PROGRAMMING WITH OBJECTS LABORATORY

III Semester: Common for CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AITC03	**Core**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	0	3	1.5	30	70	100
Contact Classes: NIL	**Tutorial Classes: NIL**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: Programming for Problem Solving using C								

I. COURSE OVERVIEW:

This course provides students with hands-on experience in developing programs and applications using object oriented programming. It covers classes, objects, inheritance, polymorphism, exception handling, files, multi-threading, database connectivity and AWT. It helps the students to develop real-world applications and enhances their programming skills.

II. COURSES OBJECTIVES:

The students will try to learn

I. The basic concepts of object oriented programming.
II. The application of object oriented features for developing flexible and extensible applications.
III. The Graphical User Interface (GUI) with database connectivity to develop web applications.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1 Demonstrate object oriented programming concepts that helps to organize complex problems solving.
CO 2 Make use of the programming constructs like control Structures, arrays, parameter passing techniques and constructors to solve the real time problems.
CO 3 Utilize the abstraction, encapsulation and polymorphism Techniques to solve different complex problems.
CO 4 Experiment all threading and thread synchronization problems in soft real time systems.
CO 5 Make use of inheritance, interfaces, packages and files to implement reusability in soft real time systems.
CO 6 Construct GUI based applications along with Exception handling using AWT, Swings and JDBC connectivity.

IV. COURSE CONTENT:

PROGRAMMING WITH OBJECTS LABORATORY

1. Getting Started Exercises

1.1 HelloWorld

1. Install JDK on your machine. Follow the instructions in "How to Install JDK".
2. Write a Hello-world program using JDK and a source-code editor, such as:
   - For All Platforms: Sublime Text, Atom
   - For Windows: TextPad, NotePad++
   - For macOS: jEdit, gedit
   - For Ubuntu: gedit

1.2 CheckOddEven

Write a program called **CheckOddEven** which prints "`Odd Number`" if the `int` variable "`number`" is odd, or "`Even Number`" otherwise. The program shall always print "`bye!`" before exiting.

Hints

n is an even number if (`n % 2`) is `0`; otherwise, it is an odd number. Use == for comparison, e.g., (`n % 2) == 0`.

```
/**
 * Trying if-else statement and modulus (%) operator.
 */
public class CheckOddEven {   // Save as "CheckOddEven.java"
   public static void main(String[] args) {  // Program entry point
      int number = 49;       // Set the value of "number" here!
      System.out.println("The number is " + number);
      if ( ...... ) {
         System.out.println( ...... );   // even number
      } else {
         System.out.println( ...... );   // odd number
      }
      System.out.println( ...... );
   }
}
```

Try

`number = 0`, `1`, `88`, `99`, `-1`, `-2` and verify your results.

Again, take note of the source-code indentation! Make it a good habit to ident your code properly, for ease of reading your program.

1.3 PrintDayInWord

Write a program called **`PrintDayInWord`** which prints "`Sunday`", "`Monday`", ... "`Saturday`" if the `int` variable "`dayNumber`" is `0`, `1`, ..., `6`, respectively. Otherwise, it shall print "Not a valid day". Use (a) a "nested-`if`" statement; (b) a "`switch-case-default`" statement.

Try

`dayNumber = 0`, 1, 2, 3, 4, 5, 6, 7 and verify your results.

1.4 `Fibonacci` (Decision & Loop)

Write a program called **`Fibonacci`** to print the first `20` Fibonacci numbers `F(n)`, where `F(n)=F(n-1)+F(n-2)` and `F(1)=F(2)=1`. Also compute their average. The output shall look like:

```
The first 20 Fibonacci numbers are:
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
The average is 885.5
```

Hints

```
/**
 * Print first 20 Fibonacci numbers and their average
 */
public class Fibonacci {
   public static void main (String[] args) {
      int n = 3;           // The index n for F(n), starting from n=3, as n=1 and
n=2 are pre-defined
      int fn;             // F(n) to be computed
      int fnMinus1 = 1;   // F(n-1), init to F(2)
      int fnMinus2 = 1;   // F(n-2), init to F(1)
      int nMax = 20;      // maximum n, inclusive
      int sum = fnMinus1 + fnMinus2;  // Need sum to compute average
      double average;

      System.out.println("The first " + nMax + " Fibonacci numbers are:");
      ......

      while (n <= nMax) {  // n starts from 3
            // n = 3, 4, 5, ..., nMax
         // Compute F(n), print it and add to sum
         ......
         // Increment the index n and shift the numbers for the next iteration
         ++n;
         fnMinus2 = fnMinus1;
         fnMinus1 = fn;
      }

      // Compute and display the average (=sum/nMax).
      // Beware that int/int gives int.
      ......
   }
}
```

Try

1. *Tribonacci numbers* are a sequence of numbers `T(n)` similar to *Fibonacci numbers*, except that a number is formed by adding the three previous numbers, i.e., `T(n)=T(n-1)+T(n-2)+T(n-3)`, `T(1)=T(2)=1`, and `T(3)=2`. Write a program called **`Tribonacci`** to produce the first twenty Tribonacci numbers.

1.5 `ExtractDigits` (Decision & Loop)

Write a program called **`ExtractDigits`** to extract each digit from an `int`, in the reverse order. For example, if the `int` is 15423, the output shall be "3 2 4 5 1", with a space separating the digits.

Hints

The *coding pattern* for extracting individual digits from an integer *n* is:

1. Use `(n % 10)` to extract the last (least-significant) digit.
2. Use `n = n / 10` to drop the last (least-significant) digit.
3. Repeat if `(n > 0)`, i.e., more digits to extract.

Take note that *n* is destroyed in the process. You may need to clone a copy.

```
int n = ...;
while (n > 0) {
   int digit = n % 10;  // Extract the least-significant digit
   // Print this digit
   ......
   n = n / 10;  // Drop the least-significant digit and repeat the loop
}
```

Try

Write a program that prompts user for a positive integer. The program shall read the input as `int`; compute and print the sum of all its digits

1.6 `InputValidation` (Loop with `boolean` flag)

Your program often needs to validate the user's inputs, e.g., marks shall be between 0 and 100.
Write a program that prompts user for an integer between `0-10` or `90-100`. The program shall read the input as `int`; and repeat until the user enters a valid input. For examples,

```
Enter a number between 0-10 or 90-100: -1
Invalid input, try again...
Enter a number between 0-10 or 90-100: 50
Invalid input, try again...
Enter a number between 0-10 or 90-100: 101
Invalid input, try again...
Enter a number between 0-10 or 90-100: 95
You have entered: 95
```

Hints

Use the following *coding pattern* which uses a do-while loop controlled by a `boolean` flag to do input validation. We use a do-while instead of while-do loop as we need to execute the body to prompt and process the input at least once.

```
// Declare variables
int numberIn;      // to be input
boolean isValid;   // boolean flag to control the loop
......

// Use a do-while loop controlled by a boolean flag
// to repeatably read the input until a valid input is entered
isValid = false;   // default assuming input is not valid
do {
   // Prompt and read input
   ......

   // Validate input by setting the boolean flag accordingly
   if (numberIn ......) {
      isValid = true;   // exit the loop
   } else {
      System.out.println(......);  // Print error message and repeat
   }
} while (!isValid);
......
```

Try
Write a program that prompts user for the mark (between `0-100` in `int`) of 3 students; computes the average (in `double`); and prints the result rounded to 2 decimal places. Your program needs to perform input validation.

Hints

```
// Declare constant
final int NUM_STUDENTS = 3;

// Declare variables
int numberIn;
boolean isValid;   // boolean flag to control the input validation loop
int sum = 0;
double average;
......

for (int studentNo = 1; studentNo <= NUM_STUDENTS; ++studentNo) {
    // Prompt user for mark with input validation
    ......
    isValid = false;   // reset assuming input is not valid
    do {
       ......
    } while (!isValid);

    sum += ......;
}
......
```

1.7 IncomeTaxCalculator (Decision)

The progressive income tax rate is mandated as follows:

Taxable Income	Rate (%)
First $20,000	0
Next $20,000	10
Next $20,000	20
The remaining	30

For example, suppose that the taxable income is `$85000`, the income tax payable is `$20000*0% + $20000*10% + $20000*20% + $25000*30%`.

Write a program called **`IncomeTaxCalculator`** that reads the taxable income (in `int`). The program shall calculate the income tax payable (in `double`); and print the result rounded to 2 decimal places. For examples,

```
Enter the taxable income: $41234
The income tax payable is: $2246.80

Enter the taxable income: $67891
The income tax payable is: $8367.30

Enter the taxable income: $85432
The income tax payable is: $13629.60

Enter the taxable income: $12345
The income tax payable is: $0.00
```

Hints

```
// Declare constants first (variables may use these constants)
// The keyword "final" marked these as constant (i.e., cannot be changed).
// Use uppercase words joined with underscore to name constants
final double TAX_RATE_ABOVE_20K = 0.1;
final double TAX_RATE_ABOVE_40K = 0.2;
final double TAX_RATE_ABOVE_60K = 0.3;

// Declare variables
int taxableIncome;
double taxPayable;
......

// Compute tax payable in "double" using a nested-if to handle 4 cases
if (taxableIncome <= 20000) {         // [0, 20000]
   taxPayable = ......;
} else if (taxableIncome <= 40000) {  // [20001, 40000]
   taxPayable = ......;
} else if (taxableIncome <= 60000) {  // [40001, 60000]
   taxPayable = ......;
} else {                              // [60001, ]
   taxPayable = ......;
}
// Alternatively, you could use the following nested-if conditions
// but the above follows the table data
```

```
      //if (taxableIncome > 60000) {          // [60001, ]
      //   ......
      //} else if (taxableIncome > 40000) {   // [40001, 60000]
      //   ......
      //} else if (taxableIncome > 20000) {   // [20001, 40000]
      //   ......
      //} else {                              // [0, 20000]
      //   ......
      //}

      // Print results rounded to 2 decimal places
      System.out.printf("The income tax payable is: $%.2f%n", ...);
```

Try

Suppose that a `10%` tax rebate is announced for the income tax payable, capped at `$1,000`, modify your program to handle the tax rebate. For example, suppose that the tax payable is `$12,000`, the rebate is `$1,000`, as `10%` of `$12,000` exceeds the cap.

1.8 `IncomeTaxCalculatorWithSentinel` (Decision & Loop)

Based on the previous exercise, write a program called `IncomeTaxCalculatorWithSentinel` which shall repeat the calculation until user enter `-1`. For example,

```
Enter the taxable income (or -1 to end): $41000
The income tax payable is: $2200.00

Enter the taxable income (or -1 to end): $62000
The income tax payable is: $6600.00

Enter the taxable income (or -1 to end): $73123
The income tax payable is: $9936.90

Enter the taxable income (or -1 to end): $84328
The income tax payable is: $13298.40

Enter the taxable income: $-1
bye!
```

The `-1` is known as the *sentinel value*. (Wiki: In programming, a *sentinel value*, also referred to as a flag value, trip value, rogue value, signal value, or dummy data, is a special value which uses its presence as a condition of termination.)

Hints

The *coding pattern* for handling input with sentinel value is as follows:

```
      // Declare constants first
      final int SENTINEL = -1;    // Terminating value for input
      ......

      // Declare variables
      int taxableIncome;
      double taxPayable;
      ......

      // Read the first input to "seed" the while loop
      System.out.print("Enter the taxable income (or -1 to end): $");
      taxableIncome = in.nextInt();
```

```
      while (taxableIncome != SENTINEL) {
         // Compute tax payable
         ......
         // Print result
         ......

         // Read the next input
         System.out.print("Enter the taxable income (or -1 to end): $");
         taxableIncome = in.nextInt();
            // Repeat the loop body, only if the input is not the SENTINEL value.
            // Take note that you need to repeat these two statements inside/outside
the loop!
      }
      System.out.println("bye!");
```

Take note that we repeat the input statements inside and outside the loop. Repeating statements is NOT a good programming practice. This is because it is easy to repeat (Cntl-C/Cntl-V), but hard to maintain and synchronize the repeated statements. In this case, we have no better choices!

2. Exercises on Patterns and Arrays

2.1 SquarePattern (nested-loop)

Write a program called **SquarePattern** that prompts user for the size (a non-negative integer in `int`); and prints the following square pattern using two nested for-loops.

```
Enter the size: 5
# # # # #
# # # # #
# # # # #
# # # # #
# # # # #
```

Hints

The *code pattern* for printing 2D patterns using nested loops is:

```
      // Outer loop to print each of the rows
      for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
         // Inner loop to print each of the columns of a particular row
         for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
            System.out.print( ...... );   // Use print() without newline inside the
inner loop
            ......
         }
         // Print a newline after printing all the columns
         System.out.println();
      }
```

Notes

1. You should name the loop indexes `row` and `col`, NOT `i` and `j`, or `x` and `y`, or `a` and `b`, which are meaningless.
2. The `row` and `col` could start at 1 (and upto `size`), or start at `0` (and upto `size-1`). As computer counts from `0`, it is probably more efficient to start from `0`. However, since humans counts from `1`, it is easier to read if you start from `1`.

Try

Rewrite the above program using nested while-do loops.

2.2 `CheckerPattern` (nested-loop)

Write a program called **`CheckerPattern`** that prompts user for the size (a non-negative integer in `int`); and prints the following checkerboard pattern.

```
Enter the size: 7
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
```

Hints

```
      // Outer loop to print each of the rows
      for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
         // Inner loop to print each of the columns of a particular row
         for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
            if ((row % 2) == 0) {   // row 2, 4, 6, ...
               ......
            }
            System.out.print( ...... );   // Use print() without newline inside the
inner loop
            ......
         }
         // Print a newline after printing all the columns
         System.out.println();
      }
```

2.3 `MultiplicationTable` (nested-loop)

Write a program called **`MultiplicationTable`** that prompts user for the size (a positive integer in `int`); and prints the multiplication table as shown:

```
Enter the size: 10
 * |   1   2   3   4   5   6   7   8   9  10
---------------------------------------------
 1 |   1   2   3   4   5   6   7   8   9  10
 2 |   2   4   6   8  10  12  14  16  18  20
 3 |   3   6   9  12  15  18  21  24  27  30
 4 |   4   8  12  16  20  24  28  32  36  40
 5 |   5  10  15  20  25  30  35  40  45  50
 6 |   6  12  18  24  30  36  42  48  54  60
 7 |   7  14  21  28  35  42  49  56  63  70
 8 |   8  16  24  32  40  48  56  64  72  80
 9 |   9  18  27  36  45  54  63  72  81  90
10 |  10  20  30  40  50  60  70  80  90 100
```

Hints

Use `printf()` to format the output, e.g., each cell is %4d.

2.4 TriangularPattern (nested-loop)

Write 4 programs called **`TriangularPatternX`** (`X = A, B, C, D`) that prompts user for the size (a non-negative integer in `int`); and prints each of the patterns as shown:

```
Enter the size: 8

#                   # # # # # # # #       # # # # # # # #                   #
# #                 # # # # # # #           # # # # # # #                 # #
# # #               # # # # # #               # # # # # #               # # #
# # # #             # # # # #                   # # # # #             # # # #
# # # # #           # # # #                       # # # #           # # # # #
# # # # # #         # # #                           # # #         # # # # # #
# # # # # # #       # #                               # #       # # # # # # #
# # # # # # # #     #                                   #     # # # # # # # #
     (a)                 (b)                 (c)                 (d)
```

Hints

1. On the main diagonal, `row = col`. On the opposite diagonal, `row + col = size + 1`, where `row` and `col` begin from 1.
2. You need to print the *leading* blanks, in order to push the `#` to the right. The *trailing* blanks are optional, which does not affect the pattern.
3. For pattern (a), `if (row >= col) print #`. Trailing blanks are optional.
4. For pattern (b), `if (row + col <= size + 1) print #`. Trailing blanks are optional.
5. For pattern (c), `if (row >= col) print #; else print blank`. Need to print the *leading* blanks.
6. For pattern (d), `if (row + col >= size + 1) print #; else print blank`. Need to print the *leading* blanks.
7. The *coding pattern* is:

```
// Outer loop to print each of the rows
      for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
         // Inner loop to print each of the columns of a particular row
         for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
            if (......) {
                System.out.print("# ");
            } else {
                System.out.print("  ");  // Need to print the "leading" blanks
            }
         }
         // Print a newline after printing all the columns
         System.out.println();
      }
```

2.5 BoxPattern (nested-loop)

Write 4 programs called **`BoxPatternX`** (`X = A, B, C, D`) that prompts user for the size (a non-negative integer in `int`); and prints the pattern as shown:

```
Enter the size: 8

# # # # # # #    # # # # # # #    # # # # # # #    # # # # # # #    # # # # # # #
#           #      #           #        #       #       #      # #       # #
#           #        #         #          #       #   #        #   #   #   #
#           #          #       #            #       #           #     #     #
#           #            #     #              #   #  #          #   #   #   #
#           #              #   #                #       #      # #       # #
# # # # # # #    # # # # # # #    # # # # # # #    # # # # # # #    # # # # # # #
     (a)              (b)             (c)             (d)             (e)
```

Hints

1. On the main diagonal, `row = col`. On the opposite diagonal, `row + col = size + 1`, where `row` and `col` begin from 1.
2. For pattern (a), `if (row == 1 || row == size || col == 1 || col == size) print #; else print blank`. Need to print the intermediate blanks.
3. For pattern (b), `if (row == 1 || row == size || row == col) print #; else print blank`.

