

# CS1020: Data Structures and Algorithms

## Tutorial 1 – Java and Simple OOP

(27<sup>th</sup> January 2012)

### Suggested Solutions

#### 1. [Basic Concepts and Coding]

- a) Match the terms in the left column to their corresponding definitions/functionalities in the right column:

a. Class
b. An object
c. Identifier
d. Variable
e. Method
f. Parameter
g. Constructor

1. Create a new object instance and initialize the instance variables.
2. A name given to a package, class, interface, method, or variable
3. Store data in a program.
4. User defined data type.
5. A value that is passed to the method.
6. Function, procedure.
7. Instance of a class.

#### **Solution:**

a-4: Class: User defined data type.

b-7: An object: Instance of a class.

c-2: Identifier: A name given to a package, class, interface, method, or variable.

d-3: Variable: Store data in a program.

e-6: Method: Function, procedure.

f-5: Parameter: A value that is passed to the method.

g-1: Constructor: Create a new object instance and initialize the instance variables.

**Note:** There are a few types of variable in Java:

- **Instance Variables (Non-Static Fields)** Technically speaking, objects store their individual states in "non-static fields", that is, fields declared without the static keyword. Non-static fields are also known as **instance variables** because their values are unique to each *instance* of a class (to each object, in other words); the `currentSpeed` of one bicycle is independent from the `currentSpeed` of another.
- **Class Variables (Static Fields)** A *class variable* is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated. A field defining the number of gears for a particular kind of bicycle could be marked as static since conceptually the same number of gears will apply to all instances. The code `static int numGears = 6;` would create such a static field. Additionally, the keyword `final` could be added to indicate that the number of gears will never change.
- **Local Variables** Similar to how an object stores its state in fields, a method will often store its temporary state in *local variable*. The syntax for declaring a local variable is similar to declaring a field (for example, `int count = 0;`). There is no special keyword designating a variable as local; that determination comes entirely from the location in which the variable is declared — which is between the opening and closing braces of a method. As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.

# CS1020: Data Structures and Algorithms

b) Write a Java method to test if a year falls in a leap year.

```
public static boolean isLeapYear(int year) {
    //Your code here
}
```

**Solution:**

```
public static boolean isLeapYear(int year) {
    return (year % 400 == 0 ||
            (year % 4 == 0 && year % 100 != 0));
}
```

Note: This method is declared to be static because it does not require an instance to invoke it.

c) Consider a date that consists of a day, month, and year, represented in integers. Write a Java method to print any given date that is advanced by one day. (You are to use the *isLeapYear* method written in part (b) and *switch* statements for comparing the months. You are not allowed to use any data structures like, array, linked-link, etc.)

```
class MyDate {
    public static void advDate(int day, int month, int year) {
        //Your code here
        System.out.println("Advanced Date: "+day+"/"+month+"/"+year);
    }
}
```

**Solution:**

```
public static void advDate(int day, int month, int year) {
    if (day < 28)
        day++;
    else {
        switch (month) {
            case 2:
                if (isLeapYear(year) && day < 29)
                    day++;
                else {
                    month++; day = 1;
                }
                break;

            case 4:
            case 6:
            case 9:
            case 11:
                if (day < 30)
                    day++;
                else {
                    month++; day = 1;
                }
                break;
        }
    }
}
```

## CS1020: Data Structures and Algorithms

```
        default:
            if (day < 31)
                day++;
            else {
                month++; day = 1;
            }
        } // switch
    }

    if (month > 12) {
        year++; month = 1;
    }

    System.out.println("Advanced Date: "+day+"/"+month+"/"+year");
}
```

- d) Improve your code in part (c) by using the Array data structure. You are to create a 2-D Integer Array (`DAY_OF_MONTH`) where its row index 0 represents a non-leap year, its row index 1 represents a leap year, and its columns represent the number of days in a month.

```
public static void advDateModified(int day, int month, int year) {
    int DAY_OF_MONTH[][] = { //Your code here };

    //Your code here

    System.out.println("Advanced Date: "+day+"/"+month+"/"+year");
}
```

