Written work

1. Which of the following are not valid Java identifiers, and why?

(i) `wolVes`

(ii) `United(there_is_only_one)`

(iii) `_87`

(iv) `5_3`

(v) `Real_ale`

(vi) `isFound?by`

(ii) Not valid because the ( ) would be expecting a parameter.
(iv) Not valid because you cannot begin an identifiers with a number.
(vi) Not valid because the ? is an illegal character.

2. A class Television has the following fields:

```
private TelevisionManufacturer supplier;
private String code;
private int screenSize; // in inches
private String type; // e.g.  plasma screen
```

Assume that the class **TelevisionManufacturer** is available and that this class contains an **equals** method.

. (i) Define a class variable, **totalTVs**, whose purpose is to keep track of the total number of Television objects constructed during execution of a program that uses the **Television** class.

```
public totalTVs()
{
     totalTVs++;
}
```

. (ii) Declare a default constructor for this class.

```
public Television()
```

. iii) Declare a constructor for this class that has a formal parameter corresponding to each field.

```
public Television( String givenSupplier, String givenCode,
int givenSize, String givenType)
{
     supplier = givenSupplier;
     code = givenCode;
     screenSize = givenSize;
     type = givenType;
}
```

. (iv) Declare an accessor method called **getScreenSize** whose purpose is to return the value of the **screenSize** field of **this Television**.

```
public int getScreenSize()
{
     return screenSize;
}
```

. (v) Declare a mutator method that sets the type of **this Television** to a given value.

```
public void settingTheType(String setType)
{
     type = setType;
}
```

. (vi) Declare a method to determine whether or not **this Television** has been supplied by a given manufacturer.

```
public boolean suppliedBy( Television givenManufacturer)
{
          return supplier.equals(givenManufacturer);
}
```

Question 1:

```
package degreeclassificationdemo;
import java.util.*;
/**
 * input mark, returns classification, returns the min and max for
 * the classification.
 * @author James Vine - 100022010
 */
public class DegreeClassificationDemo
{

    public enum Classification //5 enum types
    {
        Fail,III,II2,II1,I
    }

    public static void main(String[] args) //main method
    {
        int mark;

        Scanner scan = new Scanner(System.in);
        System.out.print("Enter result:                        ");
        mark = scan.nextInt();
        Classification classification = findClassification(mark);
        System.out.println("Classification:                      "
                           + classification );
        int[] minMax = markRange(classification);
        System.out.println("The minimum and maximum marks are: "
                           + minMax[0] + " and " + minMax[1]);
    }

    public static Classification  findClassification (int mark)
    {
        if(mark <= 39)
        {
            return Classification.Fail;
        }
        else if(mark <= 49)
        {
            return Classification.III;
        }
        else if(mark <= 59)
        {
            return Classification.II2;
        }
        else if(mark <= 69)
        {
            return Classification.II1;
        }

        return Classification.I;
    }
```

```
public static int [] markRange(Classification degreeClass)
    {
        int[] range;
        range = new int[2];
        switch (degreeClass)
        {
            case Fail:
                range[0]=0;   //array already initialises as zero
                range[1]=39;
                break;

            case III:
                range[0]=40;
                range[1]=49;
                break;

            case II2:
                range[0]=50;
                range[1]=59;
                break;

            case II1:
                range[0]=60;
                range[1]=69;
                break;

            case I:
                range[0]=70;
                range[1]=100;
                break;

            default: //throws an error but should never happen
                throw new Error("invalid classification type");
         }
        return range;
    }
}
```

```
run:
Enter result:                          34
Classification:                        Fail
The minimum and maximum marks are: 0 and 39
BUILD SUCCESSFUL (total time: 2 seconds)
```

```
run:
Enter result:                         45
Classification:                       III
The minimum and maximum marks are: 40 and 49
BUILD SUCCESSFUL (total time: 5 seconds)
```

```
run:
Enter result:                          57
Classification:                        II2
The minimum and maximum marks are: 50 and 59
BUILD SUCCESSFUL (total time: 5 seconds)
```

```
run:
Enter result:                          68
Classification:                        II1
The minimum and maximum marks are: 60 and 69
BUILD SUCCESSFUL (total time: 3 seconds)
```

```
run:
Enter result:                         88
Classification:                       I
The minimum and maximum marks are: 70 and 100
BUILD SUCCESSFUL (total time: 4 seconds)
```