2.6 `HillPattern` (nested-loop)

Write 3 programs called **HillPatternX** (X = A, B, C, D) that prompts user for the size (a non-negative integer in int); and prints the pattern as shown:

```
Enter the rows: 6

         #              # # # # # # # # # # #              #              # # # # # # # # # # #
       # # #              # # # # # # # # #              # # #            # # # # #   # # # # #
     # # # # #              # # # # # # #              # # # # #          # # # #       # # # #
   # # # # # # #              # # # # #              # # # # # # #        # # #           # # #
 # # # # # # # # #              # # #              # # # # # # # # #      # #               # #
# # # # # # # # # # #              #              # # # # # # # # # # #    #                   #
        (a)                     (b)                  # # # # # # # # #      # #               # #
                                                       # # # # # # #        # # #           # # #
                                                         # # # # #          # # # #       # # # #
                                                           # # #            # # # # #   # # # # #
                                                             #              # # # # # # # # # # #
                                                            (c)                     (d)
```

Hints

For pattern (a):

```
for (int row = 1; ......) {
         // numCol = 2*numRows - 1
         for (int col = 1; ......) {
            if ((row + col >= numRows + 1) && (row >= col - numRows + 1)) {
               ......;
            } else {
               ......;
            }
         }
         ......;
      }
```

or, use 2 sequential inner loops to print the columns:

```
      for (int row = 1; row <= rows; row++) {
         for (int col = 1; col <= rows; col++) {
```

```
            if ((row + col >= rows + 1)) {
               ......
            } else {
               ......
            }
         }
         for (int col = 2; col <= rows; col++) { // skip col = 1
            if (row >= col) {
               ......
            } else {
               ......
            }
         }
         ......
      }
```

2.7 NumberPattern (nested-loop)

Write 4 programs called **NumberPatternX** (`X = A, B, C, D`) that prompts user for the size (a non-negative integer in `int`); and prints the pattern as shown:

```
Enter the size: 8

1                   1 2 3 4 5 6 7 8                   1      8 7 6 5 4 3 2 1
1 2                   1 2 3 4 5 6 7                 2 1      7 6 5 4 3 2 1
1 2 3                   1 2 3 4 5 6               3 2 1      6 5 4 3 2 1
1 2 3 4                   1 2 3 4 5             4 3 2 1      5 4 3 2 1
1 2 3 4 5                   1 2 3 4           5 4 3 2 1      4 3 2 1
1 2 3 4 5 6                   1 2 3         6 5 4 3 2 1      3 2 1
1 2 3 4 5 6 7                   1 2       7 6 5 4 3 2 1      2 1
1 2 3 4 5 6 7 8                   1     8 7 6 5 4 3 2 1      1
     (a)                 (b)                   (c)               (d)
```

2.8 PrintArray(Array)

Write a program called `PrintArray` which prompts user for the number of items in an array (a non-negative integer), and saves it in an `int` variable called `NUM_ITEMS`. It then prompts user for the values of all the items and saves them in an `int` array called `items`. The program shall then print the contents of the array in the form of `[x1, x2, ..., xn]`. For example,

```
Enter the number of items: 5
Enter the value of all items (separated by space): 3 2 5 6 9
The values are: [3, 2, 5, 6, 9]
```

Hints

```
      // Declare variables
      tinal int NUM_ITEMS;
      int[] items;  // Declare array name, to be allocated after NUM_ITEMS is known
      ......

      // Prompt for for the number of items and read the input as "int"
      ......
      NUM_ITEMS = ......
```

```
      // Allocate the array
      items = new int[NUM_ITEMS];

      // Prompt and read the items into the "int" array, if array length > 0
      if (items.length > 0) {
         ......
         for (int i = 0; i < items.length; ++i) {  // Read all items
            ......
         }
      }

      // Print array contents, need to handle first item and subsequent items
differently
      ......
      for (int i = 0; i < items.length; ++i) {
         if (i == 0) {
            // Print the first item without a leading commas
            ......
         } else {
            // Print the subsequent items with a leading commas
            ......
         }
         // or, using a one liner
         //System.out.print((i == 0) ? ...... : ......);
      }
```

2.9 PrintArrayInStars (Array)

Write a program called **printArrayInStars** which prompts user for the number of items in an array (a non-negative integer), and saves it in an `int` variable called `NUM_ITEMS`. It then prompts user for the values of all the items (non-negative integers) and saves them in an `int` array called `items`. The program shall then print the contents of the array in a graphical form, with the array index and values represented by number of stars. For examples,

```
Enter the number of items: 5
Enter the value of all items (separated by space): 7 4 3 0 7
0: *******(7)
1: ****(4)
2: ***(3)
3: (0)
4: *******(7)
```

Hints

```
      // Declare variables
      final int NUM_ITEMS;
      int[] items;  // Declare array name, to be allocated after NUM_ITEMS is known
      ......
      ......
      // Print array in "index: number of stars" using a nested-loop
      // Take note that rows are the array indexes and columns are the value in
that index
      for (int idx = 0; idx < items.length; ++idx) {  // row
         System.out.print(idx + ": ");
         // Print value as the number of stars
         for (int starNo = 1; starNo <= items[idx]; ++starNo) {  // column
            System.out.print("*");
```

```
      }
      ......
   }
   ......
```

2.10 GradesStatistics (Array)

Write a program which prompts user for the number of students in a class (a non-negative integer), and saves it in an `int` variable called `numStudents`. It then prompts user for the grade of each of the students (integer between 0 to 100) and saves them in an `int` array called `grades`. The program shall then compute and print the average (in `double` rounded to 2 decimal places) and minimum/maximum (in `int`).

```
Enter the number of students: 5
Enter the grade for student 1: 98
Enter the grade for student 2: 78
Enter the grade for student 3: 78
Enter the grade for student 4: 87
Enter the grade for student 5: 76
The average is: 83.40
The minimum is: 76
The maximum is: 98
```

2.11 Hex2Bin (Array for Table Lookup)

Write a program called **`Hex2Bin`** that prompts user for a hexadecimal string and print its equivalent binary string. The output shall look like:

```
Enter a Hexadecimal string: 1abc
The equivalent binary for hexadecimal "1abc" is: 0001 1010 1011 1100
```

Hints

1. Use an array of 16 `String`s containing binary strings corresponding to hexadecimal number `0-9A-F` (or `a-f`), as follows

```
final String[] HEX_BITS = {"0000", "0001", "0010", "0011",
                           "0100", "0101", "0110", "0111",
                           "1000", "1001", "1010", "1011",
                           "1100", "1101", "1110", "1111"};
```

2.12 Dec2Hex (Array for Table Lookup)

Write a program called **`Dec2Hex`** that prompts user for a positive decimal number, read as `int`, and print its equivalent hexadecimal string. The output shall look like:

```
Enter a decimal number: 1234
The equivalent hexadecimal number is 4D2
```

3. Magic(Special) Numbers

3.1 Amicable umbers

Two different numbers are said to be so Amicable numbers if each sum of divisors is equal to the other number. Write a Java program to check whether the given numbers are amicable or not. For example,

```
Enter 1st number: 228
Enter 2nd number: 220
The numbers are Amicable Numbers.
```

Hints

220 and 284 are Amicable Numbers.

Divisors of 220 = 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110

1+2+4+5+10+11+20+22+44+55+110 = 284

Divisors of 284 = 1, 2, 4, 71, 142

1+2+4+71+142 = 220

Try

1. Print 5 pairs of amicable numbers.

Hints

Amicable Numbers are: (220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368).

3.2 Armstrong Number

Armstrong number is a positive number if it is equal to the sum of cubes of its digits is called Armstrong number and if its sum is not equal to the number then it's not a Armstrong number. For example,

```
Enter number:145
145 is not an Armstrong Number

Enter number: 153
153 is an Armstrong Number
```

Hints

Examples: 153 is Armstrong

(1*1*1)+(5*5*5)+(3*3*3) = 153

Try

Print all Armstrong numbers below 10000

3.3 Capricorn Number

A number is called Capricorn (or Kaprekar) number whose square is divided into two parts in any conditions and parts are added, the additions of parts is equal to the number, is called Capricorn or Kaprekar number. For example,

```
Enter a number: 45
45 is a Capricorn number
Enter a number: 297
297 is a Capricorn number
Enter a number: 44
44 is not a Capricorn number
```

Hints

```
Number = 45
(45)^2   = 2025

All parts for 2025:
202 + 5 = 207 (not 45)
20 + 25 = 45
2+ 025  = 27 (not 45)
```

From the above we can see one combination is equal to number so that 45 is Capricorn or Kaprekar number.

Try

Write a Java program to generate and show all Kaprekar numbers less than 1000.

3.4 Circular Prime

A circular prime is a prime number with the property that the number generated at each intermediate step when cyclically permuting its digits will be prime. For example, 1193 is a circular prime, since 1931, 9311 and 3119 all are also prime. For example,

```
Enter a number: 137
137 is a Circular Prime
Enter a number: 44
44 is not a Circular Prime
```

Try

Write Java code to display all circular primes from 1 to 1000.

3.5 Happy Number

A happy number is a natural number in a given number base that eventually reaches 1 when iterated over the perfect digital invariant function for. Those numbers that do not end in 1 are -unhappy numbers. For example,

```
Enter a number: 31
31 is a Happy number

Enter a number: 32
32 is not a Happy number
```

Try

Print all happy numbers from 1 to 1000.

3.6. Automorphic Number

An Automorphic number is a number whose square "ends" in the same digits as the number itself. For example,

```
Enter a number: 5
5 is a Automorphic Number

Enter a number: 25
25 is a Automorphic Number

Enter a number: 2
2 is not a Automorphic Number
```

Hints

```
5*5 = 25, 6*6 = 36, 25*25 = 625

5,6,25 are automorphic numbers
```

Try

Print all automorphic numbers from 1 to 10000

3.7 Disarium Number

A number is called Disarium number if the sum of its power of the positions from left to right is equal to the number. For example,

```
Enter a number: 135
135 is a Disarium Number

Enter a number: 32
32 is not a Disarium Number
```

Hints

$1^1 + 3^2 + 5^3 = 1 + 9 + 125 = 135$

Try

Print all Disarium numbers from 1 to 10000

3.8 Magic Number

Magic number is the sum of its digits recursively are calculated till a single digit If the single digit is 1 then the number is a magic number. Magic number is very similar with Happy Number. For example,

```
Enter a number: 226
226 is a Magic Number
Enter a number: 32
32 is not a Magic Number
```

Hints

226 is said to be a magic number

2+2+6=10 sum of digits is 10 then again 1+0=1 now we get a single digit number is 1. if we single digit number will now 1 them it would not a magic number.

Try

1. A neon number is a number where the sum of digits of square of the number is equal to the number. For example if the input number is 9, its square is 9*9 = 81 and sum of the digits is 9. i.e. 9 is a neon number. For example,

```
Enter a number: 9
9 is a Neon Number
Enter a number: 8
8 is not a Neon Number
```

2. A palindromic number is a number that remains the same when its digits are reversed. For example,

```
Enter a number: 16461
16461 is a Palindromic Number

Enter a number: 1234
1234 is not a Palindromic Number
```

3. A perfect number is a positive integer that is equal to the sum of its positive divisors, excluding the number itself. For instance, 6 has divisors 1, 2 and 3, and 1 + 2 + 3 = 6, so 6 is a perfect number. For example,

```
Enter a number: 6
6 is a Perfect Number
Enter a number: 3
3 is not a Perfect Number
```

4. A number is said to be special number when the sum of factorial of its digits is equal to the number itself. Example- 145 is a Special Number as 1!+4!+5!=145. For example,

```
Enter a number: 145
145 is a Special Number
Enter a number: 23
23 is not a Special Number
```

5. A spy number is a number where the sum of its digits equals the product of its digits. For example, 1124 is a spy number, the sum of its digits is 1+1+2+4=8 and the product of its digits is 1*1*2*4=8. For example,

```
Enter a number: 1124
1124 is a Spy Number
Enter a number: 12
12 is not a Spy Number
```

6. A number is said to be an Ugly number if positive numbers whose prime factors only include 2, 3, 5. For example, 6(2×3), 8(2x2x2), 15(3×5) are ugly numbers while 14(2×7) is not ugly since it includes another prime factor 7. Note that 1 is typically treated as an ugly number. For example,

```
Enter a number: 6
6 is an Ugly Number
```

```
Enter a number: 14
14 is not an Ugly Number
```

3.9 swap() (Array & Method)

Write a method called **`swap()`**, which takes two arrays of `int` and swap their contents if they have the same length. It shall return `true` if the contents are successfully swapped. The method's signature is as follows:

```
public static boolean swap(int[] array1, int[] array2)
```

Also write a test driver to test this method.

Hints

You need to use a temporary location to swap two storage locations.

```
// Swap item1 and item2
int item1, item2, temp;
temp = item1;
item1 = item2;
item2 = item1;
// You CANNOT simply do: item1 = item2; item2 = item2;
```

3.10 reverse() (Array & Method)

Write a method called **`reverse()`**, which takes an array of `int` and reverse its contents. For example, the reverse of `[1,2,3,4]` is `[4,3,2,1]`. The method's signature is as follows:

```
public static void reverse(int[] array)
```

Take note that the array passed into the method can be modified by the method (this is called "*pass by reference*"). On the other hand, primitives passed into a method cannot be modified. This is because a clone is created and passed into the method instead of the original copy (this is called "*pass by value*").

Also write a test driver to test this method.

Hints

You might use two indexes in the loop, one moving forward and one moving backward to point to the two elements to be swapped.

```
for (int fIdx = 0, bIdx = array.length - 1; fIdx < bIdx; ++fIdx, --bIdx) {
   // Swap array[fIdx] and array[bIdx]
   // Only need to transverse half of the array elements
}
```

You need to use a temporary location to swap two storage locations.

```
// Swap item1 and item2
int item1, item2, temp;
temp = item1;
item1 = item2;
item2 = item1;
// You CANNOT simply do: item1 = item2; item2 = item2;
```

3.11 GradesStatistics (Array & Method)

Write a program called **`GradesStatistics`**, which reads in *n* grades (of `int` between `0` and `100`, inclusive) and displays the *average*, *minimum*, *maximum*, *median* and *standard deviation*. Display the floating-point values upto 2 decimal places. Your output shall look like:

```
Enter the number of students: 4
Enter the grade for student 1: 50
Enter the grade for student 2: 51
Enter the grade for student 3: 56
Enter the grade for student 4: 53
The grades are: [50, 51, 56, 53]
The average is: 52.50
The median is: 52.00
The minimum is: 50
The maximum is: 56
The standard deviation is: 2.29
```

The formula for calculating standard deviation is:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i{}^2 - \mu^2}, \text{ where } \mu \text{ is the mean}$$

Hints:

```
public class GradesStatistics {
   public static int[] grades;  // Declare an int[], to be allocated later.
                                // This array is accessible by all the methods.

   public static void main(String[] args) {
      readGrades();  // Read and save the inputs in global int[] grades
      System.out.println("The grades are: ");
      print(grades);
      System.out.println("The average is " + average(grades));
      System.out.println("The median is " + median(grades));
      System.out.println("The minimum is " + min(grades));
      System.out.println("The maximum is " + max(grades));
      System.out.println("The standard deviation is " + stdDev(grades));
   }

   // Prompt user for the number of students and allocate the global "grades" array.
   // Then, prompt user for grade, check for valid grade, and store in "grades".
   public static void readGrades() { ..... }

   // Print the given int array in the form of [x1, x2, x3,..., xn].
   public static void print(int[] array) { ..... }

   // Return the average value of the given int[]
   public static double average(int[] array) { ..... }

   // Return the median value of the given int[]
   // Median is the center element for odd-number array,
   // or average of the two center elements for even-number array.
   // Use Arrays.sort(anArray) to sort anArray in place.
   public static double median(int[] array) { ..... }

   // Return the maximum value of the given int[]
   public static int max(int[] array) {
      int max = array[0];   // Assume that max is the first element
      // From second element, if the element is more than max, set the max to this
element.
      ......
   }
```

```
   // Return the minimum value of the given int[]
   public static int min(int[] array) { ....... }

   // Return the standard deviation of the given int[]
   public static double stdDev(int[] array) { ....... }
}
```

Take note that besides `readGrade()` that relies on global variable `grades`, all the methods are *self-contained general utilities* that operate on any given array.

3.12 GradesHistogram (Array & Method)

Write a program called **`GradesHistogram`**, which reads in *n* grades (as in the previous exercise), and displays the horizontal and vertical histograms. For example:

```
 0 -  9: ***
10 - 19: ***
20 - 29:
30 - 39:
40 - 49: *
50 - 59: *****
60 - 69:
70 - 79:
80 - 89: *
90 -100: **

                              *
                              *
 *     *                      *
 *     *                      *                      *
 *     *                  *   *                  *   *
0-9  10-19 20-29 30-39 40-49 50-59 60-69 70-79 80-89 90-100
```

4. Exercises on `String` and `char` Operations

4.1 ReverseString (String & char)

Write a program called **`ReverseString`**, which prompts user for a `String`, and prints the *reverse* of the `String` by extracting and processing each character. The output shall look like:

```
Enter a String: abcdef
The reverse of the String "abcdef" is "fedcba".
```

Hints

For a `String` called `inStr`, you can use `inStr.length()` to get the *length* of the `String`; and `inStr.charAt(idx)` to retrieve the `char` at the `idx` position, where `idx` begins at 0, up to `instr.length() - 1`.

```
      // Define variables
      String inStr;        // input String
      int inStrLen;        // length of the input String
      ......

      // Prompt and read input as "String"
      System.out.print("Enter a String: ");
      inStr = in.next();   // use next() to read a String
      inStrLen = inStr.length();

      // Use inStr.charAt(index) in a loop to extract each character
      // The String's index begins at 0 from the left.
      // Process the String from the right
      for (int charIdx = inStrLen - 1; charIdx >= 0; --charIdx) {
            // charIdx = inStrLen-1, inStrLen-2, ... ,0
         ......
      }
```

4.2 CountVowelsDigits (String & char)

Write a program called **CountVowelsDigits**, which prompts the user for a `String`, counts the number of vowels (`a`, `e`, `i`, `o`, `u`, `A`, `E`, `I`, `O`, `U`) and digits (`0-9`) contained in the string, and prints the counts and the percentages (rounded to 2 decimal places). For example,

```
Enter a String: testing12345
Number of vowels: 2 (16.67%)
Number of digits: 5 (41.67%)
```

Hints

1. To check if a `char c` is a digit, you can use `boolean` expression `(c >= '0' && c <= '9')`; or use built-in `boolean` function `Character.isDigit(c)`.
2. You could use `in.next().toLowerCase()` to convert the input `String` to lowercase to reduce the number of cases.
3. To print a `%` using `printf()`, you need to use `%%`. This is because `%` is a prefix for format specifier in `printf()`, e.g., `%d` and `%f`.

4.3 PhoneKeyPad (String & char)

On you phone keypad, the alphabets are mapped to digits as follows:

`ABC(2)`, `DEF(3)`, `GHI(4)`, `JKL(5)`, `MNO(6)`, `PQRS(7)`, `TUV(8)`, `WXYZ(9)`.

Write a program called **PhoneKeyPad**, which prompts user for a `String` (case insensitive), and converts to a sequence of keypad digits. Use (a) a nested-`if`, (b) a `switch-case-default`.

Hints

1. You can use `in.next().toLowerCase()` to read a `String` and convert it to lowercase to reduce your cases.
2. In `switch-case`, you can handle multiple cases by omitting the `break` statement, e.g.,

```
switch (inChar) {
   case 'a': case 'b': case 'c':  // No break for 'a' and 'b', fall thru 'c'
      System.out.print(2); break;
```

```
   case 'd': case 'e': case 'f':
      ......
   default:
      ......
}
```

4.4 Caesar's Code (String & char)

Caesar's Code is one of the simplest encryption techniques. Each letter in the plaintext is replaced by a letter some fixed number of position (n) down the alphabet cyclically. In this exercise, we shall pick n=3. That is, `'A'` is replaced by `'D'`, `'B'` by `'E'`, `'C'` by `'F'`, ..., `'X'` by `'A'`, ..., `'Z'` by `'C'`.

Write a program called **`CaesarCode`** to cipher the Caesar's code. The program shall prompt user for a plaintext string consisting of mix-case letters only; compute the ciphertext; and print the ciphertext in uppercase. For example,

```
Enter a plaintext string: Testing
The ciphertext string is: WHVWLQJ
```

Hints

1. Use `in.next().toUpperCase()` to read an input string and convert it into uppercase to reduce the number of cases.
2. You can use a big nested-if with 26 cases (`'A'-'Z'`). But it is much better to consider `'A'` to `'W'` as one case; `'X'`, `'Y'` and `'Z'` as 3 separate cases.
3. Take note that `char 'A'` is represented as Unicode number `65` and `char 'D'` as `68`. However, `'A' + 3 gives 68`. This is because `char + int` is implicitly casted to `int + int` which returns an `int` value. To obtain a `char` value, you need to perform explicit type casting using `(char)('A' + 3)`. Try printing `('A' + 3)` with and without type casting.

4.5 Decipher Caesar's Code (String & char)

Write a program called **`DecipherCaesarCode`** to decipher the Caesar's code described in the previous exercise. The program shall prompts user for a ciphertext string consisting of mix-case letters only; compute the plaintext; and print the plaintext in uppercase. For example,

```
Enter a ciphertext string: wHVwLQJ
The plaintext string is: TESTING
```

4.6 Exchange Cipher (String & char)

This simple cipher exchanges `'A'` and `'Z'`, `'B'` and `'Y'`, `'C'` and `'X'`, and so on.
Write a program called **`ExchangeCipher`** that prompts user for a plaintext string consisting of mix-case letters only. You program shall compute the ciphertext; and print the ciphertext in uppercase. For examples,

```
Enter a plaintext string: abcXYZ
The ciphertext string is: ZYXCBA
```

Hints

1. Use `in.next().toUpperCase()` to read an input string and convert it into uppercase to reduce the number of cases.

2. You can use a big nested-if with 26 cases (`'A'-'Z'`), or use the following relationship:

```
'A' + 'Z' == 'B' + 'Y' == 'C' + 'X' == ... == plainTextChar + cipherTextChar
Hence, cipherTextChar = 'A' + 'Z' - plainTextChar
```

4.7 TestPalindromicWord and TestPalindromicPhrase (String & char)

A word that reads the same backward as forward is called a *palindrome*, e.g., "mom", "dad", "racecar", "madam", and "Radar" (case-insensitive).

