+

Mathematical Functions, Characters, and Strings

Introduction to Computer Science CSCI-UA.0101

Lecture 4

+ Agenda Day 4

- The Math Class
- The Character Type
- The String Type
- Generating Random Numbers
- Programming Errors
- GUI Confirmation Dialogs

The Math Class

The Math class

- We can use the Math class in the Java library to perform a number of math operations not available in the language itself
- The Math class is a static class so it doesn’t need to be instantiated like the Scanner class, which requires the “new” keyword – you can just use it right out of the box.
- It exists inside the java.lang package, so you don’t need to import anything in order to use it.

Math Class: trigonometric methods

METHOD	Description	Example
Math.sin()	returns sine of an angle in radians	Math.sin(Math.PI/2) returns 0.5
Math.cos()	returns cosine of an angle in radians	Math.cos (Math.PI/3) returns 0.5
Math.tan()	returns tangent of an angle in radians	Math.tan (Math.PI/4) returns 1
Math.toRadians()	returns the value in radians of an angle in degrees	Math.toRadians(30) returns 0.5236 (= PI / 6)
Math.toDegrees()	returns the value in degrees of an angle in radians	Math.toDegrees(0.5236) returns 30
Math.asin()	returns the angle in radians that has a given sine	Math.asin(0.5) returns 0.5236 (= PI / 6)
Math.acos()	returns the angle in radians that has a given cosine	Math.acos(0.5) returns 1.0472 (= PI /3)
Math.atan()	returns the angle in radians that has a given tangent	Math.acos(0.5) returns 1.0472 (= PI / 3)

Math Class: exponent methods

METHOD	Description	Example
Math.exp(*x*)	returns *e* raised to power of *x*	Math.exp(1) returns 2.71828
Math.log(*x*)	returns natural log of *x*	Math.log(Math.E) returns 1
Math.log10(*x*)	returns the log in base 10 of *x*	Math.log10 (10) returns 1
Math.pow(*a*, *b*)	returns *a* raised to the power of *b*	Math.pow(2, 3) returns 8
Math.sqrt(*x*)	returns the square root of *x*	Math.sqrt(4) returns 2

+
Math Class: rounding methods

METHOD	Description	Example
Math.ceil(*x*)	returns *x* rounded up to its nearest integer as a double	Math.ceil(2.1) returns 3.0
Math.floor(*x*)	returns *x* rounded down to its nearest integer as a double	Math.floor(2.1) returns 2.0
Math.rint(*x*)	returns *x* rounded to its nearest integer as a double	Math.rint(2.5) returns 2.0 (if equally close to two integers, it returns the even one)
Math.round(*x*)	returns Math.floor(*x* + 0.5) as an integer if *x* is a float or as a long integer if *x* is a double.	Math.round(2.5) returns 3

+

Math Class: other methods

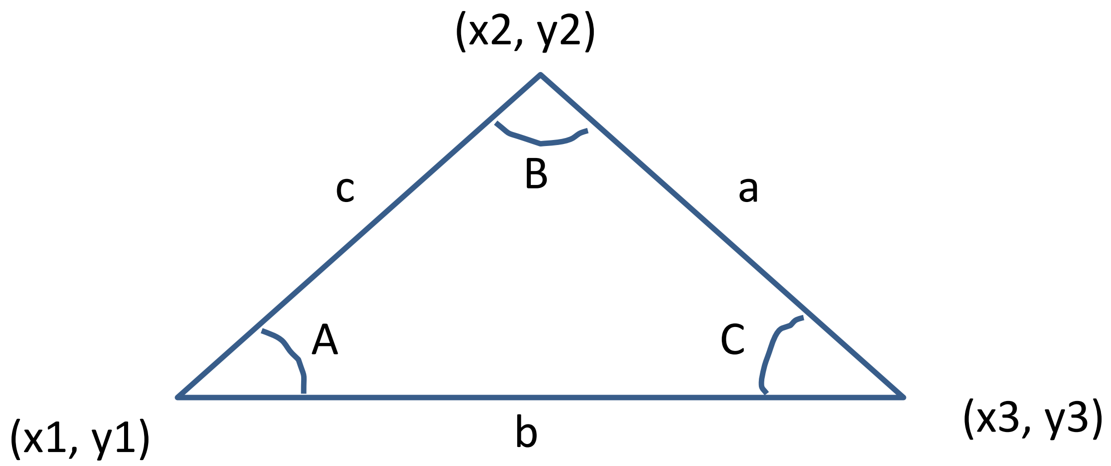
METHOD	Description	Example
Math.min(*a*, *b*)	returns the minimum of the two numbers	Math.min(2.5, 3) returns 2.5
Math.max(*a*, *b*)	returns the maximum of the two numbers	Math.max(2.5, 4.6) returns 4.6
Math.abs(*x*)	returns the absolute value of *x*	Math.abs(-2.1) returns 2.1
Math.random()	returns a random positive double >=0 and < 1. It does not take any arguments	Math.random()

