**Task 1**

**Consider the following code:**

|  |
| --- |
| **public class Quiz1** |
| **{** |
| **public static void main(String args[])** |
| **{** |
| **double [] myArray = new double[5];** |
| **String test = "";** |
| **int i = 0, j = 0, k = 18;** |
| **while (i < 4){** |
| **myArray[i] = i + 3;** |
| **test = "-->";** |
| **j = i - 1;** |
| **while (j > (i - 3) ){** |
| **if (j >= 0){** |
| **myArray[i] = myArray[i+1] - myArray[j] / 2 + 1;** |
| **}** |
| **j--;** |
| **}** |
| **test = test + myArray[i] + "-->" + j + 3 + 45;** |
| **System.out.println(test);** |
| **i++;** |
| **}** |
| **test = "-->" + myArray[i-3] + "-->" + myArray[i-1];** |
| **System.out.println(test);** |
| **}** |
| **}** |

What is the output?

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Task 2**

Consider the following class:

**class Student{**

**public String name;**

**public double cgpa;**

**}**

**Show the output of the following sequence of statements:**

|  |  |
| --- | --- |
| **Student s1 = new Student();** **Student s2 = new Student();** **Student s3;** **s1.name = "Student One";** **s1.cgpa = 2.3;** **s3 = s1;** **s2.name = "Student Two";**  **s2.cgpa = s3.cgpa++;**  **s3.name = "New Student";** **System.out.println(s1.name);** **System.out.println(s2.name);** **System.out.println(s3.name);** **System.out.println(s1.cgpa);** **System.out.println(s2.cgpa);** **System.out.println(s3.cgpa);** **s3 = s2;** **s1.name = "old student";** **s2.name = "older student";** **s3.name = "oldest student";**  **s2.cgpa = s1.cgpa - s3.cgpa + 1.3;**  **System.out.println(s1.name);** **System.out.println(s2.name);** **System.out.println(s3.name);** **System.out.println(s1.cgpa);** **System.out.println(s2.cgpa);** **System.out.println(s3.cgpa);** | **Output** |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Task 3**

**Consider the following code:**

|  |
| --- |
| **class msgClass{** |
| **public int content;** |
| **}** |
| **public class FinalT5A{** |
| **private int sum = 2;** |
| **private int y = 1;** |
| **public int x = 1;** |
| **public void methodA(){** |
| **int x=0, y =0, i = 0;** |
| **while (i < 2){** |
| **msgClass myMsg = new msgClass();** |
| **myMsg.content = this.x;** |
| **this.y = this.y + methodB(myMsg, myMsg.content);** |
| **System.out.println(x + " " + y+ " " + sum);** |
| **y = this.y / 2;** |
| **x = y + sum/2 - i;** |
| **sum = x + y + myMsg.content;** |
| **i++;** |
| **}** |
| **System.out.println(x + " " + y+ " " + sum);** |
| **}** |
| **private int methodB(msgClass mg2, int mg1){** |
| **int x = 0;** |
| **y = y + mg2.content;** |
| **mg2.content = y + mg1;** |
| **x = this.x + 3 + mg1;** |
| **sum = sum + x + y;** |
| **System.out.println(this.x + " " + this.y+ " " + sum);** |
| **mg2.content = sum - mg1 ;** |
| **return sum;** |
| **}** |
| **}** |

What is the output of the following code sequence?

|  |  |  |  |
| --- | --- | --- | --- |
| **FinalT5A fT5A = new FinalT5A();**  **fT5A.methodA();** | **x** | **y** | **sum** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

#### **Task 4**

|  |
| --- |
| **public class FinalT6A{** |
| **public static int temp = 3;** |
| **private int sum;** |
| **private int y = 2;** |
| **public FinalT6A(int x, int p){** |
| **temp+=3;** |
| **y = temp - p;** |
| **sum = temp + x;** |
| **System.out.println(x + " " + y+ " " + sum);** |
| **}** |
| **public void methodA(){** |
| **int x=0, y =0;** |
| **y = y + this.y;** |
| **x = this.y + 2 + temp;** |
| **sum = x + y + methodB(temp, y);** |
| **System.out.println(x + " " + y+ " " + sum);** |
| **}** |
| **public int methodB(int temp, int n){** |
| **int x = 0;** |
| **y = y + (++temp);** |
| **x = x + 2 + n;** |
| **sum = sum + x + y;** |
| **System.out.println(x + " " + y+ " " + sum);** |
| **return sum;** |
| **}** |
| **}** |

