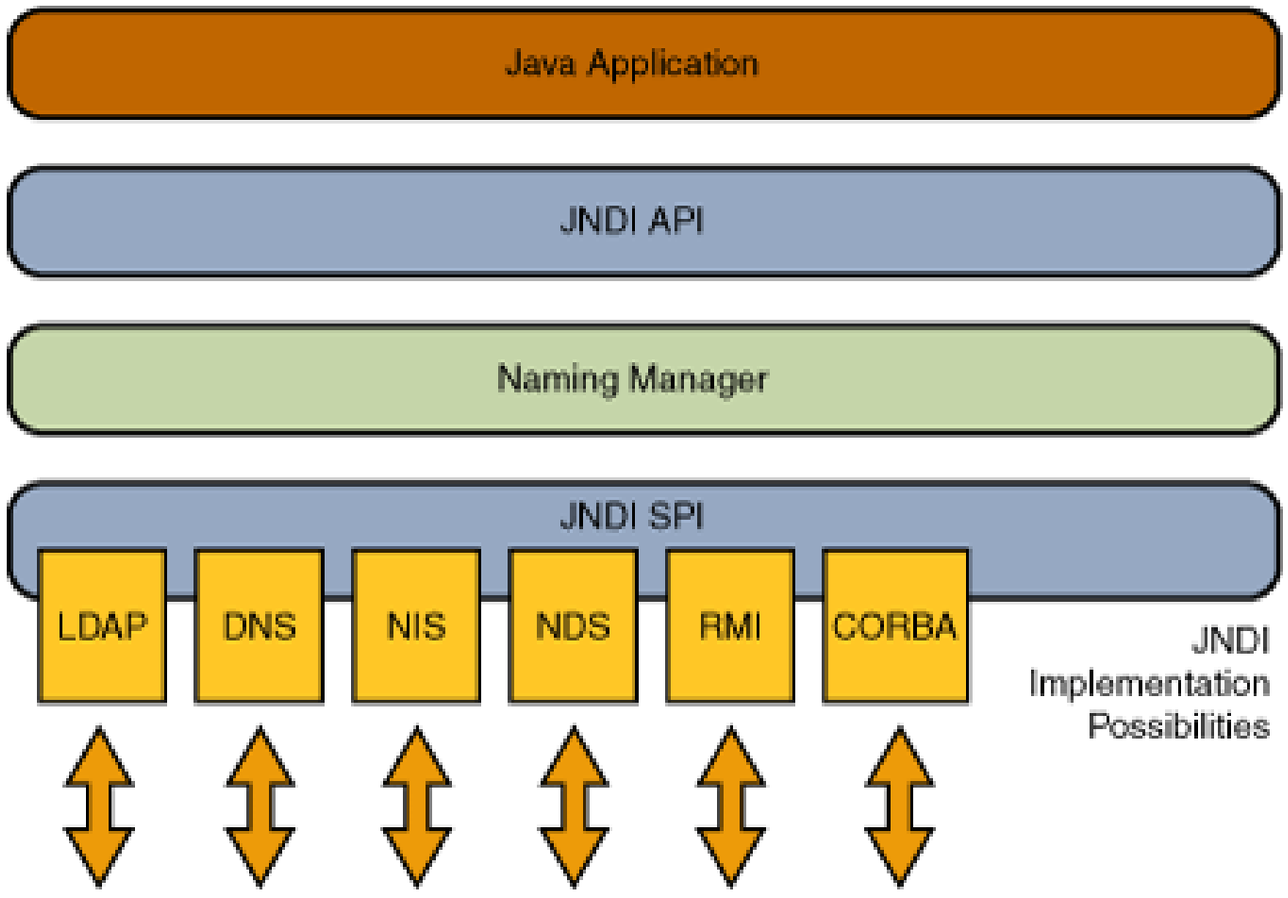
javax.sql
DataSource
- Connection Pools
- Distributed

