

Place all of your narratives and illustrations in a single Word or PDF document named E90_LastNameFirstNameHW05.docx [.pdf]. Use this assignment as the initial template. Add your steps and your code below problem statements used for that problem. Upload your homework file and your working code (e.g., filename.java) into your Assignment 5 folder. Do not include executables.

Problem 1. MySQL Community Edition.

Download and install MySQL Community Edition server to your local machine. Create a new database (your local MySQL database) within that server and a new user that has all the privileges on that database. In the database create table FLOWERS that will contain information on the name of every flower, typical height in inches and a brief description of the flower. Populate your table with information on 3 flowers. Query the data. Delete a record. Close the database connection.

[15 points]

Download and install MySQL Community Edition server to your local machine.

```
Marnie's-MacBook-Air:~ marnie$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 31
Server version: 5.6.27 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Create a new database (your local MySQL database) within that server
I chose the name marnie_local_db

```
mysql> CREATE DATABASE marnie_local_db;
Query OK, 1 row affected (0.01 sec)

mysql> use marnie_local_db;
Database changed
mysql> █
```

Create a new user that has all the privileges on that database.

I chose the user name marnie_local

```
mysql> CREATE USER marnie_local IDENTIFIED BY 'marnie123';
Query OK, 0 rows affected (0.00 sec)

mysql> █
```

```
mysql> grant usage on *.* to marnie_local@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

```
mysql> grant all privileges on marnie_local_db.* to marnie_local@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Create table FLOWERS

name of every flower,

typical height in inches

and a brief description of the flower.

```
mysql> CREATE TABLE FLOWERS (
  -> flower_name VARCHAR(30) NOT NULL,
  -> height INT,
  -> description VARCHAR(50)
  -> );
Query OK, 0 rows affected (0.01 sec)

mysql> desc flowers;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| flower_name | varchar(30) | NO | | NULL | |
| height | int(11) | YES | | NULL | |
| description | varchar(50) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> █
```

Populate your table with information on 3 flowers.

```
mysql> INSERT INTO FLOWERS VALUES ('Rose', 5, 'Red');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO FLOWERS VALUES ('Tiger lily', 15, 'Orange');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO FLOWERS VALUES ('Carnation', 10, 'Ugly and Cheap');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT Count(*) FROM FLOWERS;
+-----+
| Count(*) |
+-----+
|          3 |
+-----+
1 row in set (0.01 sec)

mysql>
```

Query the data.

```
mysql> SELECT * FROM FLOWERS;
+-----+-----+-----+
| flower_name | height | description |
+-----+-----+-----+
| Rose        | 5      | Red        |
| Tiger lily  | 15     | Orange     |
| Carnation   | 10     | Ugly and Cheap |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> █
```

Delete a record.

```
mysql> DELETE FROM FLOWERS WHERE flower_name='Carnation';
Query OK, 1 row affected (0.01 sec)

mysql> Select * FROM FLOWERS;
+-----+-----+-----+
| flower_name | height | description |
+-----+-----+-----+
| Rose        | 5      | Red        |
| Tiger lily  | 15     | Orange     |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

Close the database connection.

```
mysql > exit ;  
Bye  
Marnie s - MacBook - Air : ~ marnie $
```

Problem 2. Use any programming language (e.g., Java, .NET, Python, ...).

Starting with the attached classes PersonDAO.java and MySQLAccess.java we used in class, write new Java classes or classes in any other language of your choice that will populate and query existing table FLOWERS in your local MySQL database. Close the database connection.

[20 points]

Write new Java classes (Starting with PersonDAO.java and MySQLAccess.java) to:

Populate existing table FLOWERS in your local MySQL database

Query existing table FLOWERS in your local MySQL database

Close the database connection.

Problem_2 Java Code

Flowers.java

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;
```

```
// Renamed Class to Flowers to make it
specific to our HW
public class Flowers {
    private Connection connect = null;
    private Statement statement = null;
    private PreparedStatement
preparedStatement = null;
    private ResultSet resultSet = null;

    public void readDataBase() throws
Exception {
        try {
            // This will load the MySQL
driver, each DB has its own driver

Class.forName("com.mysql.jdbc.Driver");
            // Setup the connection with the
DB

            connect = DriverManager

                // Use Local DB for Problem
2

                .getConnection("jdbc:mysql://
localhost/marnie_local_db?" +
```

```
"user=marnie_local&password=marnie123")  
;
```

```
        //getConnection("jdbc:mysql://  
jelenainst.c5cxb8gzb9wo.us-  
east-1.rds.amazonaws.com:3306/jaca?" +  
        //  
"user=jelena&password=jelena123");
```

```
        // Statements allow to issue SQL  
queries to the database
```

```
        statement =  
connect.createStatement();  
        // Result set get the result of  
the SQL query  
        resultSet = statement  
            .executeQuery("select * from  
marnie_local_db.FLOWERS");  
        System.out.println("The Original  
Flowers before Insert");  
        writeResultSet(resultSet);  
        System.out.println("");  
        System.out.println("");  
  
        // 1st Flower added
```

```
        preparedStatement = connect
            .prepareStatement("insert
into marnie_local_db.FLOWERS values
( ?, ?, ?)");
        // ("flower_name, height,
description");

        preparedStatement.setString(1,
"Daffodil");
        preparedStatement.setInt(2, 10);

preparedStatement.setString(3, "Yellow")
;

preparedStatement.executeUpdate();

// 2nd Flower added
        preparedStatement = connect
            .prepareStatement("insert
into marnie_local_db.FLOWERS values
( ?, ?, ?)");
        // ("flower_name, height,
description");
```

```
        preparedStatement.setString(1,
"Mum");
        preparedStatement.setInt(2, 10);

preparedStatement.setString(3, "Orange")
;

preparedStatement.executeUpdate();
```

```
        // Show the added flowers
        preparedStatement = connect
            .prepareStatement("SELECT
flower_name, height, description from
marnie_local_db.FLOWERS");
        resultSet =
preparedStatement.executeQuery();
        System.out.println("The added
flowers");
        writeResultSet(resultSet);
        System.out.println("");

    } catch (Exception e) {
        throw e;
    }
```



```
    } finally {  
        // Close Database connection  
        close();  
    }  
  
