Université d'Ottawa Faculté de génie École d'ingénierie et de technologie de l'information		University of Ottawa Faculty of Engineering School of Information Technology and Engineering

ITI 1120 Fall 2009
Introduction to Computing I
Final Examination - Solution

Length of Examination: 3 hours **December, 19th 2009**

Professor: Gilbert Arbez and Mohamad Eid

IDENTIFICATION (Please complete in ink)

Family Name ____

Given Names ____

Student Number___________________ Signature_________________________

Instructions: Please read carefully!

1. This is a closed-book test. **No books, papers, calculators or other electronic devices are permitted.**
2. Answer all questions on the question sheet in the area provided. Les réponses en français sont acceptées. Questions answered in pencil will not be re-graded even if there is a marking error.
3. Algorithms are to be described using the format from the lectures and the notes.
4. You can use the back of the question sheet pages, or page 5 of the annex, for calculations and other work.
5. If a question seems unclear, state a hypothesis, and proceed to answer the question.
6. No student will be allowed to leave the room in the last 15 minutes of the exam.
7. When time is called at the end of the exam, **turn your paper over immediately. Stay in your place and remain quit until a proctor have collected all papers** and given clearance to leave.

For use of grader:

1 (10)	2 (10)	3 (15)	4 (15)	5 (10)	6 (15)	7 (25)	Tot (100)

Question 1A) (4 marks)

In this question, use only the following (**algorithm format**) Boolean expressions.

- comparison operators: <, >, =, ≤, ≥, and ≠
- Boolean connectors: NOT, AND, OR
- arithmetic operators: +, -, *, /, and MOD (modulo)
- variable names and constants.

Use parentheses where necessary. Do **not** use Java syntax!

Building Classifications are used to define structural, fire and life safety requirements. The variable *type* defines the basic type of a building and can be set to one of the following constant values: BANK, BARN, CHEMICAL_STORAGE, CHURCH, CLINIC, CONFERENCE, DAYCARE, DORMITORIES, ELEMENTARY_SCHOOL, FIRE_STATION, GARAGE, HEATH_CARE, HIGH_SCHOOL, HOSPITAL, HOTEL, LECTURE_HALL, LIBRARY, MUSEUM, MEETING, NURSING_HOME, OFFICE, PAINTING, PARTS_ASSEMBLY, PETROLEUM_STORAGE, POLICE_STATION, POST_OFFICE, RESTAURANT, RETAIL_STORE, ROOMING_HOUSE, STADIUM, STORAGE_SPACE, UNIVERSITY_HALL, WAREHOUSE, WOODWORKING.

The classification of a building is dependent on its *type* as shown in the following table:

Building Classification	Classification Descriptin	Building types
GROUPA	Assembly Occupancies	CONFERENCE, CHURCH, LECTURE_HALL, LIBRARY, MEETING, MUSEUM, RESTAURANT, STADIUM, UNIVERSITY_HALL
GROUPB	Business Occupancies	BANK, CLINIC, FIRE_STATION , OFFICE, POLICE_STATION, POST_OFFICE
GROUPE	Educational	DAYCARE, ELEMENTARY_SCHOOL, HIGH_SCHOOL
GROUPF	Factory and Industrial	PAINTING, PARTS_ASSEMBLY, WOODWORKING
GROUPH	Hazardous Occupancies	CHEMICAL_STORAGE, PETROLEUM_STORAGE
GROUPI	Institutional Occupancies	HEATH_CARE, HOSPITAL, NURSING_HOME
GROUPM	Mercantile	RETAIL_STORE
GROUPR	Residential Occupancies	DORMITORIES, HOTEL, ROOMING_HOUSE
GROUPS	Storage	STORAGE_SPACE, WAREHOUSE
GROUPU	Utility Occupancies	GARAGE, BARN

Provide an **algorithm** Boolean expression that is TRUE when the building has a classification of GROUPF, or a classification of GROUPH, or a classification of GROUPS, or a classification of GROUPU.

