**Section 1: Define / Answer**

Logical Operators

**& - bitwise and**

**- bitwise or**

**^ - bitwise xor**

**-logical or**

**&& - logical and**

**! - not**

Difference between & and &&?

& is bitwise. && is logical.

& evaluates both sides of the operation.  
&& evaluates the left side of the operation, if it's true, it continues and evaluates the right side.

**Complete Logic Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **P** | **S** | **P &S** | **P | S** | **P ^ S** | | **!S** | **(!(P^S)) &(P|S)** |
| **TRUE** | **FALSE** | **FALSE** | **TRUE** | **TRUE** | | **TRUE** | **FALSE** |
| **FALSE** | **FALSE** | **FALSE** | **FALSE** | **FALSE** | | **TRUE** | **FALSE** |
| **TRUE** | **TRUE** | **TRUE** | **TRUE** | **FALSE** | | **FALSE** | **TRUE** |
| **FALSE** | **TRUE** | **FALSE** | **TRUE** | **TRUE** | | **FALSE** | **FALSE** |
| **(P&S) && (P|S)** | | **(P^S) || (!P)** | | |
| **FALSE** | | **TRUE** | | |
| **FALSE** | | **FALSE** | | |
| **TRUE** | | **TRUE** | | |
| **FALSE** | | **FALSE** | | |

Data Structure:

<https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html>

1. LinkedList-an ordered set of data elements, each containing a link to its successor (and sometimes its predecessor).

Describe Differences between LinkedList and ArrayList-

LinkedList and [ArrayList](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) both implement List Interface but how they work internally is where the differences lies. Main **difference between ArrayList and LinkedLis**t is that ArrayList is implemented using re sizable array while LinkedList is implemented using doubly LinkedList. ArrayList is more popular among Java programmerthan LinkedList as there are few scenarios on which LinkedList is a suitable collection than ArrayList. 

Pg. 577**,** Java Programming *A comprehensive Introduction*

[**http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#format**](http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#format)(Detailed explanation of Java documentation)

[**http://www.tutorialspoint.com/java/java\_documentation.htm**](http://www.tutorialspoint.com/java/java_documentation.htm)

[**http://www.liferay.com/community/wiki/-/wiki/Main/Javadoc+Guidelines#section-Javadoc+Guidelines-Class+Comments**](http://www.liferay.com/community/wiki/-/wiki/Main/Javadoc+Guidelines#section-Javadoc+Guidelines-Class+Comments)

Internal Documentation- the notes on how and why various parts of code operate is included within the [source code](https://en.wikipedia.org/wiki/Source_code) as comments. It is often combined with meaningful [variable](https://en.wikipedia.org/wiki/Variable_(programming)) names with the intention of providing potential future programmers a means of understanding the workings of the code.

Internal documentation would be comments and remarks made by the programmer in the form of line comments and boiler plates.

External Documentation- External documentation would be things like flow charts, UML diagrams, requirements documents, design documents etc.

Java Doc Tags-  is a [documentation generator](https://en.wikipedia.org/wiki/Documentation_generator) from [Oracle Corporation](https://en.wikipedia.org/wiki/Oracle_Corporation) for generating [API](https://en.wikipedia.org/wiki/Application_programming_interface" \o "Application programming interface)documentation in [HTML](https://en.wikipedia.org/wiki/HTML) format from [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) source code. The HTML format is used to add the convenience of being able to [hyperlink](https://en.wikipedia.org/wiki/Hyperlink) related documents together.[[2]](https://en.wikipedia.org/wiki/Javadoc#cite_note-2)