Write a program called **TestPalindromicWord**, that prompts user for a word and prints ""xxx" is|is not a palindrome".

A phrase that reads the same backward as forward is also called a palindrome, e.g., "Madam, I'm Adam", "A man, a plan, a canal - Panama!" (ignoring punctuation and capitalization).

Modify your program (called **TestPalindromicPhrase**) to check for palindromic phrase. Use in.nextLine() to read a line of input.

Hints

1. ***Maintain two indexes, `forwardIndex (fIdx)` and `backwardIndex (bIdx)`, to scan the phrase forward and backward.***

```
int fIdx = 0, bIdx = strLen - 1;
while (fIdx < bIdx) {
   ......
   ++fIdx;
   --bIdx;
}
// or
for (int fIdx = 0, bIdx = strLen - 1; fIdx < bIdx; ++fIdx, --bIdx) {
   ......
}
```

2. You can check if a `char c` is a letter either using built-in `boolean` function `Character.isLetter(c)`; or `boolean` expression `(c >= 'a' && c <= 'z')`. Skip the index if it does not contain a letter.

4.8 CheckBinStr (String & char)

The binary number system uses 2 symbols, 0 and 1. Write a program called **`CheckBinStr`** to verify a binary string. The program shall prompt user for a binary string; and decide if the input string is a valid binary string. For example,

```
Enter a binary string: 10101100
"10101100" is a binary string

Enter a binary string: 10120000
"10120000" is NOT a binary string
```

Hints

Use the following coding pattern which involves a `boolean` flag to check the input string.

```
// Declare variables
      String inStr;     // The input string
      int inStrLen;     // The length of the input string
      char inChar;      // Each char of the input string
      boolean isValid;  // "is" or "is not" a valid binary string?
      ......

      isValid = true;  // Assume that the input is valid, unless our check fails
      for (......) {
         inChar = ......;
         if (!(inChar == '0' || inChar == '1')) {
            isValid = false;
            break;   // break the loop upon first error, no need to continue for
more errors
                     // If this is not encountered, isValid remains true after the
loop.
         }
      }
      if (isValid) {
         System.out.println(......);
      } else {
         System.out.println(......);
      }
      // or using one liner
      //System.out.println(isValid ? ... : ...);
```

4.9 CheckHexStr (String & char)

The hexadecimal (hex) number system uses 16 symbols, `0-9` and `A-F` (or `a-f`). Write a program to verify a hex string. The program shall prompt user for a hex string; and decide if the input string is a valid hex string. For examples,

```
Enter a hex string: 123aBc
"123aBc" is a hex string

Enter a hex string: 123aBcx
"123aBcx" is NOT a hex string
```

Hints

```
if (!((inChar >= '0' && inChar <= '9')
      || (inChar >= 'A' && inChar <= 'F')
      || (inChar >= 'a' && inChar <= 'f'))) { // Use positive logic and then reverse
   ......
}
```

4.10 Bin2Dec (String & char)

Write a program called `Bin2Dec` to convert an input binary string into its equivalent decimal number. Your output shall look like:

```
Enter a Binary string: 1011
The equivalent decimal number for binary "1011" is: 11

Enter a Binary string: 1234
error: invalid binary string "1234"
```

4.11 Hex2Dec (String & char)

Write a program called **Hex2Dec** to convert an input hexadecimal string into its equivalent decimal number. Your output shall look like:

```
Enter a Hexadecimal string: 1a
The equivalent decimal number for hexadecimal "1a" is: 26

Enter a Hexadecimal string: 1y3
error: invalid hexadecimal string "1y3"
```

4.12 Oct2Dec (String & char)

Write a program called **Oct2Dec** to convert an input Octal string into its equivalent decimal number. For example,

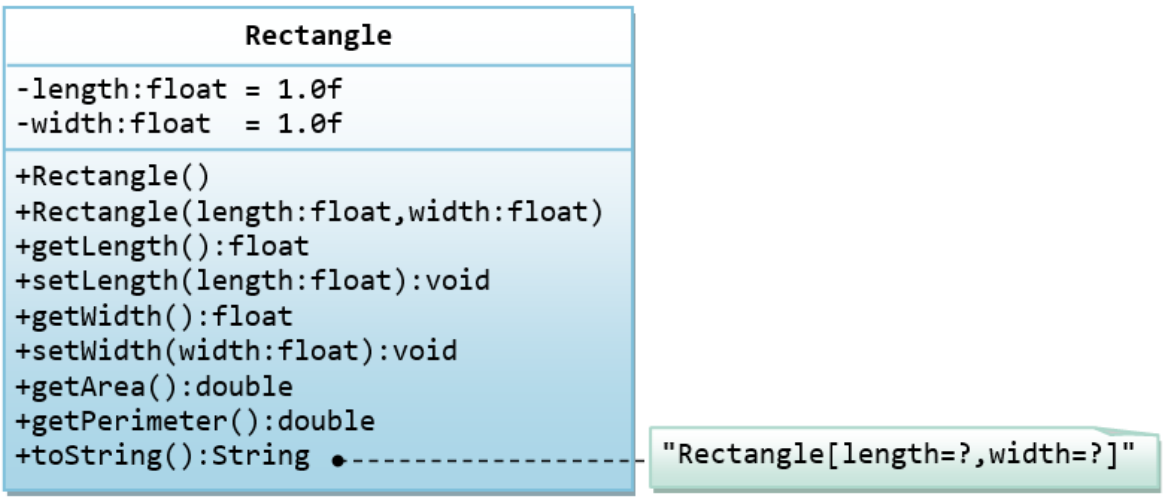
```
Ente an Octal string: 147
The equivalent decimal number "147" is: 103
```

5. Exercises on Classes and Objects

5.1 The Rectangle Class

A class called `Rectangle`, which models a rectangle with a length and a width (in `float`), is designed as shown in the following class diagram. Write the `Rectangle` class.

Hints:



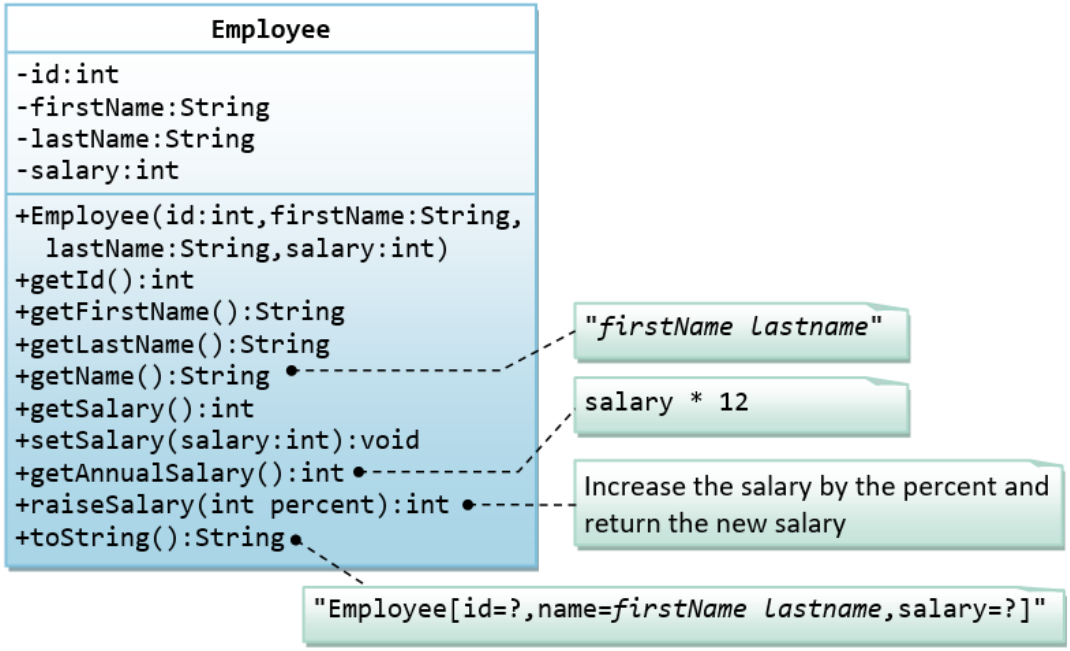
The expected output is:

```
Rectangle[length=1.2,width=3.4]
Rectangle[length=1.0,width=1.0]
Rectangle[length=5.6,width=7.8]
length is: 5.6
width is: 7.8
area is: 43.68
perimeter is: 26.80
```

5.2 The `Employee` Class

A class called `Employee`, which models an employee with an ID, name and salary, is designed as shown in the following class diagram. The method `raiseSalary(percent)` increases the salary by the given percentage. Write the `Employee` class.

Hints:



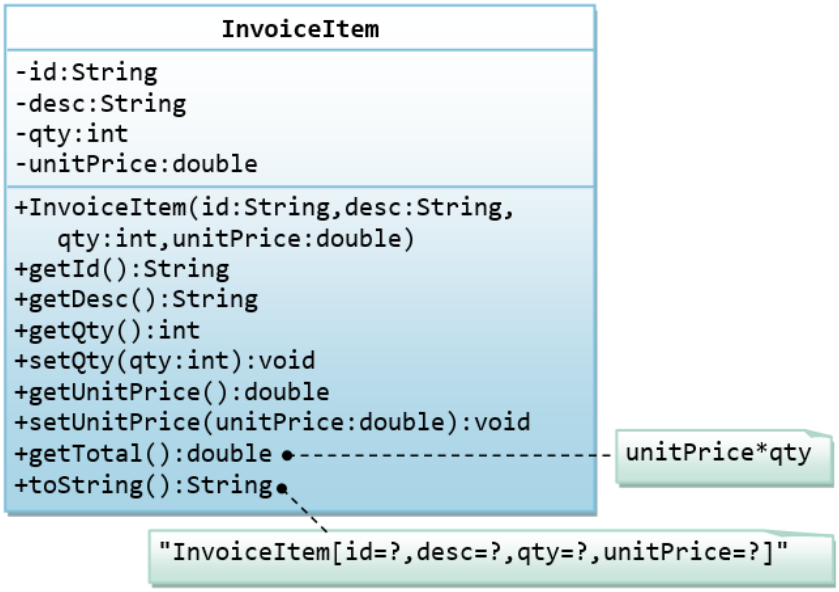
The expected out is:

```
Employee[id=8,name=Peter Tan,salary=2500]
Employee[id=8,name=Peter Tan,salary=999]
id is: 8
firstname is: Peter
lastname is: Tan
salary is: 999
name is: Peter Tan
annual salary is: 11988
1098
Employee[id=8,name=Peter Tan,salary=1098]
```

5.3 The `InvoiceItem` Class

A class called `InvoiceItem`, which models an item of an invoice, with ID, description, quantity and unit price, is designed as shown in the following class diagram. Write the `InvoiceItem` class.

Hints:



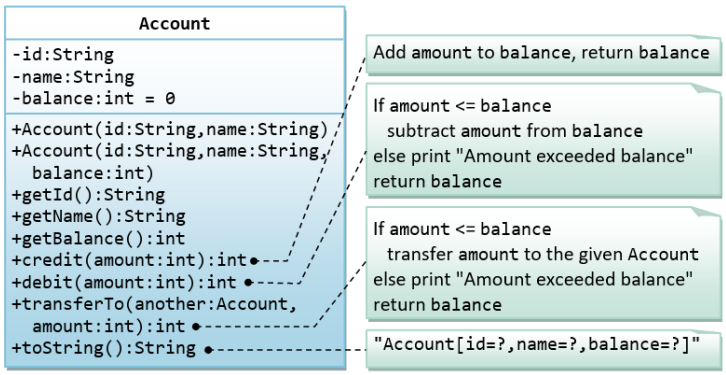
The expected output is:

```
InvoiceItem[id=A101,desc=Pen Red,qty=888,unitPrice=0.08]
InvoiceItem[id=A101,desc=Pen Red,qty=999,unitPrice=0.99]
id is: A101
desc is: Pen Red
qty is: 999
unitPrice is: 0.99
The total is: 989.01
```

5.4 The `Account` Class

A class called `Account`, which models a bank account of a customer, is designed as shown in the following class diagram. The methods `credit(amount)` and `debit(amount)` add or subtract the given `amount` to the `balance`. The method `transferTo(anotherAccount, amount)` transfers the given `amount` from this `Account` to the given `anotherAccount`. Write the `Account` class.

Hints:



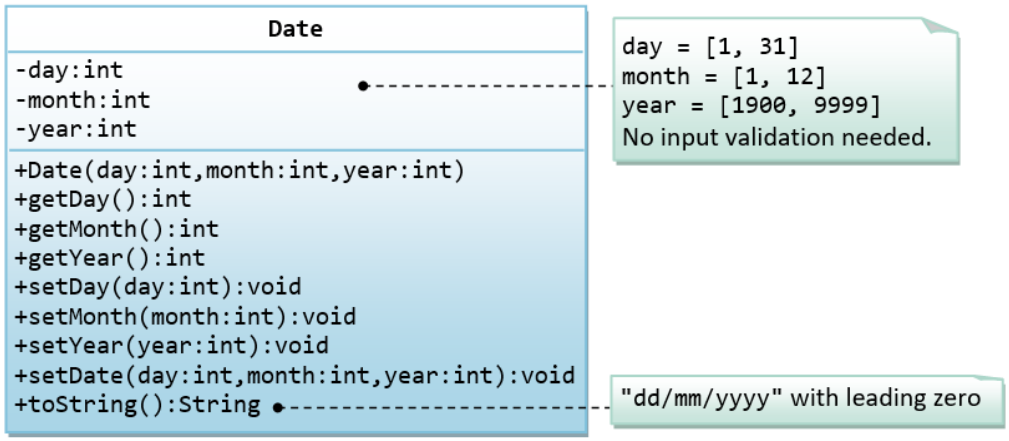
The expected output is:

```
Account[id=A101,name=Tan Ah Teck, balance=88]
Account[id=A102,name=Kumar, balance=0]
ID: A101
Name: Tan Ah Teck
Balance: 88
Account[id=A101,name=Tan Ah Teck, balance=188]
Account[id=A101,name=Tan Ah Teck, balance=138]
Amount exceeded balance
Account[id=A101,name=Tan Ah Teck, balance=138]
Account[id=A101,name=Tan Ah Teck, balance=38]
Account[id=A102,name=Kumar, balance=100]
```

5.5 The `Date` Class

A class called `Date`, which models a calendar date, is designed as shown in the following class diagram. Write the `Date` class.

Hints:



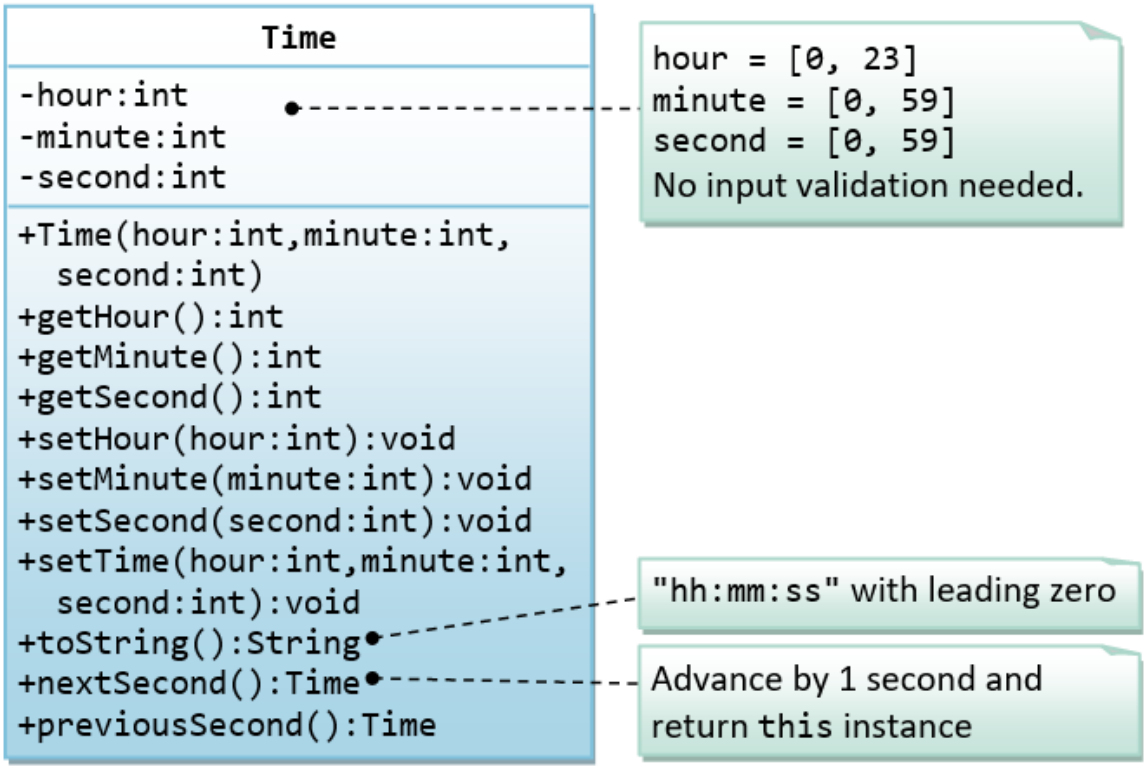
The expected output is:

```
01/02/2014
09/12/2099
Month: 12
Day: 9
Year: 2099
03/04/2016
```

5.6 Ex: The `Time` Class

A class called `Time`, which models a time instance, is designed as shown in the following class diagram. The methods `nextSecond()` and `previousSecond()` shall advance or rewind this instance by one second, and return this instance, so as to support chaining operation such as `t1.nextSecond().nextSecond()`. Write the `Time` class.

Hints:



The expected output is:

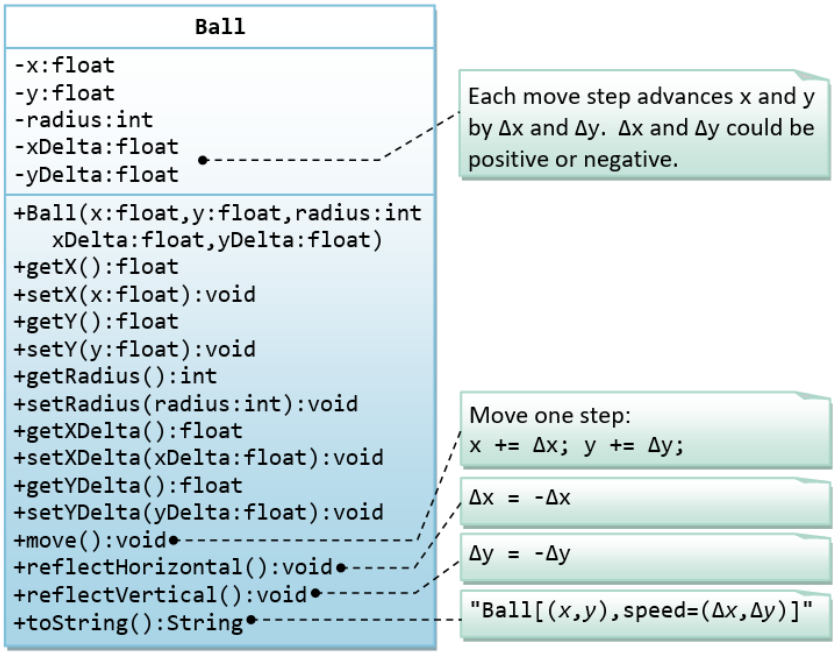
```
01:02:03
04:05:06
Hour: 4
Minute: 5
Second: 6
23:59:58
23:59:59
00:00:01
00:00:00
23:59:58
```

5.7 The `Ball` Class

A class called `Ball`, which models a bouncing ball, is designed as shown in the following class diagram. It contains its radius, x and y position. Each move-step advances the x and y by delta-x and delta-y, respectively. delta-x and delta-y could be positive or negative.

The `reflectHorizontal()` and `reflectVertical()` methods could be used to bounce the ball off the walls. Write the `Ball` class. Study the test driver on how the ball bounces.

