Q1 (a) The Java application outlined in Figure 1 is single threaded and results in the

output shown in Figure 2.

Rewrite the application using the Java Concurrency Framework, replacing the

**display**’ method with ‘**run**’, ensuring that the threads run concurrently and the

resulting output is similar to that shown in Figure3.

**class** Base{

**public** String first;

**public** Base(String f){

first=f;

}

**public** String getName(){**return** first;}

}

**class** Display **extends** Base{

**public** String second;

**public** Display(String f, String s){

**super**(f);

second=s;

}

**public** **void** display(){

**for**(**int** i=0; i<4;i++)

{ System.*out*.println(**super**.getName() +" " + second+": "+i);

**try**{ Thread.*sleep*(1000);}

c**atch**(Exception e){}

}

}

}

**public** **class** Q1a

{

**public** **static** **void** main(String[] args)

{ Display d1=**new** Display("John","Smith");

Display d2=**new** Display("Peter","Doyle");

d1.display();

d2.display();

}

}

Figure 1

Figure 1

John Smith: 0

Peter Doyle: 0

John Smith: 1

Peter Doyle: 1

John Smith: 2

Peter Doyle: 2

John Smith: 3

Peter Doyle: 3

John Smith: 0

John Smith: 1

John Smith: 2

John Smith: 3

Peter Doyle: 0

Peter Doyle: 1

Peter Doyle: 2

Peter Doyle: 3

Figure 2 Figure 3 (10 Marks)

(b) The Go method in the following class is not thread safe

**class** ThreadUnsafe **extends** Thread {

**static** **int** *val1*=9, *val2*=2, *delay*=10;

**void** Go(){

**if** (*val2* != 0) {

**try** {Thread.*sleep*(*delay*);}**catch**(Exception e){}

System.*out*.println( *val1* / *val2*);

*val2*=0;}

**else** { System.*out*.println( "Val2=0");}

}

**public** **void** run(){

Go();

}

**public** **static** **void** main( String[] args )

{ ThreadUnsafe t1=**new** ThreadUnsafe();

ThreadUnsafe t2=**new** ThreadUnsafe();

*val1*=9; *val2*=2;

t1.start();

*delay*=1;

t2.start();

}

}

1. Explain how each of the following results can occur

4

Exception in thread "Thread-0" java.lang.ArithmeticException: / by zero

at ThreadUnsafe.Go(ThreadUnsafe.java:8)

at ThreadUnsafe.run(ThreadUnsafe.java:13)

4

4

4

Val2=0

(5 marks)

1. Show how the lock construct can solve this problem and explain how it works.

(5 marks)

**[20 marks]**

Q2 (a) Given the following incomplete Java code for a Binary Search Tree.

class Node {

int key;

String name;

Node left;

Node right;

Node(int k, String nm, Node ll, Node rr)

{ key = k;

name = nm;

left = ll; right = rr; }

}

public class Tree

{ private Node head;

private Node tail;

public Tree() { tail = new Node(0, null, null, null);

head = new Node(-1, null, null, tail);}

public **void insert(int k, String nm)**

:

1. Complete the Java implementation for the **insert**  function

(6 marks)

1. Briefly explain why the performance of the search function is

~ O( 1 + log2 N)

(6 marks)

(b) Given the following incomplete Java application which uses the TreeSet Collection

public class TreeSet1

{

static int sum(TreeSet set){ **// to be completed** }

static void print(TreeSet set) { **// to be complete**

}

public static void main( String[] args )

{

TreeSet <Integer> set = new TreeSet();

Collections.*addAll*(set , 4,5,3,2,3,1,3);

*print*(set);

**int** res1=*max*(set);

System.*out*.println("Maximum element="+res1);

**int** res2=*sum*(set);

System.*out*.println("Sum of element="+res2);

}

}

Outline a java implementation for the two functions

1. **int sum(TreeSet set)**

returns the sum of the elements of the set (15 in the example above)

1. **void print(TreeSet set)**

prints out the elements of the set in the format: { 1, 2, 3, 4, 5 }

(8 Marks)

**[20 Marks]**

Q3(a) The following class defines an array of integer values and has member functions to

access the array as shown:

class ArrayTest

{

private int[] values;

public ArrayTest(int[] v)