**Javadoc tags (Examples)**

|  |  |  |
| --- | --- | --- |
| **Tag** | **Description** | **Syntax** |
| @author | Adds the author of a class. | @author name-text |
| {@code} | Displays text in code font without interpreting the text as HTML markup or nested javadoc tags. | {@code text} |
| {@docRoot} | Represents the relative path to the generated document's root directory from any generated page | {@docRoot} |
| @deprecated | Adds a comment indicating that this API should no longer be used. | @deprecated deprecated-text |
| @exception | Adds a **Throws** subheading to the generated documentation, with the class-name and description text. | @exception class-name description |
| {@inheritDoc} | Inherits a comment from the **nearest** inheritable class or implementable interface | Inherits a comment from the immediate surperclass. |
| {@link} | Inserts an in-line link with visible text label that points to the documentation for the specified package, class or member name of a referenced class. T | {@link package.class#member label} |
| {@linkplain} | Identical to {@link}, except the link's label is displayed in plain text than code font. | {@linkplain package.class#member label} |
| @param | Adds a parameter with the specified parameter-name followed by the specified description to the "Parameters" section. | @param parameter-name description |
| @return | Adds a "Returns" section with the description text. | @return description |
| @see | Adds a "See Also" heading with a link or text entry that points to reference. | @see reference |
| @serial | Used in the doc comment for a default serializable field. | @serial field-description | include | exclude |
| @serialData | Documents the data written by the writeObject( ) or writeExternal( ) methods | @serialData data-description |
| @serialField | Documents an ObjectStreamField component. | @serialField field-name field-type field-description |
| @since | Adds a "Since" heading with the specified since-text to the generated documentation. | @since release |
| @throws | The @throws and @exception tags are synonyms. | @throws class-name description |
| {@value} | When {@value} is used in the doc comment of a static field, it displays the value of that constant: | {@value package.class#field} |
| @version | Adds a "Version" subheading with the specified version-text to the generated docs when the -version option is used. | @version version-text |

**Programming Assignment**

Task 1- (Using the &&, || and ^ operators) Write a program that prompts the user to enter an integer and determines whether it is divisible by 5 or 6, whether by 5 or 6, and whether it is divisible by 5 or 6, but not both.

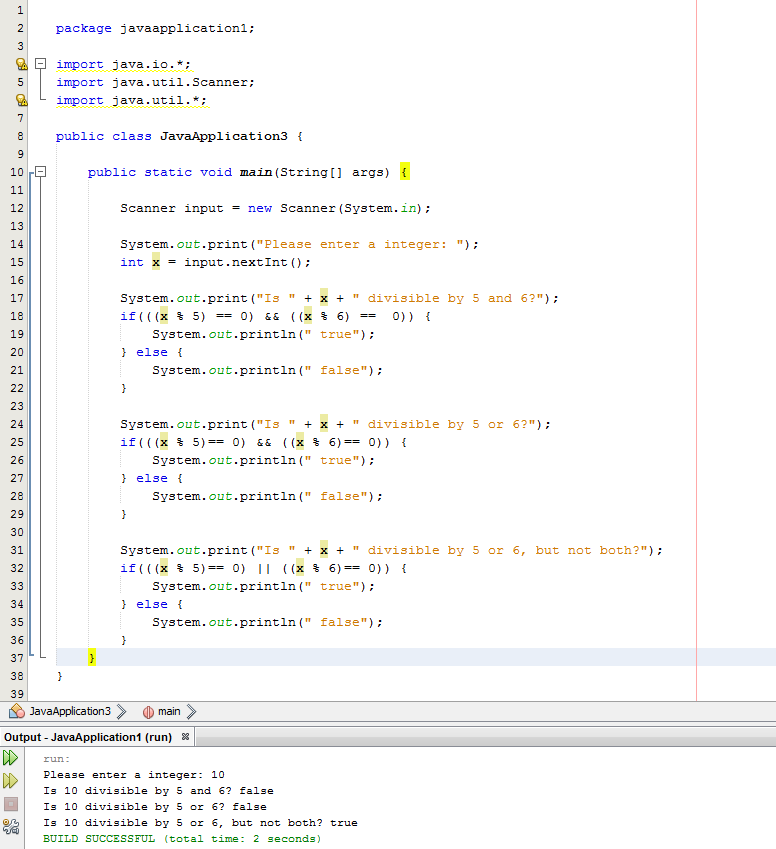
**Sample Output**

Enter an Integer: 10

Is 10 divisible by 5 and 6? false

Is 10 divisible by 5 or 6? false

Is 10 divisible by 5 or 6, but not both? true



Task 2- Create a program that uses a **switch** statement and **case** statements to output the number of days in any given month based upon a users input.

February is a unique month. So the **case** for February will needa **if**… **else** to capture the two possible options for the number of days in February based upon leap year. Include an ability to input the year if the user chooses February.

If February is a leap year you will need logical operators and circuit operators to print the number of days in a leap year.