Hints:



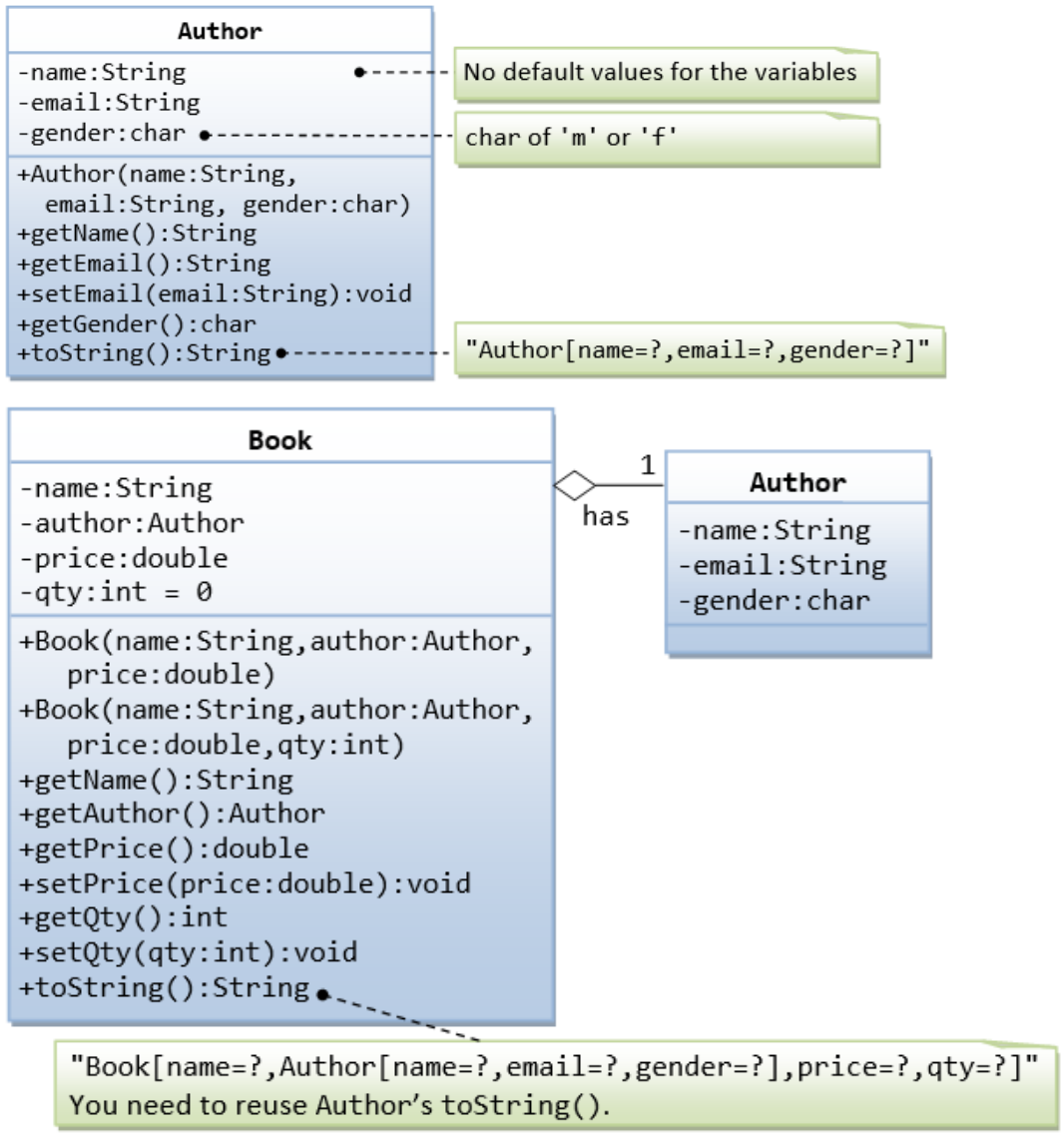
The expected output is:

```
Ball[(1.1,2.2),speed=(3.3,4.4)]
Ball[(80.0,35.0),speed=(4.0,6.0)]
x is: 80.0
y is: 35.0
radius is: 5
xDelta is: 4.0
yDelta is: 6.0
Ball[(84.0,41.0),speed=(4.0,6.0)]
Ball[(88.0,47.0),speed=(4.0,6.0)]
Ball[(92.0,41.0),speed=(4.0,-6.0)]
Ball[(96.0,35.0),speed=(4.0,-6.0)]
Ball[(92.0,29.0),speed=(-4.0,-6.0)]
Ball[(88.0,23.0),speed=(-4.0,-6.0)]
Ball[(84.0,17.0),speed=(-4.0,-6.0)]
Ball[(80.0,11.0),speed=(-4.0,-6.0)]
Ball[(76.0,5.0),speed=(-4.0,-6.0)]
Ball[(72.0,-1.0),speed=(-4.0,-6.0)]
Ball[(68.0,5.0),speed=(-4.0,6.0)]
Ball[(64.0,11.0),speed=(-4.0,6.0)]
Ball[(60.0,17.0),speed=(-4.0,6.0)]
Ball[(56.0,23.0),speed=(-4.0,6.0)]
Ball[(52.0,29.0),speed=(-4.0,6.0)]
```

6. Exercises on Composition

6.1 The Author and Book Classes

This first exercise shall lead you through all the concepts involved in OOP Composition.

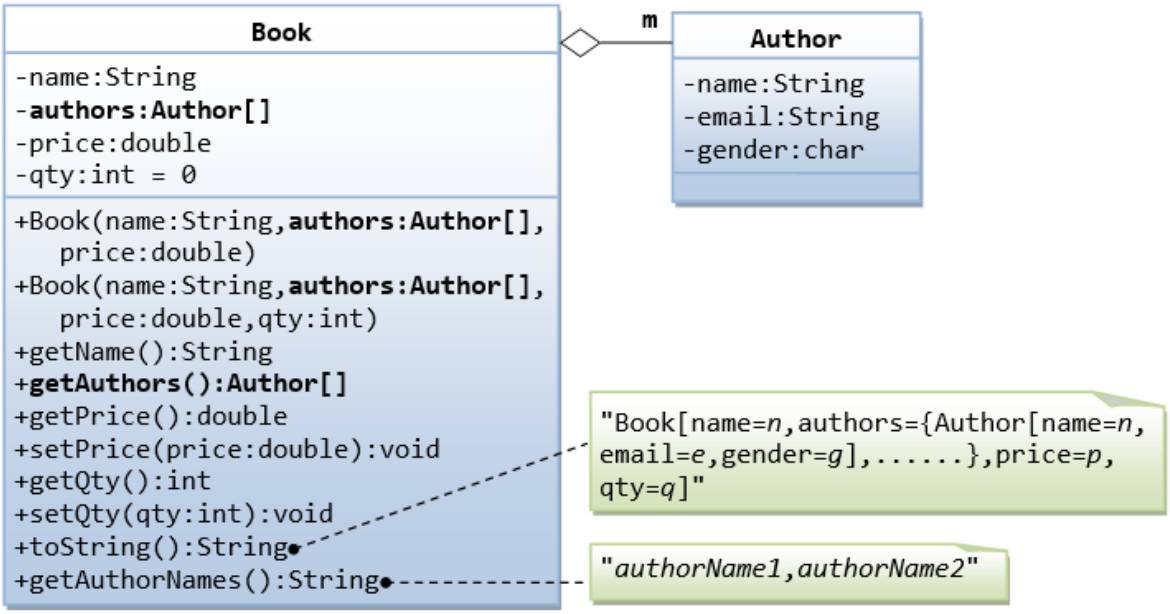


TRY

1. Printing the `name` and `email` of the author from a `Book` instance.
   (Hint: `aBook.getAuthor().getName()`, `aBook.getAuthor().getEmail()`).
2. Introduce new methods called `getAuthorName()`, `getAuthorEmail()`, `getAuthorGender()` in the `Book` class to return the `name`, `email` and `gender` of the author of the book. For example,

```
public String getAuthorName() {
   return author.getName();  // cannot use author.name as name is private in Author class
}
```

6.2 The Author and Book Classes - An Array of Objects as an Instance Variable



In the earlier exercise, a book is written by one and only one author. In reality, a book can be written by one or more author. Modify the Book class to support one or more authors by changing the instance variable `authors` to an `Author` array.

Notes:

- The constructors take an array of `Author` (i.e., `Author[]`), instead of an `Author` instance. In this design, once a Book instance is constructor, you cannot add or remove author.
- The `toString()` method shall return **`"Book[name=?,authors={Author[name=?,email=?,gender=?],.....},price=?,qty=?]".`**

You are required to:

1. Write the code for the Book class. You shall re-use the `Author` class written earlier.
2. Write a test driver (called `TestBook`) to test the Book class.

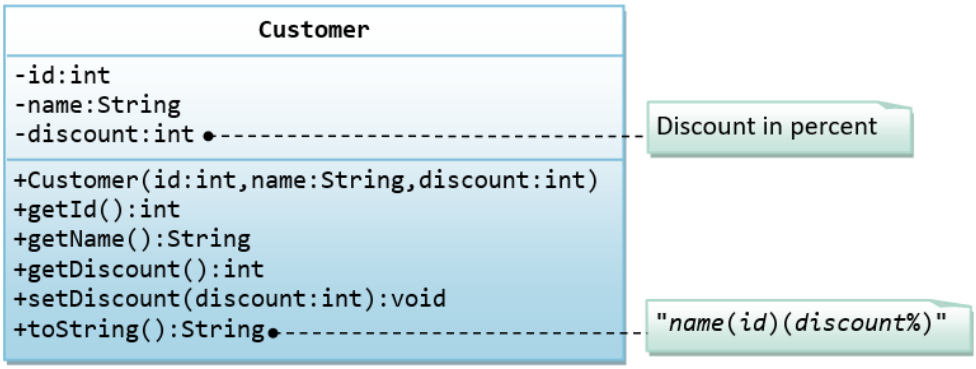
Hints

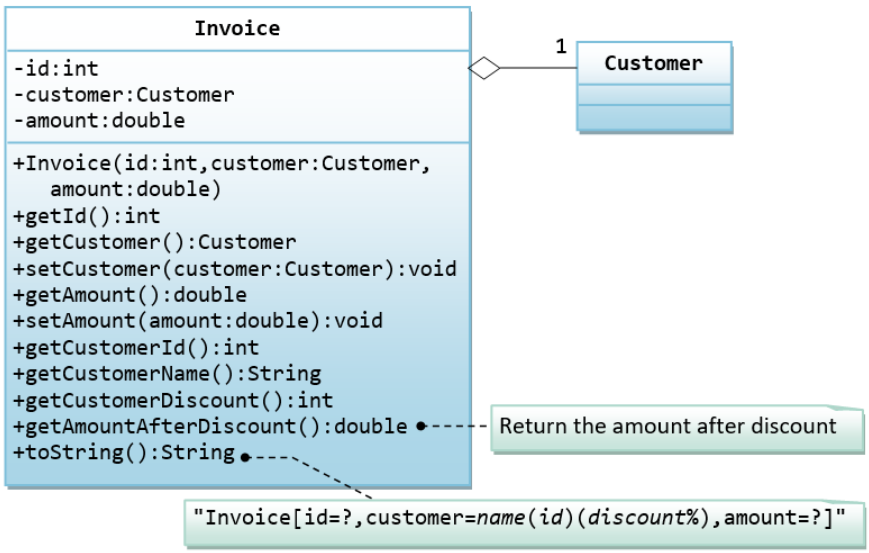
```
// Declare and allocate an array of Authors
Author[] authors = new Author[2];
authors[0] = new Author("Tan Ah Teck", "AhTeck@somewhere.com", 'm');
authors[1] = new Author("Paul Tan", "Paul@nowhere.com", 'm');

// Declare and allocate a Book instance
Book javaDummy = new Book("Java for Dummy", authors, 19.99, 99);
System.out.println(javaDummy);  // toString()
```

6.3 The `Customer` and `Invoice` classes

A class called `Customer`, which models a customer in a transaction, is designed as shown in the class diagram. A class called `Invoice`, which models an invoice for a particular customer and composes an instance of `Customer` as its instance variable, is also shown. Write the `Customer` and `Invoice` classes.

Hints:

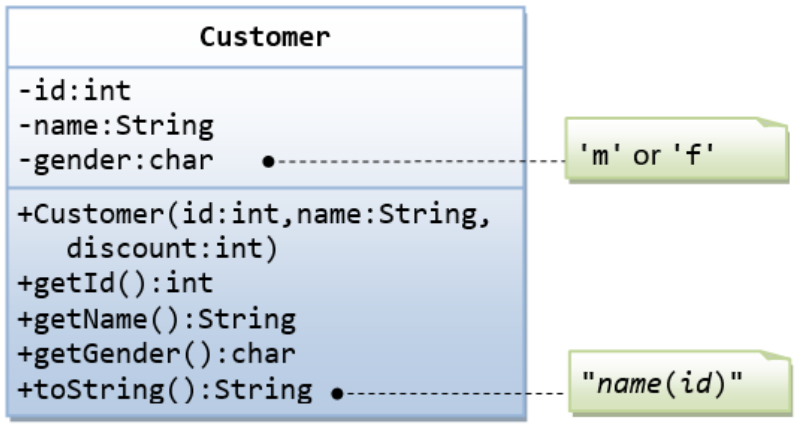




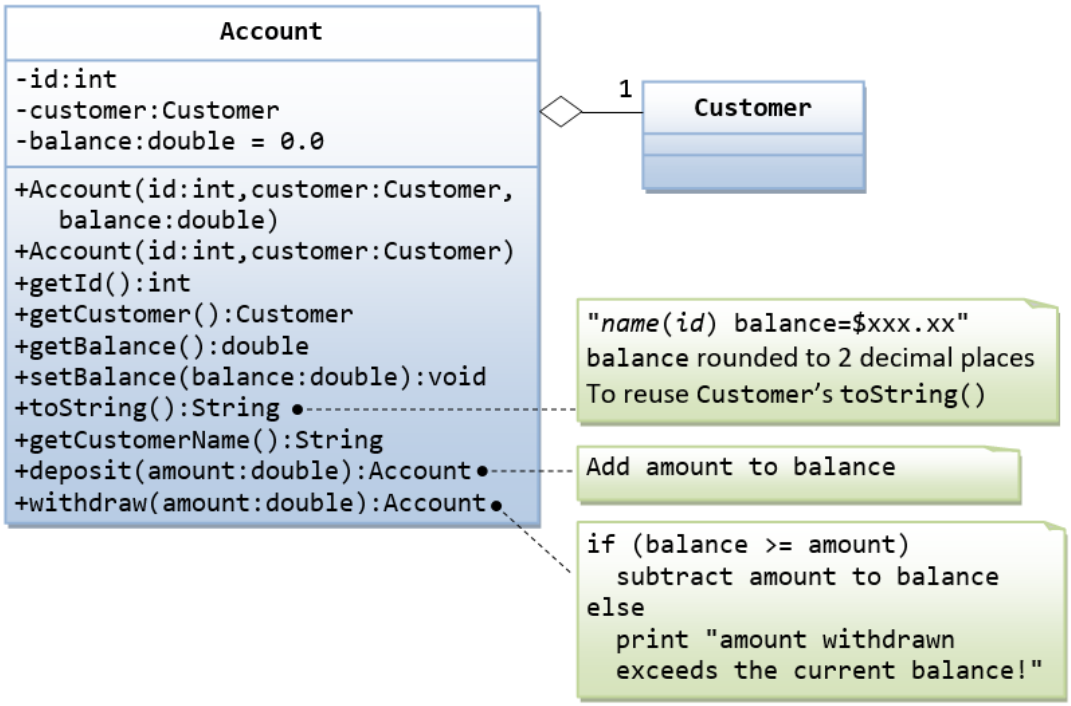
The expected output is:

```
Tan Ah Teck(88)(10%)
Tan Ah Teck(88)(8%)
id is: 88
name is: Tan Ah Teck
discount is: 8
Invoice[id=101,customer=Tan Ah Teck(88)(8%),amount=888.8]
Invoice[id=101,customer=Tan Ah Teck(88)(8%),amount=999.9]
id is: 101
customer is: Tan Ah Teck(88)(8%)
amount is: 999.9
customer's id is: 88
customer's name is: Tan Ah Teck
customer's discount is: 8
amount after discount is: 919.91
```

6.4 Ex: The `Customer` and `Account` classes



The `Customer` class models a customer is design as shown in the class diagram. Write the codes for the `Customer` class and a test driver to test all the `public` methods.



The `Account` class models a bank account, design as shown in the class diagram, composes a `Customer` instance (written earlier) as its member. Write the codes for the `Account` class and a test driver to test all the `public` methods.

It contains:

- A method called `distance(int x, int y)` that returns the distance from *this* point to another point at the given `(x, y)` coordinates, e.g.,

```
MyPoint p1 = new MyPoint(3, 4);
System.out.println(p1.distance(5, 6));
```

- An overloaded `distance(MyPoint another)` that returns the distance from *this* point to the given `MyPoint` instance (called `another`), e.g.,

```
MyPoint p1 = new MyPoint(3, 4);
MyPoint p2 = new MyPoint(5, 6);
System.out.println(p1.distance(p2));
```

- Another overloaded `distance()` method that returns the distance from `this` point to the origin `(0,0)`, e.g.

```
MyPoint p1 = new MyPoint(3, 4);
System.out.println(p1.distance());
```

You are required to:

1. Write the code for the class `MyPoint`. Also write a test program (called `TestMyPoint`) to test all the methods defined in the class,

Hints:

```
// Overloading method distance()
// This version takes two ints as arguments
public double distance(int x, int y) {
   int xDiff = this.x - x;
   int yDiff = ......
   return Math.sqrt(xDiff*xDiff + yDiff*yDiff);
}

// This version takes a MyPoint instance as argument
public double distance(MyPoint another) {
   int xDiff = this.x - another.x;
   .......
}
```

Try

Write a program that allocates `10` points in an array of `MyPoint`, and initializes to `(1, 1)`, `(2, 2)`, ... `(10, 10)`.

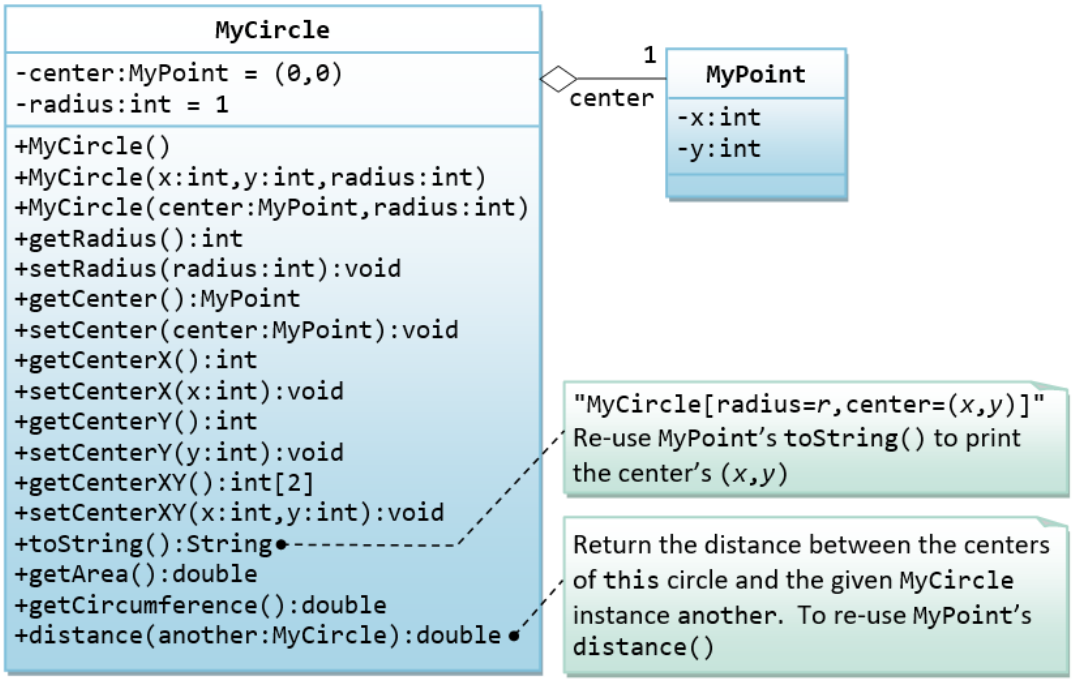
Hints

You need to allocate the array, as well as each of the 10 `MyPoint` instances. In other words, you need to issue 11 `new`, 1 for the array and 10 for the `MyPoint` instances.

```
MyPoint[] points = new MyPoint[10];  // Declare and allocate an array of MyPoint
for (int i = 0; i < points.length; i++) {
   points[i] = new MyPoint(...);    // Allocate each of MyPoint instances
}
// use a loop to print all the points
```

6.5 Ex: The `MyCircle` and `MyPoint` Classes

A class called `MyCircle`, which models a circle with a center and a radius, is designed as shown in the class diagram. The `MyCircle` class uses a `MyPoint` instance (written in the earlier exercise) as its center.