}  
  
private void writeResultSet(ResultSet  
resultSet) throws SQLException {  
    // ResultSet is initially before  
the first data set  
    while (resultSet.next()) {  
        // It is possible to get the  
columns via name  
        // also possible to get the  
columns via the column number  
        // which starts at 1  
        // e.g. resultSet.getString(2);  
        String flower_name =  
resultSet.getString("flower_name");  
        Integer height =  
resultSet.getInt("height");  
        String description =  
resultSet.getString("description");
```

```
        System.out.println("Flower Name:
" + flower_name);
        System.out.println("Height: " +
height);
        System.out.println("Description:
" + description);
    }
}
```

// You need to close the resultSet

```
private void close() {
    try {
        if (resultSet != null) {
            resultSet.close();
        }

        if (statement != null) {
            statement.close();
        }

        if (connect != null) {
            connect.close();
        }
    } catch (Exception e) {
```

```
}  
}  
  
}
```

MySQLAccess.java

```
public class MySQLAccess {  
    public static void main(String[] args) throws Exception {  
        // Changed PersonDAO and dao to Flowers and flower  
        Flowers flower = new Flowers();  
        flower.readDataBase();  
    }  
}
```

Result of Successful Code run in Console

```
46 preparedStatement.setInt(2, 10);  
47 preparedStatement.setString(3, "Yellow");  
48 preparedStatement.executeUpdate();  
49  
50 // 2nd Flower added  
51 preparedStatement = connect
```

<terminated> MySQLAccess [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (Oct 9, 2015, 9:48:30 PM)
The Original Flowers before Insert
Flower Name: Rose
Height: 5
Description: Red
Flower Name: Tiger lily
Height: 15
Description: Orange
The added Flowers
Flower Name: Rose
Height: 5
Description: Red
Flower Name: Tiger lily
Height: 15
Description: Orange
Flower Name: Daffodil
Height: 10
Description: Yellow
Flower Name: Mum
Height: 10
Description: Orange

Text Output

The Original Flowers before Insert

Flower Name: Rose

Height: 5

Description: Red

Flower Name: Tiger lily
Height: 15
Description: Orange

The added flowers

Flower Name: Rose
Height: 5
Description: Red
Flower Name: Tiger lily
Height: 15
Description: Orange
Flower Name: Daffodil
Height: 10
Description: Yellow
Flower Name: Mum
Height: 10
Description: Orange

Problem 3. Use MySQL Client and any programming language (e.g., Java, .NET, Python, ...).

Manually delete table FLOWERS (MySQL Client) and then extend the program from Problem 2 (in the language of your choice) so that when connected to the database it can recreate table FLOWERS and perform: insert a few records, query the data, delete a record. Close the database connection.

[15 points]

Manually delete table FLOWERS (MySQL Client)

```
mysql> select * FROM FLOWERS;
+-----+-----+-----+
| flower_name | height | description |
+-----+-----+-----+
| Rose       | 5      | Red         |
| Tiger lily | 15     | Orange      |
| Daffodil   | 10     | Yellow      |
| Mum        | 10     | Orange      |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> DROP TABLE FLOWERS;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
Empty set (0.00 sec)

mysql> █
```

Manually Delete Flowers table

```
mysql> select * FROM FLOWERS;
+-----+-----+-----+
| flower_name | height | description |
+-----+-----+-----+
| Rose       | 5      | Red         |
| Tiger lily | 15     | Orange      |
| Daffodil   | 10     | Yellow      |
| Mum        | 10     | Orange      |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> DROP TABLE FLOWERS;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
Empty set (0.00 sec)

mysql> █
```

Extend the program from Problem 2 to:

- Connect to the database
- Recreate table FLOWERS
- Insert a few records
- Query the data
- Delete a record
- Close the database connection

Flowers.java from Problem 3

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

// Renamed Class to Flowers to make it specific to our HW

```
public class Flowers {
    private Connection connect = null;
    private Statement statement = null;
```

```
private PreparedStatement preparedStatement = null;
private ResultSet resultSet = null;
private String sql= null;

public void readDataBase() throws Exception {
    try {
        // This will load the MySQL driver, each DB has its own driver
        Class.forName("com.mysql.jdbc.Driver");
        // Setup the connection with the DB
        connect = DriverManager

                // Use Local DB for Problem 3
// Connect to Database
        .getConnection("jdbc:mysql://localhost/marnie_local_db?" +
            "user=marnie_local&password=marnie123");

        // .getConnection("jdbc:mysql://jelenainst.c5cxb8gzb9wo.us-
        east-1.rds.amazonaws.com:3306/jaca?" +
            // "user=jelena&password=jelena123");

// Recreate the table flowers
        statement = connect.createStatement();
        sql = "CREATE TABLE FLOWERS " +
            "(flower_name VARCHAR(30), " +
            " height INT, " +
            " description VARCHAR(30))";
        statement.executeUpdate(sql);

        // Show contents of flowers table
        statement = connect.createStatement();
        // Result set get the result of the SQL query
        resultSet = statement
            .executeQuery("select * from marnie_local_db.FLOWERS");
        System.out.println("Flowers table before Insert");
        writeResultSet(resultSet);

        System.out.println("");

// Insert a few records

        // 1st Flower added
        preparedStatement = connect
```

```

        .prepareStatement("insert into marnie_local_db.FLOWERS values ( ?, ?, ?)");
// ("flower_name, height, description");

preparedStatement.setString(1, "Daffodil");
preparedStatement.setInt(2, 10);
preparedStatement.setString(3, "Yellow");
preparedStatement.executeUpdate();

// 2nd Flower added
preparedStatement = connect
        .prepareStatement("insert into marnie_local_db.FLOWERS values ( ?, ?, ?)");
// ("flower_name, height, description");

preparedStatement.setString(1, "Mum");
preparedStatement.setInt(2, 10);
preparedStatement.setString(3, "Orange");
preparedStatement.executeUpdate();

// Show the added flowers
preparedStatement = connect
        .prepareStatement("SELECT flower_name, height, description from
marnie_local_db.FLOWERS");
resultSet = preparedStatement.executeQuery();
System.out.println("The added flowers");
writeResultSet(resultSet);
System.out.println("");

// Delete a Record
statement = connect.createStatement();
sql = "DELETE FROM FLOWERS " +
        "WHERE flower_name = 'Mum' ";
statement.executeUpdate(sql);

// Show the flowers after the delete
preparedStatement = connect
        .prepareStatement("SELECT flower_name, height, description from
marnie_local_db.FLOWERS");
resultSet = preparedStatement.executeQuery();
System.out.println("The flowers after delete");
writeResultSet(resultSet);
System.out.println("");

```