Transactions
- Normally uses JNDI

JNDI

Java Naming and Directory Interface

Need JNDi Service Provider



http://java.sun.com/docs/books/tutorial/jndi/overview/index.html

Queries

executeUpdate

Use for INSERT, UPDATE, DELETE or SQL that return nothing

executeQuery

Use for SQL (SELECT) that return a result set

execute

Use for SQL that return multiple result sets

Uncommon

ResultSet

ResultSet - Result of a Query

JDBC returns a ResultSet as a result of a query

A ResultSet contains all the rows and columns that satisfy the SQL statement

A cursor is maintained to the current row of the data

The cursor is valid until the ResultSet object or its Statement object is closed

next() method advances the cursor to the next row

You can access columns of the current row by index or name

ResultSet has getXXX methods that:

- have either a column name or column index as argument
- return the data in that column converted to type XXX

getObject

A replacement for the getXXX methods

Rather than

```
ResultSet tableList =
        getTables.executeQuery("SELECT * FROM name");
String firstName = tableList.getString( 1);
```

Can use

```
ResultSet tableList =
        getTables.executeQuery("SELECT * FROM name");
String firstName = (String) tableList.getObject( 1);
```

getObject(int k) returns the object in the k’th column of the current row

getObject(String columnName) returns the object in the named column

Data Conversion

SQL type	Java type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Some Result Set Issues

What happens when we call next() too many times?

What happens if we try to access data before we call next?

In both cases an java.sql.SQLException is thrown

Mixing ResultSets

Can't have two active result sets on same statement

```
Connection rugby;
rugby = DriverManager.getConnection( dbUrl, user, password);
Statement getTables = rugby.createStatement();
ResultSet count =
        getTables.executeQuery("SELECT COUNT(*) FROM name");
ResultSet tableList =
        getTables.executeQuery("SELECT * FROM name");

while (tableList.next() )
        System.out.println("Last Name: " + tableList.getObject(1) + '\t' +
                                "First Name: " + tableList.getObject( "first_name"));

// Raises java.sql.SQLException
count.getObject(1);

rugby.close();
```

this can happen when two threads have access to the same statement

Two Statements on one Connection work

```
Connection rugby;
rugby = DriverManager.getConnection( dbUrl, user, password);
Statement getTables = rugby.createStatement();
Statement tableSize = rugby.createStatement();

ResultSet count =
        getTables.executeQuery("SELECT COUNT(*) FROM name");
ResultSet tableList =
        tableSize.executeQuery("SELECT * FROM name");

while (tableList.next() )
        System.out.println("Last Name: " + tableList.getObject(1) + '\t' +
                                "First Name: " +
tableList.getObject( "first_name"));
        count.next();
        System.out.println("Count: " + count.getObject(1) );
        count.close();
        tableList.close();
        rugby.close();
```

Threads & Connections

Some JDBC drivers are not thread safe

If two threads access the same connection results may get mixed up

PostgreSQL & MySql drivers are thread safe

When two threads make a request on the same connection

The second thread blocks until the first thread get it its results

Can use more than one connection but

Each connection requires a process on the database

SSL

SSL & TLS

Secure Socket Layer (SSL)

- SSL1 never release (Netscape)
- SSL2 (1995)
- SSL3 (1996)

Use Public Key encryption
To pass private key

Client checks server certificate

Transport Layer Security (TLS)

- TLS1 (1999)
- TLS1.1 (2006)
- TLS1.2 (2008)

TLS allows server to check client certificate

X.509 Certificates

Pairs public key to a Name

Certificate contents

Version
Serial Number
Algorithm ID
Issuer
Validity
    Not Before
    Not After
Subject
Subject Public Key Info
    Public Key Algorithm
    Subject Public Key
Issuer Unique Identifier (Optional)
Subject Unique Identifier (Optional)
Extensions (Optional)
Certificate Signature Algorithm
Certificate Signature

Certificate Authority (CA)

Trusted companies/agencies that issue certificates

VeriSign (57% of market)

Microsoft Corporation Incident

2001 VeriSign issued certificate named "Microsoft Corporation" to person

Trusted CAs & Web Browsers

Web browsers have a list of trusted CAs

User gets warning if site uses certificate browser can't validate

Root Certificates

Certificates are signed using private key of issuer

Use public key to validate signature

Web browsers contain certificates of CAs (issuers)

