**Performance Based Programming**

**Assignment**

**Liam Dorrian**

**L00108099**

**Using a stopwatch to determine if using secure coding, mutability and garbage collector effects performance levels of a program.**

## 

## 1.0Introduction

This assignment was to develop a software artefact using techniques to help make the code more secure. The topics chosen for this artefact where secure coding, garbage collector and mutability. The Artefact which was created contained a number of classes that used different techniques to help resolve different threading problems.

In order to check if making code more secure does affect performance two artefacts will have to be created. One will use a prepared statement and will have secure coding procedures, garbage collector and mutability. The other programs will use the standard jdbc method and will have no other features. Both artefact will allow the user to insert, delete and update records and uses a switch to allow easy navigation through the artefact.

A stopwatch class was also created the stopwatch class will be used to tell the time it took for each of insertions, deletion and updated to occur. The stopwatch class will be used in both versions of the artefact. A user class was also created which was used to help print out all the details from the database. This will only be used for the prepared statement version of the artefact. A sample database called user details was created and it contained over 10,000 records as this would be a reasonable size to measure the performance levels.

## 1.1Title

Using a stopwatch to determine if using secure coding, mutability and garbage collector effects performance levels of a program.

**2.1Research overview**

### **2.2 Garbage Collector**

In the Java programming language, dynamic allocation of objects is achieved using the garbage collector. The Garbage collector is a memory management feature of the java programming language. The Garbage collector combats heap fragmentation (Venners,2015). An example of a garbage collector is a daemon thread. The garbage collector also helps ensure program integrity, ensuring better performance and security measures. It makes java memory more efficient because garbage collector removes the unreferenced objects from heap memory. It allows the user to not worry about freeing up allocated memory. Figure 2.1 below, demonstrates the Garage collector in conjunction with used memory within the system, providing the figures before and after the method is called.

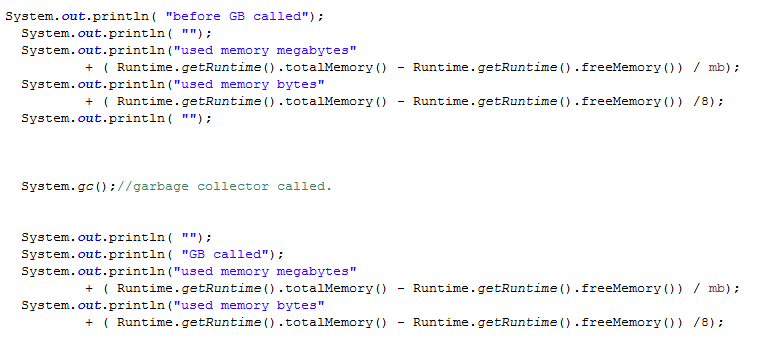


Fig1.1 Code used for the garbage collector.

### **2.3 Secure Coding**

Secure coding can be most intuitively defined as the act if writing secure programs. Secure programs are programs that cannot be manipulated into performing illegal operations. (Bar-el,2015). Secure coding has become big in recent years as security gaps in software can have many implications for companies. Last year TalkTalk lost over 60 million and lost nearly 100,000 after a cyber-attack (Farrell, 2016). Also the use of prepared statements for the artefact will ensure that dos (denial of service) attacks are much hard to take place. Dedicated website such as cert have been setup to help teach secure coding practices and have setup 10 principal techniques ( Seacord,2011).  Here are some of the techniques that were used in the artefact.

1. **Validate input.** Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software [vulnerabilities](https://www.securecoding.cert.org/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability). Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files (Seacord, 2005).

**Keep it simple.** Keep the design as simple and small as possible (Saltzer ,1975). Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex.

### **2.4 Mutability**

For this artefact immutable objects will be used as they make the code more secure. The following sentences explain is the difference between the two objects.

Immutable Objects are when you have a reference to an instance of an object, the contents of that instance cannot be altered. The opposite of immutable is Mutable Objects. Mutable objects are When you have a reference to an instance of an object, the contents of that instance can be altered (O'Meara,2003).

Normally immutability in java is achieved through following steps (Gupta,2013).

1. Don’t provide mutator methods for any field
2. Make all fields final and private
3. Don’t allow subclasses by declaring the class final itself
4. Return deep cloned objects with copied content for all mutable fields in class

Using a final String means that the string cannot be changed which is good for security reasons. Fig 1.2 shows how in this artefact how immutable objects were used in thus artefact. The user class also had no setters which is another part of immutability.