Programming Challenge

- Write a program to calculate the angles of a triangle from the coordinates of its vertices, using the following formulas:

  A = acos((a*a – b*b – c*c) / (-2 * b * c))
  B = acos((b*b – a*a – c*c) / (-2 * a * c))
  C = acos((c*c – b*b – a*a) / (-2 * a * b))

- **ComputeAngles.java**



+ Character Data Type

The Character Data Type

- In addition to numbers and Strings, Java also supports the ability to process individual characters
- The character data type (abbreviated "char") is one of Java's built-in primitive data types, along with int and double
- In fact, the String data type is actually made up of an "array" (Python: "list") of characters

Defining a character

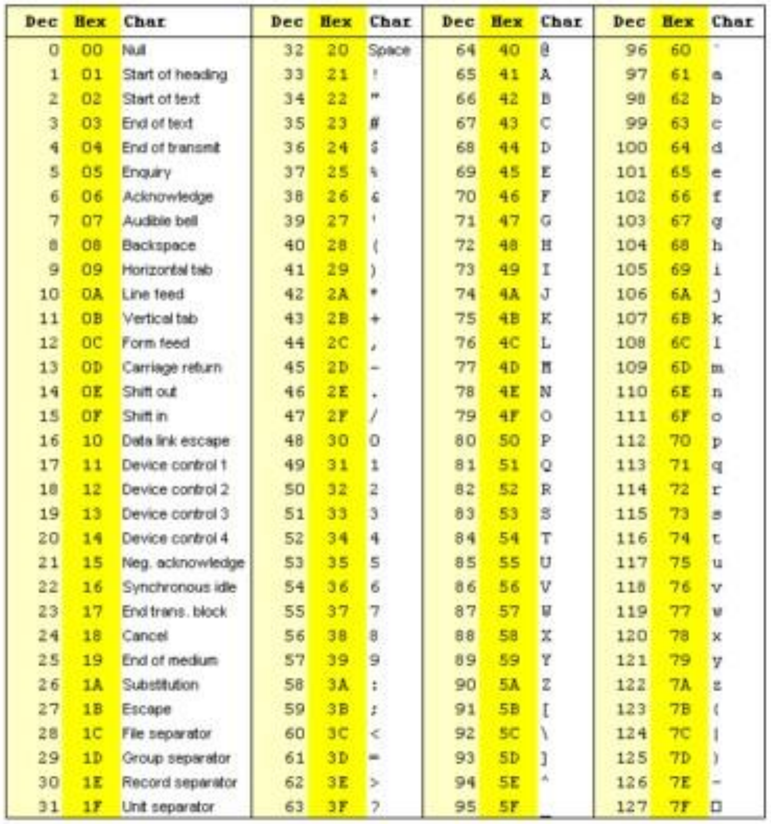
- You can define a character using the same syntax as when you define an integer:

```
char a = ‘C’;
```

- Note the delimiters - characters must be delimited with the single tick, not the double quote character

The ASCII Chart

- Remember, everything is just zeros and ones!
- Java uses the ASCII chart (to the right) to encode characters
- This means that when you define a character to be the char 'A' you are referencing item #65 on the ASCII chart.



Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Chars and ASCII

- You can convert between chars and integer data types very easily in Java. For example:

```
char a = 65;
System.out.println(a); // A

int b = ’C’;
System.out.println(b) // 67;
```

+
Working with Time values

- All computers have a standard time keeping mechanism based on the "UNIX Epoch"
- The "UNIX Epoch" is 00:00:00 on January 1st, 1970 GMT. All computers count forward from this time, and it serves as a common reference point for time calculations.
- You can access the current GMT time via Java using the following method call:

```
long currentTime = System.currentTimeMillis();
```

- This method call will return the number of milliseconds that have elapsed since the UNIX Epoch.

Programming Challenge

- Write a program that asks the user to enter in their first name and their last name
- Calculate how long it takes to complete this task to and report the result to the user once they have finished inputting the requested data.
- **TimeChallenge.java.**

+ String Data Type

+
The String class

- String is a pre-defined class in Java. You don’t need to import the String class into your programs.

- The String type is not a primitive type like int, char or double. String is a reference type that can be used to create a reference variable that references a string object.

- For example, in the statement [**String message = “Hello”;**], message is a reference variable that references a string object that contains the string “Hello”.

- For the time being you need to know only how to declare a String variable, how to assign a string to the variable, and how to use the methods available in the String class to your string objects.

String Class: methods for string processing

METHOD	Description
str.length()	returns the number of characters in the string str
str.charAt(index)	returns the character at the specified index in the string str (the index of the first character is 0)
str.concat(s1)	returns a new string that concatenates str with s1
str.toUpperCase()	returns a new string with all characters of string str converted to upper case.
str.toLowerCase()	returns a new string with all characters of string str converted to lower case.
str.trim()	returns a new string eliminating whitespace characters at both ends of the string str.
str.indexOf(s1)	returns the index of the first occurrence in string str of substring s1 (-1 if there is no such occurrence)
str.substring(index1, index2)	Returns a substring of string str starting at index1 and ending at index2.

Programming Challenge

- Write a program that ask the user to enter his/her favorite city and calculates:

  -the number of characters of the string
  -the city in upper case letters
  -the city in lower case letters
  -the first character of the city name
  -the last character of the city name

- **StringManipulation.java**

+

String Class: methods for comparing strings

METHOD	Description
str.equals(s1)	returns true if str is equal to s1
str.equalsIgnoreCase(s1)	returns true if str is equal to s1; it is case insensitive
str.compareTo(s1)	returns an integer >0, 0, or <0 to indicate whether this string is lexicographically >, =, or < than s1
str.compareToIgnoreCase(s1)	same as compareTo() except that the comparison is case insensitive
str.startsWith(prefix)	returns true if string str starts with the specified prefix
str.endsWith(suffix)	returns true if string str ends with the specified suffix
str.contains(s1)	returns true is string str contains substring s1

+
Programming Challenge

- The US Constitution establishes that only a natural born citizen who is at least 35 years old is eligible to be President. Write a program to help a user determine is s/he is eligible to be POTUS.

- **President.java**

+ Programming Challenge

- Write a program to order the names of two cities entered by the user in alphabetical order

- **OrderTwoCities.java**

Generating Random Numbers

+ Using Math.random()

- Math.random() generates a random double >= 0.0 and < 1.0
- Example 1:

```
(int) (Math.random() * 10)          //returns a radom int between 0 and 9
```

- Example 2:

```
50 + (int) (Math.random() * 50)  //returns a radom int between 50 and 99
```

Using the Random class

- The Random class can also be used to generate a random number
- Before using the Random class we need to import it:

  import java.util.Random;

- We also need to create an object of the Random class:

  Random randomNum = new Random(); //randomNum is a generator

- Methods of the Random class:

  randomNum.nextDouble(); //generates random double >= 0.0 and < 1.0

  randomNum.nextInt(n); //generates random int >= 0 and < n

+ Generating Random Ranges

- You can also use the Math class to generate random numbers in customized ranges(i.e. generate#'s between 5 and 25 instead of 0.0 and 1.0)
- To do this you can apply the following algorithm:
  - Identify the Max and Min values of the range
  - Construct a statement like the following:

```
Random r = new Random();

int num = r.nextInt(4) + 3

(num will be from 3,4,5 or 6)
```

+ Programming Challenge: Lottery

- A lottery is a game of chance that lets the user select a series of integers
- Write a program that generates the winning two digits, and allows a user to select two digits.
- Allocate winnings based on the following chart:

  >>Exact match: $10,000

  >>Match all digits: $ 3,000

  >>Match one digit: $1,000

  >>No match: $ 0.00

- ***LoteryUsingStrings.java***

+

Formatting Console Output

Formatting Console Output

- During the output phase of your programs you will often want to ensure that your output is formatted in a certain way. For example:

```
Input a price value:  1.00

Your item, with tax, will cost $1.07
```

System.out.printf()

- System.out.printf() is a method that can be used to print formatted text to the console.
- The method takes at least two arguments - a formatting pattern and the data elements to be formatted.
- For example, if you wanted to format a floating point number (either a double or a float) to 2 decimal places you could do the following:

```
double num = 5.0 / 9.0;
System.out.printf("%.2f", num);
```

+ Formatting Patterns

- Formatting patterns always begin with a percent sign
- Formatting patterns always end with a data type description represented as a single character. For example:

```
%f = floating point
%d = decimal integer
%s = string
```

+ Formatting Patterns: Width

- Between the percent sign and the data type descriptor you can specify "width" and "precision" values for the element in question.
- Width represents the number of characters to use when printing out the element in question. Java will "pad" the output with spaces at the beginning if the element ends up having less characters than the number specified.

```
int num = 5;

System.out.printf("%5d", num);
System.out.println();
System.out.printf("%10d",  num);
System.out.println();
System.out.printf("%15d",  num);
System.out.println();
```

```
____5
_________5
______________5
```

("_" = "space")

+
Formatting Patterns: Precision

- Between the percent sign and the data type descriptor you can specify "width" and "precision" values for the element in question.
- Precision represents the number of decimal places to display. You can only specify precision with numbers that contain decimal values. To set precision simply type a decimal point and the number of decimal values to round to.

```
double num = 5.0 / 9.0;

System.out.printf("%.2f",  num);
System.out.println();
System.out.printf("%.4f",  num);
System.out.println();
System.out.printf("%.6f",  num);
System.out.println();
```

```
0.56
0.5556
0.555556
```

Formatting Patterns: Combine Width and Precision

- For floating point values you can combine both width and precision. For example:

```
double num = 5.0 / 9.0;

System.out.printf("%10.2f", num);
System.out.println();
System.out.printf("%10.4f", num);
System.out.println();
System.out.printf("%10.6f", num);
System.out.println();
```

- Will produce:

```
      0.56       // 6 leading spaces
    0.5556       // 4 leading spaces
  0.555556       // 2 leading spaces
```

Combining formatting with standard output

- The formatting pattern used in calls to System.out.printf() does not need to only contain a single formatting pattern.
- You can combine non-formatting pattern characters in this String as well. For example:

```
double price = 1.99;
double taxRate = 0.07;
double total = price + (price * taxRate);

System.out.printf("Your item will cost $ %.2f", total);
```

Formatting multiple elements

- You can use multiple formatting patterns to format multiple items inside the same call to System.out.printf(). By default is right justified.
- If you do this you must ensure that you pass additional elements to your call to System.out.printf(). For example:

```
// print a header row
System.out.printf("%10s%10s", "Base Price", "Total");
System.out.println();

double price = 1.99;
double taxRate = 0.07;
double total = price + (price * taxRate);

// print a data row
System.out.printf("%10.2f%10.2f", price, total);
```

