ICE 2: Classes & Objects

Resource: http://blue.smu.edu.sg/is442/ice2-resource.zip

Note:

1. The api documentation of the required classes are available in the folder named "api" in the provided zip. Double click on "index.html" to open all documentation.
2. Zip up your Java files and submit one zip file.

1. **[Difficulty: *]** Study the `Shirt` API documentation. Write a program called `ShirtTest` to do the following:
    a. Create the following objects using the appropriate constructor:

Variable Name	Brand	Price	Color
shirt1	SMUgger	85.50	Red
shirt2	SMUgger	90.40	Blue
shirt3	(no brand)	77.60	Green

    Use the specific constructor that takes in 2 arguments for `shirt3`.

    b. Checks whether `shirt1` and `shirt2` are of the same brand using an appropriate method.

    c. Check whether `shirt2` and `shirt3` are of the same brand using an appropriate method.

    d. Set the color of `shirt2` to red.

    e. Print to the console the textual representation of `shirt1`, `shirt2` and `shirt3`.

    Your output should be as follows:

```
s1 same brand as s2:true
s2 same brand as s3:false
Shirt[colour=R,Price=85.5,Brand=SMUgger]
Shirt[colour=R,Price=90.4,Brand=SMUgger]
Shirt[colour=G,Price=77.6,Brand=waterBrand]
```

2. **[Difficulty: *]** Study the `BubbleTea` API documentation. Complete the file called `BubbleTeaTest` to do the following:
    a. Complete the `displayHasPearlsTest()` method. This method will display "`{brand} has pearls in it `" or "`{brand} has no pearls in it`".

    b. Complete the `displayTooSweetTest()` method. This method will display "`{brand} is too sweet `" or "`{brand} is low in sugar`".

    c. Complete the `displaySweetnessComparison()` method. This method will print "`{brand a} is sweeter than {brand b}`". For example, "`COIL is sweeter than Ho Lai Ho Kee`".

Your output should be as follows:

```
1. Pearl Test
Kong Cha has pearls in it
COIL has pearls in it
Ho Lai Ho Kee has no pearls in it

2. Sweetness Test
Kong Cha is too sweet
COIL is low in sugar
Ho Lai Ho Kee is low in sugar

3. Sweetness Comparison Test
Kong Cha is sweeter than COIL
Kong Cha is sweeter than Ho Lai Ho Kee
COIL is sweeter than Ho Lai Ho Kee

3. Sweetness Comparison Test after increase in cup3 sugar level
Kong Cha is sweeter than COIL
Kong Cha is sweeter than Ho Lai Ho Kee
Ho Lai Ho Kee is sweeter than COIL
```

Note: You can try using the printf method. The following code:

```
String name = "Ah Beng";
int age = 18;
System.out.printf("%s is of age %d%n", name, age);
```

will generate the following output:

```
Ah Beng is of age 18
```

Reference:
https://www.cs.colostate.edu/~cs160/.Summer16/resources/Java_printf_method_quick_reference.pdf

3. **[Difficulty: *]** Study the `NewYearGoodie` and `CaloriesCalculator` API documentation. Write a program called `CaloriesCalculatorTest` to do the following:
    a. Create the following objects using appropriate constructors:

Variable Name	Name	Calories	Weight (in g)
tart	Pineapple Tarts	82.5	20
bakKwa	Bak Kwa	115	28
loveLetters	Love Letters	56.5	13

    b. Checks whether `tart` or `bakKwa` is a more sinful new year goodie. Use an appropriate method and print the answer to the console.
    c. Find the calories per gram for the three `NewYearGoodie` objects and print them to the console. Use `System.out.printf` to format the output to 2 decimal places.

    d. Create an instance of the `CaloriesCalculator` for yourself.
    e. You have eaten 2 pieces of pineapple tarts, 3 pieces of bak kwa, and 5 love letters. Calculate the total calories using appropriate methods in the `CaloriesCalculator`. Print the result to the console.
    f. Using an appropriate method in the `CaloriesCalculator`, find out the most sinful new year goodie(the highest number of calories per gram). Its name should be displayed on the console.