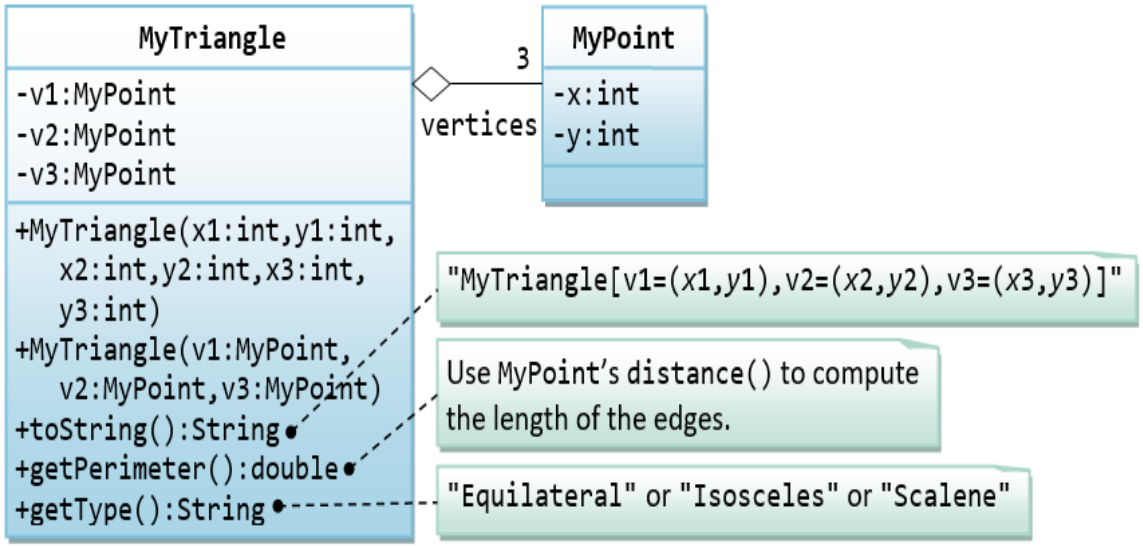


Hints:

```
// Constructors
public MyCircle(int x, int y, int radius) {
   // Need to construct an instance of MyPoint for the variable center
   center = new MyPoint(x, y);
   this.radius = radius;
}
public MyCircle(MyPoint center, int radius) {
   // An instance of MyPoint already constructed by caller; simply assign.
   this.center = center;
   ......
}
public MyCircle() {
   center = new MyPoint(.....);  // construct MyPoint instance
   this.radius = ......
}

// Returns the x-coordinate of the center of this MyCircle
public int getCenterX() {
   return center.getX();   // cannot use center.x and x is private in MyPoint
}

// Returns the distance of the center for this MyCircle and another MyCircle
public double distance(MyCircle another) {
   return center.distance(another.center); // use distance() of MyPoint
}
```

6.6 Ex: The `MyTriangle` and `MyPoint` Classes

A class called `MyTriangle`, which models a triangle with 3 vertices, is designed as shown in the class diagram. The `MyTriangle` class uses three `MyPoint` instances (created in the earlier exercise) as the three vertices.



It contains:

- Three `private` instance variables `v1`, `v2`, `v3` (instances of `MyPoint`), for the three vertices.
- A constructor that constructs a `MyTriangle` with three set of coordinates, `v1=(x1, y1)`, `v2=(x2, y2)`, `v3=(x3, y3)`.
- An overloaded constructor that constructs a `MyTriangle` given three instances of `MyPoint`.
- A `toString()` method that returns a string description of the instance in the format "`MyTriangle[v1=(x1,y1),v2=(x2,y2),v3=(x3,y3)]`".
- A `getPerimeter()` method that returns the length of the perimeter in double. You should use the `distance()` method of `MyPoint` to compute the perimeter.
- A method `printType()`, which prints "`equilateral`" if all the three sides are equal, "`isosceles`" if any two of the three sides are equal, or "`scalene`" if the three sides are different.

Write the `MyTriangle` class. Also write a test driver (called `TestMyTriangle`) to test all the `public` methods defined in the class.

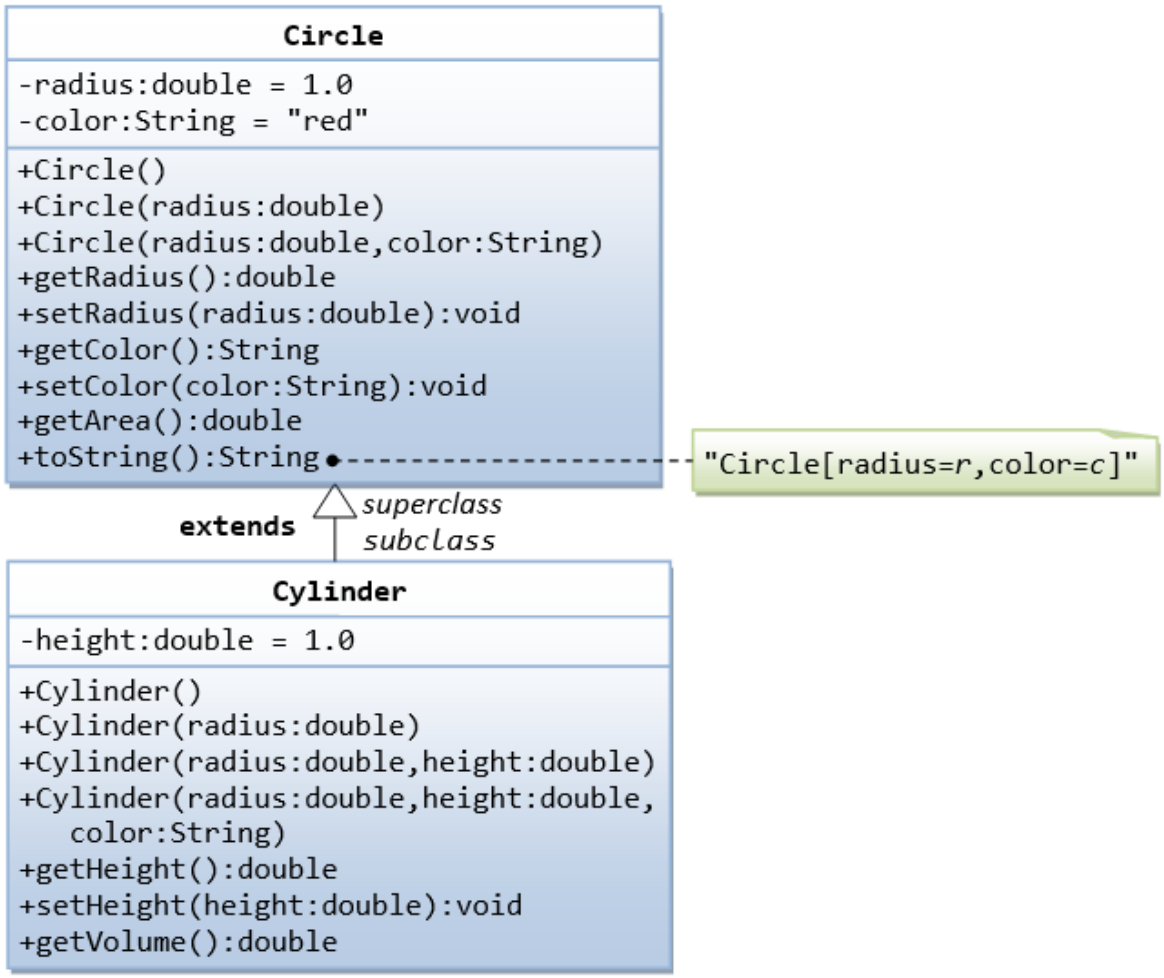
Try

Design a `MyRectangle` class which is composed of two `MyPoint` instances as its *top-left* and *bottom-right* corners. Draw the class diagrams, write the codes, and write the test drivers.

7. Exercises on Inheritance

7.1 An Introduction to OOP Inheritance: The `Circle` and `Cylinder` Classes

This exercise shall guide you through the important concepts in inheritance.



In this exercise, a subclass called `Cylinder` is derived from the superclass `Circle` as shown in the class diagram (where an an arrow pointing up from the subclass to its superclass). Study how the subclass `Cylinder` invokes the superclass' constructors (via `super()` and `super(radius)`) and inherits the variables and methods from the superclass `Circle`.

You can reuse the `Circle` class that you have created in the previous exercise. Make sure that you keep "`Circle.class`" in the same directory.

```
public class Cylinder extends Circle {  // Save as "Cylinder.java"
   private double height;  // private variable

   // Constructor with default color, radius and height
   public Cylinder() {
      super();        // call superclass no-arg constructor Circle()
      height = 1.0;
   }
   // Constructor with default radius, color but given height
   public Cylinder(double height) {
```

```
      super();        // call superclass no-arg constructor Circle()
      this.height = height;
   }
   // Constructor with default color, but given radius, height
   public Cylinder(double radius, double height) {
      super(radius);  // call superclass constructor Circle(r)
      this.height = height;
   }

   // A public method for retrieving the height
   public double getHeight() {
      return height;
   }

   // A public method for computing the volume of cylinder
   //  use superclass method getArea() to get the base area
   public double getVolume() {
      return getArea()*height;
   }
}
```

Method Overriding and "Super": The subclass `Cylinder` inherits `getArea()` method from its superclass Circle. Try *overriding* the `getArea()` method in the subclass `Cylinder` to compute the surface area (=2π×radius×height + 2×base-area) of the cylinder instead of base area. That is, if `getArea()` is called by a `Circle` instance, it returns the area. If `getArea()` is called by a `Cylinder` instance, it returns the surface area of the cylinder.

If you override the `getArea()` in the subclass `Cylinder`, the `getVolume()` no longer works. This is because the `getVolume()` uses the *overridden* `getArea()` method found in the same class. (Java runtime will search the superclass only if it cannot locate the method in this class). Fix the `getVolume()`.

Hints: After overridding the `getArea()` in subclass `Cylinder`, you can choose to invoke the `getArea()` of the superclass `Circle` by calling `super.getArea()`.

Try

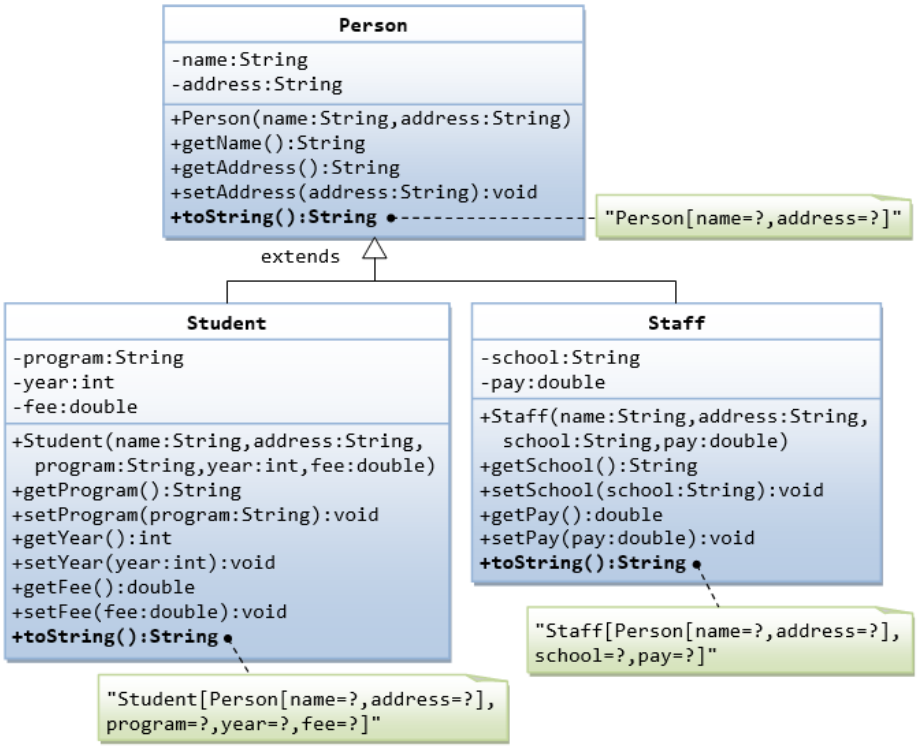
Provide a `toString()` method to the `Cylinder` class, which overrides the `toString()` inherited from the superclass `Circle`, e.g.,

```
@Override
public String toString() {      // in Cylinder class
   return "Cylinder: subclass of " + super.toString()  // use Circle's toString()
          + " height=" + height;
}
```

Try out the `toString()` method in `TestCylinder`.

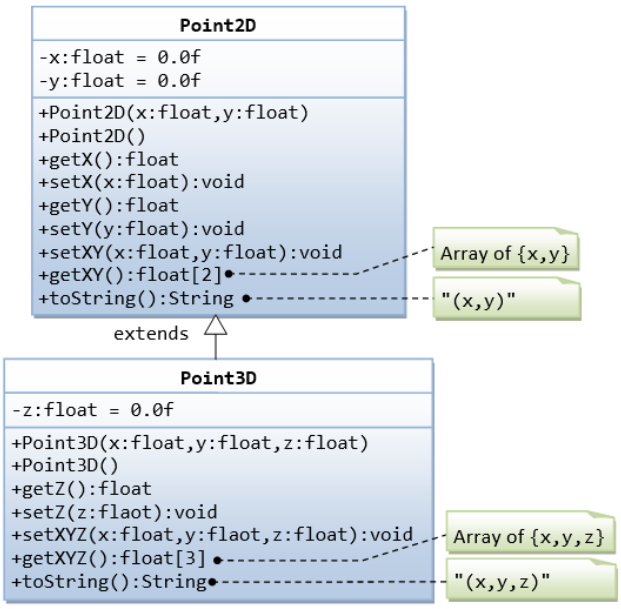
7.2 Superclass `Person` and its subclasses

Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation `@Override`.



7.3 `Point2D` and `Point3D`

Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation `@Override`.



Hints:

1. You cannot assign floating-point literal say `1.1` (which is a `double`) to a `float` variable, you need to add a suffix f, e.g. `0.0f`, `1.1f`.
2. The instance variables `x` and `y` are `private` in `Point2D` and cannot be accessed directly in the subclass `Point3D`. You need to access via the `public` getters and setters. For example,

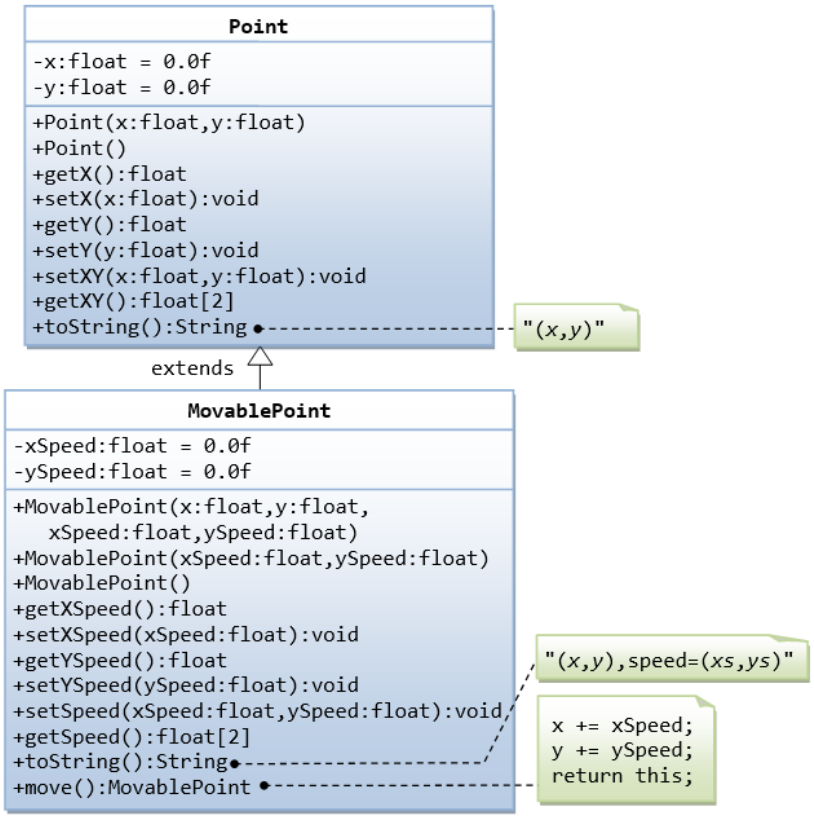
```
public void setXYZ(float x, float y, float z) {
       setX(x);     // or super.setX(x), use setter in superclass
       setY(y);
       this.z = z;
    }
```

3. The method `getXY()` shall return a `float` array:

```
public float[] getXY() {
       float[] result = new float[2];  // construct an array of 2 elements
       result[0] = ...
       result[1] = ...
       return result;  // return the array
    }
```

7.4 `Point` and `MovablePoint`

Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation `@Override`.



Hints

1. You cannot assign floating-point literal say `1.1` (which is a `double`) to a `float` variable, you need to add a suffix f, e.g. `0.0f`, `1.1f`.
2. The instance variables `x` and `y` are `private` in `Point` and cannot be accessed directly in the subclass `MovablePoint`. You need to access via the `public` getters and setters. For example, you cannot write `x += xSpeed`, you need to write `setX(getX() + xSpeed)`.

7.5 Superclass `Shape` and its subclasses `Circle`, `Rectangle` and `Square`

- Write a superclass called Shape (as shown in the class diagram)
- Write a test program to test all the methods defined in Shape.
- Write two subclasses of Shape called `Circle` and `Rectangle`, as shown in the class diagram.
- Write a class called `Square`, as a subclass of `Rectangle`. Convince yourself that `Square` can be modeled as a subclass of `Rectangle`. `Square` has no instance variable, but inherits the instance variables width and length from its superclass Rectangle.

Provide the appropriate constructors (as shown in the class diagram).

Hints:

```
public Square(double side) {
   super(side, side);  // Call superclass Rectangle(double, double)
}
```

- Override the `toString()` method to return "`A Square with side=xxx, which is a subclass of yyy`", where `yyy` is the output of the `toString()` method from the superclass.
- Do you need to override the `getArea()` and `getPerimeter()`? Try them out.
- Override the `setLength()` and `setWidth()` to change both the `width` and `length`, so as to maintain the square geometry.

Exercises on Composition vs Inheritance

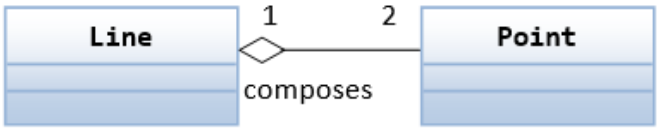
7.6 The `Point` and `Line` Classes

They are two ways to reuse a class in your applications: *composition* and *inheritance*.

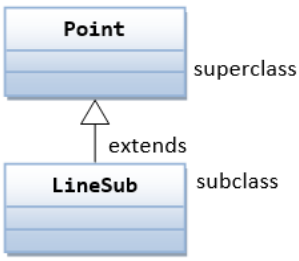
Let us begin with *composition* with the statement "a line composes of two points".