### **Solution:**

```
public static void advDateModified(int day, int month, int year) {
    int DAY_OF_MONTH[][] = {{31,28,31,30,31,30,31,31,30,31,30,31},
                             {31,29,31,30,31,30,31,31,30,31,30,31}};

    // leapYearIndex = 0 (not a leap year)
    // leapYearIndex = 1 (leap year)
    int leapYearIndex = 0;
    if (isLeapYear(year))
        leapYearIndex = 1;

    if (++day > DAY_OF_MONTH[leapYearIndex][month-1]) {
        month++; day = 1;
    }

    if (month > 12) {
        year++; month = 1;
    }

    System.out.println("Advanced Date: "+day+"/"+month+"/"+year");
}
```

## CS1020: Data Structures and Algorithms

2. **[Simple OOP]** The following class represents a point in two-dimensional space. It consists of two variables namely the x- and y-coordinates. Your task is to replace all of the `/*Fill in here*/` phrases found in the *Constructors, Accessors, Mutators* and *other necessary methods* with your own code.

(Hint: You may need to use the Math class in the Java API to calculate square root: <http://docs.oracle.com/javase/7/docs/api/> (Java API) and <http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html> (Math class). Also, comparing two doubles using `'=='` is not a good choice since rounding errors might occur during the computation in which will make the result incorrect. Therefore, in this exercise, we consider two points are equal if their distance is smaller than 0.000001).

```
class Point{

    //The x and y coordinate of the point

    private double x, y;

    //Construct and initializes a point at the origin (0, 0) of
    //the coordinate space.

    public Point(){
        /*Fill in here*/
    }

    //Construct and initialize a point at the specified (x, y)
    //location in the coordinate space

    public Point(/*Fill in here*/){
        /*Fill in here*/
    }

    //Construct and initialize a point with the same location
    //as the specified Point object.

    public Point(/*Fill in here*/){
        /*Fill in here*/
    }

    //Get the x coordinate

    public double getX(){
        /*Fill in here*/
    }

    //Get the y coordinate

    public double getY(){
        /*Fill in here*/
    }

    //Set the x coordinate

    public void setX(/*Fill in here*/){
        /*Fill in here*/
    }

    //Set the y coordinate
```

## CS1020: Data Structures and Algorithms

```
public void setY(/*Fill in here*/){
    /*Fill in here*/
}

//Set the location of this point to the specified double
//coordinates(newX,newY).

public void setLocation(/*Fill in here*/){
    /*Fill in here*/
}

//Set the location of this point to the specified point.

public void setLocation(/*Fill in here*/){
    /*Fill in here*/
}

//Calculate the Euclidean distance between the current point
//and the point p

public double distance(Point p){
    /*Fill in here*/
}

//Check whether the current point and the point p refer to
//the same location

public boolean equals(Point p){
    /*Fill in here*/
}

//Calculate the Euclidean distance between 2 points p1 and p2

public static double distance(Point p1, Point p2){
    /*Fill in here*/
}

//Check whether the two point p1 p2 refer to the same location

public static boolean equals(Point p1, Point p2){
    /*Fill in here*/
}

//Print the coordinates of the point (Example: (3, 5))

public String toString(){
    /*Fill in here*/
}
}
```

# CS1020: Data Structures and Algorithms

## *Solution:*

```
class Point{

    private double x, y;

    //Default Constructor
    //Called another constructor and setting x,y = 0
    //this(0,0) means Point(0,0) because this is referring to the
    //current executing object which is a Point object.

    public Point() {
        this(0,0);
    }

    //Constructor with parameters

    public Point(double a, double b) {
        x = a;
        y = b;
    }

    //Yet another constructor that constructs a Point based on another Point

    public Point(Point p) {
        this(p.getX(), p.getY());
    }

    //Accessors and Mutators

    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }

    //The keyword "this" represents the current executing object
    //As this is an instance method, you need an instance of a Point object
    //(lets call it aPoint) to invoke this method. Hence, aPoint is the
    //current executing object, and this.x is equivalent to aPoint.x.

    public void setX(double x){
        this.x = x;
    }
    public void setY(double y){
        this.y = y;
    }

    //Mutator that sets two coordinates at once

    public void setLocation(double newX, double newY) {
        x = newX;
        y = newY;
    }

    //Overloading Method
    //Yet another mutator that sets the corordinates of the current
    //Point based on another Point

    public void setLocation(Point p) {
        x = p.getX();
        y = p.getY();
    }

    //Calculate the distance between two Points
    //Instance Method: you need an instance of Point to invoke this method
```