```

    } catch (Exception e) {
        throw e;
    } finally {
        // Close Database connection
        close();
    }
}

private void writeResultSet(ResultSet resultSet) throws SQLException {
    // ResultSet is initially before the first data set
    while (resultSet.next()) {
        // It is possible to get the columns via name
        // also possible to get the columns via the column number
        // which starts at 1
        // e.g. resultSet.getString(2);
        String flower_name = resultSet.getString("flower_name");
        Integer height = resultSet.getInt("height");
        String description = resultSet.getString("description");
        System.out.println("Flower Name: " + flower_name);
        System.out.println("Height: " + height);
        System.out.println("Description: " + description);
    }
}

// You need to close the resultSet
private void close() {
    try {
        if (resultSet != null) {
            resultSet.close();
        }

        if (statement != null) {
            statement.close();
        }

        if (connect != null) {
            connect.close();
        }
    } catch (Exception e) {
    }
}

```



```

38 // Show contents of flowers table
39 statement = connect.createStatement();
40 // Result set get the result of the SQL query
41 resultSet = statement
42     .executeQuery("select * from marnie_local_db.FLOWERS");
43 System.out.println("Flowers table before Insert");
44 writeResultSet(resultSet);
45
46 System.out.println("");
47
48 // 1st Flower added
49 preparedStatement = connect

```

<terminated>- MySQLAccess (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (Oct 9, 2015, 10:22:34 PM)
 Flowers table before Insert
 The added flowers
 Flower Name: Daffodil
 Height: 10
 Description: Yellow
 Flower Name: Mum
 Height: 10
 Description: Orange
 The flowers after delete
 Flower Name: Daffodil
 Height: 10
 Description: Yellow

Step 1: Select a VPC Configuration

VPC with a Single Public Subnet

VPC with Public and Private Subnets

VPC with Public and Private Subnets and Hardware VPN Access

VPC with a Private Subnet Only and Hardware VPN Access

In addition to containing a public subnet, this configuration adds a private subnet whose instances are not addressable from the Internet. Instances in the private subnet can establish outbound connections to the Internet via the public subnet using Network Address Translation (NAT).

Creates:

A /16 network with two /24 subnets. Public subnet instances use Elastic IPs to access the Internet. Private subnet instances access the Internet via a Network Address Translation (NAT) instance in the public subnet. (Hourly charges for NAT instances apply.)

Select

[Cancel and Exit](#)

}

The Output Text in the Console

Flowers table before Insert

The added flowers

Flower Name: Daffodil

Height: 10

Description: Yellow

Flower Name: Mum

Height: 10

Description: Orange

The flowers after delete

Flower Name: Daffodil

Height: 10

Description: Yellow

Problem 4. Use AWS Console and MySQL Client.

1. Create new VPC with two subnets (one public and one private). Create a new security group associated with your new VPC, and adjust permissions (CIDR) of that security group so that you can access future RDS from your local machine.
2. Create a new micro instance of MySQL database in Amazon RDS service – using AWS Console.
3. Demonstrate that you can connect to the remote RDS database using your local MySQL client.

[15 points]

1. Create new VPC with two subnets (one public and one private)

The screenshot shows the AWS Management Console interface for creating a new VPC. The top navigation bar includes the AWS logo, 'Services', 'Edit', and user information 'Marnie Scully', 'N. Virginia', and 'Support'. The main heading is 'Step 2: VPC with Public and Private Subnets'.

The form contains the following sections:

- VPC Configuration:**
 - IP CIDR block:** 10.0.0.0/16 (65531 IP addresses available)
 - VPC name:** VPC_marnie
- Subnet Configuration:**
 - Public subnet:** 10.0.0.0/24 (251 IP addresses available)
 - Availability Zone:** us-east-1a
 - Public subnet name:** Public subnet
 - Private subnet:** 10.0.1.0/24 (251 IP addresses available)
 - Availability Zone:** us-east-1b
 - Private subnet name:** Private subnet

You can add more subnets after AWS creates the VPC.