System.out.printf() and line breaks

- System.out.printf() will not generate a line break for you
- You can generate a line break with a call to System.out.println() on the line following your call to System.out.printf() [as in the previous examples]
- You can also place a "\n" escape character into your formatting pattern. For example:

```
double num = 5.0 / 9.0;
System.out.printf("%10.2f\n", num);
System.out.printf("%10.4f\n", num);
System.out.printf("%10.6f\n", num);
```

+
Programming Challenge

- Demonstration of the use of System.out.printf()
- Create a table to display the degrees, radians, sin, cosine and tangent of angles 0, 30, 60 and 90
- **FormatDemo.java**

Programming Errors

Types of Errors

- **Syntax errors**: The code does not follow the rules of the language; for example, a single quote is used where a double quote is needed; a semi-colon is missing; a keyword is used as a variable name.
- **Runtime errors**: In this case, your code is fine but the program does not run as expected (it "crashes"). For example, if your program is meant to divide two numbers, but does not test for a zero divisor, a run-time error would occur when the program attempts to divide by zero.
- **Logic errors**: These can be the hardest to find. In this case, the program is correct from a syntax perspective; and it runs; but the result is unanticipated or outright wrong. For example, if your program prints "2+2 = 5" the answer is clearly wrong . .

Example Errors

```
System.out.printtln(“hi”);
```

+
Example Errors

```
System.out.printtln("hi");
```

Syntax error:

"printtln" should be "println"

+
Example Errors

```
public class Welcome {
  public static main (String[] args) {
      System.out.println(“hi there”)
      System.out.println(“bye!”)
  }

}
```

+
Example Errors

```
public class Welcome {
  public static main (String[] args) {
      System.out.println(“hi there”)
      System.out.println(“bye!”)
  }

}
```

Syntax Errors:

1. The “void” keyword is missing before the main method declaration
2. Missing semicolons after the two System.out.println method calls

+
Example Errors

```
import java.util.Scanner;

public class Welcome {
  public static void main (String[] args) {
     Scanner numberScanner = new Scanner(System.in);
     int a = numberScanner.nextInt(); // a = 0
     System.out.println( 100 / a );
  }

}
```

+
Example Errors

```
import java.util.Scanner;

public class Welcome {
  public static void main (String[] args) {
     Scanner numberScanner = new Scanner(System.in);
     int a = numberScanner.nextInt(); // a = 0
     System.out.println( 100 / a );
  }

}
```

Runtime Error:

  Cannot divide a number by zero.

+
Example Errors

```
public class Welcome {
 public static main (String[] args) {
    System.out.print(“Test score average: “);
    System.out.println( (100+95+87) / 2 );
  }

}
```

Example Errors

```
public class Welcome {
  public static void main (String[] args) {
      System.out.print(“Test score average: “);
      System.out.println( (100+95+87) / 2 );
  }

}
```

Logic Error:

Program will run, but the answer that is being given is incorrect

+ Debugging Syntax Errors

- Syntax errors are usually the easiest types of errors to correct
- The compiler can detect syntax errors and indicate the possible error and the ***approximate*** site where the errors are located. Sometimes the error message produced by the compiler is cryptic.
- When getting several errors, try to fix the first error on the list first. Fixing that first error can also resolve several other subsequent errors.

+ Debugging Runtime Errors

- Runtime errors are also fairly easy to identify.
- The Java interpreter will display runtime errors to the console as they occur. Errors reported in this way usually contain descriptive information regarding the nature of the error.

Debugging Logic Errors

- Logic errors, or “bugs,” can be difficulty to identify and correct since they generally do not halt the execution of your program or prevent it from compiling.