(type=PAINTING OR type=PARTS_ASSEMBLY OR type=WOODWORKING) OR
(type=CHEMICAL_STORAGE OR type=PETROLEUM_STORAGE) OR
(type=STORAGE_SPACE OR type=WAREHOUSE) OR
(type=GARAGE OR type=BARN)

Question 1B) (6 marks)

Consider the following two classes:

```
class Airplane
{
  private int ID;
  private double X;
  private double Y;
  private static int numberOfPlanes;
  public double computeDistance()
  {
   …
  }

  public static void getNumberOfPlanes()
  {
   …
  }
}
```

```
class Zone
{
  private String ID;
  public static double limit;
  private Airplane[] planes =
                    new Airplane[10];

  public Zone () { }
  public Zone (double l)
  {
   …
  }
  public void checkDistances()
  {
    …
  }
}
```

Suppose that the following instructions are used in the `main()` method in a class `Test`. Each case should be considered independently – as it if were in its own `main()` method. Circle the letter of ALL cases which CAUSE a compilation error and provide a short description of what causes the error.

a) `Airplane.numberOfPlanes = 5+2;`

// numberOfPlanes is a private variable

b) `Zone z = new Zone (300, "Zone1");`

// There is no constructor defined with 2 arguments

c)
```
Zone ref1 = new Zone();
ref1.limit = 1.25;
```

// accepted

d) `Airplane.getNumberOfPlanes ();`

// accepted

e)
```
Zone zone1;
Airplane plane1 = new Airplane();
zone1.planes[0] = plane1;
```

// The array of Airplane objects is private and can't be accessed from outside the class

f)
```
Zone instance = new Zone(800);
Zone.checkDistances();
```

// can't call an instance method using the class name

Question 2: (10 marks)

On the following page, show how the contents of the working and global memory are changed by the execution of the main method model. Show all assignments to variables/parameters. Use the following as a guideline to show successive assignments.

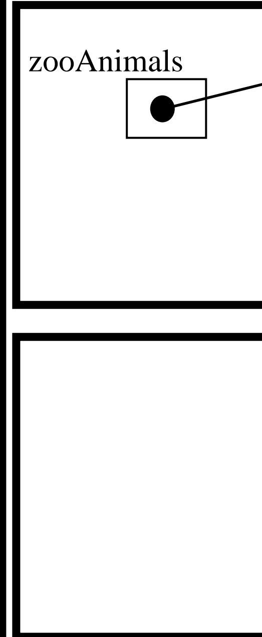
Variable	~~2~~, ~~6~~, ~~4~~, 10

Also show what the program displays in on the console window.

