Department of Computer Science, HKBU

COMP2026 Problem Solving Using Object Oriented Programming

Laboratory 8

Part A Discovery Exercises

Task 1: Bee Object

Refer to the **`Bee.java`** example and answer the following questions.

a) Write statements to create 3 Bee objects with the following values.

Object name	x-coordinate	y-coordinate	dx
b1	50	60	2
b2	97	54	5
b3	7	8	7

Then, write statements to move the Bee objects horizontally and print the Bee objects.

b) Create a project in IntelliJ and put **`Bee.java`** and **`BeeTester.java`** into the **`src`** folder of the project. Test part (a) in the **`runApp()`** method in **`BeeTester.java`**. Run the program and paste the screenshot of the output below.

c) Write statements to update the x and y coordinates of the Bee objects in part (a) by using the **`setX()`** and **`setY()`** methods and then print the objects.

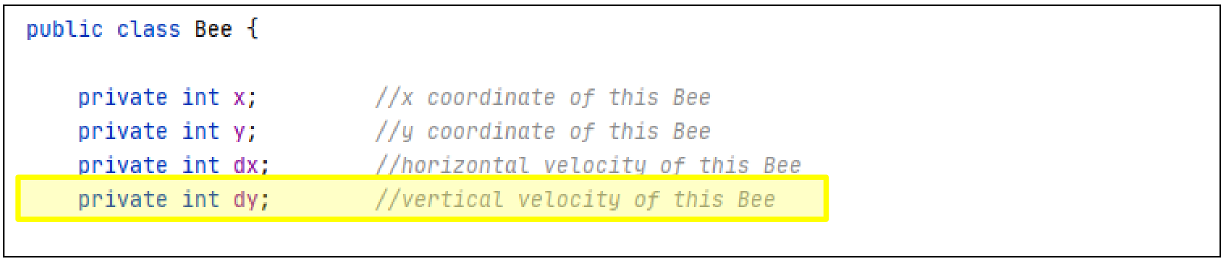
Object name	New x-coordinate	New y-coordinate
b1	12	34
b2	56	78
b3	90	11

d) Test part (c) in the **`runApp()`** method in **`BeeTester.java`**. Run the program and paste the screenshot of the output below.

e) Write statements to get the x coordinates of the 3 Bee objects by using **`getX()`** and then print the sum of them.

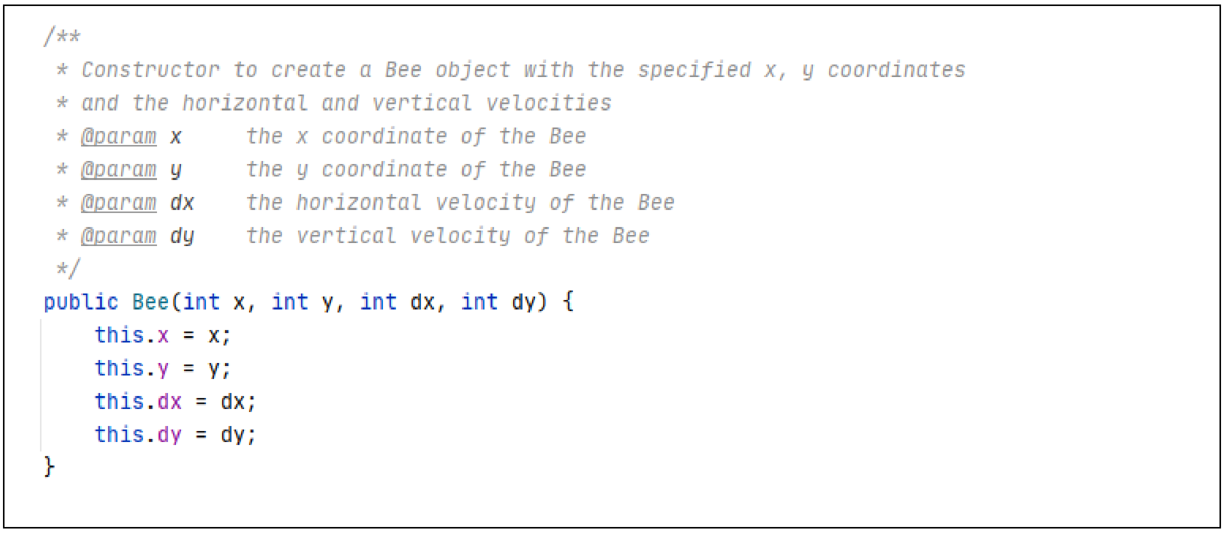
f) Test part (e) in the **`runApp()`** method in **`BeeTester.java`**. Run the program and paste the screenshot of the output below.

