ASSIGNEMNT 1	**Chapter 3 – Selections** **Chapter 4 – Mathematical Functions, Characters and Strings** **Chapter 5 – Loops** **Chapter 6 - Methods**
Due Jan 18 at 11:30 pm	**Objectives** ■ To declare **boolean** variables and write Boolean expressions using relational operators ■ To implement selection control using one-way **if** statements ■ To implement selection control using two-way **if-else** statements ■ To implement selection control using nested **if** and multi-way **if** statements ■ To avoid common errors and pitfalls in **if** statements ■ To generate random numbers using the **Math.random()** method ■ To combine conditions using logical operators (**!**, **&&**, **||**, and **^**) ■ To program using selection statements with combined conditions ■ To solve mathematical problems by using the methods in the Math class ■ To represent characters using the char type ■ To encode characters using ASCII and Unicode ■ To represent strings using the String object

***3.1** (*Algebra: solve quadratic equations*) The two roots of a quadratic equation $ax^2 + bx + c = 0$ can be obtained using the following formula:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

$b^2 - 4ac$ is called the discriminant of the quadratic equation. If it is positive, the equation has two real roots. If it is zero, the equation has one root. If it is negative, the equation has no real roots.

Write a program that prompts the user to enter values for a, b, and c and displays the result based on the discriminant. If the discriminant is positive, display two roots. If the discriminant is 0, display one root. Otherwise, display "The equation has no real roots".

Note that you can use `Math.pow(x, 0.5)` to compute $\sqrt{x}$. Here are some sample runs.

```
Enter a, b, c: 1.0 3 1 Enter
The equation has two roots -0.381966 and -2.61803
```

```
Enter a, b, c: 1 2.0 1 Enter
The equation has one root -1
```

```
Enter a, b, c: 1 2 3 Enter
The equation has no real roots
```

**3.15 (*Game: lottery*) Revise Listing 3.8, Lottery.java, to generate a lottery of a three-digit number. The program prompts the user to enter a three-digit number and determines whether the user wins according to the following rules:

1. If the user input matches the lottery number in the exact order, the award is $10,000.
2. If all digits in the user input match all digits in the lottery number, the award is $3,000.
3. If one digit in the user input matches a digit in the lottery number, the award is $1,000.

**3.21 (*Science: day of the week*) Zeller's congruence is an algorithm developed by Christian Zeller to calculate the day of the week. The formula is

$$h = \left(q + \frac{26(m+1)}{10} + k + \frac{k}{4} + \frac{j}{4} + 5j\right) \% 7$$

where

- `h` is the day of the week (0: Saturday, 1: Sunday, 2: Monday, 3: Tuesday, 4: Wednesday, 5: Thursday, 6: Friday).
- `q` is the day of the month.
- `m` is the month (3: March, 4: April, …, 12: December). January and February are counted as months 13 and 14 of the previous year.
- `j` is the century (i.e., $\frac{year}{100}$).
- `k` is the year of the century (i.e., *year* % 100).

Note that the division in the formula performs an integer division. Write a program that prompts the user to enter a year, month, and day of the month, and displays the name of the day of the week. Here are some sample runs:

```
Enter year: (e.g., 2012): 2015 Enter
Enter month: 1-12:  1 Enter
Enter the day of the month: 1-31: 25 Enter
Day of the week is Sunday
```

```
Enter year: (e.g., 2012): 2012 Enter
Enter month: 1-12: 5 Enter
Enter the day of the month: 1-31: 12 Enter
Day of the week is Saturday
```

(*Hint*: January and February are counted as 13 and 14 in the formula, so you need to convert the user input 1 to 13 and 2 to 14 for the month and change the year to the previous year.)

**3.28 (*Geometry: two rectangles*) Write a program that prompts the user to enter the center *x*-, *y*-coordinates, width, and height of two rectangles and determines whether the second rectangle is inside the first or overlaps with the first, as shown in Figure 3.9. Test your program to cover all cases.

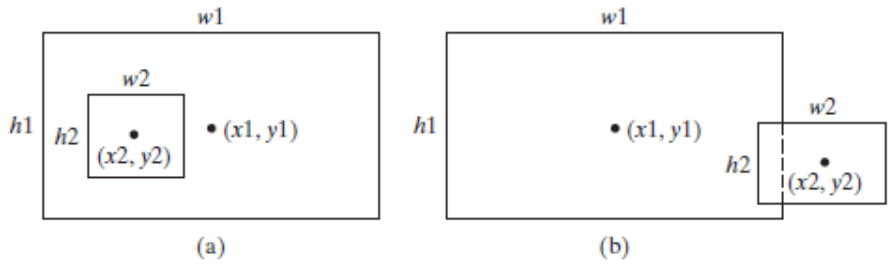


FIGURE 3.9 (a) A rectangle is inside another one. (b) A rectangle overlaps another one.