What is the output of the following code sequence?

|  |  |  |  |
| --- | --- | --- | --- |
| **FinalT6A q1 = new FinalT6A(2,1);**  **q1.methodA();**  **q1.methodA();** | **x** | **y** | **sum** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

#### **Task 5**

**public class Quiz1{**

**public static int temp = 4;**

**public int sum;**

**public int y;**

**public Quiz1(){**

**y = temp - 1;**

**sum = temp + 1;**

**temp+=2;**

**}**

**public Quiz1(int p){**

**y = temp + p ;**

**sum = p+ temp + 1;**

**temp-=1;**

**}**

**public void methodA(){**

**int x=0, y =0;**

**y = y + this.y;**

**x = this.y + 2 + temp;**

**sum = x + y + methodB(x, y);**

**System.out.println(x + " " + y+ " " + sum);**

**}**

**public int methodB(int m, int n){**

**int x = 0;**

**y = y + m + (++temp);**

**x = x + 2 + n;**

**sum = sum + x + y;**

**System.out.println(x + " " + y+ " " + sum);**

**return sum;**

**}**

**}**

**Consider the following code:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Quiz1 q1 = new Quiz1();**  **q1.methodA();**  **q1.methodA();**  **Quiz1.temp+= 2;**  **Quiz1 q2 = new Quiz1(2);**  **q2.methodA();**  **q2.methodA();** | **x** | **y** | **sum** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Task 6**

|  |
| --- |
| **class A{** |
| **public static int temp = 4;** |
| **public int sum;** |
| **public int y;** |
| **public A(){** |
| **y = temp - 2;** |
| **sum = temp + 1;** |
| **temp-=2;** |
| **}** |
| **public void methodA(int m, int n){** |
| **int x = 0;** |
| **y = y + m + (temp++);** |
| **x = x + 1 + n;** |
| **sum = sum + x + y;** |
| **System.out.println(x + " " + y+ " " + sum);** |
| **}** |
| **}** |
| **class B{** |
| **public static int x;** |
| **public int y = 5;** |
| **public int temp = -5;** |
| **public int sum = 2;** |
| **public B(){** |
| **y = temp + 3 ;** |
| **sum = 3 + temp + 2;** |
| **temp-=2;** |
| **}** |
| **public B(B b){** |
| **sum = b.sum;** |
| **x = b.x;** |
| **b.methodB(2,3);** |
| **}** |
| **public void methodA(int m, int n){** |
| **int x = 2;** |
| **y = y + m + (temp++);** |
| **x = x + 5 + n;** |
| **sum = sum + x + y;** |
| **System.out.println(x + " " + y+ " " + sum);** |
| **}** |
| **public void methodB(int m, int n){** |
| **int y = 0;** |
| **y = y + this.y;** |
| **x = this.y + 2 + temp;** |
| **methodA(x, y);** |
| **sum = x + y + sum;** |
| **System.out.println(x + " " + y+ " " + sum);** |
| **}** |
| **}** |

**Consider the following code:**

|  |  |  |  |
| --- | --- | --- | --- |
| **A a1 = new A();**  **B b1 = new B();**  **B b2 = new B(b1);**  **b1.methodA(1, 2);**  **b2.methodB(3, 2);** | **x** | **y** | **sum** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Task 7**