g) In **`Bee.java`**, add a new instance variable called **`dy`**, which stand for the vertical velocity of the bee, as follows.



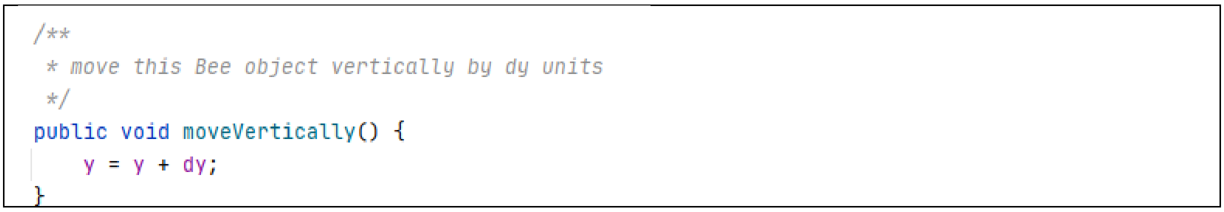
```
public class Bee {

    private int x;          //x coordinate of this Bee
    private int y;          //y coordinate of this Bee
    private int dx;         //horizontal velocity of this Bee
    private int dy;         //vertical velocity of this Bee
```

h) In **Bee.java**, add a new constructor to construct a bee with the specified x and y coordinates, the horizonal and vertical velocities as follows.



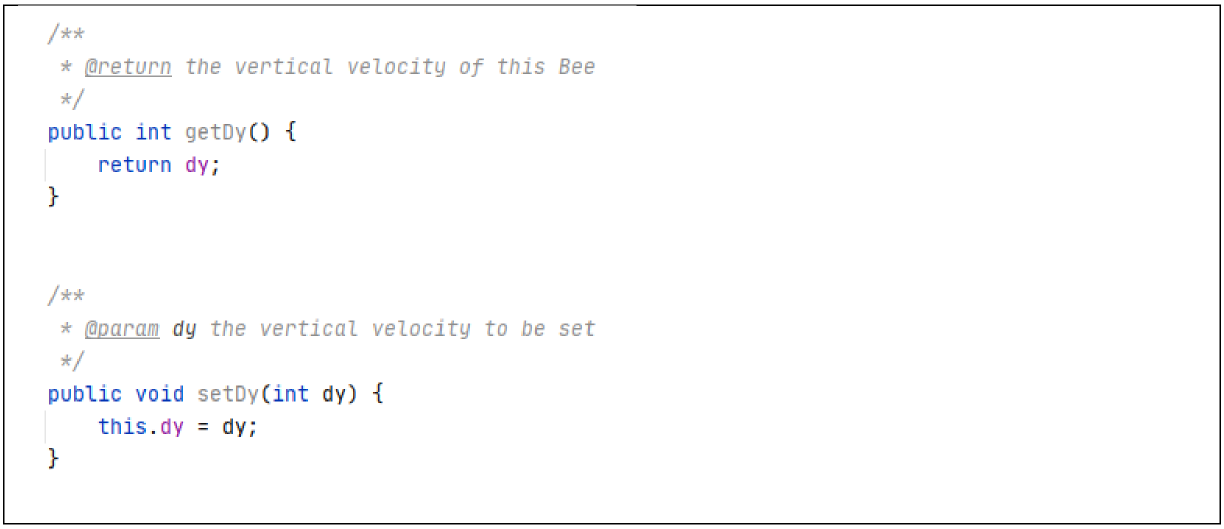
```java
/**
 * Constructor to create a Bee object with the specified x, y coordinates
 * and the horizontal and vertical velocities
 * @param x     the x coordinate of the Bee
 * @param y     the y coordinate of the Bee
 * @param dx    the horizontal velocity of the Bee
 * @param dy    the vertical velocity of the Bee
 */
public Bee(int x, int y, int dx, int dy) {
    this.x = x;
    this.y = y;
    this.dx = dx;
    this.dy = dy;
}
```