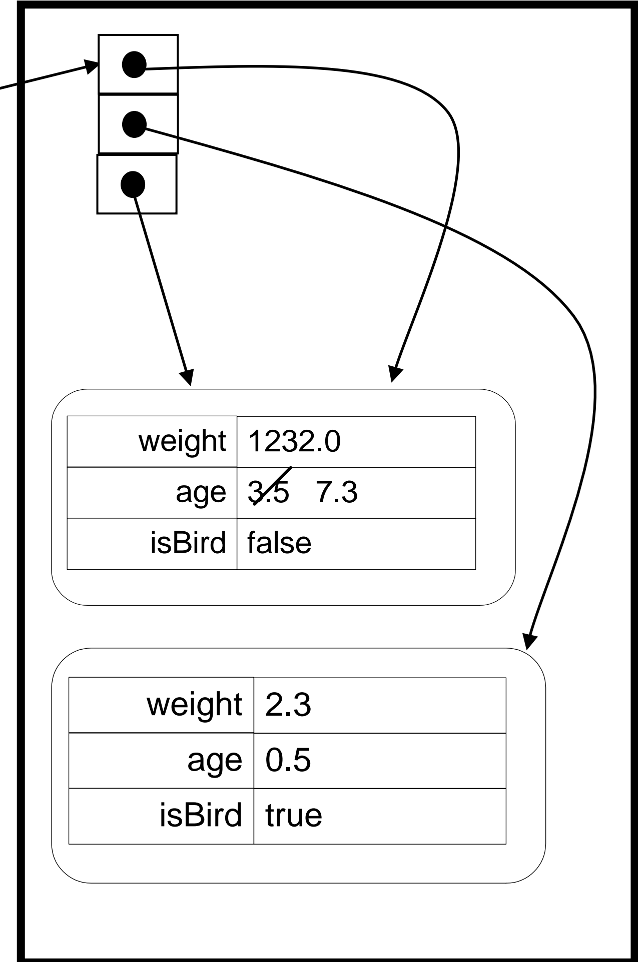
Program Memory

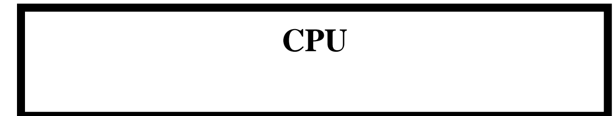
```
class Question2
{
  public static void main(String args[])
  {
    Animal[] zooAnimals;
    zooAnimals = new Animal[3];
    zooAnimals[0] = new Animal(1232.0, 3.5, false);
    zooAnimals[1] = new Animal(2.3, 0.5, true);
    zooAnimals[2] = zooAnimals[0];
    zooAnimals[2].age = 7.3;
    System.out.println("Animal zero's age is "
                                    +zooAnimals[0].age);
    System.out.println("Animal one's age is "+
                                     zooAnimals[1].age);
    System.out.println("Animal two's age is "+
                                     zooAnimals[2].age);
  }
}

class Animal
{
  public double weight;
  public double age;
  public boolean isBird;

  public Animal(double weight, double years, boolean bird)
  {
    this.weight = weight;
    age = years;
    isBird = bird;
  }
```

Working memory



Global Memory





Console Display:

```
Animal zero's age is 7.3
Animal one's age is 0.5
Animal two's age is 7.3
```

Question 3: (15 marks)

The algorithm in the annex prints a pyramid with the following format:

```
                               1
                           1   2   1
                       1   2   4   2   1
                   1   2   4   8   4   2   1
               1   2   4   8  16   8   4   2   1
           1   2   4   8  16  32  16   8   4   2   1
       1   2   4   8  16  32  64  32  16   8   4   2   1
   1   2   4   8  16  32  64 128  64  32  16   8   4   2   1
```

The algorithm takes one parameter, n, that gives the number of lines to print in the pyramid. Its value should be less than or equal to 10. Translate the algorithm in the annex to a Java **method**.
The Java method `Math.pow(double a, double b)` can be used to find a^b. The following new algorithm model is called:

PrintNum(num) prints the number within a field of 4 characters, that is, precedes the number with the appropriate number of spaces to print a 4 character string. Translate a call to this algorithm to a call to the Java method *System.out.printf("%4d",num)* where *num* corresponds to the argument of *PrintNum*.

Question 3 – continued

```
    public static void pyramid(int n)
    {
      // Declare variables
      int line;
      int i;
      int two2i;

      if(n>0  &&  n<11)
      {
        line=1;
        while(line <= n)
        {
          // Prints blank spaces at the start of the line
          System.out.print(" ");
          i=0;
          while(i<n-line)
          {
            System.out.print("    ");
            i=i+1;
          }
          // Print first half of the numbers
          i=0;
          while(i<line)
          {
            two2i = (int) Math.pow(2,i);
            System.out.printf("%4d", two2i);
            i=i+1;
          }
          // Print second half of the numbers
          i=line-2;
          while(i >= 0)
          {
            two2i = (int) Math.pow(2,i);
            System.out.printf("%4d", two2i);
            i=i-1;
          }
          System.out.println();
          line=line+1;
        }
      }
      else
      {
        System.out.println("n is invalid");
      }
    }
```

Question 4: (15 marks)

An N X N square matrix is "**upper triangular**" when, all the elements below the top-left-to-lower-right diagonal are zeros. The following 3 X 3 matrix is an example of an upper triangular matrix.

$$\mathbf{A} = \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & 0 & x \end{bmatrix}$$

Write a Java method that will take any square matrix of integers **A** and returns true if **A** is an upper triangular matrix and false otherwise. Your method should be as efficient as possible.

```
public static boolean isUpperTriangular( int[][] a)
{

   boolean isUpperTriangular = true;
   int i, j;

   for (i = 1; i < a.length && isUpperTriangular; i = i+1)
   {
      for (j = 0; j < i && isUpperTriangular; j = j+1)
      {
            if (a[i][j] != 0)
            {
                  isUpperTriangular = false;
            }
            else
            {
                  /* do nothing */
            }
      }
   }
   return (isUpperTriangular);
}
```

Question 5: (10 marks)
Write a recursive method, *dip(n)*, that produces the following output when n=5:

```
*****
****
***
**
*
*
**
***
****
*****
```

(Note that you may use a loop to print an individual line. Up to 3 bonus marks will be assigned if you use a second recursive method for printing each individual line, that is, your solution does not use any loops.)