Here are the sample runs:

```
Enter r1's center x-, y-coordinates, width, and height: 2.5 4 2.5 43 Enter
Enter r2's center x-, y-coordinates, width, and height: 1.5 5 0.5 3 Enter
r2 is inside r1
```

```
Enter r1's center x-, y-coordinates, width, and height: 1 2 3 5.5 Enter
Enter r2's center x-, y-coordinates, width, and height: 3 4 4.5 5 Enter
r2 overlaps r1
```

```
Enter r1's center x-, y-coordinates, width, and height: 1 2 3 3 Enter
Enter r2's center x-, y-coordinates, width, and height: 40 45 3 2 Enter
r2 does not overlap r1
```

*4.17 (*Days of a month*) Write a program that prompts the user to enter a year and the first three letters of a month name (with the first letter in uppercase) and displays the number of days in the month. Here is a sample run:

```
Enter a year: 2001 Enter
Enter a month: Jan Enter
Jan 2001 has 31 days
```

***4.21** (*Check SSN*) Write a program that prompts the user to enter a Social Security number in the format DDD-DD-DDDD, where D is a digit. Your program should check whether the input is valid. Here are sample runs:

```
Enter a SSN: 232-23-5435 Enter
232-23-5435 is a valid social security number
```

```
Enter a SSN: 23-23-5435 Enter
23-23-5435 is an invalid social security number
```

***4.23** (*Financial application: payroll*) Write a program that reads the following information and prints a payroll statement:

Employee's name (e.g., Smith)
Number of hours worked in a week (e.g., 10)
Hourly pay rate (e.g., 9.75)
Federal tax withholding rate (e.g., 20%)
State tax withholding rate (e.g., 9%)

A sample run is shown below:

```
Enter employee's name: Smith Enter
Enter number of hours worked in a week: 10 Enter
Enter hourly pay rate: 9.75 Enter
Enter federal tax withholding rate: 0.20 Enter
Enter state tax withholding rate: 0.09 Enter

Employee Name: Smith
Hours Worked: 10.0
Pay Rate: $9.75
Gross Pay: $97.5
Deductions:
  Federal Withholding (20.0%): $19.5
  State Withholding (9.0%): $8.77
  Total Deduction: $28.27
Net Pay: $69.22
```

5.29 (*Display calendars*) Write a program that prompts the user to enter the year and first day of the year and displays the calendar table for the year on the console. For example, if the user entered the year 2013, and 2 for Tuesday, January 1, 2013, your program should display the calendar for each month in the year, as follows:

January 2013

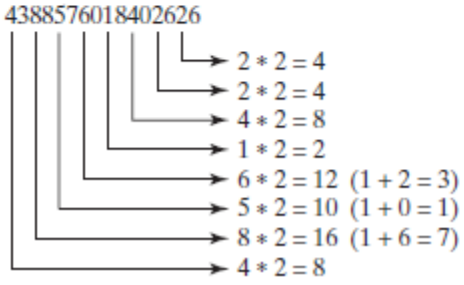
Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

6.31 (*Financial: credit card number validation*) Credit card numbers follow certain patterns. A credit card number must have between 13 and 16 digits. It must start with:

- 4 for Visa cards
- 5 for Master cards
- 37 for American Express cards
- 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the *Luhn check* or the *Mod 10 check*, which can be described as follows (for illustration, consider the card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.



2. Now add all single-digit numbers from Step 1.

$$4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37$$

3. Add all digits in the odd places from right to left in the card number.

$$6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38$$

4. Sum the results from Step 2 and Step 3.

$$37 + 38 = 75$$

5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid.

Write a program that prompts the user to enter a credit card number as a `long` integer. Display whether the number is valid or invalid. Design your program to use the following methods:

```
/** Return true if the card number is valid */
public static boolean isValid(long number)

/** Get the result from Step 2 */
public static int sumOfDoubleEvenPlace(long number)

/** Return this number if it is a single digit, otherwise,
 * return the sum of the two digits */
public static int getDigit(int number)

/** Return sum of odd-place digits in number */
public static int sumOfOddPlace(long number)

/** Return true if the digit d is a prefix for number */
public static boolean prefixMatched(long number, int d)

/** Return the number of digits in d */
public static int getSize(long d)

/** Return the first k number of digits from number. If the
 * number of digits in number is less than k, return number. */
public static long getPrefix(long number, int k)
```

Here are sample runs of the program: (You may also implement this program by reading the input as a string and processing the string to validate the credit card.)

```
Enter a credit card number as a long integer:
  4388576018410707 Enter
4388576018410707 is valid
```

```
Enter a credit card number as a long integer:
  4388576018402626 Enter
4388576018402626 is invalid
```

TASK LIST

You are required to complete the following activities by the deadlines specified and submit the appropriate *deliverables* through eLearning.

ACTIVITY	DEADLINE
Create Class Files having its own main method for each of the problems in this assignment. Compile, Execute and Test your program	Jun 18 at 11:30 pm
Zip the Java Files and submit on eLearning	

GUIDELINES

You will be graded according to the following guidelines:

- You are required to submit files through eLearning by the specified deadline. You can earn a maximum of 100 points.
- If the files do not compile, you will receive a 0 for the program.
- You are graded primarily on the design of your class and the adequate testing of your class. Poor design or inadequate testing will result in loss of points.
- Your files should be adequately commented. Up to -15 points will be deducted for poor indentation and documentation.