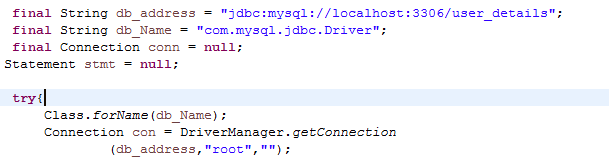


Fig1.2 example of code use to make object immutable

**3.0 Tests conducted**

**3.1 Test One**

The first test that was carried out was a simple but effective and was see how easy it was to crash the two versions of the artefact by using unit testing. Some of the test that where carried where check to what happens when an into is entered in for a string and vice versa. The more exception handling in the code decrease the chance of it been broken. Additional teste where done on the version 2 of the artefact. The test where conducted where too see what happens when the user tries to delete a user\_id not present in the database.

**3.2 Results test one**

The test was a very simple but worthwhile exercise. The artefact without any exception handling was very easy to break which was expected. Whereas the second version of the artefact was a lot more difficult as there was a lot of exception handling and also a try catch finally was used which is another additionally security measure was used. When the user tries to delete a record not present a simple message comes up on the screen asking them to enter a valid number and keeps looping around until a correct user\_id is found. Fig2.1 shows an example of the exception handling being used.

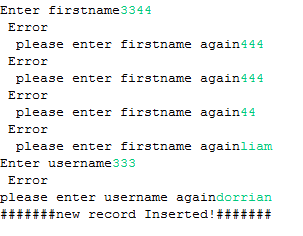


Fig2.1 showing Exception handling in action

**Test Two**

The second test carried out is too see how adding all these extra security features will have an impact on the performance on the artefact. Version will use the jdbc connection method and has no other extra functionality added to it. The stopwatch class was used to time how long each the insert, delete, and update methods took to perform.

In version two of the artefact it used a prepared statement and had extra functionality such as secure coding, garbage collector and immutable objects. The stopwatch was again used for this version to time the objects

**2.3 Results test one**

The results from this test shows that using a program that contains secure coding, mutability and garbage collector do have an effect on the performance. All the times where calculated in milli seconds. For the Jdbc connection the insert only took 6 seconds. Whereas the prepared statement took 54 seconds which is a big gap between them both. The delete method had similar results as the prepared statement took 37 seconds which was 30 seconds slower than the jdbc. The prepared statement was again roughly 30 seconds slower again then the jdbc. The select all form the database gave a surprise results as the prepared statement was a lot quicker than the jdbc statement. A factor in it being quicker was creating the user class which was used to help print out everything from the database. The prepared statement was 700 milliseconds quicker than the jdbc connection. Also in the second version, once the garbage collector is called the number of megabytes and bytes used decreased which was expected. In each of the methods we can see the used bytes and megabytes before and after the garbage collector was called

**4.0 Conclusions**

Overall I feel I have a greater understanding into how adding security measures to code will have adverse effects for performance. I also have gained important information into the topic chosen which where secure coding, garbage collector and mutability. It was also the first time I have properly used the garbage collector and mutability and it was a steep learning curve. Hopefully this knowledge gained will be useful for the exam in January. I found my lectures notes on blackboard useful especially the ones on the garbage collector. I found the secure coding part of this artefact very interesting as it was challenging to try and get my system to be unbreakable but one that was very interesting. Before this assignment very little thought went into how I could make me code secure. But now I will have to take secure coding as a must for all my programs. Lastly making objects immutable was very worthwhile, as I never heard of immutable objects before. At the start I was getting a lot of errors while working with mainly they were due to private final instead of jest final

Overall this artefact has been challenging but will help me understand why we code things in certain way and how we can improve our code.

**5.0 References.**

David O'Meara. (2003). *Mutable and Immutable Objects.* Available: http://www.javaranch.com/journal/2003/04/immutable.htm. Last accessed 06 dec 2016.

Hagai bar-el. (2015). *The important of secure cosing.* Available: http://www.infosecwriters.com/text\_resources/pdf/On\_The\_Importance\_of\_Secure\_Coding.pdf. Last accessed 06 dec 2016.

Lokesh Gupta. (2013). *Why Strings are Immutable in Java?.* Available: http://howtodoinjava.com/core-java/string-class/java-interview-question-why-strings-are-immutable/. Last accessed 06 dec 2016.

[Saltzer 74] Saltzer, J. H. "Protection and the Control of Information Sharing in Multics." Communications of the ACM 17, 7 (July 1974): 388-402.