- NAT Instance Configuration:**

Specify the details of your NAT instance.

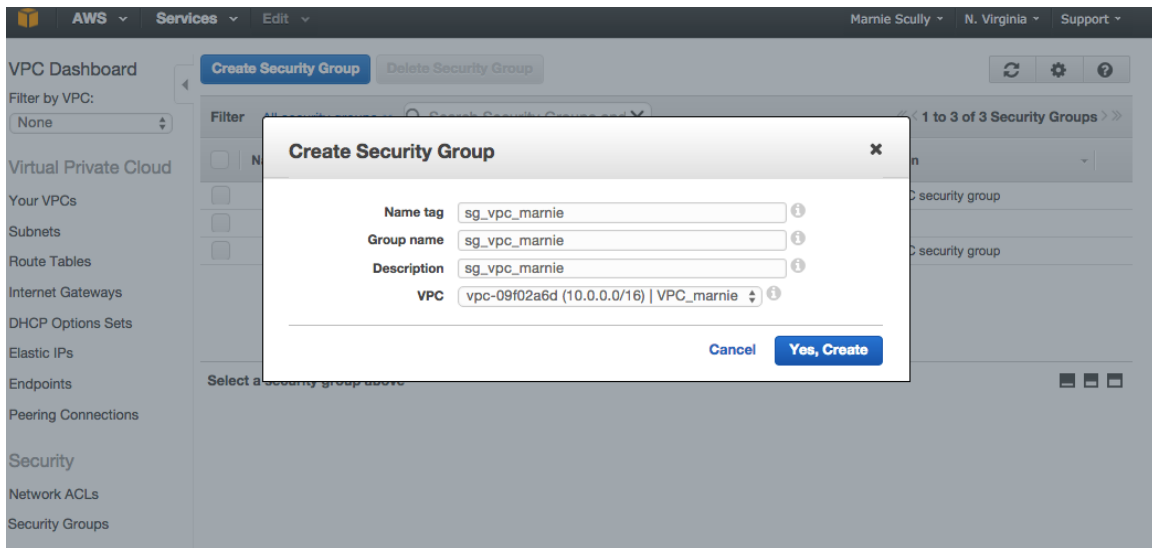
 - Instance type:** m1.small
 - Key pair name:** CitiKeyPair
 - Note: Instance rates apply. [View Rates.](#)
- S3 Endpoints:**

Add endpoints for S3 to your subnets

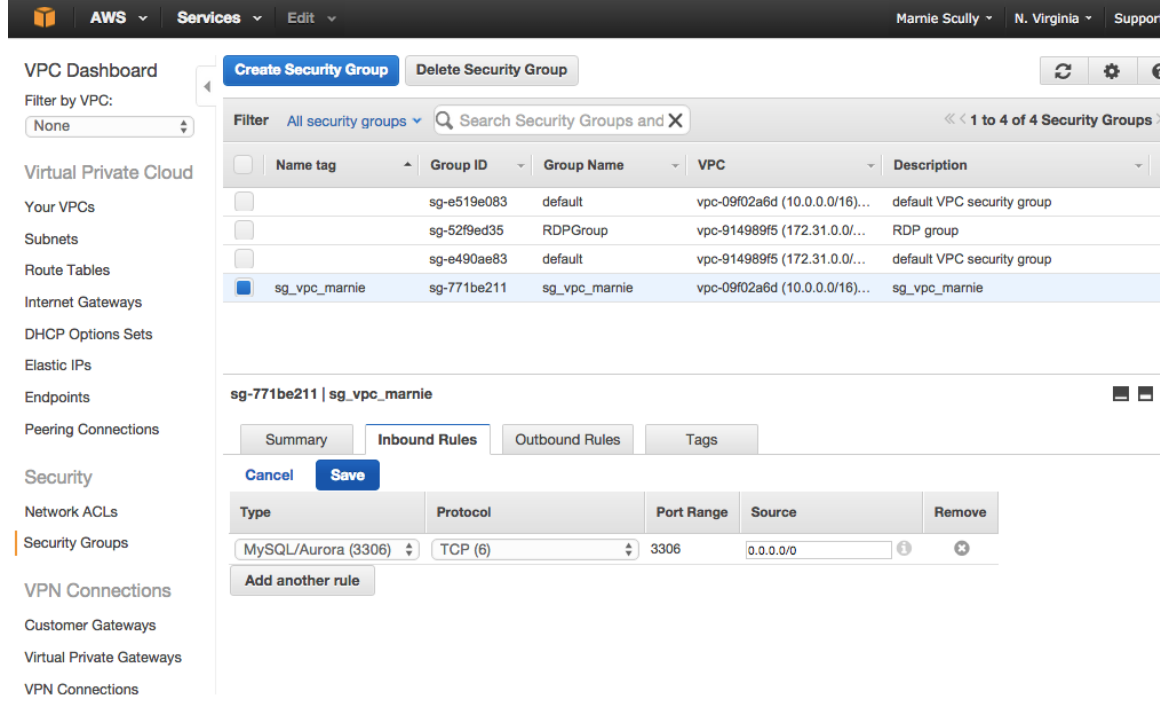
 - Subnet:** None
- Optional Settings:**
 - Enable DNS hostnames:** Yes (selected) / No
 - Hardware tenancy:** Default

At the bottom right, there are three buttons: 'Cancel and Exit', 'Back', and 'Create VPC'.

2. Create a new security group associated with your new VPC



3. Adjust permissions (CIDR) of that security group so that you can access future RDS from your local machine.



4. Create a new micro instance of MySQL database in Amazon RDS service – using AWS Console.

The screenshot shows the 'Select Engine' step in the AWS Management Console. The left sidebar indicates 'Step 1: Select Engine'. The main content area has the heading 'Select Engine' and a subtext 'To get started, choose a DB Engine below and click Select.' Below this, there are five database engine options: Amazon Aurora, MariaDB, MySQL, PostgreSQL, and ORACLE. The MySQL option is selected, showing 'MySQL MySQL Community Edition' with a 'Select' button.

The screenshot shows the 'Do you plan to use this database for production purposes?' step. The left sidebar shows 'Step 1: Select Engine', 'Step 2: Production?', 'Step 3: Specify DB Details', and 'Step 4: Configure Advanced Settings'. The main content area has the heading 'Do you plan to use this database for production purposes?'. A blue box contains a message: 'For databases used in production or pre-production we recommend: Multi-AZ Deployment for high availability (99.95% monthly up time SLA) and Provisioned IOPS Storage for fast, consistent performance. Billing is based upon the RDS pricing table. An instance which uses these features is not eligible for the RDS Free Usage Tier.' Below this, there are two radio button options: 'Yes, use Multi-AZ Deployment and Provisioned IOPS Storage as defaults while creating this instance' and 'No, this instance is intended for use outside of production or under the RDS Free Usage Tier'. The 'No' option is selected. At the bottom right are 'Cancel', 'Previous', and 'Next Step' buttons.

The screenshot shows the 'Specify DB Details' step. The left sidebar shows 'Step 3: Specify DB Details' and 'Step 4: Configure Advanced Settings'. The main content area has the heading 'Specify DB Details'. It contains several configuration options: 'DB Engine' (mysql), 'License Model' (general-public-license), 'DB Engine Version' (5.6.23), 'DB Instance Class' (db.t2.micro – 1 vCPU, 1 GiB RAM), 'Multi-AZ Deployment' (No), 'Storage Type' (General Purpose (SSD)), and 'Allocated Storage*' (5 GB). A red warning box states: 'Provisioning less than 100 GB of General Purpose (SSD) storage for high throughput workloads could result in higher latencies upon exhaustion of the initial General Purpose (SSD) IO credit balance. Click here for more details.' Below these options is a 'Settings' section with 'DB Instance Identifier*' (marnie_inst), 'Master Username*' (marnie_rds), and 'Master Password*' (masked with dots). A note on the right says 'Retype the value you specified for Master Password.' At the bottom right are 'Previous' and 'Next Step' buttons.

Make sure you choose the VPC and Security Group created earlier

Step 4: Configure Advanced Settings

If you are using SSL to connect to this instance, you should use the [new certificate bundle](#). Learn more [here](#).

VPC*

Subnet Group

Publicly Accessible

Availability Zone

VPC Security Group(s)

Database Options

Database Name

Note: If no database name is specified then no initial MySQL database will be created on the DB Instance.

Database Port

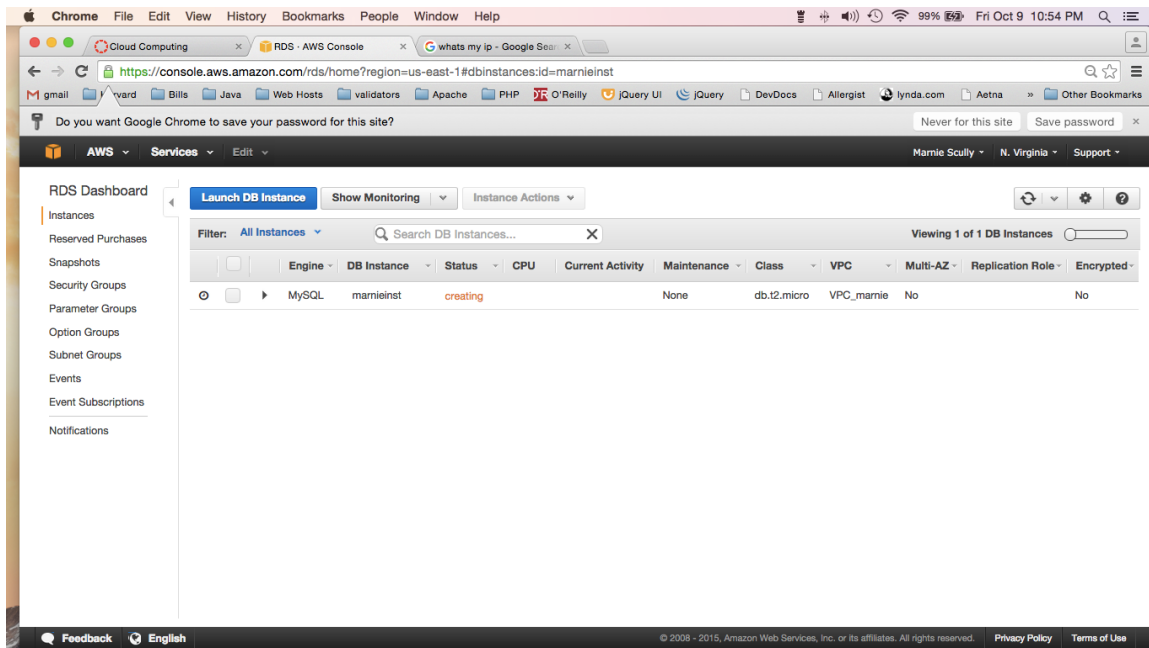
DB Parameter Group

Option Group

Copy Tags To Snapshots ☐

Enable Encryption

Specify a string of up to 8 alpha-numeric characters that define the name given to a database that Amazon RDS creates when it creates the DB instance, as in "mydb". If you do not specify a database name, Amazon RDS does not create a database when it creates the DB instance.



5. Demonstrate that you can connect to the remote RDS database using your local MySQL client.