{

values = v;

}

public int first()throws Exception

{

if (values.length == 0) throw new Exception();

return values[0];

}

public int max()throws Exception

{

if (values.length == 0) throw new Exception();

int res = values[0];

for (int i=0;i< values.length;i++)

{

if (res< values[i]) res = values[i];

}

return res;

}

public void printAll()

{

System.out.println(); System.out.print("[");

for (int i=0;i< values.length;i++)

System.out.print(" " + values[i]);

System.out.println("]");

}

}

An object of this type can be defined and used as follows:

int[] a = { 2, 6, 3, 5 };

ArrayTest at = new ArrayTest(a);

int res1 = at.first();

System.out.println("First=" + res1);

int res2 = at.max();

System.out.println("max=" + res2);

at.printAll();

1. Use java Generics to rewrite the ‘ArrayTest’ class so it can be used with any numeric type ‘int’, ‘float’, ‘double’ etc
2. Outline code that defines and uses an object of this new class.

(11 Marks)

(b) Given the following incomplete Java application which uses the ArrayList Collection

**public** **class** ArrayList1

{

**static** **int** sum(ArrayList list){ **// to be completed**

}

**public** **static** **void** main( String[] args )

{

ArrayList<Integer> list = **new** ArrayList();

Collections.addAll(list , 6, 3, 1, 4);

**int** res1=sum(**new** ArrayList(list));

System.out.println("Sum of all elements=" +res1);

}

}

Outine a java **recursive** implementation for the function

**static int sum(ArrayList list)**

which returns the sum of all element in the list

i.e. sum([6,3,1,4]) returns 14

(9 Marks)

**[20 Marks]**

Q4(a) In the proposed application new Customers enter their name and age and are then

allocated a base price for a product. Customers under the age of 17 are classified as

children and qualify for a 50% price reduction.

Based on these requirements and the UML class diagram below outline a complete

application demonstrating how Java realizes the Factory Pattern.

The main method is given in **appendix1** but you must complete the Customer,

RegularCustomer, ChildCustomer and CustomerFactory classes.

<<abstract>>

Customer

string name

int age

double price

Customer(String,int,double)

void setNewPrice(double p)

abstract double readPrice()

RegularCustomer

RegularCustomer(String,int,double)

double readPrice()

ChildCustomer

ChildCustomer(String,int,double)

double readPrice()

(10 marks)

(b) The Java application outlined below allows any number of Server objects to be

created. Show how the application can be modified using the Singleton pattern to

ensure that only one Server object will ever be created.

class Server

{ private String ipnumber="172.16.254.4";

private String name="Gamma";

private String location="Athlone";

private int sequence=0;

private static int *count*=0;

public Server()

{ *count*++;

sequence=count;

}

public String readNumber(){return ipnumber;}

public void print\_details()

{

System.*out*.println("Name:= " + name);

System.*out*.println("ipNumber:= " + ipnumber);

System.*out*.println ("Location:= " + location);

System.*out*.println ("Sequence:= "+sequence);

System.*out*.println ();

}

}

public class SingletonTest{

public static void main(String[] args)

{

Server s1 = new Server();

Server s2 = new Server();

s1.print\_details();

s2.print\_details();

}

}

(10 marks)

**[20 Marks]**

**Appendix 1**

**public** **class** Ex1{

**public** **static** **void** main(String[] args ){

**double** origPrice=62.48;

CustomerFactory cfactory = **new** CustomerFactory();

String name=Console.*readLine*("Enter Name:");

**int** age=Console.*readInt*("Enter Age:");

Customer c = cfactory.getCustomer(name,age,origPrice);

**int** choice=1; **double** val=0;

**while**(choice!=3){

System.*out*.println("1: Enter New Price");

System.*out*.println("2: Read Price for Customer:");

System.*out*.println("3: Exit");

System.*out*.println();

choice=Console.*readInt*("Enter Choice:");

**switch**(choice){

**case** 1:

val=Console.*readDouble*("Enter Price");

c.setNewPrice(val); **break**;

**case** 2: **double** res=(**double**)c.readPrice();

System.*out*.println("Price: "+res);

**break**; }