COMPUTER SCIENCE
HIGHER LEVEL
PAPER 2

MOCK EXAM

1 hour 20 minutes

INSTRUCTIONS TO CANDIDATES

- Do not turn over this examination paper until instructed to do so.
- **Answer all questions.**

Option D - Java and Object Oriented Programming

System Overview

A school maintains a CONTACTS database of **contact information**. This is used to send e-mails and/or make phone calls to **students, teachers and parents.**

By using this database, the school can do all of the following more efficiently:
- send emails to large groups, like "all parents" or "all grade 12 students" or "Upper School teachers"
- quickly look up a phone number to contact parents in an emergency like an injured child
- teachers can contact students in their classes
- students and their parents can contact specific teachers

The database has a user-friendly **GUI** interface, allowing administrative staff to add and edit data, as well as allowing all users (parents, teachers and students) to look up contact information.

Objects

There are several objects in the CONTACTS database:

Object	***Description***
Person	A base class for any person : **name , phone , email address, ID (a unique integer)**
Student	Any student enrolled at school : **[Person]** + **grade , homeroom teacher**
Parent	Any parent of a student : **[Person]** + **business phone, business email**
Teacher	A teacher working at school : **[Person]** + **class room phone, homeroom grade**
List	A LinkedList of Objects - could be a list of Students or Parents or Teachers or any mixture of these Objects

Phone Numbers

The **phone** field contains the home phone number, as a complete phone number like "603-1234".
The **business phone** is also a complete phone number, e.g. "987-6543"
The **classroom phone** is a 3 digit internal extension, like "276" - this only works inside the school.

Code

Part of the code for each Class is shown below. The code may be incomplete in places where the information is not needed for this exam, or where the code must be completed as part of the exam.

```
public class Person {
    private String name = "";
    private String phone = "";
    private String email = "";
    private int ID = 0;

    public Person()
    { }

    public Person(String n, String p, String e, int i)
    {
        setName(n);
        setPhone(p);
        setEmail(e);
        setID(i);
    }

    public void setName(String n) { name = n; }
    public void setPhone(String p) { phone = p; }
    public void setEmail(String e)
    {  if(checkEmail(e) == true)
       {  email = e; }
       else
       {  email = ""; }
    }
    public void setID(int i) { ID = i; }

    public String getName() { return name; }
    public String getPhone() { return phone; }
    public String getEmail() { return email; }
    public int getID() { return ID; }

    public boolean checkEmail(String e)
    {
        if( e.indexOf('@') > 0 )          // find @ sign
        {  return true; }
        else
        {  return false; }
    }
}
```

```
public class Student extends Person
{
    private int grade = 0;
    private String homeroom = "";

    public Student(String n, String e, String p, int i, int g, String h)
    {
        setName(n);
        setEmail(e);
        setPhone(p);
        setID(i);
        setGrade(g);
        setHomeroom(h);
    }

    public void setGrade(int g)
    { if(g > 0 && g < 13)
      {  grade = g; }
    }
    public void setHomeroom(String h) { homeroom = h; }

    public int getGrade() { return grade; }
    public String getHomeroom() { return homeroom; }

}
```

```
public class Teacher extends Person
{
    private int grade = 0;
    private String homeroomPhone = "";

    public Teacher(String n, String e, String p, int i, int g, String hp)
    {
        setName(n);
        setEmail(e);
        setPhone(p);
        setID(i);
        setGrade(g);
        setHomeroomPhone(hp);
    }

    public void setGrade(int g)
    { if(g > 0 && g < 13)
      {  grade = g; }
    }
    public void setHomeroomPhone(String hp) { homeroomPhone = hp; }

    public int getGrade() { return grade; }
    public String getHomeroom() { return homeroomPhone; }
}
```