```
Marnie's-MacBook-Air:~ marnie$ mysql -h marnieinst.cnvmcqwxml0.us-east-1.rds.amazonaws.com -u marnie_rds -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 5.6.23-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Problem 5. Use any programming language (e.g., Java, .NET, Python, ...).

1. Modify the code developed in Problem 3 so that it can now connect to your Amazon's RDS database instance that you created in problem 4.
2. Demonstrate that your client code could perform on remote RDS database instance the same operations it performed on the local database (create table, insert a few records, query the data, delete a record (on your remote RDS database instance). Close the database connection.

[15 points]

1. Modify the code developed in Problem 3 so that it can now connect to your Amazon's RDS database instance that you created in problem 4
2. On remote RDS database instance:
 - a. create table
 - b. insert a few records
 - c. query the data
 - d. delete a record
3. Close the database connection

Modified Flowers.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

// Renamed Class to Flowers to make it specific to our HW
public class Flowers {
    private Connection connect = null;
    private Statement statement = null;
    private PreparedStatement preparedStatement = null;
    private ResultSet resultSet = null;
    private String sql= null;

    public void readDataBase() throws Exception {
        try {
            // This will load the MySQL driver, each DB has its own driver
            Class.forName("com.mysql.jdbc.Driver");
            // Setup the connection with the DB
            connect = DriverManager

                                // Use Remote DB for Problem 5
                                //.getConnection("jdbc:mysql://localhost/marnie_local_db?" +
                                "user=marnie_local&password=marnie123");

            .getConnection("jdbc:mysql://marnieinst.cnvmcqwxml0.us-
east-1.rds.amazonaws.com/marnie_rds?" +
                "user=marnie_rds&password=marnie123");

            // Create Table Flowers
            statement = connect.createStatement();
            sql = "CREATE TABLE FLOWERS " +
                "(flower_name VARCHAR(30), " +
                " height INT, " +
                " description VARCHAR(30))";
            statement.executeUpdate(sql);

            // Show contents of flowers table
```

```
statement = connect.createStatement();
// Result set get the result of the SQL query
resultSet = statement
    .executeQuery("select * from marnie_rds.FLOWERS");
System.out.println("Flowers table before Insert");
writeResultSet(resultSet);

System.out.println("");

// insert a few records
// 1st Flower added
preparedStatement = connect
    .prepareStatement("insert into marnie_rds.FLOWERS values ( ?, ?, ?)");
// ("flower_name, height, description");

preparedStatement.setString(1, "Daffodil");
preparedStatement.setInt(2, 10);
preparedStatement.setString(3, "Yellow");
preparedStatement.executeUpdate();

// 2nd Flower added
preparedStatement = connect
    .prepareStatement("insert into marnie_rds.FLOWERS values ( ?, ?, ?)");
// ("flower_name, height, description");

preparedStatement.setString(1, "Mum");
preparedStatement.setInt(2, 10);
preparedStatement.setString(3, "Orange");
preparedStatement.executeUpdate();

// Show the added flowers
preparedStatement = connect
    .prepareStatement("SELECT flower_name, height, description from
marnie_rds.FLOWERS");
resultSet = preparedStatement.executeQuery();
System.out.println("The added flowers");
writeResultSet(resultSet);
System.out.println("");

// Delete a Record
statement = connect.createStatement();
sql = "DELETE FROM FLOWERS " +
```



```

        "WHERE flower_name = 'Mum'" ;
statement.executeUpdate(sql);

// Show the flowers after the delete
preparedStatement = connect
    .prepareStatement("SELECT flower_name, height, description from
marnie_rds.FLOWERS");
resultSet = preparedStatement.executeQuery();
System.out.println("The flowers after delete");
writeResultSet(resultSet);
System.out.println("");

    } catch (Exception e) {
        throw e;
    } finally {
        // Close Database connection
        close();
    }

