420-201-RE	Assignment 1	Due Date: 5 September 2016

Program 1: "Java String literals"

Write a new Java program that

1. displays the four text lines boxed below using exactly *four* **System.out.println** statements,

```
In Java, a String literal is a sequence of characters inside
a pair of double quotation marks, such as "Java is fun".
Java programmers should understand the difference between "
and \" as well as the difference between ' and ".
```

2. displays the same four text lines boxed above using exactly *one* **System.out.println** statement, and
3. displays the following pattern:

```
   /\
  //\\
 /\/\/\
+------+
|      |
|  GO  |
+------+
|      |
| Java |
+------+
|      |
|  GO  |
+------+
   /\
  //\\
 /\/\/\
```

Program 2: Integer Division

The purpose here is for you to learn about *integer* division in Java, which is different from what you are accustomed to from algebra:

In Java, an **int** divided by another **int** produces an **int** *quotient* and an **int** *remainder*. Examples:

$7/2 = 3$, the *quotient* of 7 divided by 2

$7\%2 = 1$, the *remainder* of 7 divided by 2

Expression	=	in Algebra	in Java
7/2	=	3.5	**3**
7.0/2	=	3.5	3.5
7/2.0	=	3.5	3.5
7.0/2.0	=	3.5	3.5

Write a Java program that prompts for an integer that represents an amount in cents and then computes and displays minimum number of coins that make change for the given amount. Assume you have infinite supply of toonies (200¢), loonies (100¢), quarters (25¢), dimes (10¢), nickels (5¢), and pennies (1¢).

Use a variety of input values to verify that your program runs correctly.

A sample run of your program should produce the following output, where the user's input is shown in red for clarity:

```
Enter change in cents: 5763

A minimum of 35 coins to make change for 5763 cents:

Toonies : 28
Loonies : 1
Quarters: 2
Dimes   : 1
Nickels : 0
Cents   : 3
```

Hint Compute the to total number of like coins with / and the remaining change with **%**.

For example, there are $5763/200 = 28$ toonies in 5763 cents with $5763\%200 = 163$ remaining cents. Then, there are $163/100 = 1$ loonies in 163 cents with $163\%100 = 63$ remaining cents. Next, there are $63/25 = 2$ quarters in 63 cents with $63\%25 = 13$ remaining cents. And so on.

Program 3: Integer Division, Again

Write a Java program that inputs a ten-digit phone number, and then turns that ten-digit phone number into a more readable string with a pair of parentheses, a dash, and a space.

For example, if the user enters the value **5147447500** as input, your program should print the string **(514) 744-7500**.

Hint use *integer division* and *integer remainder* operations to compute the three required blocks of numbers: area code **514**, prefix **744**, and line number **7500**.

For example, in Java, **5147447500 / 10000000** is equal to **514**.

A sample run of your program should produce the following output, where the user's input is shown in red for clarity:

```
Enter a 10-digit phone numbers: 5147447500
area code   : 514
prefix      : 744
line number : 7500
phone number: (514) 744-7500
```

Program 4: double to int Conversion

The idea here is to practice type casting in Java.

Recall that Java lets you assign a **int** value to a **double** variable, but not vice versa. To assign a **double** value to an **int** variable, you need to convince the Java compiler to trust you by explicit type casting:

```
int i;
i = 12;           // ok, int variable <-- int value
i = 12.3         // not ok, int variable <-- double value
i = (int) 12.3; // ok, drops .3 and assigns 12 to i
                // this is an example of type casting, effectively saying:
                // dear Java compile, trust me, I know I'm going to loose
                // precision here, but that's ok ... I know what I'm doing
```

Write a Java program that

1. inputs a price as a floating-point value of type **double**, and then
2. separates and prints the dollars (an **int**) and cents (another **int**) from that value.

A sample run of your program should produce the following output, where the user's input is shown in red for clarity:

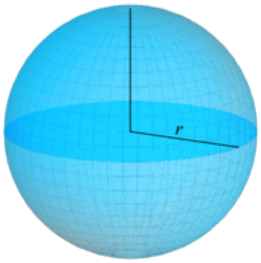
```
Enter a price: 248.75
The price is 248 dolors and 75 cents.
```

Program 5: Floating-Point Operations

Recall that the volume (V) and surface area (A) of a sphere with radius r are given by the following formulas:

$$V = (4/3)\pi r^3$$

$$A = 4\pi r^2$$



Write a Java program that prompts the user to enter a value for the radius of a sphere, reads it, and then computes and prints the volume and surface area of that sphere.

Make sure you choose meaningful names for the identifiers in your program; for example, prefer the name **redius** to a name like **r**.

Here is the output of a sample run of the program, with the user input shown in red for clarity.

```
Enter the radius of a sphere :  8.55
Sphere radius : 8.55
Volume : 2618.1
Surface area : 918.633
Press any key to continue . . .
```

Note that the output of your program may produce slightly different values for the volume and surface area depending on how you implement the formulas.

Also note the difference between algebraic and Java notations:

Algebra	Java
π	`Math.PI`
r^3	`Math.pow(r, 3)` or `(r * r * r)`
b^x	`Math.pow(b, x)`

Program 6: More Floating-Point Operations

Write a program that will calculate the monthly payment for a loan using the formula

$$P = \frac{iA}{1 - (1+i)^{-N}}$$

where

- A is the loan amount
- i is the monthly interest rate
- N is the total number of monthly payments for the entire loan term.
- P is the amount of each monthly payment.

Here is how a sample run of your program should look like, where the user inputs is shown in red for clarity:

```
Enter loan amount: 225000.0
Enter term in years: 30
Enter annual interest rate (%): 7.8

Your Monthly payment: $1619.71
```

Note 1: Your program will prompt the user for the value of the annual interest rate instead of monthly interest rate. That means you will need to convert the input annual interest rate to monthly interest rate before you can use it in the formula above. For example, if the annual interest rate is 7.8%, then the monthly interest rate is $i = \frac{7.8\%}{12} = \frac{7.8}{1200}$.

Note 2: Your program will prompt the user for the value of the loan term in years. That means you will need to convert the input loan term in years to loan term in months before you can use it in the formula above. For example, for a loan term of 30 years, there is a total of $30 \times 12 = 360$ monthly payments.

Make sure you choose meaningful names for A, i, N, and P in the formula above. Here are some suggestions:

Variable in Formula	Sample Variable Name in Java	Another Sample Variable Name in Java
A	loan_amount	loanAmount
i	monthly_interest_rate	monthlyInterestRate
N	loan_term_in_months	loanTermInMonths
P	monthly_payment	monthlyPayment