[Saltzer 75] Saltzer, J. H. & Schroeder, M. D. "The Protection of Information in Computer Systems." Proceedings of the IEEE 63, 9 (September 1975), 1278-1308.

[Seacord 05] Seacord, R. Secure Coding in C and C++. Upper Saddle River, NJ: Addison-Wesley, 2006 (ISBN 0321335724).

[Swiderski 04] Swiderski, F. & Snyder, W. Threat Modeling. Redmond, WA: Microsoft Press, 2004.

Sean Farrell. (2016). *TalkTalk counts costs of cyber-attack.* Available: https://www.theguardian.com/business/2016/feb/02/talktalk-cyberattack-costs-customers-leave. Last accessed 6 dec 2016.

Robert Seacord, (2011). *Top 10 Secure Coding Practices.* Available: https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices. Last accessed 05 dec 2016.

Venners, B. (2015). Why Garbage Collection?. [online] Artima.com. Available at: https://www.artima.com/insidejvm/ed2/gc.html [Accessed 29 Nov. 2016].

**6.0 Appendix**

**Stopwatch class**

**package** core;

**public** **class** Stopwatch {

**private** **long** elapsedTime;

**private** **long** startTime;

**private** **boolean** isRunning;

//Constructor set to null

**public** Stopwatch()

{

reset();

}

//Starts the stopwatch. Time starts accumulating now.

**public** **void** start()

{

**if** (isRunning)

**return**;

isRunning = **true**;

startTime = System.*currentTimeMillis*();

}

//Stops the stopwatch. Time stops accumulating & is added to the elapsed time.\_

**public** **void** stop()

{

**if** (!isRunning)

**return**;

isRunning = **false**;

**long** endTime = System.*currentTimeMillis*();

elapsedTime = elapsedTime + endTime - startTime;

}

//Returns the total elapsed time

**public** **long** getElapsedTime()

{

**if** (isRunning)

{

**long** endTime = System.*currentTimeMillis*();

**return** elapsedTime + endTime - startTime;

}

**else**

**return** elapsedTime;

}

// Stops the watch and resets the elapsed time to 0

**public** **void** reset()

{

elapsedTime = 0;

isRunning = **false**;

}

}

**User class**

**package** core;

**public** **class** User\_class {

**int** id;

String fname;

String uname;

/\*\*

\* **@param** id

\* **@param** fname

\* **@param** uname

\*/

**public** User\_class(**int** id, String fname, String uname) {

**super**();

**this**.id = id;

**this**.fname = fname;

**this**.uname = uname;

}

**public** User\_class(){

}

**private** **int** getId() {

**return** id;

}

**private** String getFname() {

**return** fname;

}

**private** String getUname() {

**return** uname;

}

@Override

**public** String toString() {

**return** "User\_class [id=" + id + ", fname=" + fname + ", uname=" + uname + "]";

}

}

**Version 1 class**

**package** core;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.util.Scanner;

**public** **class** Assignment2

{

// JDBC driver name and database URL

**static** **final** String ***JDBC\_DRIVER*** = "com.mysql.jdbc.Driver";

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost/user\_details";

// Database credentials

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "";

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, SQLException {

Connection conn = **null**;

Statement stmt = **null**;

Scanner kb = **new** Scanner(System.***in***);

Stopwatch stopStackInser = **new** Stopwatch();

String sql = **null**;

//STEP 2: Register JDBC driver

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

//STEP 3: Open a connection

System.***out***.println("Connecting to a selected database...");

conn = DriverManager.*getConnection*(***DB\_URL***,***USER***,***PASS***);

System.***out***.println("Connected database successfully...");

//STEP 2: Register JDBC driver

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

//STEP 3: Open a connection

stmt = conn.createStatement();

System.***out***.println("#### Options#######");

System.***out***.println("1)Add records");

System.***out***.println("2)delete records");

System.***out***.println("3)update records");

System.***out***.println("4)view records");

System.***out***.println("5)Exit");

System.***out***.print("please eneter record");

**int** option=kb.nextInt();

**switch**(option)

{

**case** 1:

System.***out***.print("Enter firstname");

kb.nextLine();

String fname =kb.nextLine();

System.***out***.print("Enter usertname");

String uname =kb.nextLine();

stopStackInser.start();

sql="insert into user\_details values( "+0+" , ' "+fname+" ', ' "+uname+" ' )";

stmt.execute(sql);