}

private void writeResultSet(ResultSet resultSet) throws SQLException {
    // ResultSet is initially before the first data set
    while (resultSet.next()) {
        // It is possible to get the columns via name
        // also possible to get the columns via the column number
        // which starts at 1
        // e.g. resultSet.getString(2);
        String flower_name = resultSet.getString("flower_name");
        Integer height = resultSet.getInt("height");
        String description = resultSet.getString("description");
        System.out.println("Flower Name: " + flower_name);
        System.out.println("Height: " + height);
        System.out.println("Description: " + description);
    }
}

// You need to close the resultSet
private void close() {
    try {
        if (resultSet != null) {
            resultSet.close();
        }
    }
}

```

```
        if (statement != null) {  
            statement.close();  
        }  
  
        if (connect != null) {  
            connect.close();  
        }  
    } catch (Exception e) {  
  
    }  
}  
  
}
```

Output in Console

Flowers table before Insert

The added flowers

Flower Name: Daffodil

Height: 10

Description: Yellow

Flower Name: Mum

Height: 10

Description: Orange

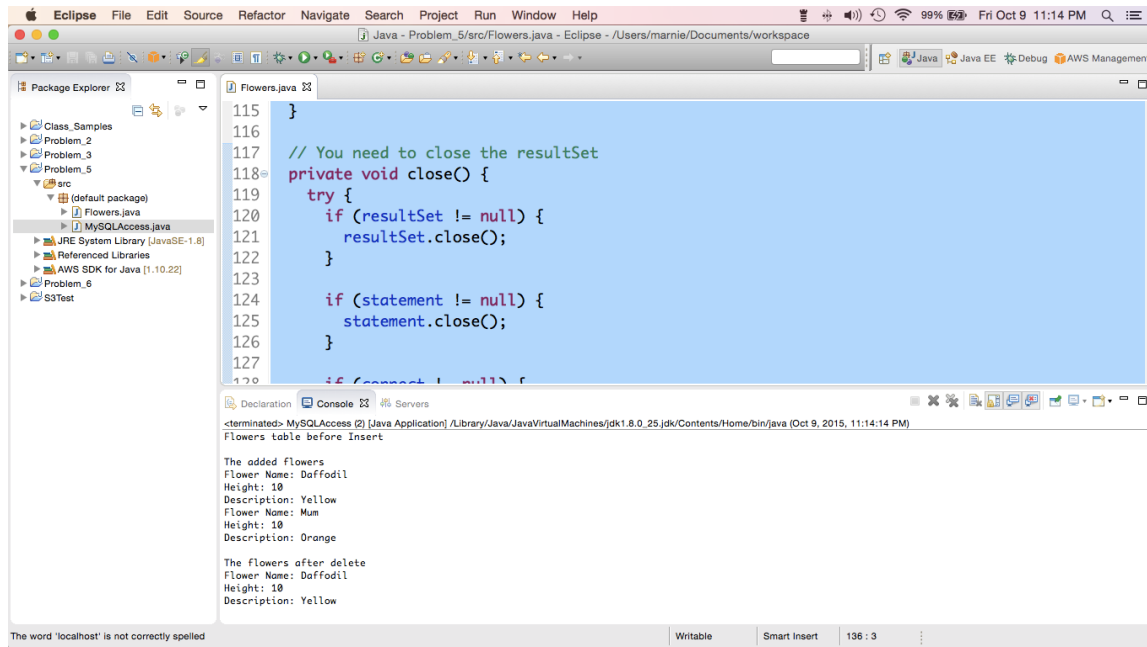
The flowers after delete

Flower Name: Daffodil

Height: 10

Description: Yellow

Screen Shot in Eclipse



Problem 6. Use AWS SDK for any programming language (e.g., Java, .NET, Python, ...).

Write a program that when given a bucket name will go out to S3 and wipe that bucket out. That program has to locate all objects in all the folders, first delete those objects, then delete those folders and finally delete the bucket. Try first with a single S3 Bucket folder with several objects (files). Finally write a program that could handle an arbitrarily nested folder structure with objects of any S3 bucket.

[20 points]

Write a program that when given a bucket name will go out to S3 and wipe a bucket out that contains a bucket with a single folder with several objects.

S3DeleteBucket.java

```
import java.io.BufferedReader;
```

```
import java.io.File;
```

```
import java.io.FileOutputStream;
```

```
import java.io.IOException;
```

E90_ScullyMarnieHW05.pdf

```
import java.io.InputStream;
```

```
import java.io.InputStreamReader;
```

```
import java.io.OutputStreamWriter;
```

```
import java.io.Writer;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.UUID;
```

```
import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.auth.AWSCredentials;
```

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
```

```
import com.amazonaws.regions.Region;
```

```
import com.amazonaws.regions.Regions;
```

```
import com.amazonaws.services.s3.AmazonS3;
```

```
import com.amazonaws.services.s3.AmazonS3Client;
```

```
import com.amazonaws.services.s3.model.Bucket;
```

```
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
```

```
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
```

```
import com.amazonaws.services.s3.model.DeleteObjectsResult;
```

```
import com.amazonaws.services.s3.model.GetObjectRequest;
```

```
import com.amazonaws.services.s3.model.ListObjectsRequest;

import com.amazonaws.services.s3.model.MultiObjectDeleteException;

import com.amazonaws.services.s3.model.ObjectListing;

import com.amazonaws.services.s3.model.PutObjectRequest;

import com.amazonaws.services.s3.model.S3Object;

import com.amazonaws.services.s3.model.S3ObjectSummary;


// Gratefully adapted from the Sample for S3 from amazon

public class S3DeleteBucket {

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

        /*

        * The ProfileCredentialsProvider will return your [marniescully]

        * credential profile by reading from the credentials file located at

        * (/Users/marnie/.aws/credentials).

        */

        AWSCredentials credentials = null;

        try {

            credentials = new ProfileCredentialsProvider("marniescully").getCredentials();

        } catch (Exception e) {

            throw new AmazonClientException(

                "Cannot load the credentials from the credential profiles file. " +
```

```
"Please make sure that your credentials file is at the correct " +  
"location (/Users/marnie/.aws/credentials), and is in valid format.",  
e);  
  
}  
  
AmazonS3 s3 = new AmazonS3Client(credentials);  
  
Region usEast1 = Region.getRegion(Regions.US_EAST_1);  
  
s3.setRegion(usEast1);  
  
String bucketName = "my-first-s3-bucket-" + UUID.randomUUID();  
  
String key1 = "MyObjectKey1";  
  
String key2 = "MyObjectKey2";  
  
String key3 = "MyObjectKey3";  
  
  
  
System.out.println("=====  
");  
  
System.out.println("Getting Started with Amazon S3");  
  
System.out.println("=====  
\n");  
  
try {  
  
    /*
```

- * Create a new S3 bucket - Amazon S3 bucket names are globally unique,
- * so once a bucket name has been taken by any user, you can't create
- * another bucket with that same name.