Question 2:

```
package librarysimulation;
import java.util.*;
/**
 * outputs classification, whether it's a check in or check out,
 * the classification after the check in or check out,
 *  and the status of the book.
 * @author James Vine - 100022010
 */
public class LibrarySimulation
{


    public static void main(String[] args)
    {
        String[] events = runSimulation( generateBookStock(),20);
        for(int i = 0; i < events.length; i ++)
        {
            System.out.println(events[i]);
        }
    }

/**
* A method to generate a collection of LibraryBook objects to use as
* test data in your simulation
* @return      an array of LibraryBook objects
*
*/
public static LibraryBook [] generateBookStock()
```

```
{
   String [] authorsList = {"Lewis and Loftus",  "Mitrani",
                   "Goodrich","Lippman", "Gross", "Baase",
                   "Maclane", "Dahlquist", "Stimson", "Knuth",
                   "Hahn", "Cormen and Leiserson",
                   "Menzes", "Garey and Johnson"};
   String [] titlesList = {"Java Software Solutions", "Simulation",
                   "Data Structures", "C++ Primer", "Graph Theory",
                   "Computer Algorithms", "Algebra",
                   "Numerical Methods", "Cryptography",
                   "Semi-Numerical Algorithms",
                   "Essential MATLAB", "Introduction to Algorithms",
                   "Handbook of Applied Cryptography",
                   "Computers and Intractability"};
   int [] pagesList = {832, 185, 695, 614, 586, 685, 590, 573, 475,
                       685, 301, 1175, 820, 338};
   int n = authorsList.length;
   LibraryBook [] bookStock = new LibraryBook[n];
       for(int i = 0; i < n; i++)
       {
           bookStock[i] = new LibraryBook(authorsList[i],
                                          titlesList[i], pagesList[i]);
       }
//       set library classification for half of the LIbraryBooks
       for(int i = 0; i < n; i=i+2)
       {
           bookStock[i].setClassification("QA" + (99 - i));
       }
//        set approx. two thirds of LIbraryBooks in test data as
//        lending runList
       for(int i = 0; i < 2*n/3; i++)
           bookStock[i].setAsForLending();
//        set approx. one third of LibraryBooks in test data as
//        reference-only
       for(int i = 2*n/3; i < n; i++)
           bookStock[i].setAsReferenceOnly();
       return bookStock;
}

/**
* A method to derive the type of Event depending on the book's
* different variables.
* @return      the type of book classification
*
*/
public static String deriveEvent(LibraryBook book, boolean checkIn,
                                 String classification)
{
    System.out.println(book.classification + " " + checkIn
                       + " " + classification + " " + book.status);
    if(book.classification==null)
    {
        book.setClassification(classification);
        return "BOOK IS CLASSIFIED";
    }
```

```
    else if(book.status==LibraryBook.BookStatus.REFERENCE_ONLY)
    {
        return "REFERENCE ONLY BOOK";
    }
    else if(book.status==LibraryBook.BookStatus.AVAILABLE_FOR_LENDING
            && !checkIn)
    {
        return "BOOK IS LOANED OUT";
    }
    else if(book.status==LibraryBook.BookStatus.ON_LOAN &&
            !checkIn)
    {
        if(book.reserveBook())
        {
          return "RESERVATION PLACED FOR ON LOAN BOOK";
        }
        else
        {
            return "BOOK IS ON LOAN BUT CANNOT BE RESERVED";
        }
    }
    else if(checkIn)
    {
//        coursework sheet didn't mention
//        what happens if book is available and returned?
        return "BOOK IS RETURNED";
    }
    throw new Error("Unable to determine event type");
}
/**
* @param bookStock      the stock of LibraryBooks in the library
* @param numberOfEvents the size of the events table to be generated
*                       table of events generated during the simulation
* @return
*/
public static String[] runSimulation(LibraryBook[] bookStock,
                                                    int numberOfEvents)
{
    String[] runList = new String[numberOfEvents];
    for(int i = 0; i < numberOfEvents; i ++)
    {
       Random eventChoice = new Random();
//       0 = check in, 1 = check out
       int randomNum = eventChoice.nextInt(2);
       Random bookChoice = new Random();
       int randomBookNum = bookChoice.nextInt(bookStock.length-1);
       String eventString = deriveEvent(bookStock[randomBookNum],
                                        randomNum == 0, "QA"
                                        + randomBookNum);
       runList[i] = Integer.toString(i) + " "
                    + Integer.toString(randomBookNum) + " "
                    + bookStock[randomBookNum].classification + " "
                    + eventString;
       System.out.println(bookStock[randomBookNum]);
    }
    return runList;
}
}
```