# CS1020: Data Structures and Algorithms

```
public double distance(Point p) {
    return Math.sqrt(Math.pow(x - p.getX(),2) + (Math.pow(y - p.getY(),2)));
}

//Determine if two Points lie on the same spot

public boolean equals(Point p) {
    double epsilon = 0.000001;
    return distance(p) < epsilon;
}

//Overloading Method
//Static Method: You do not need to have an instance in order
//to invoke static method

public static double distance(Point p1, Point p2) {
    return Math.sqrt(Math.pow(p1.getX() - p2.getX(),2) + (Math.pow(p1.getY()
    - p2.getY(),2)));
}

//Overloading Method

public static boolean equals(Point p1, Point p2) {
    double epsilon = 0.000001;
    return distance(p1,p2) < epsilon;
}

//Print the coordinates of the point. Example: (3, 5)

public String toString() {
    return "(" + x + ", " + y + ")";
}
}
```

## CS1020: Data Structures and Algorithms

3. **[Instantiation]** The following questions are related to Question 2. After you have written your Point class, now it is time to use it.

Assuming that the following *modify* method has been added into the Point class:

```
//This instance method modifies the x-coordinate of
//a point by x*y

public void modify() {
    this.setX(this.getX() * this.getY());
}
```

- a) Write a Java program TestPoint.java with a *main* method that creates 2 points ptA (3, 5) and ptB (5, 3) and tests if they reside at the same location using both the *equals* methods. You should store the results in two *boolean* variables, test1 and test2 respectively. What are their values?

```
class TestPoint {

    public static void main(String[] args) {
        /*Fill in here*/
    }
}
```

### **Solution:**

```
class TestPoint {

    public static void main(String[] args) {
        Point ptA = new Point(3, 5);
        Point ptB = new Point(5, 3);

        boolean test1 = ptA.equals(ptB);
        boolean test2 = Point.equals(ptA, ptB);

        System.out.println("test1 = " + test1 +
                            "; test2 = " + test2);
    }
}
```

**Note:** The value stored in the variables, test1 and test2, should both be false. This question shows you the use of both instance and static methods. Instance method needs a Point instance to invoke and takes in another point as a parameter. Static method, on the other hand, does not require a Point instance to invoke, but it requires two points as parameters.

- b) What is the output of the following code?

```
class TestPoint {

    public static void main(String[] args) {
        Point ptC = new Point(3, 5);
        Point ptD = ptC;
        System.out.println("ptC: " + ptC);
        System.out.println("ptD: " + ptD);
        System.out.println();
    }
}
```

## CS1020: Data Structures and Algorithms

```
    ptD.modify();
    System.out.println("ptC: " + ptC);
    System.out.println("ptD: " + ptD);
}
}
```

### *Solution:*

```
ptC: (3.0, 5.0)
ptD: (3.0, 5.0)
```

```
ptC: (15.0, 5.0)
ptD: (15.0, 5.0)
```

**Note:** Both ptC and ptD are referencing to the same instance object in the memory. Hence, by changing the contents of ptD, the contents of ptC will also be changed.

Note that ptC and ptD in the two print statements automatically call the toString method in the class Point in Question 2.

c) What is the output of the following code?

```
class TestPoint {
    public static void main(String[] args) {
        Point ptE = new Point(3, 5);
        Point ptF = new Point(ptE);
        System.out.println("ptE: " + ptE);
        System.out.println("ptF: " + ptF);
        System.out.println();

        ptF.modify();
        System.out.println("ptE: " + ptE);
        System.out.println("ptF: " + ptF);
    }
}
```

### *Solution:*

```
ptE: (3.0, 5.0)
ptF: (3.0, 5.0)
```

```
ptE: (3.0, 5.0)
ptF: (15.0, 5.0)
```

**Note:** The keyword “new” creates a new Point instance for ptF, which also means that a new memory location (different from ptE) is allocated to store the contents of ptF. Hence, modifying the contents of ptF will not affect the contents of ptE as they reside in two different locations in the memory.