#1

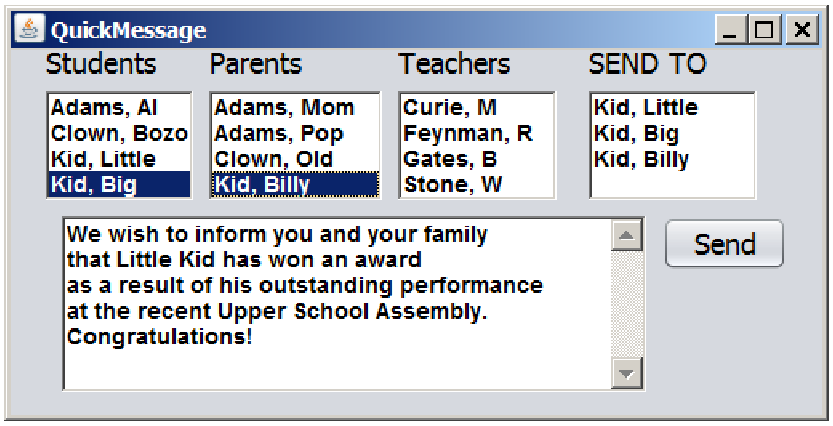
(a) Explain what a **constructor method** is, including a specific example chosen from the code on the previous pages, as well as a brief explanation of how and when the **constructor** functions. *[3 marks]*

(b) Explain what **data validation** code is, and state a specific example that occurs in the code on the previous pages. *[2 marks]*

(c) Explain the term **encapsulation**, making specific reference to the sample code on the previous pages. Include an explanation of the connection between **private** data members and **set** methods. *[4 marks]*

(d) Construct the **Parent** class, written in Java, including all appropriate **get** methods, **set** methods, and properties (variables). You do NOT need to write any **validation** code. *[6 marks]*

===



Students	Parents	Teachers	SEND TO
Adams, Al	Adams, Mom	Curie, M	Kid, Little
Clown, Bozo	Adams, Pop	Feynman, R	Kid, Big
Kid, Little	Clown, Old	Gates, B	Kid, Billy
Kid, Big	Kid, Billy	Stone, W	

```
We wish to inform you and your family
that Little Kid has won an award
as a result of his outstanding performance
at the recent Upper School Assembly.
Congratulations!
```

The diagram above shows the QuickMessage interface. It contains a list with the names of all the Students in the school, a list with all the Parents, and a list with all the Teachers. The user can click on Students, Parents and/or Teachers, thus copying the names (and the corresponding Objects) into the SEND TO box. When the user clicks the [Send] button, an email is sent automatically to **all** the people in the SEND TO box.

Here is some of the code in the QuickMessage class (this code is incomplete):

```
public QuickMessage()
{
    students = load("students");   // loads Student objects into students LinkedList
    parents = load("parents");     // loads Parent objects into parents LinkedList
    teachers = load("teachers");   // loads Teacher objects into teachers LinkedList
    display(students);
    display(parents);              // displays the LinkedList into GUI List boxes
    display(teachers);
}

// == The "load" and "display" methods exist, but are not shown here ==//
```

When the user clicks the [Send] button, the following method sends emails:

```
//== Before running sendEmails, the LinkedList sendTo must contain ==/
//== numerous Person, Student, Parent and/or Teacher Objects       ==/
//==  and message must contain text to be sent in the email        ==/

public void sendEmails(LinkedList sendTo, String message)
{
    for(int c=0; c < sendTo.size(); c = c+1)
    {
        Person p = (Person)sendTo.get(c);
        sendMail( p , message);
    }
}

//== The sendMail method exists, but is not shown here ==//
```

#2

(a) Outline **two major differences** between a **LinkedList** and an **Array** in Java. *[3 marks]*

(b) Construct a method that searches through the **students LinkedList**,
finds a specific name, and outputs the corresponding email address.
You must write your solution in Java. *[5 marks]*

(c) Construct a method that searches through the **students LinkedList** for all
the students who are **in grade 12**. Whenever it finds a grade 12 student,
it **adds** that object to the **sendTo** LinkedList. Write your solution in Java. *[7 marks]*