The header of the method should be the following:

```
public static void dip(int n)
{
  int i;  // number of the term

  dipline(n);

  if(n == 1) // Base case
  {
    // do nothing
  }
  else
  {
    dip(n-1);
  }

  dipline(n);
}

public static void dipline(int n)
{
  if(n == 0)
  {
    System.out.println();
  }
  else
  {
    System.out.print("*");
    dipline(n-1);
  }
}
```

Question 6: (15 marks)

Comparing loans with various interest rates:

Provide an algorithm model with GIVENS that receive a loan amount in dollars and a loan period in number of years. The algorithm model displays the monthly payments and the total paid for each interest

rate starting from 5% to 8%, with an increment of 1/8%. For example, for a loan amount of 10,000 dollars for five year loan period, the algorithm should display the following:

Loan Amount: 10000
Number of years: 5

Interest Rate	Monthly Payment	Total Paid
5%	188.71	11322.74
5.125%	189.28	11357.13
5.25%	189.85	11391.59
…		
7.85%	202.16	12129.97
8.0%	202.76	12165.83

The monthly payment, *m*, is computed as follows:

$$m = \frac{p * r * q}{12 * (q - 1)}$$

where

- *r* is the interest rate in decimal form (ex. r=0.05 for an interest rate of 5%)
- *p* is the principal amount of the loan, i.e. the amount of money loaned (in the example above p=10000)
- *q* is computed as $\left(1 + \frac{r}{12}\right)^n$
- *n* is the loan period in months (for example 20 years, n = 240 months)

[Do not be concerned with aligning the columns exactly. Use next two pages to answer this question]

Question 6 continued

GIVENS:

  p (principle amount of the loan)
  nYears (loan period in years)

RESULT:

  none

INTERMEDIATES:

  nMonths (loan period in months)
  r (interest rate)
  q (q factor as defined)
  m (monthly payment)

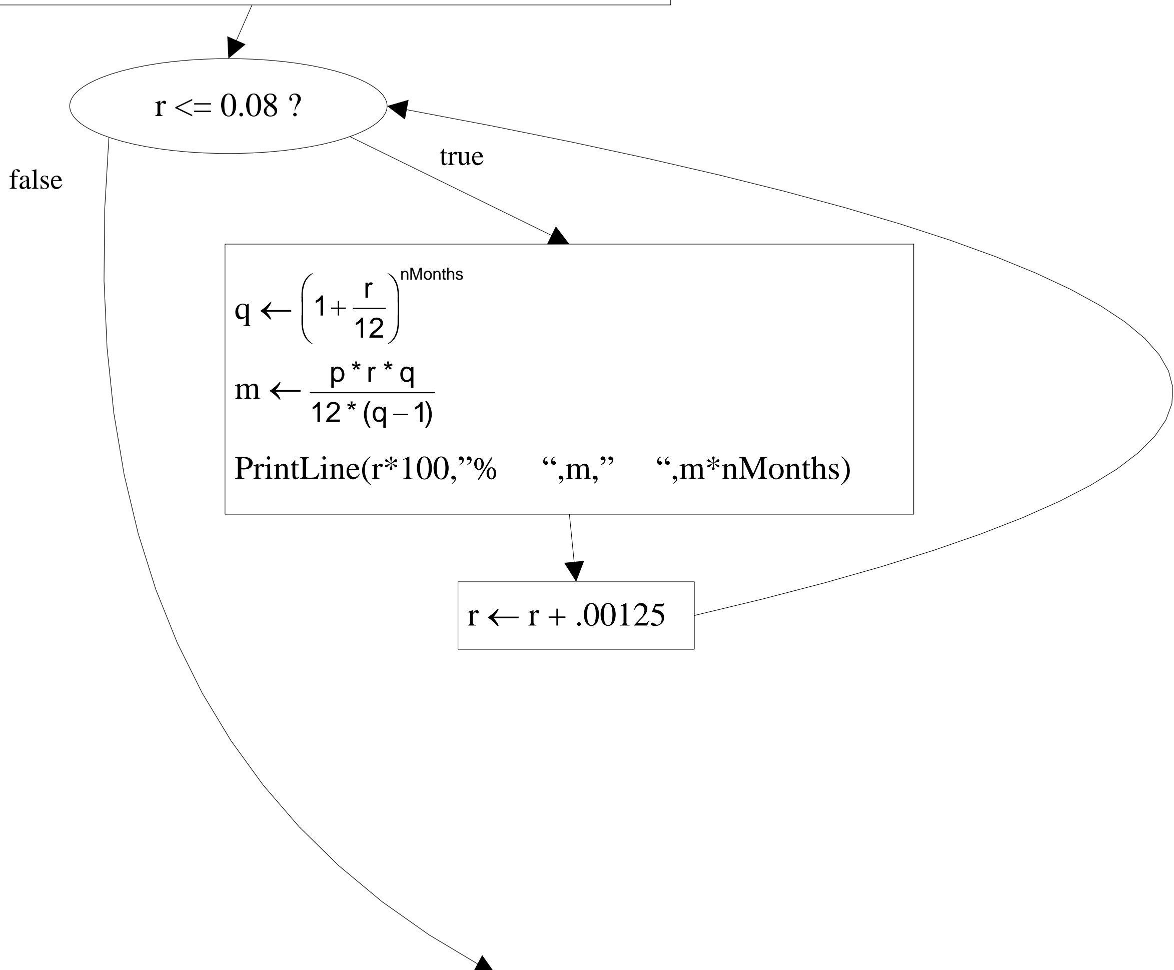
ASSUMPTIONS:

  Monthly payment, m, is computed as: $m = \frac{p*r*q}{12*(q-1)}$ **, where q=** $\left(1+\frac{r}{12}\right)^{nMonths}$

  Total payment is computed as m * nMonths

HEADER: loanTable(p, nYears)

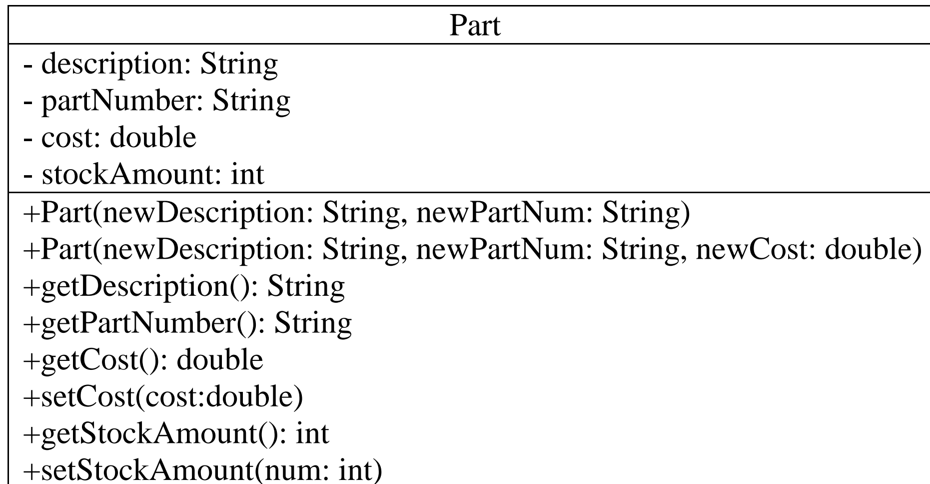
BODY:



Question 7 (25 marks)

The class `PartsInventory` shall be designed to contain a number of `Part` objects.

The class `Part` has already been implemented and is provided for your use (see annex). The class `Part` includes the attributes: part description, part number, cost, and amount in stock in the inventory. The class provides two constructors, four accessor methods, and two modifier methods. The first constructor sets the *cost* attribute to 0. Both constructors set the *stockAmount* attribute to 0. The *setCost* and *setStockAmount* methods are used to update the values of the *cost* and *stockAmount* attributes respectively. A UML class diagram for the class *Part* is as follows:



On the next two pages, you shall complete the methods for the `PartsInventory` class. Your `PartsInventory` class should provide five public methods (provide a constructor if required) that would permit the following class `TestPartsInventory` to execute:

```
class TestPartsInventory
{
   public static void main (String[] args)
   {
      PartsInventory inventory = new PartsInventory( );
      inventory.addPart( "Dakota Starter, 3.9L, 5.2L, 5.9L", "F5000-137756");
      inventory.addPart( "Dakota Water Pump, 3.9L", "AWAW7160");
      inventory.addPart ( "Dakota Alternator 136 Amp", “MG1338”, 399.95);
      inventory.addPart ( "Dakota Master Brake Cylindre", "CE131.67025", 124.94);
      inventory.addPart ( "Dakota Roll Up Tonneau Cover”, "A7414079");
      inventory.updatePartCost( "Dakota Starter, 3.9L, 5.2L, 5.9L",169.95);
      inventory.updatePartCost( "MG1338",78.95);
      inventory.updatePartStockAmount ( "Invalid",5);  // Note the invalid string
      inventory.updatePartStockAmount ( "AWAW7160",5);
      inventory.updatePartStockAmount ("F5000-137756",3);
      inventory.updatePartStockAmount ("A7414079",1);
      inventory.printInventory();
   }
}
```