*

- * You can optionally specify a location for your bucket if you want to
- * keep your data closer to your applications or users.

*/

```
System.out.println("Creating bucket " + bucketName + "\n");
```

```
s3.createBucket(bucketName);
```

```
/*
```

- * List the buckets in your account

```
*/
```

```
System.out.println("Listing buckets");
```

```
for (Bucket bucket : s3.listBuckets()) {
```

```
    System.out.println(" - " + bucket.getName());
```

```
}
```

```
System.out.println();
```

```
/*
```

- * Upload an object to your bucket - You can easily upload a file to

- * S3, or upload directly an InputStream if you know the length of
- * the data in the stream. You can also specify your own metadata
- * when uploading to S3, which allows you set a variety of options
- * like content-type and content-encoding, plus additional metadata
- * specific to your applications.

*/

```
System.out.println("Uploading a new object to S3 from a file\n");
```

```
s3.putObject(new PutObjectRequest(bucketName, key1, createSampleFile()));
```

```
System.out.println("Uploading a new object to S3 from a file\n");
```

```
s3.putObject(new PutObjectRequest(bucketName, key2, createSampleFile()));
```

```
System.out.println("Uploading a new object to S3 from a file\n");
```

```
s3.putObject(new PutObjectRequest(bucketName, key3, createSampleFile()));
```

/*

- * List objects in your bucket by prefix - There are many options for
- * listing the objects in your bucket. Keep in mind that buckets with
- * many objects might truncate their results when listing their objects,
- * so be sure to check if the returned object listing is truncated, and
- * use the AmazonS3.listNextBatchOfObjects(...) operation to retrieve


```
* additional results.

*/

System.out.println("Listing objects");

ObjectListing objectListing = s3.listObjects(new ListObjectsRequest()

    .withBucketName(bucketName)

    .withPrefix("My"));

for (S3ObjectSummary objectSummary : objectListing.getObjectSummaries())
{

    System.out.println(" - " + objectSummary.getKey() + " " +

        "(size = " + objectSummary.getSize() + ")");

}

System.out.println();

/*

* This snippet was from the Amazon help file

* http://docs.aws.amazon.com/AmazonS3/latest/dev/DeletingMultipleObjectsUsingJava.html

*/

DeleteObjectsRequest multiObjectDeleteRequest = new
DeleteObjectsRequest(bucketName);

List<KeyVersion> keys = new ArrayList<KeyVersion>();
```

```
keys.add(new KeyVersion(key1));
```

```
keys.add(new KeyVersion(key2));
```

```
keys.add(new KeyVersion(key3));
```

```
multiObjectDeleteRequest.setKeys(keys);
```

```
try {
```

```
    DeleteObjectsResult delObjRes =  
s3.deleteObjects(multiObjectDeleteRequest);
```

```
    System.out.format("Successfully deleted all the %s items.\n",  
delObjRes.getDeletedObjects().size());
```

```
} catch (MultiObjectDeleteException e) {
```

```
    // Process exception.
```

```
}
```

```
/*
```

```
* Delete a bucket - A bucket must be completely empty before it can be
```

```
* deleted, so remember to delete any objects from your buckets before
```

```
* you try to delete them.
```

```
*/
```

```
        System.out.println("Deleting bucket " + bucketName + "\n");

        s3.deleteBucket(bucketName);

    } catch (AmazonServiceException ase) {

        System.out.println("Caught an AmazonServiceException, which means your
request made it "

            + "to Amazon S3, but was rejected with an error response for some
reason.");

        System.out.println("Error Message:  " + ase.getMessage());

        System.out.println("HTTP Status Code: " + ase.getStatusCode());

        System.out.println("AWS Error Code:  " + ase.getErrorCode());

        System.out.println("Error Type:      " + ase.getErrorType());

        System.out.println("Request ID:     " + ase.getRequestId());

    } catch (AmazonClientException ace) {

        System.out.println("Caught an AmazonClientException, which means the client
encountered "

            + "a serious internal problem while trying to communicate with S3, "

            + "such as not being able to access the network.");

        System.out.println("Error Message: " + ace.getMessage());

    }

}

private static File createSampleFile() throws IOException {
```

```
File file = File.createTempFile("aws-java-sdk-", ".txt");

file.deleteOnExit();

Writer writer = new OutputStreamWriter(new FileOutputStream(file));

writer.write("abcdefghijklmnopqrstuvwxyzn");

writer.write("01234567890112345678901234\n");

writer.write("!@#$%^&*()-=[]{};:'.<>/?\n");

writer.write("01234567890112345678901234\n");

writer.write("abcdefghijklmnopqrstuvwxyzn");

writer.close();

return file;

}

/**

 * Displays the contents of the specified input stream as text.

 *

 * @param input

 *         The input stream to display as text.

 *

 * @throws IOException
```

E90_ScullyMarnieHW05.pdf

*/

```
private static void displayTextInputStream(InputStream input) throws IOException {
```

```
    BufferedReader reader = new BufferedReader(new InputStreamReader(input));
```

```
    while (true) {
```

```
        String line = reader.readLine();
```

```
        if (line == null) break;
```