System.***out***.println("Time Taken "+stopStackInser.getElapsedTime());

stopStackInser.reset();

**break**;

**case** 2:

System.***out***.print("Enter id to delete");

**int** id =kb.nextInt();

stopStackInser.start();

sql = "DELETE FROM user\_details WHERE user\_id = "+id;

stmt.executeUpdate(sql);

stopStackInser.stop();

System.***out***.println("Time Taken "+stopStackInser.getElapsedTime());

stopStackInser.reset();

**break**;

**case** 3:

System.***out***.print("Enter id to update");

kb.nextLine();

id =kb.nextInt();

System.***out***.print("Enter firstname");

kb.nextLine();

fname =kb.nextLine();

System.***out***.print("Enter usertname");

uname =kb.nextLine();

stopStackInser.start();

sql ="UPDATE user\_details set values firstname='"+fname+"'username='"+uname+"' WHERE user\_id= '"+id+"'";

stmt.execute(sql);

stopStackInser.stop();

System.***out***.println("Time Taken "+stopStackInser.getElapsedTime());

stopStackInser.reset();

**break**;

**case** 4:

String sql1 = "SELECT \* FROM user\_details";

stopStackInser.start();

ResultSet rs = **null**;

rs = stmt.executeQuery(sql1);

//Processing the ResultSet object

**while** (rs.next())

{

System.***out***.println("ID :"+rs.getInt(1));

System.***out***.println("First Name : "+rs.getString(2));

System.***out***.println("Last Name :"+rs.getString(3));

System.***out***.println("-------------------");

}

System.***out***.println("Time Taken "+stopStackInser.getElapsedTime());

stopStackInser.reset();

**break**;

**default** :System.***out***.println("Invalid grade");

}

}

}

**Version 2 class**

package core;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.InputMismatchException;

import java.util.Scanner;

//liam dorrian

//Applied Computing

public class Assignment2\_New {

public static void main(String[] args)

{

final String db\_address = "jdbc:mysql://localhost:3306/user\_details";

final String db\_Name = "com.mysql.jdbc.Driver";

final Connection conn = null;

Statement stmt = null;

try{

Class.forName(db\_Name);

Connection con = DriverManager.getConnection

(db\_address,"root","");

int id = 0;

String fname;

String uname;

final int mb =1024\*1024;

int rowsUpdated;

int option = 0;

Stopwatch stopStackInser = new Stopwatch();

Scanner kb = new Scanner(System.in);

do{

//Menu System

System.out.println("#### Options#######");

System.out.println("1)Add records");

System.out.println("2)delete records");

System.out.println("3)update records");

System.out.println("4)view records");

System.out.println("5)Exit");

System.out.println("#################");

System.out.println("Please input an integer value [1-5]");

while(!kb.hasNextInt())

{

kb.next();

System.out.println(" Error ");

System.out.print("Please input an integer value [1-5]");

}

option = kb.nextInt();

//switch statement

switch(option)

{

case 1:

System.out.println("");

System.out.println( "before GB called");

System.out.println("used memory megabytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / mb);

System.out.println("used memory bytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) /8);

System.out.println( "");

System.out.print("Enter firstname");

kb.nextLine();

fname =kb.nextLine();

//making sure no int can be passed in

while(!fname.matches("[a-zA-Z]+$"))

{

System.out.println(" Error ");

System.out.print(" please enter firstname again");

fname =kb.nextLine();

}

System.out.print("Enter username");

uname =kb.nextLine();

//making sure no int can be passed in

while(!uname.matches("[a-zA-Z]+$"))

{

System.out.println(" Error ");

System.out.print("please enter username again");

uname =kb.nextLine();

}

PreparedStatement addeemp = con.prepareStatement

("insert into user\_details values(?,?,?)");

addeemp.setInt(1,0);

addeemp.setString(2, fname);

addeemp.setString(3, uname);

stopStackInser.start();

rowsUpdated = addeemp.executeUpdate();

//ensuring record in inserted.

if (rowsUpdated > 0) {

System.out.println("#######new record Inserted!#######");

}

else

{

System.out.println("########RECORD NOT INSERTED#######");

}

System.gc();//calling garbage collector

System.out.println( "");

System.out.println( "GB called");

System.out.println("used memory megabytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / mb);

System.out.println("used memory bytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) /8);

stopStackInser.stop();

System.out.println( "");