Complete the definition of the following two classes: `Point` and `Line`. The class `Line` composes 2 instances of class `Point`, representing the beginning and ending points of the line. Also write test classes for `Point` and `Line` (says `TestPoint` and `TestLine`).

The class diagram for *composition* is as follows (where a diamond-hollow-head arrow pointing to its constituents):



Instead of *composition*, we can design a `Line` class using `inheritance`. Instead of "a line composes of two points", we can say that "a line is a point extended by another point", as shown in the following class diagram:

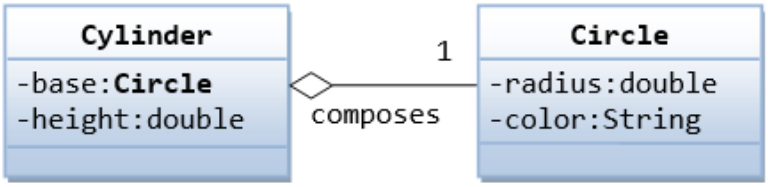


Let's re-design the `Line` class (called `LineSub`) as a subclass of class `Point`. `LineSub` inherits the starting point from its superclass `Point`, and adds an ending point. Complete the class definition. Write a testing class called `TestLineSub` to test `LineSub`.

Try

There are two approaches that you can design a line, `composition` or `inheritance`. "A line composes two points" or "A line is a point extended with another point"".
Compare the `Line` and `LineSub` designs: `Line` uses *composition* and `LineSub` uses *inheritance*. Which design is better?

7.7 The `Circle` and `Cylinder` Classes Using Composition



Try

rewriting the `Circle-Cylinder` of the previous exercise using *composition* (as shown in the class diagram) instead of *inheritance*. That is, "a cylinder is composed of a base circle and a height".

```
public class Cylinder {
   private Circle base;   // Base circle, an instance of Circle class
   private double height;

   // Constructor with default color, radius and height
   public Cylinder() {
      base = new Circle(); // Call the constructor to construct the Circle
      height = 1.0;
   }
   ......
}
```

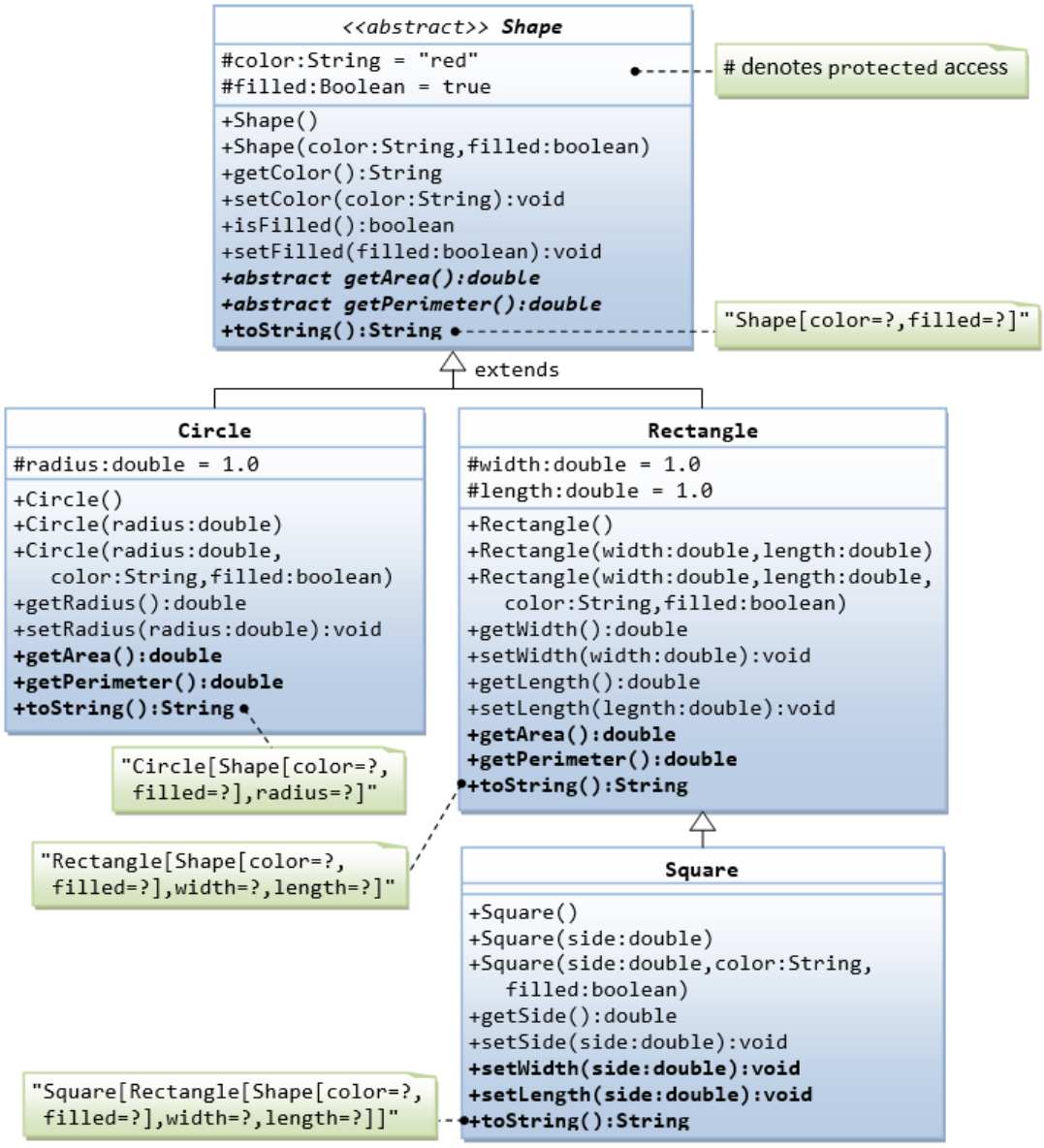
Which design (inheritance or composition) is better?

8. Exercises on Polymorphism, Abstract Classes and Interfaces

8.1 Ex: Abstract Superclass Shape and Its Concrete Subclasses

Rewrite the superclass `Shape` and its subclasses `Circle`, `Rectangle` and `Square`, as shown in the class diagram.

`Shape` is an `abstract` class containing 2 `abstract` methods: `getArea()` and `getPerimeter()`, where its concrete subclasses must provide its implementation. All instance variables shall have `protected` access, i.e., accessible by its subclasses and classes in the same package. Mark all the overridden methods with annotation `@Override`.



In this exercise, Shape shall be defined as an `abstract` class, which contains:

- Two protected instance variables color(String) and filled(boolean). The `protected` variables can be accessed by its subclasses and classes in the same package. They are denoted with a `'#'` sign in the class diagram.
- Getter and setter for all the instance variables, and `toString()`.
- Two `abstract` methods `getArea()` and `getPerimeter()` (shown in italics in the class diagram).

The subclasses `Circle` and `Rectangle` shall *override* the `abstract` methods `getArea()` and `getPerimeter()` and provide the proper implementation. They also *override* the `toString()`.

Write a test class to test these statements involving polymorphism and explain the outputs. Some statements may trigger compilation errors. Explain the errors, if any.

```
Shape s1 = new Circle(5.5, "red", false);  // Upcast Circle to Shape
System.out.println(s1);                    // which version?
System.out.println(s1.getArea());          // which version?
System.out.println(s1.getPerimeter());     // which version?
```

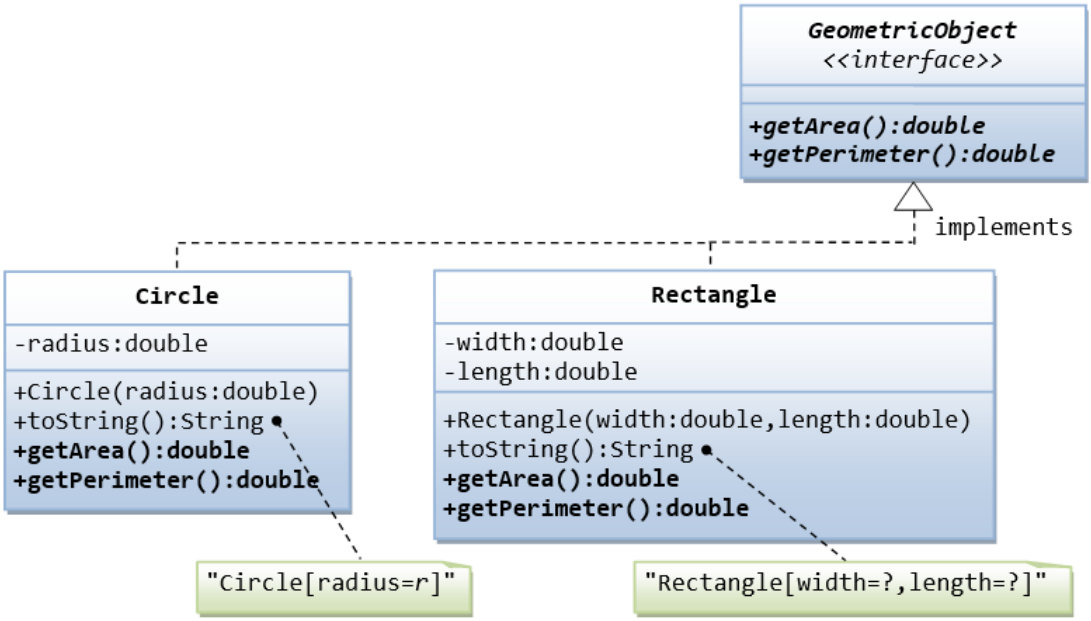
```
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle c1 = (Circle)s1;                    // Downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());

Shape s2 = new Shape();

Shape s3 = new Rectangle(1.0, 2.0, "red", false);   // Upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3;   // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());

Shape s4 = new Square(6.6);     // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());

// Take note that we downcast Shape s4 to Rectangle,
//  which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

Try

Explain the usage of the `abstract` method and `abstract` class?

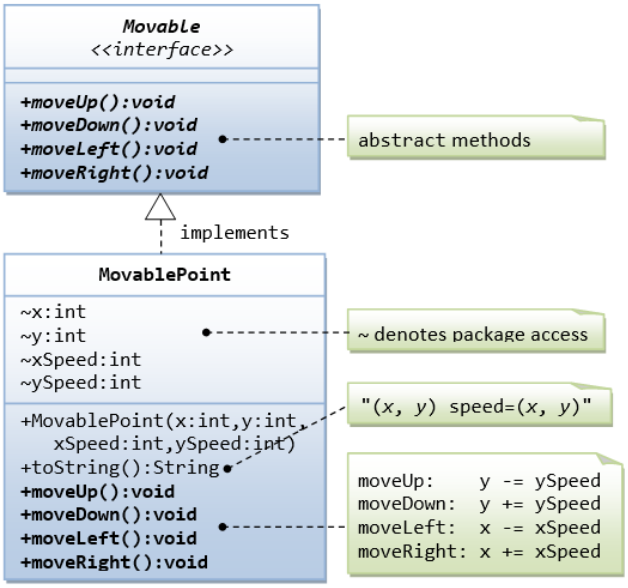
8.2 `GeometricObject` Interface and its Implementation Classes `Circle` and `Rectangle`

Write an interface called `GeometricObject`, which contains 2 abstract methods: getArea() and `getPerimeter()`, as shown in the class diagram. Also write an implementation class called `Circle`. Mark all the overridden methods with annotation `@Override`.



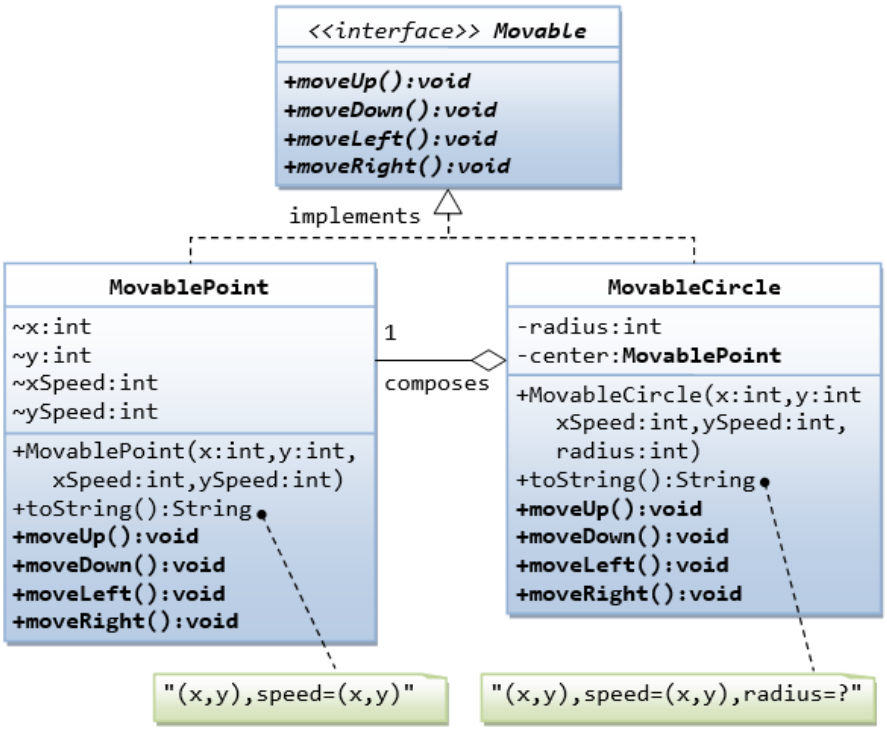
8.3 Ex: `Movable` Interface and its Implementation `MovablePoint` Class

Write an interface called `Movaable`, which contains 4 abstract methods moveUp(), moveDown(), moveLeft() and `moveRight()`, as shown in the class diagram. Also write an implementation class called `MovablePoint`. Mark all the overridden methods with annotation `@Override`.

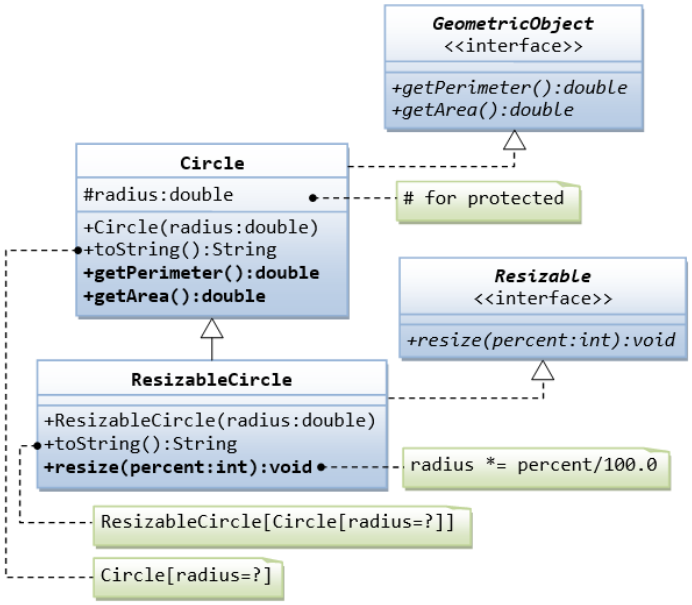


8.4 `Movable` Interface and Classes `MovablePoint` and `MovableCircle`

Write an interface called `Movable`, which contains 4 abstract methods `moveUp()`, `moveDown()`, `moveLeft()` and `moveRight()`, as shown in the class diagram. Also write the implementation classes called `MovablePoint` and `MovableCircle`. Mark all the overridden methods with annotation `@Override`.



8.5 Interfaces `Resizable` and `GeometricObject`



Write the `interface` called `GeometricObject`, which declares two abstract methods: getParameter() and `getArea()`, as specified in the class diagram.

Hints:

```
public interface GeometricObject {
   public double getPerimeter();
   ......
}
```

Write the implementation class `Circle`, with a protected variable `radius`, which implements the interface `GeometricObject`.

Hints:

```
public class Circle implements GeometricObject {
   // Private variable
   ......

   // Constructor
   ......

   // Implement methods defined in the interface GeometricObject
   @Override
   public double getPerimeter() { ...... }

   ......
}
```

Write a test program called `TestCircle` to test the methods defined in `Circle`.

The class `ResizableCircle` is defined as a subclass of the class `Circle`, which also implements an interface called `Resizable`, as shown in class diagram. The interface `Resizable` declares an `abstract` method `resize()`, which modifies the dimension (such as `radius`) by the given percentage. Write the interface `Resizable` and the class `ResizableCircle`.

Hints:

```
public interface Resizable {
   public double resize(...);
}
```

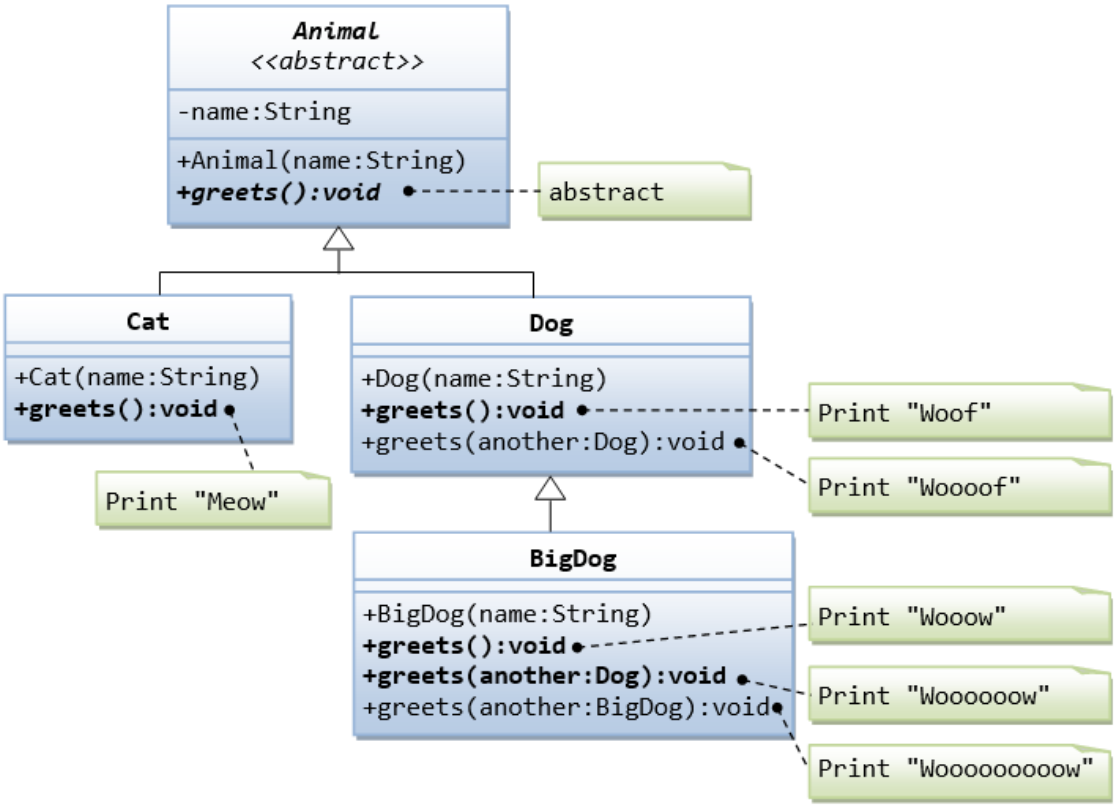
```
public class ResizableCircle extends Circle implements Resizeable {

   // Constructor
   public ResizableCircle(double radius) {
      super(...);
   }

   // Implement methods defined in the interface Resizable
   @Override
   public double resize(int percent) { ...... }
}
```

Try

Write a test program called `TestResizableCircle` to test the methods defined in `ResizableCircle`.

8.6 Abstract Superclass `Animal` and its Implementation Subclasses

Write the codes for all the classes shown in the class diagram. Mark all the overridden methods with annotation @Override.



8.7 Another View of Abstract Superclass `Animal` and its Implementation Subclasses

Examine the following codes and draw the class diagram.