|  |
| --- |
| **class A{** |
| **public static int temp = 3;** |
| **public int sum;** |
| **public int y;** |
| **public int x;** |
| **public A(){** |
| **int temp = -3;** |
| **int sum = 7;** |
| **y = temp - 5;** |
| **sum = temp + 2;** |
| **temp-=2;** |
| **this.x = sum + temp + y;** |
| **}** |
| **public A(int y, int temp){** |
| **y = temp - 1+ x;** |
| **sum = temp + 2 -x;** |
| **temp-=2;** |
| **}** |
| **public void methodA(int m, int [] n){** |
| **int x = 0;** |
| **y = y + m + methodB(x,m)+(temp++)+y;** |
| **x = this.x + 2 + (++n[0]);** |
| **sum = sum + x + y;** |
| **n[0] = sum + 2;** |
| **System.out.println(n[0] + x + " " + y+ " " + sum + n[0]);** |
| **}** |
| **public int methodB(int m, int n){** |
| **int [] y = {0};** |
| **this.y = y[0] + this.y + m;** |
| **x = this.y + 2 + temp - n;** |
| **sum = x + y[0] + this.sum;** |
| **System.out.println(y[0]+ x + "this.temp" + y[0]+ " " +sum+ y[0]);** |
| **return y[0];** |
| **}** |
| **}** |
| **class B{** |
| **public int y=1;** |
| **public int temp=-3;** |
| **public int x = 1;** |
| **public static int sum = 2;** |
| **public B(){** |
| **y = temp + 1 ;** |
| **x = 3 + temp + x;** |
| **temp-=2;** |
|  |
| **}** |
| **public B(B b){** |
| **sum = b.sum + this.sum;** |
| **x = b.x + x;** |
| **b.methodB(3,5);** |
| **}** |
| **public void methodA(int m, int [] n){** |
| **int x = 0;** |
| **y = y + m + (temp++);** |
| **x = x + 2 + (++n[0]);** |
| **sum = sum + x + y;** |
| **n[0] = sum + 2;** |
| **System.out.println(temp + x + " " + y+ " " + sum + temp);** |
| **}** |
| **public void methodB(int m, int n){** |
| **int [] y = {0};** |
| **this.y = y[0] + this.y + m;** |
| **x = this.y + 2 + temp - n;** |
| **methodA(x, y);** |
| **sum = x + y[0] + this.sum;** |
| **System.out.println(n + x + " " + y[0]+ " " + sum + n);** |
| **}** |
| **}** |
| **Consider the following code:** |
| **int x[] = {35};** |
| **A a1 = new A();** |
| **A a2 = new A(-5,-7);** |
| **B b1 = new B();** |
| **B b2 = new B(b1);** |
| **a1.methodA(1, x);** |
| **b2.methodB(3, 2);** |
| **a2.methodA(1, x);** |

#### **Task 8**

**Study the following output**

|  |  |
| --- | --- |
| **Code** | **Output** |
| **public class StudentTest{**  **public static void main(String [] args){**  **Student s1 = new Student();**  **System.out.println(s1.getName());**  **Student s2 = new Student("Matin");**  **System.out.println(s2.getName());**  **Student s3 = new Student("Saad");**  **System.out.println(s3.getName());**  **System.out.println(Student.numberOfStudents);**  **}**  **}** | **default name**  **Matin**  **Saad**  **3** |

Write the code for the Student class so that the StudentTest class generates the output shown above.

#### **Task 9**

Write a **Java program** takes a positive number ***n*** typed by the user and then, using nested iterations (loops), outputs the following triangle of numbers. You may assume that as the value of ***n***, user may enter any number between 1 and 9.

**[Answer on the answer-script]**

1

22

333

…..

…..

nnn…n

#### **Task 10**

Generate the following pattern (e.g using ‘loop’ and ‘break’ operation)

0

0 1

0 2 4

0 3 6 9

0 4 8 12 16

0 5 10 15 20 25

0 6 12 18 24 30 36

0 7 14 21 28 35 42 49

0 8 16 24 32 40 48 56 64

0 9 18 27 36 45 54 63 72 81

**Task 11**

Write a Java program that will ask a BU student for his/her Name and BU ID and then show the student:

1) His/her first name

2) His/her last name

3) Middle initials

4) His/her department

5) His/her first year at BU

6) His/her first semester at BU

7) Till now how many semesters he/she has finished at BU.

8) Assuming it takes 12 semesters to graduate show the approximate semester and year and semester the student is going to graduate in.

**TASK 12**

Create a class called BankAccount as described below:

* **Fields:**name, address, accountID, balance
* **Methods:**   
  public String getName()  
  public void setName(String n)  
  public String getAccountID()  
  public void setAccountID(String i)  
  public String getAddress()  
  public void setAddress(String a)  
  public double getBalance()  
  public void setBalance(double c)  
  public void addInterest() //adds 7% of the balance

Write a class called AccountTester to write a main() method:

* public static void main(String[] args){  
    
  }
* Inside the main() method
  + Create 3 objects/instances of BankAccount called acc1, acc2 and acc3
  + Set their fields to some value using the public methods.
  + Call addInterest() on acc1 and acc3
  + Print the information of each BankAccount using System.out.println()

Add constructors to BankAccount and use the constructor to set the field values.

##### Task 13

Create a class **SavingsAccount,** which will use a **static** class variable to store the **annualInterestRate** for all account holders.