Your output should look like this:

```
Pineapple Tarts more sinful than Bak Kwa:false

Pineapple Tarts (calories per gram) :4.13
Bak Kwa (calories per gram) :4.11
Love Letters (calories per gram) :4.35

Total Calories :792.50
Most sinful goodie :Love Letters
```

4. **[Difficulty: *]** You are given the `Substance` API documentation and class file. Complete the code for the `calculateMass` method inside the given file called `RadioActiveTest.java` Your output should look like this:

```
D:\ICE2>java RadioActiveTest
Expected:295.245
Actual  :295.245
```

5. **[Difficulty: *]** You are given the API documentation of the following classes: `Employee`, `Spouse` and `Company`. Write a program called `CompanyTest` to do the following:
    a. Create a `Company` object (which already has some `Employee` objects inside) with the name `"UMS"`.

    b. Add the following employee to the company:
    Name: John
    Employee ID: 5
    Gender: M
    Salary: $2,500
    Spouse's name: Kate
    Spouse's age: 37

    c. Print out the total salary of all the employees in the company.

    d. Prompt the user for a new employee's name, employee ID, salary and gender, and create a new `Employee` object without a spouse. Add this employee to the company.

    e. Now print out the total salary of the company to see whether the change is correct.

    f. Retrieve the `Employee` object whose ID is 4. Print out the name and salary of this employee. Check whether this employee has a spouse. If he has, print out the name of the spouse.

    g. Retrieve the `Employee` object whose ID is 5. Print out the name and salary of this employee. Check whether this employee has a spouse. If he has, print out the name of the spouse.

    h. You may notice that your code for step (f) and step (g) above is very similar. Now instead of writing repetitive code in the `main` method, implement the following static method in `CompanyTest`. Then call this method appropriately in `main` to replace your old code for step (f) and step (g).

```
/*
This method retrieves the employee with the specified id in the    specified
company and prints out his/her name and salary.
It then checks whether this employee has a spouse and prints out either "He/she
doesn't have a spouse" (if there's no spouse) or the name of the spouse (if
there is a spouse).
*/
public static void checkEmployee(Company comp, int id) {
    // ...
}
```

See the sample run of the program below.

```
D:\IS442\ICE2>java CompanyApp
Total salary(before): $10700.0

Enter ID: 6
Enter name: Nick Brown
Enter salary: 3400
Enter gender: M

Total salary(after addition): $14100.0
```

```
Name: Darren, Salary: $3000.00, Status: Single

Name: John, Salary: $2500.00, Status: Married
Spouse's name: Kate
```

6. **[Difficulty: **]** You are given the API documentation of the following Java classes and their byte code (.class) files (**in q6 folder**). Study the API documentation of these classes.
    - `CommunityGarden`
    - `Crop`
    - `Farmer`
    - `Plot`

    A `main` method has been written for you in each of the parts. This `main` method is for testing purpose only. You can use them to test your code.

    Your job is to implement the following four static methods.
    A. `insertPlot`
    B. `getNumSmallLandPlots`
    C. `getFarmerDetails`
    D. `getPercentageOfLeasedPlotsWithCrop`

    Descriptions and requirements of the four methods are given above the method signatures. Strictly follow the requirements stated. DO NOT change the signature (parameter and return type) of each of the methods.

    Note: DO NOT HARDCODE your methods.

OPTIONAL

7. **[Difficulty: *]** Implement a class called `Person`. The `Person` class should have the following attributes: `firstName`, `lastName` and `age`. It should also contain the following constructor/methods:
    a. A constructor to initialize the attributes `firstName`, `lastName` and `age`
    b. A second constructor to initialize only the attributes `firstName`, `lastName`. This constructor sets the value of `age` to `0`.
    c. `getFirstName`: returns the first name of the `Person` object
    d. `getLastName`: returns the last name of the `Person` object
    e. `getAge`: returns the age of the `Person` object
    f. `setAge`: sets the age of the `Person` object
    g. `toString()`: returns the textual representation of the `Person` object in this format: `Person[name=<firstName> <lastName>, age=<age>]`

    Write a `PersonTest` class to check that the `Person` class works. `PersonTest` should do the following in its `main` method:
    a. Prompt the user to enter the first name, last name and age of a person
    b. Create an instance of the Person class (called `aPerson`) with values entered by the user in the previous step
    c. Invoke `toString()` method on `aPerson` object to display the textual representations.
    d. Prompt the user to input only the first name and the last name of a second person
    e. Create another instance of the Person class (called `otherPerson`) with values entered by the user in the previous step.
    f. Invoke `toString()` method on `otherPerson` object to display the textual representations.
    g. Prompt the user to enter the age of `otherPerson`. Use the `setAge()` method to change the age of `otherPerson` to the user-entered value.
    h. Now invoke the `toString()` method on `otherPerson` object again and print out the returned String.