```
abstract public class Animal {
   abstract public void greeting();
}
```

```
public class Cat extends Animal {
   @Override
   public void greeting() {
      System.out.println("Meow!");
   }
}
```

```
public class Dog extends Animal {
   @Override
   public void greeting() {
      System.out.println("Woof!");
   }

   public void greeting(Dog another) {
      System.out.println("Wooooooooooof!");
   }
}
```

```
public class BigDog extends Dog {
   @Override
   public void greeting() {
      System.out.println("Woow!");
   }

   @Override
   public void greeting(Dog another) {
      System.out.println("Wooooooowwwww!");
   }
}
```

Try

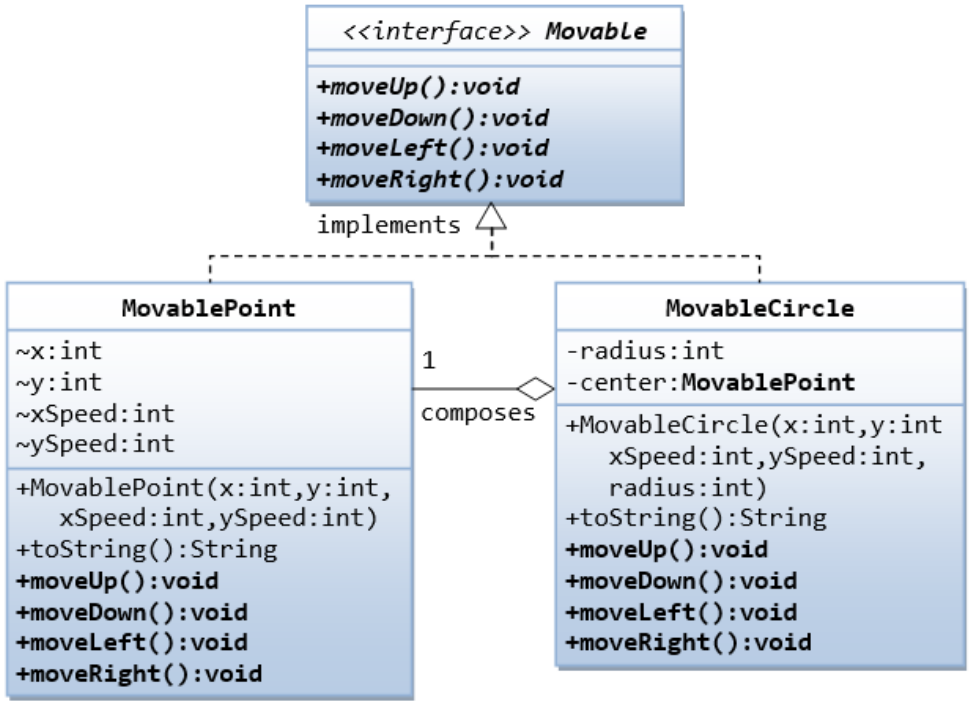
Explain the outputs (or error) for the following test program.

```
public class TestAnimal {
   public static void main(String[] args) {
      // Using the subclasses
      Cat cat1 = new Cat();
      cat1.greeting();
      Dog dog1 = new Dog();
      dog1.greeting();
      BigDog bigDog1 = new BigDog();
      bigDog1.greeting();

      // Using Polymorphism
      Animal animal1 = new Cat();
      animal1.greeting();
      Animal animal2 = new Dog();
      animal2.greeting();
      Animal animal3 = new BigDog();
      animal3.greeting();
      Animal animal4 = new Animal();

      // Downcast
      Dog dog2 = (Dog)animal2;
      BigDog bigDog2 = (BigDog)animal3;
      Dog dog3 = (Dog)animal3;
      Cat cat2 = (Cat)animal2;
      dog2.greeting(dog3);
      dog3.greeting(dog2);
      dog2.greeting(bigDog2);
      bigDog2.greeting(dog2);
      bigDog2.greeting(bigDog1);
   }
}
```

8.8 Interface `Movable` and its subclasses MovablePoint & MovableCircle

Suppose that we have a set of objects with some common behaviors: they could move up, down, left or right. The exact behaviors (such as how to move and how far to move) depend on the objects themselves. One common way to model these common behaviors is to define an *interface* called `Movable`,
with `abstract` methods `moveUp()`, `moveDown()`, `moveLeft()` and `moveRight()`. The classes that implement the `Movable` interface will provide actual implementation to these `abstract` methods.

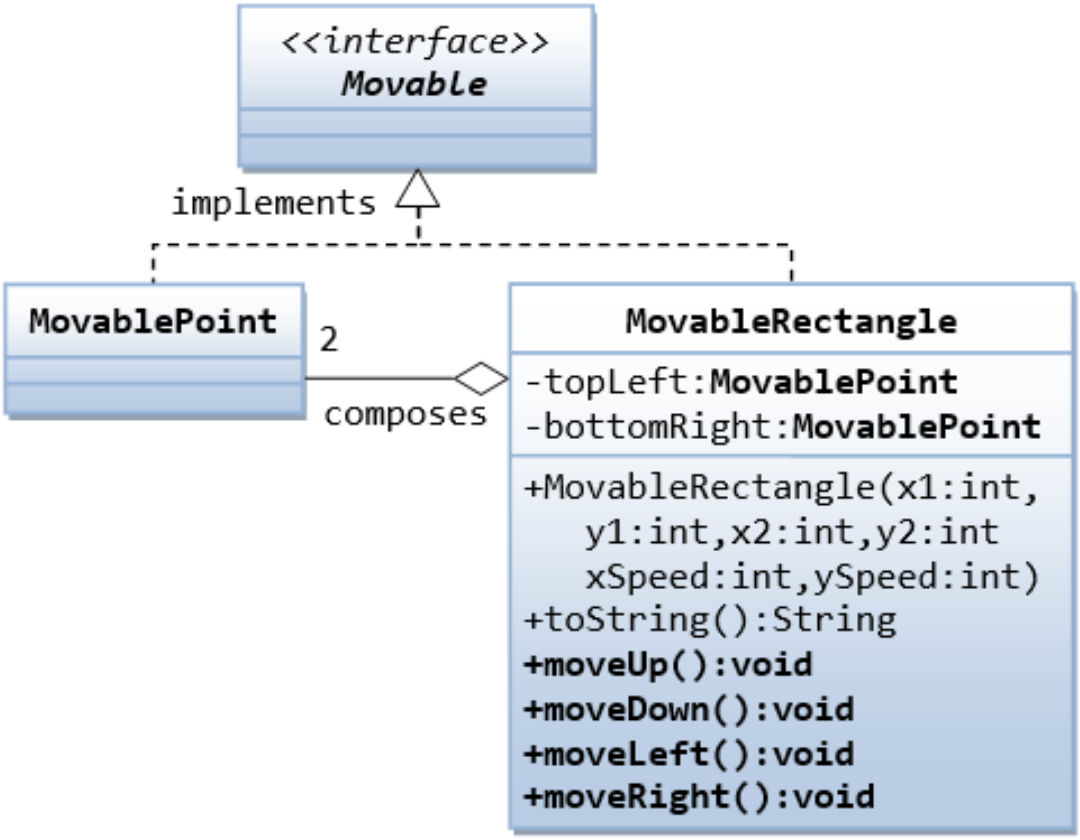
Let's write two concrete classes - `MovablePoint` and `MovableCircle` - that implement the Movable interface.



Write a test program and try out these statements:

```
Movable m1 = new MovablePoint(5, 6, 10, 15);     // upcast
System.out.println(m1);
m1.moveLeft();
System.out.println(m1);

Movable m2 = new MovableCircle(1, 2, 3, 4, 20);  // upcast
System.out.println(m2);
m2.moveRight();
System.out.println(m2);
```

Write a new class called `MovableRectangle`, which composes two `MovablePoints` (representing the top-left and bottom-right corners) and implementing the `Movable` Interface. Make sure that the two points has the same speed.



Try
Develop classes that shows the difference between an interface and an abstract class

9. Exercises on Exception Handling

9.1 Usage of the try and catch and finally block.

In this example, we are implementing try and catch block to handle the exception. The error code written is in try block and catch block handles the raised exception. The finally block will be executed on every condition.

Hints:

```
class ExceptionTest{
        public static void main(String[] args){
                int a = 40, b = 4, c = 4;
                int result;
                try{
                        result = a / (b-c);
                }
                catch (...){
                        ...;
                }
                finally{
                        ...;
                }
                ...;
                System.out.println("Result: "+result);
```

```
        }
}
```

Try

Experiment

- A scenario where NumberFormatException occurs
- A scenario where NullPointerException occurs
- A scenario where ArrayIndexOutOfBoundsException occurs

9.2 Multiple catch block using command line argument

The catch block is used to handle the exception which is raised in try block. A single try block may contain more than one catch block. Below example shows how to use to multiple catch block.

Hints:

```
class Check_Exception{
        public static void main(String[] args){
                try{
                        int a = ...;
                        int b = ...;
                        int c = a / b;
                        System.out.println("Result: "+c);
                }
                catch (...){

                }
                catch(...){

                }
                catch(NumberFormatException ne){

                }
                finally{
                        ...
                }
        }
}
```

9.3 Java throw Keyword

Create a validate method that takes integer value (age) as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

Hints

The syntax of the Java throw keyword is given below.

```
throw new exception_class("error message");
```

Example:

```
throw new IOException("sorry device error");
```

Try
Implement both unchecked and checked exceptions using throw keyword

9.4 Java throws keyword

The throws keyword can be used to declare multiple exceptions, separated by a comma. Whichever exception occurs, if matched with the declared ones, is thrown automatically then. Write Java code that demonstrates the working of throws keyword in exception handling.

Hints

Syntax:

```
return_type method_name() throws exception_class_name{
        //method code
}
```

Advantages:
1. Checked Exception can be propagated (forwarded in call stack).
2. It provides information to the caller of the method about the exception.

9.5 Chained Exceptions

Java allows relating one exception with another exception. i.e. one exception describes the cause of another exception. Write a program to explain the chained exception in Java.

9.6 Custom Exceptions

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

Following are few of the reasons to use custom exceptions:

- To catch and provide specific treatment to a subset of existing Java exceptions.
- Business logic exceptions: These are the exceptions related to business logic and workflow. It is useful for the application users or the developers to understand the exact problem.

In order to create custom exception, we need to extend Exception class that belongs to java.lang package.
Develop a java program to implement custom exeptions

```
// class representing custom exception
class InvalidAgeException  extends Exception
{
    public InvalidAgeException (String str)
    {
        // calling the constructor of parent Exception
        super(str);
    }
}
```

```
// class that uses custom exception InvalidAgeException
…
// main method
…..
```

10. Exercises on File Handling

10.1 Reading text file

Consider a text file abc.txt that contains data. Now your task is to extract the content of the file and display it

Hints:

1. Your program should accept the path to the text file as a command-line argument.
2. Implement proper error handling to account for file-related exceptions.
3. Design and implement a function/method for reading and displaying the content of the text file.

10.2 Reading file content line by line

The file file.txt is a text file that contains a list of names. Each line in the file contains a single name. The names in the file are separated by newline characters.
Write a Java program to read the content of the file file.txt line by line. For each line, print the line to the console.

Hints:

1. Create a BufferedReader object to read the file file.txt.
2. Use a while loop to read each line from the file.
3. For each line, print the line to the console.

10.3 Appending data to an existing file.

Develop a program that allows users to add new content to an already existing file. This problem aims to assess your understanding of file handling, proper exception management, and effective modification of file content.

Hints

1. Create a Java program that accepts the following inputs from the user:
    The path to the existing text file.
    The new content that the user wants to append to the file.
2. Open the existing file and append the provided content to it.
3. Implement appropriate error handling to manage exceptions during file operations.
4. After appending the data, display a confirmation message to the user.

10.4 Reading first 4 lines from a text file

Your task is to develop java program that reads and displays the first four lines of a text file.

Hints

1. Take the path to the text file as a command-line argument.
2. Implement error handling to address potential file-related exceptions.
3. Design a function/method that reads and displays the first four lines of the text file.

10.5 Copy the content of one file to another file

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, FileInputStream and FileOutputStream. Develop java code to copy the content of one file to another.

TRY

Implement the same using Character Streams like FileReader and FileWriter.

10.6 List files and Directories

Write a Java program to get a list of all file/directory names in the given directory.

TRY

1. Check if a file or directory specified by pathname exists or not.
2. Get specific files with extensions from a specified folder.
3. Check if a file or directory has read and write permissions.

11. Exercises on Multithreading

11.1 Concurrent Thread Increment

Create and Start Multiple Threads - Write a Java program to create and start multiple threads that increment a shared counter variable concurrently.

Hints

1. First, define a "Counter" class that represents a shared counter variable. It has a synchronized increment() method that increments the counter variable by one.
2. Next define an "IncrementThread" class that extends Thread. Each IncrementThread instance increments the shared counter by a specified number of increments.
3. In the Main class, we create a 'Counter' object, specify the number of threads and increments per thread, and create an array of 'IncrementThread' objects. We then iterate over the array, creating and starting each thread.
4. After starting all the threads, we use the join() method to wait for each thread to finish before proceeding. After all threads have finished, we print the shared counter's final count.

11.2 wait() and notify() for Thread Synchronization

Write a Java program to create a producer-consumer scenario using the wait() and notify() methods for thread synchronization.

Hints

1. The "Producer" class implements the Runnable interface and represents the producer thread. It continuously produces items by adding values to the shared buffer. When the buffer is full, the producer waits until the consumer consumes an item and notifies it.
2. The "Consumer" class also implements the Runnable interface and represents the consumer thread. It continuously consumes items by removing values from the shared buffer. As soon as the buffer is empty, the consumer uses the wait() method to wait until a new item is produced by the producer.

3. In the main() method, we create instances of the Producer and Consumer classes as separate threads and start them concurrently.

11.3 Synchronizing Threads with Reentrant for Shared Resource

A reentrant mutual exclusion Lock with the same basic behavior and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities.

A ReentrantLock is owned by the thread last successfully locking, but not yet unlocking it. A thread invoking lock will return, successfully acquiring the lock, when the lock is not owned by another thread. The method will return immediately if the current thread already owns the lock. This can be checked using methods isHeldByCurrentThread(), and getHoldCount().

Write a Java program that uses the ReentrantLock class to synchronize access to a shared resource among multiple threads.

11.4 Thread Synchronization with Semaphores

In computer science, a semaphore is a variable or abstract data type used to control access to a common resource by multiple threads and avoid critical section problems in a concurrent system such as a multitasking operating system. Semaphores are a type of synchronization primitive.

Write a Java program to demonstrate Semaphore usage for thread synchronization.

11.5 Concurrent Read-Write Access with ReadWriteLock

A ReadWriteLock maintains a pair of associated locks, one for read-only operations and one for writing. The read lock may be held simultaneously by multiple reader threads, so long as there are no writers. The write lock is exclusive.

All ReadWriteLock implementations must guarantee that the memory synchronization effects of writeLock operations (as specified in the Lock interface) also hold with respect to the associated readLock. That is, a thread successfully acquiring the read lock will see all updates made upon previous release of the write lock.

A read-write lock allows for a greater level of concurrency in accessing shared data than that permitted by a mutual exclusion lock. It exploits the fact that while only a single thread at a time (a writer thread) can modify the shared data, in many cases any number of threads can concurrently read the data (hence reader threads). In theory, the increase in concurrency permitted by the use of a read-write lock will lead to performance improvements over the use of a mutual exclusion lock. In practice this increase in concurrency will only be fully realized on a multi-processor, and then only if the access patterns for the shared data are suitable.

Write a Java program to illustrate the usage of the ReadWriteLock interface for concurrent read-write access to a shared resource.

12. JDBC Programming

12.1 Database Connection and Query

Create a Java program that establishes a connection to a database using JDBC. You are required to perform a basic SQL query to retrieve information from a table and display the results.

Hints

1. Set up a local database (e.g., MySQL) with a sample table.
2. Develop a Java program that establishes a JDBC connection to the database.
3. Write an SQL query to retrieve specific information from the table.
4. Execute the query and display the results in a readable format.
5. Handle any potential exceptions that may occur during database operations.

12.2 Prepared Statements and Parameterized Queries

Design a Java program that utilizes prepared statements for executing parameterized queries. Your program should demonstrate the use of placeholders to execute safe and efficient database operations.

Hints

1. Modify the previous program to use prepared statements for SQL queries.
2. Implement parameterized queries by using placeholders for dynamic values.
3. Allow the user to input values for the query parameters.
4. Execute the prepared statement and display the results.

12.3 Transaction Management

Develop a Java program that demonstrates the concept of transaction management in JDBC. Your program should include operations that simulate a transaction, with both successful and failed cases.

Hints

1. Create a sample database table suitable for transaction simulation.
2. Implement a transaction that involves multiple SQL statements.
3. Perform operations that simulate a successful transaction.
4. Perform operations that simulate a failed transaction (e.g., due to an error or violation).
5. Handle transaction rollbacks and commits accordingly.

12.4 Inserting Multiple Records

Design a Java program that showcases batch processing using JDBC. Your program should allow the insertion of multiple records into a database table in a single operation.

Hints

1. Create a suitable database table for record insertion.
2. Develop a Java program that inserts multiple records into the table.
2. Allow the user to input data for multiple records.
3. Use JDBC techniques to efficiently insert records in a single operation.

12.5 ResultSet and Data Retrieval

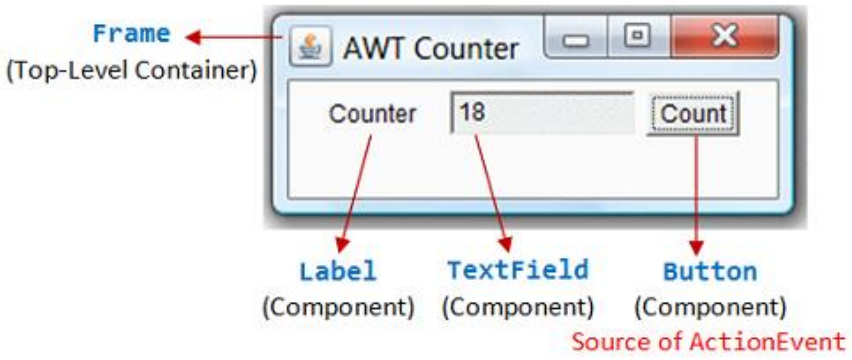
Create a Java program that focuses on retrieving and processing data from a database using JDBC's ResultSet. Your program should demonstrate the fetching and manipulation of data retrieved from a database table.

Hints

1. Set up a database table with relevant sample data.
2. Develop a Java program that connects to the database and retrieves data using a SQL query.
3. Process and manipulate the retrieved data using the ResultSet interface.
4. Display the processed data in a clear and organized manner.

13. AWT GUI Applications

13.1 AWTCounter



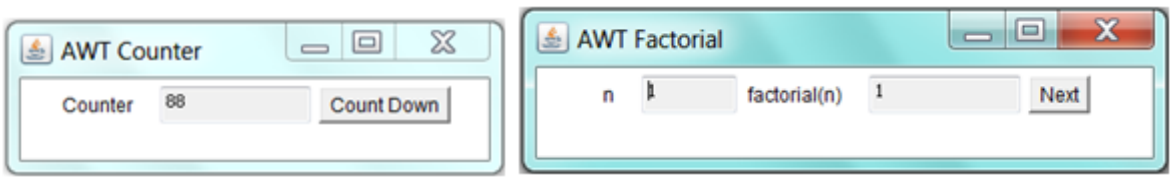
Write an AWT GUI application (called `AWTCounter`) as shown in the Figure. Each time the "Count" button is clicked, the counter value shall increase by 1.

The program has three components:

1. a `java.awt.Label` "`Counter`";
2. a non-editable `java.awt.TextField` to display the counter value; and
3. a `java.awt.Button` "`Count`".

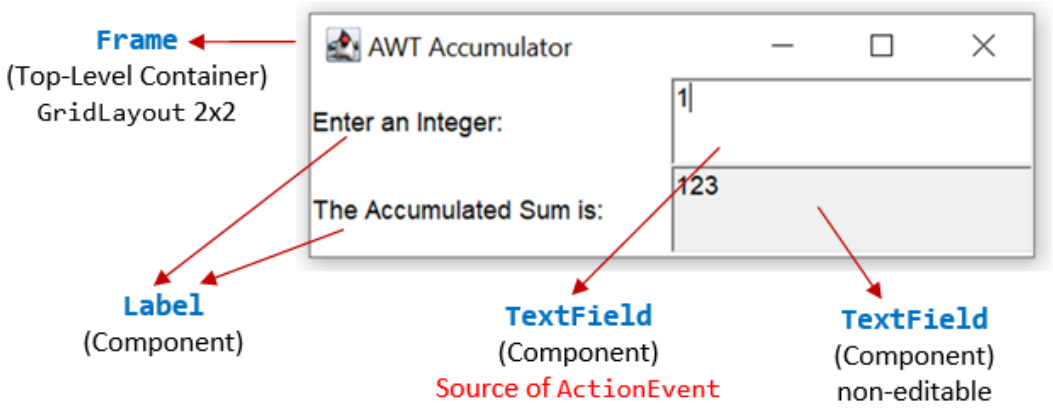
The components are placed inside the top-level AWT container `java.awt.Frame`, arranged in `FlowLayout`.
You have to use control-c, or "close" the CMD shell, or hit the "terminate" button on Eclipse's Console to terminate the program. This is because the program does not process the `WindowEvent` fired by the "window-close" button.

Try



1. Modify the program (called `AWTCounterDown`) to count down, with an initial value of 88, as shown.
2. Modify the program (called `AWTFactorial`) to display n and factorial of n, as shown. Clicking the "Next" button shall increase n by 1. n shall begin at 1.

13.2 AWTAccumulator



Write an AWT GUI application called `AWTAccumulator`, which has four components:

1. a `java.awt.Label` "Enter an integer and press enter";
2. an input `java.awt.TextField`;
3. a `java.awt.Label` "The accumulated sum is", and
4. a protected (read-only) `java.awt.TextField` for displaying the accumulated sum.

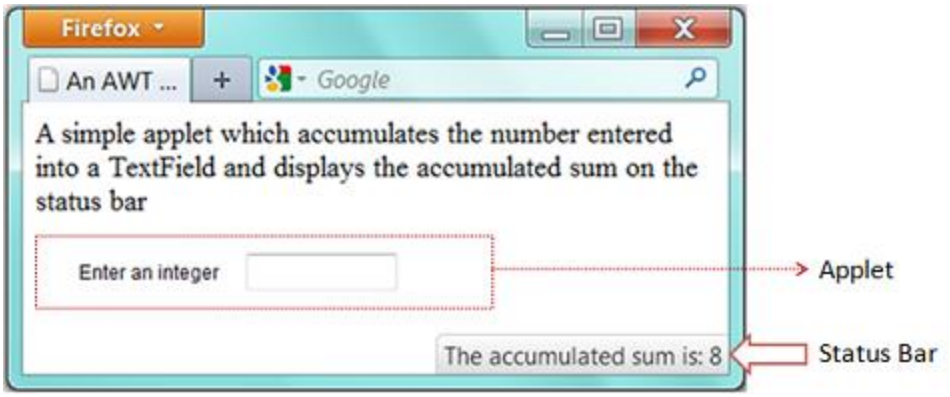
The four GUI components are placed inside a container `java.awt.Frame`, arranged in `GridLayout` of 2 rows 2 columns. The program shall accumulate the numbers entered into the input `TextField`, and display the accumulated sum on the display `TextField`.

Try

1. Modify the program (called `AWTAccumulatorLabel`) to display the sum using a `Label` instead of a protected `TextField`, as shown (using `FlowLayout`).

2. Modify the program (called `AWTFactorialTextField`) to display the factorial of the input number, as shown (using `FlowLayout`).

13.3 AWTAccumulatorApplet (Obsolete)



NOTE: Most browsers today do not support Java applets anymore. Keep this section for nostalgia.

An Java *applet* is a graphics program run inside a browser.
Write a Java applet (called `AWTAccumulatorApplet`) which contains:

1. a `label` "Enter an integer:",
2. a `TextField` for user to enter a number.
3. The applet shall accumulate all the integers entered and show it on the status bar of the browser's window.

Note:

- An applet extends from `java.applet.Applet`, whereas a standalone GUI application extends from `java.awt.Frame`. You cannot `setTitle()` and `setSize()` on Applet.
- Applet uses `init()` to create the GUI, while standalone GUI application uses the constructor (invoked in `main()`).

HTML codes: AWTAccumulatorApplet.html

Applet runs inside a web browser. A separate HTML script (says `AWTAccumulatorApplet.html`) is required, which uses an <applet> tag to embed the applet

Try

1. Modify the applet to run the "Counter" application (as in `AWTCounter`).
2. Modify the applet to run the "Factorial" application (as in `AWTFactorial`).

13.4 WindowEvent and WindowListener

Modify the `AWTCounter` program (called `AWTCounterWithClose`) to process the "Window-Close" button.

14. Swing GUI Applications

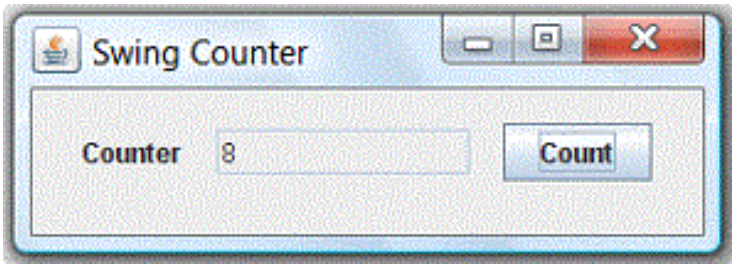
14.1 Converting from AWT to Swing

Convert all the previous AWT exercises (`AWTCounter`, `AWTAccumulator`, `AWTFactorial`, etc.) to Swing applications (called `SwingCounter`, `SwingAccumulator`, `SwingFactorial`, etc.).

Notes:

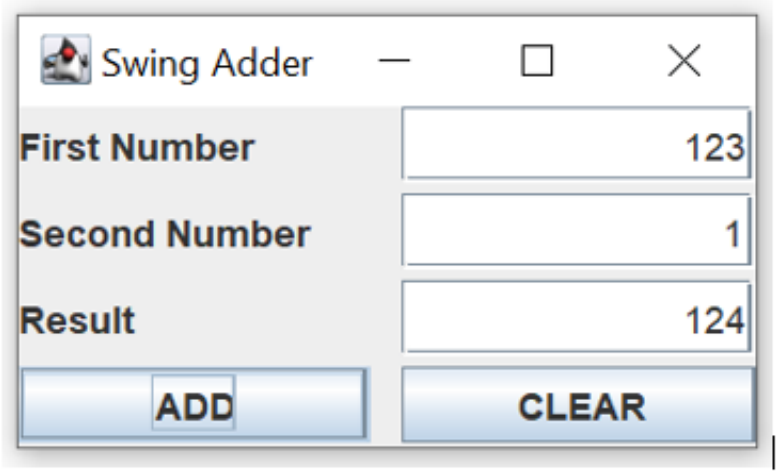
- Swing Components are kept in package `javax.swing`. They begin with a prefix "`J`", e.g., `JButton`, `JLabel`, `JFrame`.
- Swing Components are to be added onto the `ContentPane` of the top-level container `JFrame`. You can retrieve the `ContentPane` via method `getContentPane()` from a `JFrame`.