i) In **Bee.java**, add a **moveVertically()** method that moves the bee vertically by **dy**.



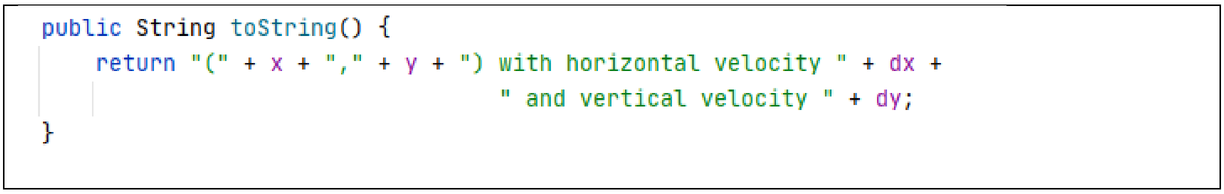
```java
/**
 * move this Bee object vertically by dy units
 */
public void moveVertically() {
    y = y + dy;
}
```

j) In **Bee.java**, add the get and set methods for **dy**.



```java
/**
 * @return the vertical velocity of this Bee
 */
public int getDy() {
    return dy;
}


/**
 * @param dy the vertical velocity to be set
 */
public void setDy(int dy) {
    this.dy = dy;
}
```

k) In **Bee.java**, modify the **toString()** methods to make it also prints the vertical velocity **dy**.



```java
public String toString() {
    return "(" + x + "," + y + ") with horizontal velocity " + dx +
                                " and vertical velocity " + dy;
}
```

l) In **`BeeTester.java`**, write statements to create 3 more Bee objects with the following values.

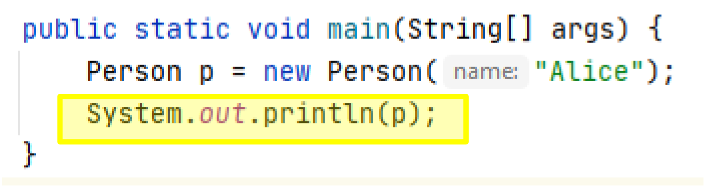
Object name	x-coordinate	y-coordinate	dx	dy
b4	11	22	1	2
b5	33	44	3	4
b6	55	66	5	6

Then, write statements to move these Bee objects vertically and print the Bee objects.

m) Test part (l) in the **`runApp()`** method in **`BeeTester.java`**. Run the program and paste the screenshot of the output below.

Task 2: toString Method

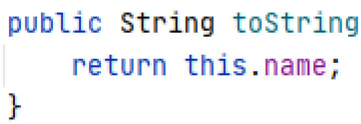
a) Add the following print statement to the **main** of the given **Person.java** to print the person object p.



```
public static void main(String[] args) {
    Person p = new Person( name: "Alice");
    System.out.println(p);
}
```

b) Run the program and paste the screenshot of the output below.