    A sample run of `PersonTest` is shown below:

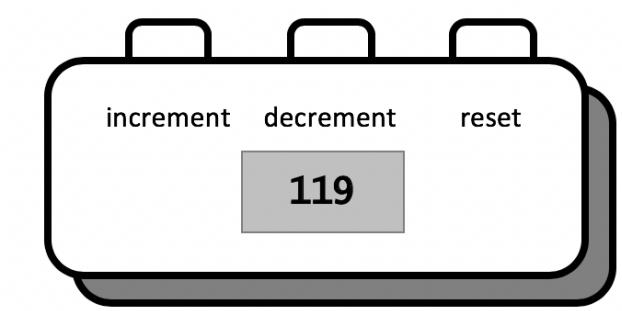
```
D:\IS442\ICE2>java PersonTest
Enter details of the first person:
Enter first name : Kree
Enter last name : Harrison
Enter age : 23
Person[name = Kree Harrison, age = 23]

Enter details of the second person:
Enter first name : Ming
Enter last name : Bridges
Person[name = Ming Bridges, age = 0]

Enter age of second person : 20

After setting age of otherPerson
Person[name=Ming Bridges, age = 20]
```

8. **[Difficulty: *]** Your little sister has a problem with counting. Build a counter that will help her to count. You remember that you used to own a real counter that looked something like this:



It has an increment button, a decrement button and a reset button that resets the value of the counter to zero. The `Counter` class should have the following attributes/methods:

a. An attribute called `value`: that keeps track of the counter number
b. A default constructor which initializes `value` to 0
c. A specific constructor which takes in one parameter say `v` and initializes `value` to `v`
d. Getter for `value` called `getValue()`: returns the value of the counter
e. Setter for `value` called `setValue(int newValue)`: sets the value of the counter to `newValue`
f. `void increment()`: increments the value of the counter by 1
g. `void decrement()`: decrements the value of the counter by 1
h. `void reset()`: sets the value of the counter to 0

Write a test class `CounterTest`. In its `main` method

a. create two instances of the `Counter` class named `firstCounter` and `secondCounter`. Use the default constructor to create the `firstCounter` instance, and the specific constructor to create the `secondCounter` instance with a value of 7.
b. Increment the `firstCounter` counter five times. Decrement the `secondCounter` counter once. Use methods `increment()` and `decrement()` to increment and decrement the counters. Print out the values before and after the increment/decrement.
c. Invoke the reset method on the `secondCounter` counter. Print out the value after the reset.

A sample output of the program is shown below:

```
Before increment – First Counter value: 0
After incrementing 5 times – First Counter value: 5

Before decrement – Second Counter value: 7
After decrement – Second Counter value: 6

After reset – Second Counter value: 0
```

Additional things to try:

A. Give a default value to instance variable at declaration as

```
private int value = 2;
```

Create a `Counter` object - using `Counter` class' default constructor. Use method `getValue()` to check if the counter value has a value of `2` or `0`. What do you observe?

Create a `Counter` object using the `Counter` class' specific constructor to initialize `value` to 5. Use method `getValue()` to check if the counter value takes a value of `2` or `5`. What does this imply?

B. Remove the statement in the *default constructor*

```
value = 0;
```

Create a `Counter` object - using `Counter` class' default constructor. Use method `getValue()` to check what is the counter value. What do you observe?

9. **[Difficulty: *]** Implement the `BankAccount` class. The `BankAccount` class should have the following:
   a. An instance variable of type `double` called `balance`

   b. A default constructor that sets the opening balance to 500

   c. A specific constructor that takes in an initial balance and initializes `balance` to this value

   d. A method which returns the current balance of the bank account

```
public double getBalance()
```

   e. A method to simulate depositing `amount` into the bank account

```
public void deposit(double amount)
```

   f. A method to simulate withdrawing `amount` from the bank account. The method should return `true` if the withdrawal is successful, else `false`. Withdrawal fails if there is insufficient balance in the account i.e. balance in the account is less than the amount to be withdrawn.

```
public boolean withdraw(double amount)
```

   g. A method that simulates the transfer of money from the current account to another account. The method should return `true` if the transfer is successful, else `false`. Transfer fails if there is insufficient balance in the current account from which transfer has to be made.

```
public boolean transfer(double amount, BankAccount
        otherAccount)
```