System.out.println("Time Taken "+stopStackInser.getElapsedTime());

stopStackInser.reset();

break;

case 2:

System.out.println( "before GB called");

System.out.println( "");

System.out.println("used memory megabytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / mb);

System.out.println("used memory bytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) /8);

System.out.println( "");

System.out.print("Enter id to delete");

//making sure no string can be passed in

while(!kb.hasNextInt())

{

kb.next();

System.out.println(" Error ");

System.out.print("Enter a valid id to delete");

}

id = kb.nextInt();

System.out.print("");

stopStackInser.start();

PreparedStatement deleteeemp = con.prepareStatement

("DELETE from USER\_DETAILS WHERE USER\_ID =?");

deleteeemp.setInt(1,id);

rowsUpdated = deleteeemp.executeUpdate();

// checking if record went trough

if (rowsUpdated > 0)

{

System.out.println("new record Deleted!");

}

else

{

System.out.println("########RECORD NOT FOUND#######");

}

System.gc();//garbage collecter called

System.out.println( "");

System.out.println( "GB called");

System.out.println("used memory megabytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / mb);

System.out.println("used memory bytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) /8);

stopStackInser.stop();

System.out.println( "");

System.out.println("Time Taken "+stopStackInser.getElapsedTime());

stopStackInser.reset();

break;

case 3:

System.out.println( "before GB called");

System.out.println("used memory megabytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / mb);

System.out.println("used memory bytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) /8);

System.out.println("");

System.out.print("Enter id to update");

while(!kb.hasNextInt())

{

kb.next();

System.out.println(" Error ");

System.out.print("Id not found please try again");

}

id = kb.nextInt();

kb.nextLine();

System.out.println("Enter firstname");

fname =kb.nextLine();

System.out.print("");

while(!fname.matches("[a-zA-Z]+$"))

{

System.out.println(" Error ");

System.out.print(" please enter firstname again");

fname =kb.nextLine();

}

System.out.print("Enter username");

uname =kb.nextLine();

while(!uname.matches("[a-zA-Z]+$"))

{

System.out.print(" ERROR ,please enter username again");

uname =kb.nextLine();

}

stopStackInser.start();

PreparedStatement updateemp = con.prepareStatement

("UPDATE user\_details SET username = ? , first\_name = ? where user\_id = ?");

updateemp.setString(1, uname);

updateemp.setString(2,fname);

updateemp.setInt(3,id);

rowsUpdated = updateemp.executeUpdate();

if (rowsUpdated > 0)

{

System.out.println("new record Updated!");

}

else

{

System.out.println("########RECORD NOT UPDATED#######");

}

System.gc();//garbage collector called.

System.out.println( "");

System.out.println( "GB called");

System.out.println("used memory megabytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / mb);

System.out.println("used memory bytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) /8);

stopStackInser.stop();

System.out.println("");

System.out.println("Time Taken "+stopStackInser.getElapsedTime());

stopStackInser.reset();

break;

case 4:

System.out.println( "before GB called");

System.out.println( "");

System.out.println("used memory megabytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / mb);

System.out.println("used memory bytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) /8);

System.out.println( "");

stmt = con.createStatement();

String query = "select \* from user\_details";

System.gc();//garbage collector called.

stopStackInser.start();

ResultSet rs = stmt.executeQuery(query);

User\_class user = new User\_class();

while (rs.next())

{

user.id = rs.getInt("user\_id");

user.uname = rs.getString("username");

user.fname = rs.getString("first\_name");

System.out.println( user.id + " " + user.uname+" "+ user.fname);

stopStackInser.stop();

}

System.out.println( "");

System.out.println( "GB called");

System.out.println("used memory megabytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / mb);

System.out.println("used memory bytes"

+ ( Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) /8);

System.out.println("Time Taken "+stopStackInser.getElapsedTime());

stopStackInser.reset();

rs.close();

break;

case 5:

break;

default :System.out.println("Invalid number !! Try again");

break;

}

}while(option!=5);

}catch(SQLException se){

//Handle errors for JDBC

se.printStackTrace();

}catch(Exception e){

//Handle errors for Class.forName

e.printStackTrace();

}finally{

//finally block used to close resources

try{

if(stmt!=null)

conn.close();

}catch(SQLException se)

{

}// do nothing

try{

if(conn!=null)

conn.close();

}catch(SQLException se){

se.printStackTrace();

}//end finally try

}//end try

System.out.println("Goodbye!");

}

}