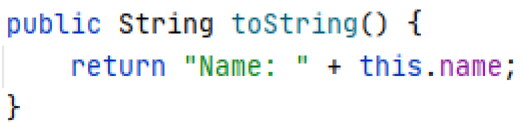
c) Add the following **toString** method to the Person class.



```
public String toString() {
    return this.name;
}
```

d) Run the program again and paste the screenshot of the output below.

e) Modify the **toString** method in the Person class as follows.

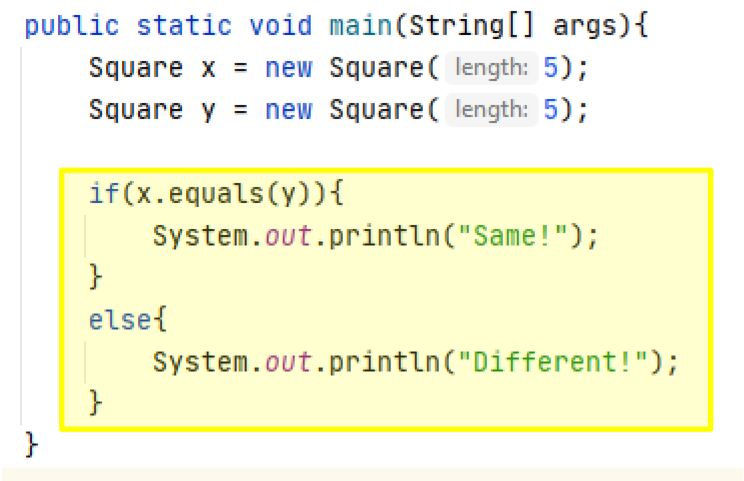


```
public String toString() {
    return "Name: " + this.name;
}
```

f) Run the program again and paste the screenshot of the output below.

Task 3: equals Method

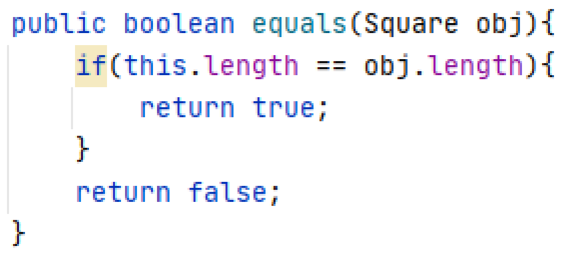
a) Add the following if statement to the **main** of the given **Square.java** to check whether the squares are equal.



```
public static void main(String[] args){
    Square x = new Square( length: 5);
    Square y = new Square( length: 5);

    if(x.equals(y)){
        System.out.println("Same!");
    }
    else{
        System.out.println("Different!");
    }
}
```

b) Run the program and paste the screenshot of the output below.

c) Add the following **equals** method to the Square class.



```
public boolean equals(Square obj){
    if(this.length == obj.length){
        return true;
    }
    return false;
}
```

d) Run the program again and paste the screenshot of the output below.

Part B Programming Exercises

Task 1: Playing Card

There are 52 cards in a standard deck. Each card belongs to one of four suits and one of 13 ranks. The suits are Diamonds, Clubs, Hearts and Spades. The ranks are Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, and King.
Write a class named **Card** to represent a card in a standard deck. The class contains:

- Two private integer fields **suit** and **rank** that represent the suit and rank of a card with the following mappings.
  - The mapping for suites:

Diamonds	→	0
Clubs	→	1
Hearts	→	2
Spades	→	3

  - The mapping for ranks:
    Each of the numerical ranks (2 through 10) maps to the corresponding integer, and for face cards:

Ace	→	1
Jack	→	11
Queen	→	12
King	→	13

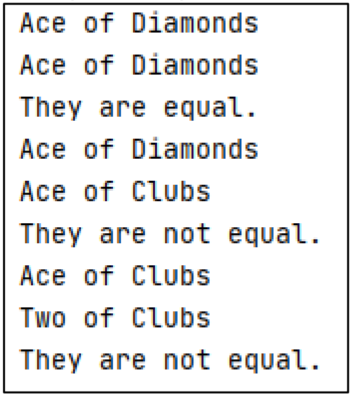
- A public constructor that constructs a card with the specified **suit** and **rank**.
- Two public get methods for fields **suit** and **rank** respectively.
- Two public set methods for fields **suit** and **rank** respectively.
- A public method named **equals** that returns whether this card equals another card of the **Card** type. Two cards are equals if they are having the same suit and rank.
- A public method named **toString** that returns the string representation of the card in the format "**Rank** of **Suit**", e.g. "Ace of Diamonds", "Ten of Hearts", "King of Clubs", etc.