```
Container cp = getContentPane();  // of JFrame
cp.setLayout(......);
cp.add(......);
```



14.2 SwingAdder

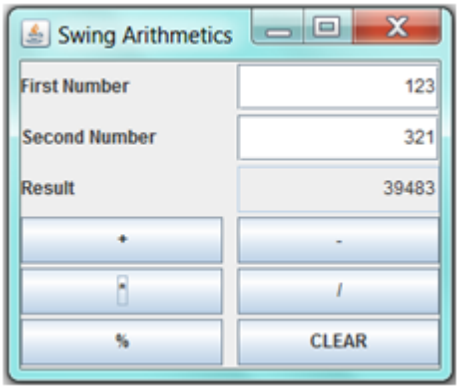
Write a Swing application called `SwingAdder` as shown. The "ADD" button adds the two integers and display the result. The "CLEAR" button shall clear all the text fields.



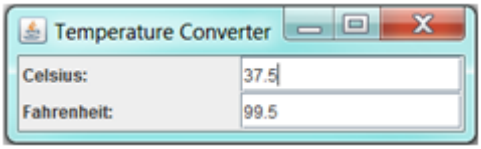
Hints: Set the content-pane to 4x2 GridLayout. The components are added from left-to-right, top-to-bottom.

Try

Modify the above exercise (called `SwingArithmetics`) to include buttons "+", "-", "*", "/", "%" (remainder) and "`CLEAR`" as shown.



14.3 SwingTemperatureConverter

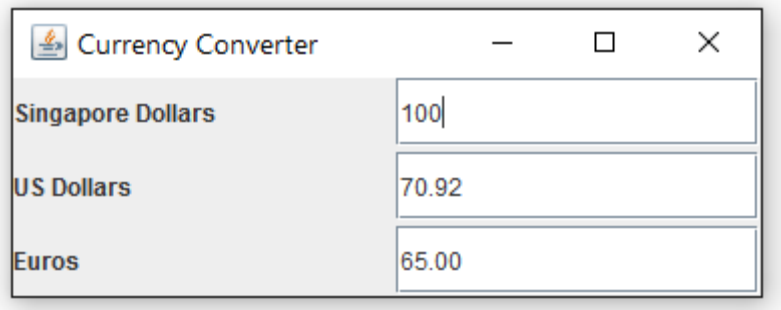


Write a GUI program called `SwingTemperatureConverter` to convert temperature values between Celsius and Fahrenheit. User can enter either the Celsius or the Fahrenheit value, in floating-point number.

Hints

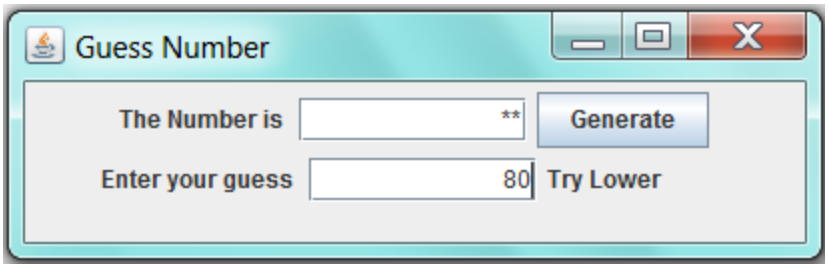
To display a floating-point number in a specific format (e.g., 1 decimal place), use the `static` method `String.format()`, which has the same form as `printf()`. For example, `String.format("%.1f", 1.234)` returns String "`1.2`".

14.4 SwingCurrencyConverter



Write a simple currency converter, as shown in the figure. User can enter the amount of "Singapore Dollars", "US Dollars", or "Euros", in floating-point number. The converted values shall be displayed to 2 decimal places. Assume that 1 USD = 1.41 SGD, 1 USD = 0.92 Euro, 1 SGD = 0.65 Euro.

14.5 SwingNumberGuess



Write a number guessing game in Swing (as shown in the Figure). The program shall generate a random number between 1 to 100. It shall mask out the random number generated and output "Yot Got it", "Try Higher" or "Try Lower" depending on the user's input.

Hints

- You can use `Math.random()` to generate a random number in `double` in the range of `[0.0, 1.0)`.

14.6 SwingPhoneApp

Write a Software Phone App using Java Swing as illustrated in the figure. The user enters the phone number and pushes the "CALL" button to start a phone call. Once the call is started, the label of the "CALL" button changes to "HANG UP". When the user hangs up, the display is cleared.

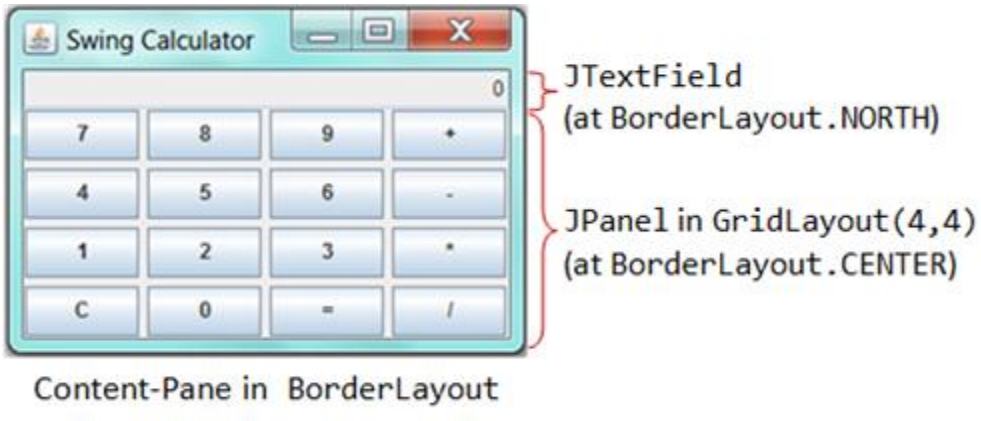
Assume that the following 2 methods are available for handling phone call:

```
public void call(String phoneNumber);  // to make a phone call with the phoneNumber
public void hangup();  // to terminate the existing call
```

Hints

- Use a 10-element `JButton` array to hold the 10 numeric buttons. Construct a common instance of a named inner class as the `ActionListener` for the 10 numeric buttons.
- Use a `boolean` flag (says `isCalling`) to keep track of the status.

14.7 SwingCalculator

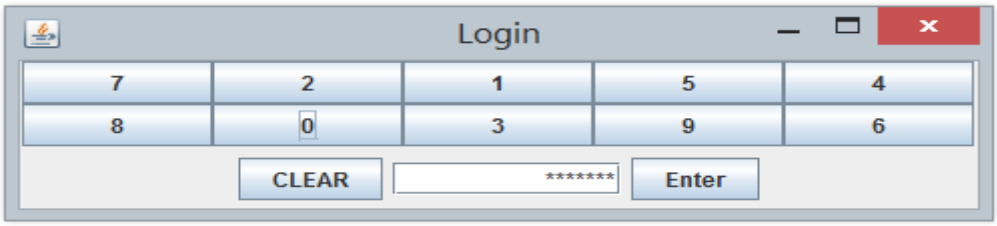


Implement a simple calculator (called SwingCalculator) as shown.

Hints

- Set the ContentPane to BorderLayout. Add a JTextField (tfDisplay) to the NORHT. Add a JPanel (panelButtons) to the CENTER. Set the JPanel to GridLayout of 4x4, and add the 16 buttons.
- All the number buttons can share the same listener as they can be processed with the same codes. Use event.getActionCommand() to get the label of the button that fires the event.
- The operator buttons "+", "-", "*", "/", "%" and "=" can share a common listener.
- Use an anonymous inner class for "C" button.
- You need to keep track of the *previous* operator. For example in "1 + 2 =", the current operator is "=", while the *previous* operator is "+". Perform the operation specified by the previous operator.

14.8 SwingLoginPanel

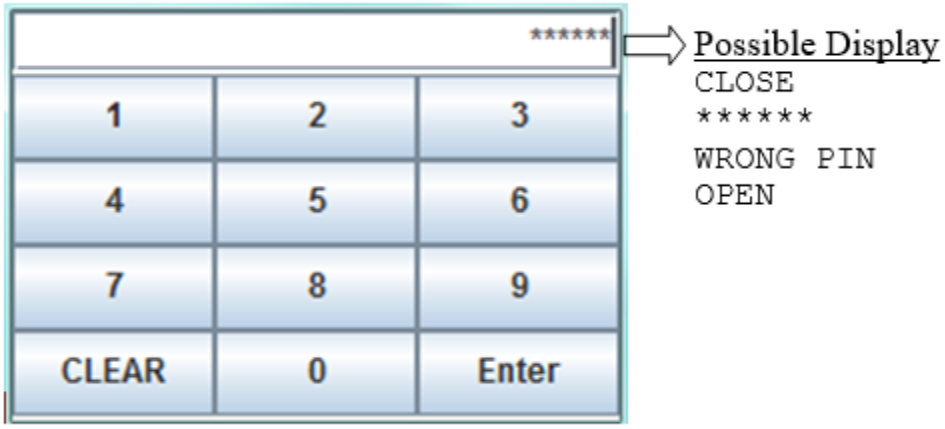


A Java Swing application has a login page as shown in the Figure. Users are required to enter the correct passcode to start the application. The system uses a scramble keypad with a randomly allocated set of numbers from 0 to 9. The display shall show "Enter passcode" initially, and show an asterisk (*) for each number entered. Upon pushing the "Enter" button, the system verifies the passcode. If the passcode is correct, the system invokes a method called `startApp()` to start the application. Otherwise, it displays "Wrong passcode". The "Clear" button shall clear the display.

Assume that the following methods are available:

```
public String getPasscode(); // return the passcode
public void startApp();      // Start the application
public void shuffleArray(int[] array)
   // Shuffle (Randomize) the given int array, e.g.,
   // int[] numbers = {1, 2, 3, 4, 5};
   // shuffleArray(numbers); // randomize the elements
```

14.9 SwingLock



Write a Java Swing application for an electronic lock as shown in the figure. The display shall show the state of either "CLOSE" or "OPEN". In the "CLOSE" state, the user types his PIN followed by the "Enter" key to unlock the system.

The display shall show an asterisk (*) for each number entered. The display shall show "WRONG PIN" if the PIN is incorrect. The "Clear" button clears the number entered (if any), locks the system and sets the display to "CLOSE".

Assume that the following methods are available:

```
public boolean checkPIN(String PIN);  // return true for correct PIN
public void unlock();   // Unlock the system
public void lock();     // Lock the system
```

Hints

- Use a 10-element `JButton` array to hold the 10 numberic buttons. Construct a common instance of a named inner class as their `ActionListener`.
- Use a `boolean` flag (says `isLocked`) to keep track of the status.

15. Final Notes

The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging.

There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)). Check out these sites:

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)
- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).
- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)

- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)
- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- OpenChallenge (https://www.openchallenge.org/)

Coding Contests Scores

Students must solve problems and attain scores in the following coding contests:

Name of the contest	Minimum number of problems to solve	Required score
CodeChef	20	200
Leetcode	20	200
GeeksforGeeks	20	200
SPOJ	5	50
InterviewBit	10	1000
Hackerrank	25	250
Codeforces	10	100
BuildIT	50	500
	Total score need to obtain	2500

Student must have any one of the following certifications:

- HackerRank – Java Basic Skills Certification
- Oracle Certified Associate Java Programmer OCAJP
- CodeChef - Learn Java Certification
- NPTEL – Programming in Java
- NPTEL – Data Structures and Algorithms in Java

V. TEXT BOOKS:

1. Schildt, Herbert. *Java: The Complete Reference* 11th Edition, McGraw-Hill Education, 2018.
2. Deitel, Paul and Deitel, Harvey. *Java: How to Program*, Pearson, 11th Edition, 2018.

VI. REFERENCE BOOKS:

1. Evans, Benjamin J. and Flanagan, David. *Java in a Nutshell*, O'Reilly Media, 7th Edition, 2018.
2. Bloch, Joshua. *Effective Java*, Addison-Wesley Professional, 3rd Edition, 2017.
3. Sierra, Kathy and Bates, Bert. *Head First Java*, O'Reilly Media, 2nd Edition, 2005
4. Farrell, Joyce.*Java Programming*, Cengage Learning B S Publishers, 8th Edition, 2020

VII. ELECTRONICS RESOURCES:

1. https://docs.oracle.com/en/java/
2. https://www.geeksforgeeks.org/java
3. https://www.tutorialspoint.com/java/index.htm
4. https://www.coursera.org/courses?query=java

VIII. MATERIALS ONLINE

1. Course Content
2. Lab manual