==

#3

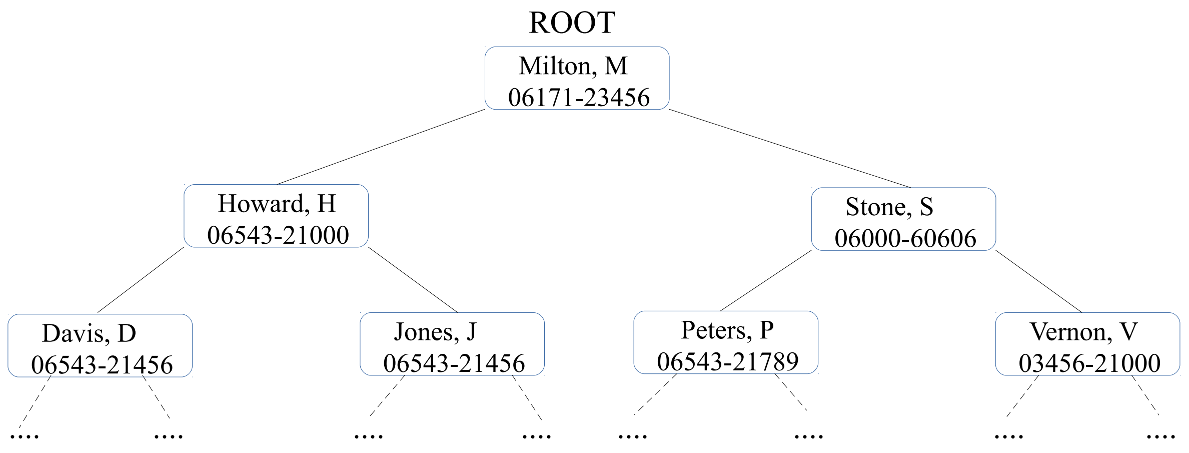
(a) Explain what the term **inheritance** means in Object Oriented Programming,
including a specific example where inheritance is used in the CONTACTS system. *[3 marks]*

(b) Outline how the use of **inheritance** makes it easier when programmers are
expanding an application to add more features. *[2 marks]*

(c) The **Person** class contains error-prevention code in the **checkEmail** method.
This prevents accidental errors when entering an email address.
This method is actually too simple. It should also check more rules.
Every email address must:
- contain exactly one '@' sign
- after the '@' sign, there must be exactly one period '.'
- there must be at least 2 characters between the '@' and the '.' period.
- there must be at least 3 characters after the '.' period
- there must be at least 3 characters before the '@' sign

Using standard Java **String** methods, construct an improved **checkEmail** method that
checks all the rules stated above. *[10 marks]*

A **telephone tree** is used during emergencies, for example when school is cancelled for a Snow Day. In a telephone tree, the first (root) person calls 2 people. Each of those calls 2 people, each of those calls 2 people, etc. The beginning of the **Teachers' tree** is shown below:



This tree contains all the **Teachers** in the school, so it goes on for quite a while.
Each box represents a **Node** object, created from the following class:

```
public class Node extends Teacher
{
     Node  leftChild = null;
     Node  rightChild = null;
}
```

(a) Assume the school has 120 teachers. Assume that the **binary search tree** for the teachers is **balanced**. State the maximum number of iterations (steps) required to find any teacher inside the tree. *[2 marks]*

(b) State the type of **traversal** required to print all the names in **alphabetical order**. *[1 mark]*

(c) State what would be displayed by the following Java command:

```
output(ROOT.leftChild.rightChild.getName());
```
[1 mark]

(d) Outline one similarity and one difference between a **binary tree** and a **linked-list.** *[2 marks]*

(e) The method below should be able to find any teacher's name inside this tree, and return the corresponding phone number. For example:

```
findName( "Peters, P" , ROOT);     // start searching at the ROOT
```

would return "06543-21789" .

Copy the code below and fill in the blanks to make it work correctly, assuming the tree already exists and ROOT points at the top node. *[6 marks]*

```
public String findName(String target, Node ROOT) extends Teacher
{
   Node temp = ROOT;

   while( temp != __________ )
   {
       if(  _______________________.equals(target) )
       {
           return __________________ ;
       }
       else if( temp.getName().compareTo( target ) > 0 )
       {
           temp = ____________________________ ;
       }
       else
       {
           temp = ____________________________ ;
       }
   }

   return __________________;
}
```

(f) Construct a **recursive** method that **counts** all the Nodes in the Teachers' tree. If it works correctly, and all the teachers are listed in the tree, it would return **120**. If some teachers are missing, it would return a smaller number. *[8 marks]*