Generating a Certificate using Java

```
Al pro 13->keytool -genkey -alias whitney -keystore exampleKeystore
Enter keystore password:
Keystore password is too short - must be at least 6 characters
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Roger Whitney
What is the name of your organizational unit?
  [Unknown]:  Computer Science
What is the name of your organization?
  [Unknown]:  SDSU
What is the name of your City or Locality?
  [Unknown]:  San Diego
What is the name of your State or Province?
  [Unknown]:  CA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Roger Whitney, OU=Computer Science, O=SDSU, L=San Diego, ST=CA, C=US correct?
  [no]:  yes

Enter key password for <whitney>
        (RETURN if same as keystore password):
```

How TLS Works

Handshake

- Client & Server use public/private key system
- Exchange secrete key

Encrypted communication

- Using secrete key encrypt/decrypt communication

How TLS Works

Handshake

Client connect to server, gives list of cipher suites it supports

Server selects strongest cipher suite both support, notifies client

Server sends its certificate to client

Client may verify certificate with CA

Client

- Generates random number,
- Encrypts number with servers public key
- Sends number to server

Server & Client use random number to generate shared secret key

Java Client Side SSL/TLS

```
public static void main(String[] args) throws UnknownHostException,
        IOException {

    int port = 443;
    String hostname = "www.sdsu.edu";
    SocketFactory sslSocketFactory = SSLSocketFactory.getDefault();
    Socket socket =  sslSocketFactory.createSocket(hostname,
            port);

    InputStream in = socket.getInputStream();
    OutputStream out = socket.getOutputStream();

    // Read/Write to server

    in.close();
    out.close();
}
```

Some SSLSocket Extras

getSupportedCipherSuites

getSupportedProtocols

```
SocketFactory sslSocketFactory = SSLSocketFactory.getDefault();
SSLSocket socket = (SSLSocket) sslSocketFactory.createSocket(hostname, port);
for (String ciper : socket.getSupportedCipherSuites())
        System.out.println(ciper);
for (String ciper : socket.getSupportedProtocols())
        System.out.println(ciper);
```

Java JDK Supported Cipher Suites

SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA

SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_WITH_AES_128_CBC_SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
TLS_KRB5_WITH_RC4_128_SHA
TLS_KRB5_WITH_RC4_128_MD5
TLS_KRB5_WITH_3DES_EDE_CBC_SHA
TLS_KRB5_WITH_3DES_EDE_CBC_MD5
TLS_KRB5_WITH_DES_CBC_SHA
TLS_KRB5_WITH_DES_CBC_MD5
TLS_KRB5_EXPORT_WITH_RC4_40_SHA
TLS_KRB5_EXPORT_WITH_RC4_40_MD5
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5

Java JDK Supported Protocols

SSLv2Hello
SSLv3
TLSv1
TLSv1.1
TLSv1.2

Handshake

```
public class ClientSocket implements HandshakeCompletedListener {
        public void handshakeCompleted(HandshakeCompletedEvent event) {
                Socket socket = event.getSocket();
                try {
                        InputStream in = socket.getInputStream();
                        OutputStream out = socket.getOutputStream();
            // do work here
                        in.close();
                        out.close();
                } catch (IOException error) { error.printStackTrace(); }
        }

        public void connect() throws UnknownHostException, IOException {
                int port = 443;
                String hostname = "www.sdsu.edu";
                SocketFactory sslSocketFactory = SSLSocketFactory.getDefault();
                SSLSocket socket = (SSLSocket) sslSocketFactory.createSocket(hostname,
    port);
                socket.addHandshakeCompletedListener(this);
                socket.startHandshake();
        }
}
```

Session Management

SSL will reuse session keys for multiple connections

- Between same client & server
- Connections made in short time span

Feature can be turned off

Client Mode

setUseClientMode(boolean)

Set client mode to false

Client will try to authenticate itself

setNeedClientAuth(boolean)

Require all clients connecting to server to authenticate themselfs

Server-side SSLSocket

Need to

- Generate public keys & certificates
- Create SSLContext for the algorithm you will use
- Create TrustedManagerFactory for source of certificate material
- Create KeyManagerFactory for type of key material
- Create KeyStore object for key & certificate database
- Fill KeyStore object
- Initialize TrustedManagerFactory & KeyManagerFactory

Then just use the socket

Java SSL and Nio

"using SSl over socket channels is extremely complicated"

"Another good question is why they provided something that requires users to be able to write a state machine and to have a working knowledge of RFC 2246 to implement correctly."

Esmond Pitt

Fundamental Networking in Java, Springer 2006