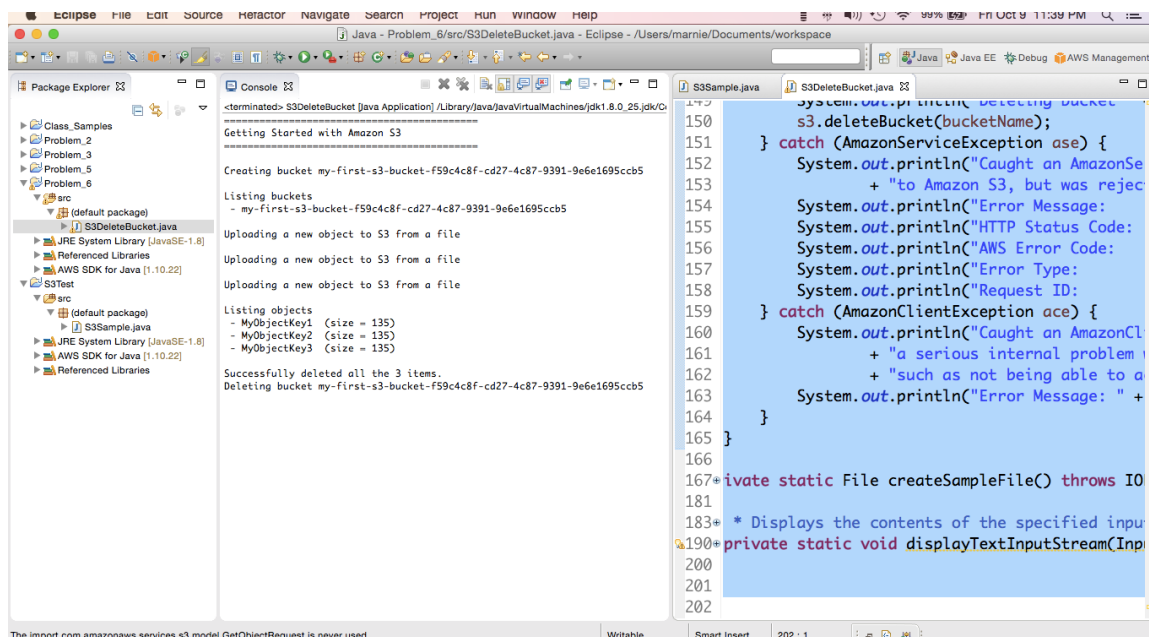
```
        System.out.println("    " + line);
```

```
    }
```

```
    System.out.println();
```

```
}
```

```
}
```



Output in Console

=====

Getting Started with Amazon S3

=====

Creating bucket my-first-s3-bucket-f59c4c8f-cd27-4c87-9391-9e6e1695ccb5

Listing buckets

- my-first-s3-bucket-f59c4c8f-cd27-4c87-9391-9e6e1695ccb5

Uploading a new object to S3 from a file

Uploading a new object to S3 from a file

Uploading a new object to S3 from a file

Listing objects

- MyObjectKey1 (size = 135)
- MyObjectKey2 (size = 135)
- MyObjectKey3 (size = 135)

Successfully deleted all the 3 items.

Deleting bucket my-first-s3-bucket-f59c4c8f-cd27-4c87-9391-9e6e1695ccb5

Write a program that when given a bucket name with an arbitrarily nested folder structure can wipe that bucket out.

I changed the code from

```
DeleteObjectsRequest multiObjectDeleteRequest = new
    DeleteObjectsRequest(bucketName);

    List<KeyVersion> keys = new ArrayList<KeyVersion>();

    keys.add(new KeyVersion(key1));

    keys.add(new KeyVersion(key2));

    keys.add(new KeyVersion(key3));

    multiObjectDeleteRequest.setKeys(keys);

    try {

        DeleteObjectsResult delObjRes =
s3.deleteObjects(multiObjectDeleteRequest);

        System.out.format("Successfully deleted all the %s items.\n",
delObjRes.getDeletedObjects().size());

    } catch (MultiObjectDeleteException e) {

        // Process exception.

    }
```

to

```
DeleteObjectsRequest multiObjectDeleteRequest = new
DeleteObjectsRequest(bucketName);

List<KeyVersion> keys = new ArrayList<KeyVersion>();

// Deletes any # of objects in bucket

for (S3ObjectSummary objectSummary : objectListing.getObjectSummaries()) {
```

```
        keys.add(new KeyVersion(objectSummary.getKey()));
    }

    multiObjectDeleteRequest.setKeys(keys);

    try {

        DeleteObjectsResult delObjRes =
            s3.deleteObjects(multiObjectDeleteRequest);

        System.out.format("Successfully deleted all the %s items.\n",
            delObjRes.getDeletedObjects().size());

    } catch (MultiObjectDeleteException e) {

        // Process exception.

    }
```

Output on Console

```
=====
Getting Started with Amazon S3
=====
```

Creating bucket my-first-s3-bucket-9e41957e-9e37-4555-8b31-ab2df6b7bb1e

Listing buckets

- my-first-s3-bucket-9e41957e-9e37-4555-8b31-ab2df6b7bb1e

Uploading a new object to S3 from a file

E90_ScullyMarnieHW05.pdf

Uploading a new object to S3 from a file

Uploading a new object to S3 from a file

Uploading a new object to S3 from a file

Listing objects

- MyObjectKey1 (size = 135)
- MyObjectKey2 (size = 135)
- MyObjectKey3 (size = 135)
- MyObjectKey4 (size = 135)

Successfully deleted all the 4 items.

Deleting bucket my-first-s3-bucket-9e41957e-9e37-4555-8b31-ab2df6b7bb1e

```
120      System.out.println();
121
122      /*
123       * This snippet was from
124       * http://docs.aws.amazon.com/
125       */
126      DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(bucketName);
127
128      List<KeyVersion> keys = new ArrayList<>();
129      keys.add(new KeyVersion("MyObjectKey1", 135));
130      keys.add(new KeyVersion("MyObjectKey2", 135));
131      keys.add(new KeyVersion("MyObjectKey3", 135));
132      keys.add(new KeyVersion("MyObjectKey4", 135));
133      multiObjectDeleteRequest.setKeys(keys);
134
135      try {
136          DeleteObjectsResult deleteObjectsResult = s3.deleteObjects(bucketName, multiObjectDeleteRequest);
137          System.out.format("Successfully deleted all the %d items.\n", deleteObjectsResult.getDeletedObjects().size());
138      } catch (MultiObjectDeleteException e) {
139          // Process exception.
140      }
141
142
143
```

Console Output:

```
<terminated> S3DeleteBucket_Arbitrary [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java
Getting Started with Amazon S3
Creating bucket my-first-s3-bucket-9e41957e-9e37-4555-8b31-ab2df6b7bb1e
Listing buckets
- my-first-s3-bucket-9e41957e-9e37-4555-8b31-ab2df6b7bb1e
Uploading a new object to S3 from a file
Uploading a new object to S3 from a file
Uploading a new object to S3 from a file
Uploading a new object to S3 from a file
Listing objects
- MyObjectKey1 (size = 135)
- MyObjectKey2 (size = 135)
- MyObjectKey3 (size = 135)
- MyObjectKey4 (size = 135)
Successfully deleted all the 4 items.
Deleting bucket my-first-s3-bucket-9e41957e-9e37-4555-8b31-ab2df6b7bb1e
```