* Each object of the class contains a **private** instance variable **savingsBalance** indicating the amount the saver currently has on deposit.
* Provide method **calculateMonthlyInterest()** to calculate the monthly interest [by multiplying the **savingsBalance** by **annualInterestRate** divided by 12], this interest should be added to **savingsBalance**.
* Provide a **static** method **modifyInterestRate()** that sets the **annualInterestRate** to a new value.

Write a driver program to test class **SavingsAccount**.

* Instantiate two **SavingsAccount** objects, **saver1** and **saver2**, with balances $20000.00 and $30000.00, respectively using constructor.
* Set **annualInterestRate** to 4.2%, then calculate the monthly interest and print the new balances for each of the savers using **printSavingsBalance( )** method.
* Then set the **annualInterestRate** to 5.5% and calculate the next month’s interest and print the new balances for each of the savers.

#### **Task 14**

Design a class an instance of which is a pair of dice. Each die has six faces, with the numbers in 1, 2, 3, 4, 5, 6 on them. ***At all times***, one face is face up. What methods will you have? You probably want one for rolling the dice, so that new faces become face up in a ***random*** manner.

[**Math.random()** Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.]

**Task 15**

Create a Java program plays tic-tac-toe (or, as the British call it, "noughts and crosses").

If you have never played tic-tac-toe, read the description of the game in the next section; otherwise, skip to the assignment specifications.

**Description**

Tic-tac-toe is played on a 3 x 3 grid. Two players take turns at selecting squares on the grid, the first marking his or her moves with an "O" (British, "nought"), the second, with "X" ("cross"). The first player to select three squares in a row -- horizontally, vertically, or diagonally -- wins the game.

**Specifications**

Your program should draw the game board using a simple representation, with the squares numbered from 1 to 9, such as:

1 | 2 | 3

-----------

4 | 5 | 6

-----------

7 | 8 | 9

It should then prompt the user to select one of the squares, by number. It should check that a valid selection has been made (i.e., the number is in range and was not previously selected by either player) and, if not, print an appropriate message and prompt again. The following example illustrates such an interaction:

X | X | 3

-----------

4 | O | 6

-----------

7 | O | 9

Your move? 2

Please enter one of the digits shown on the board.

Your move? 17

Please enter one of the digits shown on the board.

Your move? 3

Once the human player has made a valid selection, the program should choose a square for its move. It should then indicate which square was selected and print an updated representation of the board, indicating the human player's moves with O's and its own moves with X's. The next round of play then commences with another prompt to the human player, as in:

I chose square 4.

X | X | O

-----------

X | O | 6

-----------

7 | O | 9

Your move?

After each move -- both the human's and its own -- the program should check for a winner. If a player has won, an appropriate message should be printed and the game ends.

For example, if the human player wins, it might print:

X | X | O

-----------

4 | O | 6

-----------

O | 8 | 9

Congratulations. You won.

If the computer wins, it might print the following:

X | X | O

-----------

X | O | O

-----------

X | O | 9

I won. Let's play again soon.

The program should also check for a no-win condition, in which all squares have been selected, but neither player has won:

X | O | X

-----------

X | O | O

-----------

O | X | 9

Your move? 9

The game ended with no winner.

**Hints**

The playing grid could be represented internally with an array of size 9. However, because the human player marks squares with an 'O', it is not a good idea to number the squares 0 through 8 instead of 1 through 9. In order to avoid the nuisance of adjusting player selections in the range 1 ... 9 onto the array indices 0 ... 8, it is probably a good idea to create an array of size 10, and only use elements 1 ... 9.

You could represent the board with a two-dimensional (3 x 3) array, but this is probably more trouble than it is worth, because you would have to map the user's selections onto pairs of row and column numbers. It's easier to write a board printing routine that starts a new row after each set of three squares.

There are several ways one could keep track of which squares have been selected by which player and which are still available. One of the simplest is to declare the array as of type char[] (i.e., an array of characters). If the array is then initialized with the characters '1' through '9' and player moves are recorded by storing an 'X' or an 'O' in place of the appropriate digit character, it is easy to print the 3 x 3 representation of the current state of the grid.

If this method is chosen, be careful to note the distinction between the numbers 1, 2, ..., 9 and the characters '1', '2', ..., '9'.