**Sample Output**

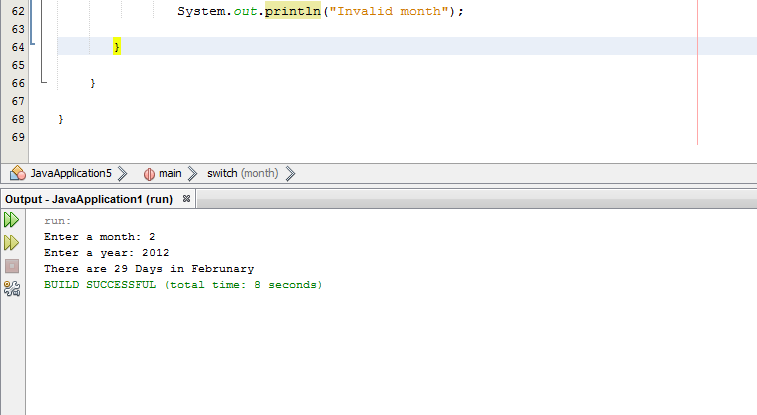
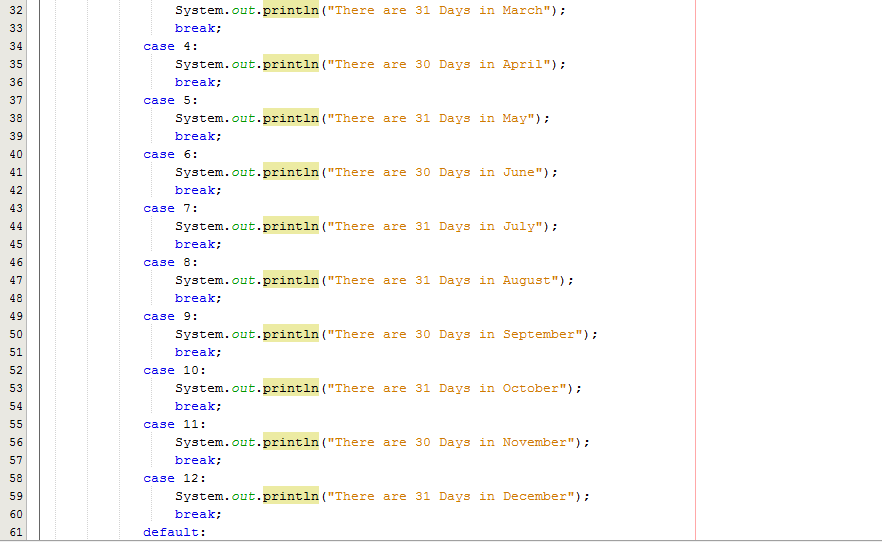
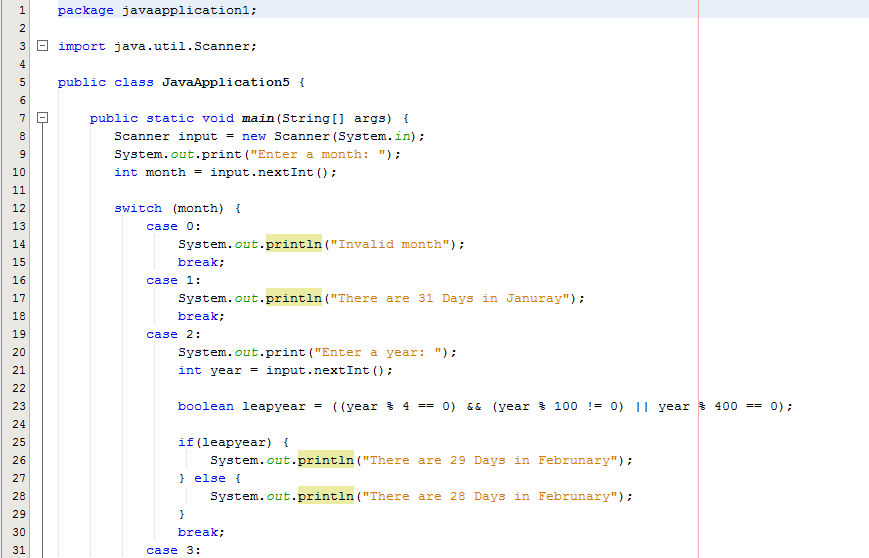
Enter a month: 6

There are 30 Days in June

Enter a month: 2

Enter a year: 2012

There 29 Days in February



Task 3-

Math.random() is a method in the Java library that computes a random **double** value between 0 and 1.

For example:

double x = Math.random();

assigns to the variable x a random **double** between 0 and 1. Write a program that tests how well **Math.randon()** works. Write a program that calls **Math.random()** 1,000 times to create 1,000 values. Store the values that are greater than 0.3 and .7 in a LinkedList and then print all the elements in the LinkedList. Store values other values in an ArrayList and then print all the values.