Test your Card class with the following **main** method:

```
public static void main(String[] args){
    Card c1 = new Card(0, 1); //suit: Diamonds, rank: Ace
    Card c2 = new Card(0, 1); //suit: Diamonds, rank: Ace
    Card c3 = new Card(1, 1); //suit: Clubs, rank: Ace
    Card c4 = new Card(1, 2); //suit: Clubs, rank: 2

    System.out.println(c1);
    System.out.println(c2);
```

```
        if(c1.equals(c2)){
            System.out.println("They are equal.");
        }
        else{
            System.out.println("They are not equal.");
        }
        System.out.println(c1);
        System.out.println(c3);
        if(c1.equals(c3)){
            System.out.println("They are equal.");
        }
        else{
            System.out.println("They are not equal.");
        }
        System.out.println(c3);
        System.out.println(c4);
        if(c3.equals(c4)){
            System.out.println("They are equal.");
        }
        else{
            System.out.println("They are not equal.");
        }
    }
```

Sample output:



```
Ace of Diamonds
Ace of Diamonds
They are equal.
Ace of Diamonds
Ace of Clubs
They are not equal.
Ace of Clubs
Two of Clubs
They are not equal.
```

Task 2: Hand of Cards

Write a class named **`Hand`** to represent a hand of cards. The cards belong to the class **`Card`**. A hand is empty when it is created, and any number of cards can be added to it. The class contains:

- A private **`Card`** array field that represents the hand of cards.
- A public constructor that constructs an array of cards with zero size for the hand.
- A public **`getCount`** method to return the number of cards in the hand.
- A public **`addCard`** method to add a **`Card`** to the hand.
- A public **`getCard`** method to return the card from the hand in the specified position, where positions are numbered starting from 0.
- A public **`removeCard`** method to remove the card in the specified position form the hand. After removing the card, the size of hand should be reduced by 1.

- A public **removeCard** method to remove the specified **Card** form the hand if the specified card is in the hand. After successfully removing the card, the size of hand should be reduced by 1.

- A public method named **toString** that returns the string representation of the hand of cards in the format “[**position**] **Card**”. For example, a hand of three cards would return a String as follows.

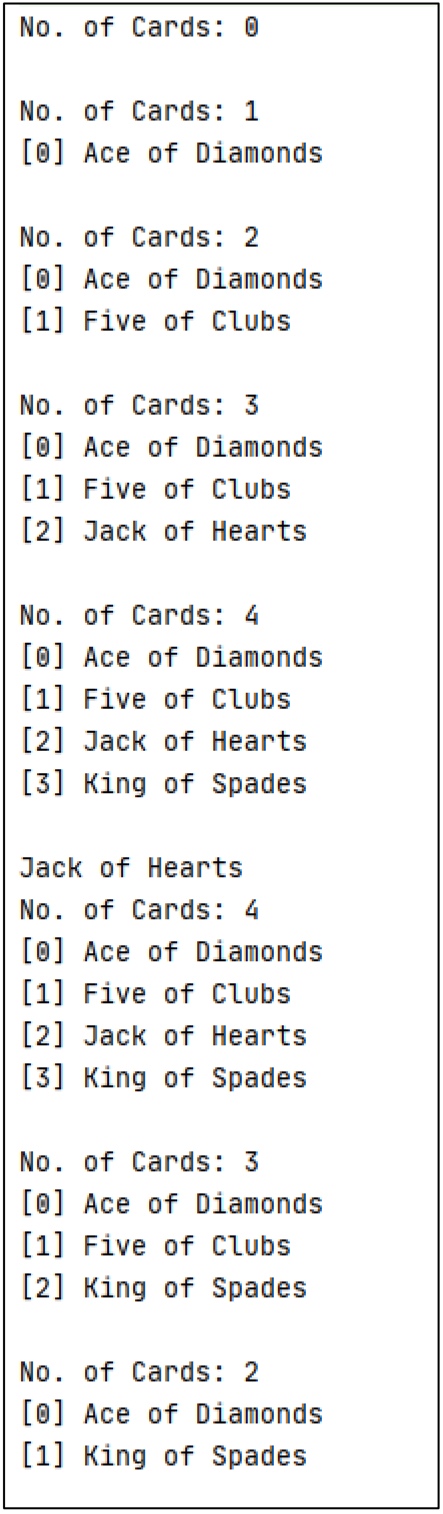
  [0] Ace of Diamonds
  [1] Ten of Hearts
  [2] King of Clubs