Compile your `BankAccount.java` against the `BankApplication.java`. Ensure that the output is as expected.

```
Balance in accountOne: $500.0
Balance in accountTwo: $1200.0

After depositing $200 in accountOne
Balance in accountOne: $700.0

After withdrawal of $250 from accountTwo
Balance in accountTwo: $950.0

After transferring $120 from accountOne into accountTwo
Balance in accountOne: $580.0
Balance in accountTwo: $1070.0

Withdrawing $600 from accountOne, successful?
```

```
false
```

10. **[Difficulty: *]** Build a `CashRegister` class that totals up sales and calculates change due to a customer. The class should contain the following:
    a. Two instance variables `purchase` and `payment` (both of type `int`). The variable `purchase` is used to store the cumulative purchase price of all items purchased in cents. The variable `payment` is used to store the amount received from the customer in cents.

    b. A default constructor that initializes all instance variables to **0**

    c. `void registerPurchase(double amount)`: adds the specified amount to `purchase`. This is a method that records the price of a newly purchased item.

    d. `void makePayment(int dollars, int cents)`: dollars and cents paid by the customer. This is a method that records the payment received from the customer and changes the value of `payment` accordingly. Payment is calculated like this:
    payment = dollars x 100 + cents,

    e. `double getPurchaseAmt()`: returns the value of `purchase`

    f. `double giveChange()`: returns the change due to the customer. Change due is calculated like this: change due = payment - purchase.

    Write a `CashRegisterTest` class to check that the `CashRegister` class works. The `CashRegisterTest` should do the following in `main`:
    a. Instantiate the `CashRegister` class
    b. Prompt the user to `'Enter the price of first item'`. Then use `registerPurchase()` to record this purchase.
    c. Prompt the user to `'Enter the price of second item'`. Similarly, use `registerPurchase()` to record this purchase.
    d. Prompt the user to `'Enter dollars received from customer'`
    e. Prompt the user to `'Enter cents received from customer'`
    f. Call method `makePayment()` to record the payment made by the customer.
    g. Display the change due to customer by invoking the `giveChange()` method

    A sample run of `CashRegisterTest` is shown below:

```
Enter the price of first item: $2.75
Enter the price of second item: $3.50

Enter dollars received from customer: 10
Enter cents received from customer: 0

Total cost of purchase: $6.25
Change due: $3.75
```

11. **[Difficulty: **]** Write a class called `Rational` which represents a rational number. A rational number is the ratio of 2 integers and is represented typically as *a / b* where *a* is the numerator and *b* is the denominator. (Put it in a simple way: a rational number is a fraction.) We assume the denominator to be non-zero.

    The basic operations using rational numbers are performed as follows:

    Addition : $\frac{a}{b}+\frac{c}{d}=\frac{ad+bc}{bd}$

    Subtraction: $\frac{a}{b}-\frac{c}{d}=\frac{ad-bc}{bd}$

    Multiplication : $\frac{a}{b}\times\frac{c}{d}=\frac{ac}{bd}$

    Division : $\frac{a}{b}\div\frac{c}{d}=\frac{ad}{bc}$

    To represent a rational number, you need to represent its numerator and denominator. This necessity implies that a class representing rational numbers needs at least two instance variables – one variable to represent the numerator and the other variable to represent the denominator of the rational number:

```
// Numerator of the rational number
private int numerator;

// Denominator of the rational number
private int denominator;
```

    The `Rational` class should provide the following constructor/methods to initialize and manipulate the rational number represented:

    a. Constructs a rational number (i) by default – in which case the numerator and denominator of the rational number are both 1 or (ii) from a specified numerator and denominator.
    b. methods for getting and setting the values of both the numerator and denominator
    c. methods for mathematical operations (add, subtract, multiply, and divide) that returns a Rational object as the result.
    d. produces a `String` representation of a rational number suitable for displaying to the screen.

    We have provided the `RationalTest.java` code. This program illustrates the look and feel that is required of `Rational`. The input and output behavior for the program is as follows:

```
Enter numerator of a rational number: 1
Enter denominator of a rational number: 2

Enter numerator of another rational number: 1
Enter denominator of another rational number: 3
For r = 1/2 and s = 1/3
    r + s = 5/6
    r - s = 1/6
    r * s = 1/6
    r / s = 3/2
```