Question 2: LibraryBook class.

```
package librarysimulation;
/**
 * LibraryBook class for LibrarySimulation
 * @author James Vine - 100022010
 */
public class LibraryBook
{
    private String author;
    private String title;
    private int pages;
    String classification;
    int borrowed;
    BookStatus status;
    int reservations;

    public enum BookStatus
    {
        REFERENCE_ONLY, ON_LOAN, AVAILABLE_FOR_LENDING
    }

     /**
     * Constructor with arguments for LibraryBook's author(s), title
     * and number of pages
     * @param bookAuthor    the names of author(s) of this LibraryBook
     * @param bookTitle     the title of this LibraryBook
     * @param bookPages     the number of pages of this LibraryBook
     */
     public LibraryBook(String bookAuthor, String bookTitle,
                                           int bookPages)
     {
         author = bookAuthor;
         title = bookTitle;
         pages = bookPages;

//       variables below already set to these by default
         classification = null;
         borrowed = 0;
         status = BookStatus.REFERENCE_ONLY;
         reservations = 0;
     }

    String getAuthor()
    {
        return author;
    }

    String getTitle()
    {
        return title;
    }

    int getPages()
    {
        return pages;
    }
```

```
String getClassification()
{
    return classification;
}
 /**
 * A  method to reset Library classification of this LibraryBook
 * @param bookClass
 * @return the proposed new classification
                        true,   if the proposed new classification
                                has at least 3 characters to which
                                the Library classification
                                is reset.
                        false,  otherwise.
 */
public boolean setClassification(String bookClass)
{
    classification = bookClass;
    if (classification.length() >3)
    {
        return false;
    }
    return true;
}

public boolean isAvailable(BookStatus status)
{
    if(status == BookStatus.REFERENCE_ONLY)
    {
        return false;
    }
    else if(status == BookStatus.ON_LOAN)
    {
        return false;
    }
    return true;
}

void setAsReferenceOnly()
{
    status = BookStatus.REFERENCE_ONLY;
}

void setAsForLending()
{
    status = BookStatus.AVAILABLE_FOR_LENDING;
}

BookStatus getStatus()
{
    return status;
}
```

```
void setReservations(int numReservations)
{
    if (numReservations > 3)
    {
        throw new Error("Number of reservations greater than 3");
    }
    reservations = numReservations;
}

int getReservations()
{
    return reservations;
}

/**
* If possible, reserves this LibraryBook.
* This is only possible if this LibraryBook is currently on loan
* and less than 3 reservations have been placed since this
* went on loan.
* @return      true,   if new reservation has been made for this.
*              false,  otherwise
*/
public boolean reserveBook()
{
    if(status == BookStatus.ON_LOAN && reservations < 3)
    {
        reservations ++;
        return true;
    }
    return false;
}

public boolean borrowBook()
{
    if(status == BookStatus.AVAILABLE_FOR_LENDING)
    {
        status = BookStatus.ON_LOAN;
        return true;
    }
    return false;
}

public boolean returnBook()
{
    if(status == BookStatus.ON_LOAN)
    {
        status = BookStatus.AVAILABLE_FOR_LENDING;
        return true;
    }
    return false;
}
```

```
    @Override
    public String toString()
    {
        String returnVal = "Title: " + title + "\n";
        returnVal = returnVal + "Author: " + author + "\n";
        returnVal = returnVal + "Pages: " + pages + "\n";
        returnVal = returnVal + "Classification: " + classification
                                                    + "\n";
        returnVal = returnVal + "Book Status: " + status + "\n";
        return returnVal;
    }
}
```

```
0 4 QA95 BOOK IS LOANED OUT
1 2 QA97 BOOK IS LOANED OUT
2 4 QA95 BOOK IS LOANED OUT
3 7 QA7 BOOK IS CLASSIFIED
4 3 QA3 BOOK IS CLASSIFIED
5 0 QA99 BOOK IS LOANED OUT
6 7 QA7 BOOK IS LOANED OUT
7 5 QA5 BOOK IS CLASSIFIED
8 3 QA3 BOOK IS LOANED OUT
9 4 QA95 BOOK IS LOANED OUT
10 4 QA95 BOOK IS LOANED OUT
11 5 QA5 BOOK IS RETURNED
12 8 QA91 BOOK IS LOANED OUT
13 10 QA89 REFERENCE ONLY BOOK
14 4 QA95 BOOK IS RETURNED
15 6 QA93 BOOK IS LOANED OUT
16 4 QA95 BOOK IS LOANED OUT
17 1 QA1 BOOK IS CLASSIFIED
18 0 QA99 BOOK IS LOANED OUT
19 10 QA89 REFERENCE ONLY BOOK
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
QA95 false QA4 AVAILABLE_FOR_LENDING
Title: Graph Theory
Author: Gross
Pages: 586
Classification: QA95
Book Status: AVAILABLE_FOR_LENDING

QA97 false QA2 AVAILABLE_FOR_LENDING
Title: Data Structures
Author: Goodrich
Pages: 695
Classification: QA97
Book Status: AVAILABLE_FOR_LENDING

QA95 false QA4 AVAILABLE_FOR_LENDING
Title: Graph Theory
Author: Gross
Pages: 586
Classification: QA95
Book Status: AVAILABLE_FOR_LENDING

null false QA7 AVAILABLE_FOR_LENDING
Title: Numerical Methods
Author: Dahlquist
Pages: 573
Classification: QA7
Book Status: AVAILABLE_FOR_LENDING
```