You are not required to make the program play intelligently. You could have it choose the first available square on each of its moves. This doesn't make it very much fun to play against the computer, but if you need to play tic-tac-toe with a computer in order to have fun, your life is too sad and miserable for you to notice that the computer is not playing intelligently.

Remember to declare all your functions to be static. Using several small functions makes it much easier to write the program. At minimum, you will want a function to print the current state of the playing grid, and another to determine if some player has won.

**Variations**

You could put the whole tic-tac-toe game procedure in an outer loop that allows the human to play multiple games against the computer, and keeps score over all games in the tournament.

You could make the computer play more intelligently by at least recognizing a threat -- two of the opponent's marks in the same row, column, or diagonal -- and blocking the win by the human player.

You could have the computer recognize a tie game before prompting the user to enter the number of the only remaining grid square. It's more work, and it won't earn you any more points, but you *could* do that.

**Task 16**

**Output:**

**========================**

**Name: Saad Abdullah**

**Department: CSE**

**List of courses**

**========================**

**CSE 110 Programming Language I**

**CSE 111 Programming Language-II**

**========================**

**========================**

**Name: Mumit Khan**

**Department: CSE**

**List of courses**

**========================**

**CSE 220 Data Structures**

**CSE 221 Algorithms**

**CCSE 230 Discrete Mathematics**

**========================**

**========================**

**Name: Sadia Kazi**

**Department: CSE**

**List of courses**

**========================**

**CSE 310 Object Oriented Programming**

**CSE 320 Data Communications**

**CSE 340 Computer Architecture**

**========================**

**Tester:**

public class TestTeacher{

public static void main(String [] args){

Teacher t1 = new Teacher("Saad Abdullah", "CSE");

Teacher t2 = new Teacher("Mumit Khan", "CSE");

Teacher t3 = new Teacher("Sadia Kazi", "CSE");

Course c1 = new Course("CSE 110 Programming Language I");

Course c2 = new Course("CSE 111 Programming Language-II");

Course c3 = new Course("CSE 220 Data Structures");

Course c4 = new Course("CSE 221 Algorithms");

Course c5 = new Course("CCSE 230 Discrete Mathematics");

Course c6 = new Course("CSE 310 Object Oriented Programming");

Course c7 = new Course("CSE 320 Data Communications");

Course c8 = new Course("CSE 340 Computer Architecture");

t1.addCourse(c1);

t1.addCourse(c2);

t2.addCourse(c3);

t2.addCourse(c4);

t2.addCourse(c5);

t3.addCourse(c6);

t3.addCourse(c7);

t3.addCourse(c8);

t1.printDetail();

t2.printDetail();

t3.printDetail();

}

}

Write the Teacher and Course classes so that the TestTeacher class produces the outputs given above

**Task 17**

The class CasModCounter abstracts a series of counters like the one you see in your car's odometer. Each counter represents one single digit. Several counters may be cascaded as shown below:

**c4 <- c3 <- c2 <- c1 <- c0**

Each counter has a modulus limit. For example, if c0 has a limit of 3, it will count between the range 0 and 2. If the count is 2 and an increment stimuli is sent to it, its value will become 0 and it will forward the increment stimuli to the next counter in line (if there is one). Similarly, if the count is 0 and a decrement is received, its value will become 2 and it will forward the decrement to the next counter.

In order to create an object of a CasModCounter, you must provide the mod limit value.

An **increment** increases the value of the counter by one. If the value rolls over the limit, then the counter forwards the increment to the next counter in line, if there is one.

A **decrement** decreases the value of the counter by one. If the value rolls under 0, then the counter forwards the decrement to the next counter in line, if there is one.

Note the Counters are cascaded right to left. A **print** function called on the right most counter will forward the request to its next in line counter and then print its value.

The **attach** function attaches the argument counter object to the counter object. If a counter is already attached, then the request is forwarded to the attached counter.

#### **Task 18**

Create a class **Queue** with method **qInsert()** and **qRemove()** to do operation of insertion of elements at the back end and extraction of elements from the front end, respectively, as a typical queue, using array of integers or floating point values.

* For **qInsert( )** method insert the random integers.
* Print all the elements of queue invoking **qRemove( )** method.
* Create al least two instances of **Queue** class, whose size can be defined by constructor, for manipulating the corresponding methods.
* It should also provide a default constructor, which will initialize the queue with default size of **10**.
* Check the provision of ‘**Overflow’** and ‘**Underflow’** during **insert** and **remove** operation.