One Strategy : Print Statements

- The most time honored debugging technique is inserting "print" statements into your code at key points to peek behind the scenes and see what is going on.
- Inserting System.out.println() statements that display the value of key variables and a short descriptive message can be valuable for seeing how a program is behaving.

+

One Strategy : Print Statements

```
Scanner numberScanner = new Scanner(System.in);

System.out.print("Give me a number: " );
int a = numberScanner.nextInt();

System.out.print("Give me a number: " );
int b = numberScanner.nextInt();



int sum = a + b*2;
```

```
Scanner numberScanner = new Scanner(System.in);

System.out.print("Give me a number: " );
int a = numberScanner.nextInt();

System.out.print("Give me a number: " );
int b = numberScanner.nextInt();

System.out.println ("a is " + a);
System.out.println ("b is " + b);

int sum = a + b*2;

System.out.println("sum is " + sum);
```

+

Operator Precedence

+

Operator Precedence

- Earlier we learned that Java supports the standard order of operations for arithmetic operations.
- The full operator precedence chart is as follows (items we have discussed already are highlighted)

Operator	Description
( )	**Parenthesis**
(type)	Casting
!	Not
***, /, %**	**Mult, Div, Mod**
+, -	**Add, Sub**
<, <=, >, >=	Comparison
==, !=	Equality
^	XOR
&&	AND
||	OR
=	Assignment

Operator Precedence

- Note that the logical operators do not occupy the same level of the precedence tree.
- This means that you should always group out your boolean expressions using parenthesis to ensure that they are evaluated in the correct order. For example:

```
true || true && false // true
(true || true) && false // false
```

Operator	Description
( )	**Parenthesis**
(type)	Casting
!	Not
***, /, %**	**Mult, Div, Mod**
+, -	**Add, Sub**
<, <=, >, >=	Comparison
==, !=	Equality
^	XOR
&&	AND
||	OR
=	Assignment

+

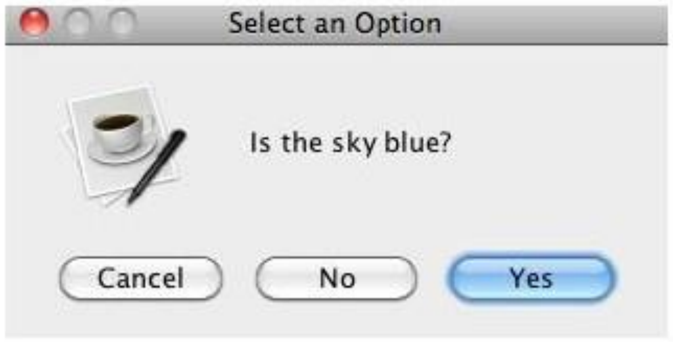
Confirmation Dialogs

+
Confirmation Dialogs

- You can use a GUI pop-up window to ask users to answer simple "Yes / No" questions.
- The syntax for launching a Confirmation Dialog is as follows:

```
int option = JOptionPane.showConfirmDialog(null, "Is the sky blue?")
```

- Which will produce a pop-up window that looks like the following:



Is the sky blue?

Confirmation Dialogs

- The JOptionPane.showConfirmDialog() method will pause and wait for the user to select an answer.
- It will then return an integer that represents the button that the user pressed, as follows:
  - Yes: 0
  - No: 1
  - Cancel: 2
- Instead of having to remember these integers you can use a series of static constants that exist in the JOptionPane class. For example:
  - JOptionPane.YES_OPTION = 0
  - JOptionPane.NO_OPTION = 1
  - JOPtionPane.CANCEL_OPTION = 2

+ Using Confirmation Dialogs

```
// ask the user to answer a question
int option  = JOptionPane.showConfirmDialog(null,  “Does 2+2 = 4?”);

// check the answer
if (option == JOptionPane.YES_OPTION)
{
	System.out.println(“Correct!”);
}
else if (option == JOptionPane.NO_OPTION)
{
	System.out.println(“Incorrect!”);
}
else
{
	System.out.println(“Cancel!”);
}
```