Executing `main()` would result in the following being printed on the screen :

```
Part Invalid does not exist.
Inventory:
   Dakota Starter, 3.9L, 5.2L, 5.9L; F5000-137756; 169.95; 3
   Dakota Water Pump, 3.9L; AWAW7160; 0.0; 5
   Dakota Alternator 136 Amp; MG1338; 78.95; 0
   Dakota Master Brake Cylindre; CE131.67025; 124.94; 0
```

```
 Dakota Roll Up Tonneau Cover; A7414079; 0.0; 1
```

Question 7:(continued)

Complete the Class `PartsInventory` on this page and the next 2 pages.

```
class PartsInventory
{
  // ATTRIBUTE DECLARATION: (3 marks)

  Part [] partList;

  // CONSTRUCTOR (if required): (2 marks)

  // Not required

 // MODIFIER METHOD addPart: (8 marks)
 // Method parameters: a description (String) and a part number (String).
 // it is possible that the method will be called with an optional
 // parameter cost (double) that sets the cost attribute of the added part.
 // The method adds a Part object to PartsInventory object.

  public void addPart(String newDescription, String newPartNum)
  {
    Part newPart;
    Part [] newPartList;
    int i;

    newPart = new Part(newDescription, newPartNum);

    if(partList == null)
    {
       partList = new Part[1];  // First creation of list
       partList[0] = newPart;
    }
    else
    {
       newPartList = new Part[partList.length+1];  // Creates new list
       for(i=0 ; i<partList.length ; i=i+1)
       {
          newPartList[i] = partList[i];
       }
       newPartList[newPartList.length-1] = newPart;
       partList = newPartList;
    }
  }

  public void addPart(String newDescription, String newPartNum, double newCost)
  {
      addPart(newDescription, newPartNum);
      updatePartCost(newPartNum,newCost);
  }
```

```
 // METHOD printInventory: (4 marks)
 // Method parameters: (none)
 // Returns: (none)
 // This method prints a list of all parts in the inventory (including all
 // information about the parts).See the TestPartsInventory sample output on
 // Page 12 for the exact format of the output.

  public void printInventory()
  {
    int i;
    System.out.println("Inventory:");
    for(i=0 ; i< partList.length ; i=i+1)
    {
       System.out.println("   "+partList[i].getDescription()+"; " +
                          partList[i].getPartNumber() + "; " +
partList[i].getCost()
                          + "; "+partList[i].getStockAmount());
    }
  }

  // METHOD updatePartCost: (4 marks)
  // Updates the cost attribute of the first Part with the specified description
  // or part number. If a part cannot be found to be updated, an error message   //
is printed (see page 12 for the exact format of the message).
  //NOTE: Recall that two strings referenced by s1 and s2 can only be compared
  // using s1.equals(s2).
  // Method parameters:
  //      descOrPartNum:a String which can match to either the description or
  //                    partNumber attributes of the Part to update
  //      newCost: and a value for the cost attribute.

   public void updatePartCost(String descOrPartNum, double newCost)
  {
    int i;
    boolean foundPart = false;

    i=0;
    while((i< partList.length) && !foundPart)
    {
       if(descOrPartNum.equals(partList[i].getDescription()) ||
          descOrPartNum.equals(partList[i].getPartNumber()) )
       {
          partList[i].setCost(newCost);
          foundPart = true;
       }
       else {/* do nothing */ }
       i = i + 1;
    }
    if(!foundPart) System.out.println("Part "+descOrPartNum+" does not exist.");
  }
```

```
  // METHOD updatePartStockAmount: (4 marks)
  // Updates the stockAmount attribute of the first Part with the specified
  // description or part number. . If a part cannot be found to be updated, an
  // error message is printed (see page 12 for the exact format of the message).
  //NOTE: Recall that two strings referenced by s1 and s2 can only be compared
  // using s1.equals(s2).
  // Method parameters:
  //      descOrPartNum:a String which can match to either the description or
  //                    partNumber attributes of the Part to update

//     newStockAmount: a value for the stockAmount attribute.
  public void updatePartStockAmount(String descOrPartNum, int newStockAmount)
  {
    int i;
    boolean foundPart = false;

    i=0;
    while((i< partList.length) && !foundPart)
    {
       if(descOrPartNum.equals(partList[i].getDescription()) ||
          descOrPartNum.equals(partList[i].getPartNumber()) )
       {
          partList[i].setStockAmount(newStockAmount);
          foundPart = true;
       }
       else {/* do nothing */ }
       i = i + 1;
    }
    if(!foundPart) System.out.println("Part "+descOrPartNum+" does not exist.");
  }

} // End of class PartsInventory
```