- *Note that you need to use the Card class in Task 1.*

Test your Hand class with the following **main** method:

```java
public static void main(String[] args){
    Hand myHand = new Hand();
    Card c1 = new Card(0, 1); //suit: Diamonds rank: Ace
    Card c2 = new Card(1, 5); //suit: Clubs rank: 5
    Card c3 = new Card(2, 11); //suit: Hearts rank: Jack
    Card c4 = new Card(3, 13); //suit: Spades rank: King

    //getCount
    System.out.println("No. of Cards: " + myHand.getCount());
    System.out.println(myHand); //empty

    //add cards
    myHand.addCard(c1);
    System.out.println("No. of Cards: " + myHand.getCount());
    System.out.println(myHand);
    myHand.addCard(c2);
    System.out.println("No. of Cards: " + myHand.getCount());
    System.out.println(myHand);
    myHand.addCard(c3);
    System.out.println("No. of Cards: " + myHand.getCount());
    System.out.println(myHand);
    myHand.addCard(c4);
    System.out.println("No. of Cards: " + myHand.getCount());
    System.out.println(myHand);

    //getCard
    System.out.println(myHand.getCard(2));
    System.out.println("No. of Cards: " + myHand.getCount());
    System.out.println(myHand);

    //remove cards
    myHand.removeCard(2);
    System.out.println("No. of Cards: " + myHand.getCount());
    System.out.println(myHand);

    myHand.removeCard(new Card(1, 5));
    System.out.println("No. of Cards: " + myHand.getCount());
    System.out.println(myHand);

}
```

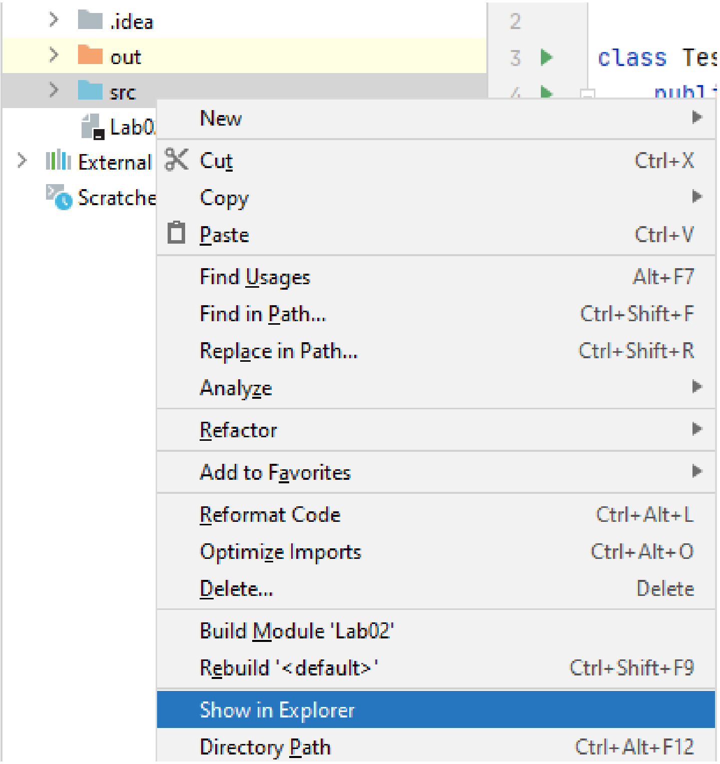
Sample output:



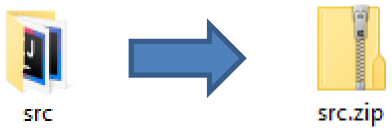
```
No. of Cards: 0

No. of Cards: 1
[0] Ace of Diamonds

No. of Cards: 2
[0] Ace of Diamonds
[1] Five of Clubs

No. of Cards: 3
[0] Ace of Diamonds
[1] Five of Clubs
[2] Jack of Hearts

No. of Cards: 4
[0] Ace of Diamonds
[1] Five of Clubs
[2] Jack of Hearts
[3] King of Spades

Jack of Hearts
No. of Cards: 4
[0] Ace of Diamonds
[1] Five of Clubs
[2] Jack of Hearts
[3] King of Spades

No. of Cards: 3
[0] Ace of Diamonds
[1] Five of Clubs
[2] King of Spades

No. of Cards: 2
[0] Ace of Diamonds
[1] King of Spades
```

Part C Submitting Exercises

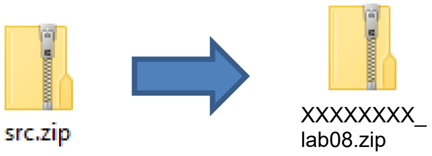
Step 1: Right-click the **src** folder and select **Show in Explorer**



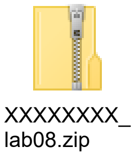
Step 2: Zip the **src** folder into **src.zip**



Step 3: Rename the src.zip file to ***XXXXXXXX_lab08.zip*** where ***XXXXXXXX*** is your **student id**



Step 4: Submit ***XXXXXXXX_lab08*.zip** and **XXXXXXXX_lab08.docx** to Moodle.



References

[1] Bravaco, R., & Simonson, C. (2009). *Java programming: From the ground up*. Dubuque, IA: McGraw-Hill.

[2] Dean, J., & Dean, R. (2008). *Introduction to programming with Java: A problem solving approach*. Boston: McGraw-Hill.

[3] Farrell, J. (2012). *Java programming. Boston, MA: Course Technology Cengage Learning*

[4] Forouzan, B. A., & Gilberg, R. F. (2007). *Computer science: A structured programming approach using C (3rd ed.)*. Boston, MA: Thomson Course Technology.

[5] Gaddis, T. (2016). *Starting out with Java (6th ed.)*. Pearson.

[6] Liang, Y. D. (2013). *Introduction to Java programming: Comprehensive version*. (8th ed.). Pearson.

[7] Schildt, H. (2006). *Java a beginner's guide*. New York: McGraw Hill.

[8] Schildt, H., & Skrien, D. J. (2013). Java programming: A comprehensive introduction. New York: McGraw-Hill.

[9] Wu, C. T. (2010). *An introduction to object-oriented programming with Java*. Boston: McGraw Hill Higher Education

[10] Xavier, C. (2011). *Java programming: A practical approach*. New Delhi: Tata McGraw Hill.

[11] yet another insignificant Programming Notes. (n.d.). Retrieved from https://www3.ntu.edu.sg/home/ehchua/programming

[12] Zakhour, S., Kannan, S., & Gallardo, R. (2013). *The Java tutorial: A short course on the basics (5th ed.)*.