Algorithm Formats

Flowchart
A flowchart models the processing flow of a module, program, or system. It is a graphical representation of the steps that describe the processing being performed.

Symbols and Semantics
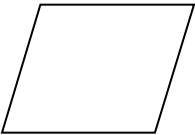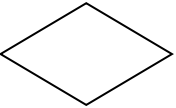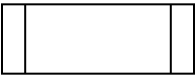A flowchart is a graph of connected symbols that describe the processing flow (i.e., control flow) of one module, program or system. The symbols used on a flowchart represent flow, terminal, processing step, input/output, decision, subroutine call, and connectors. An algorithm described using a flowchart starts and ends with a terminal symbol.
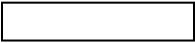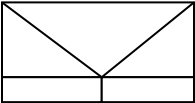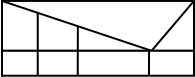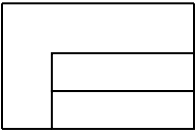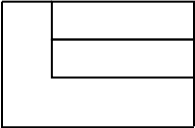
Symbol	Semantics
⟶	A *flow* is a directed line that connects two symbols within the flowchart. • The direction of the arrow indicates the processing (i.e., execution) sequence within the flowchart.
(oval)	A *terminal* is an oval labeled with the word "start", "stop", "enter", or "exit". • While "start" and "enter" have similar meaning (likewise "stop" and "exit"), convention dictates that "start" and "stop" be used when the flowchart represents the main processing of the module, program or system, and "enter" and "exit" be used when the flowchart represents a sub-processing element (i.e., subroutine) of the module, program, or system. • The "start" and "enter" terminal symbols cannot have an inbound flow and must have only one outbound flow. • The "stop" and "exit" terminal symbols can have one or more inbound flows but no outbound flows.
(rectangle)	A *processing step* is a rectangle labeled with a verb-noun-phrase that describes the processing performed at this step of the flowchart. • A processing symbol can have one or more inbound flows but must have only one outbound flow. • After the verb-noun-phrase is performed, processing continues to the symbol pointed to by the outbound flow.
(parallelogram)	An *input/output* is a parallelogram labeled with a verb-noun-phrase that describes an input or output processing statement. • An input verb-noun-phrase (e.g., "read number", "get name") indicates that data is obtained from an external entity. For example, input data may come from a user typing on the keyboard or from a data file. • An output verb-noun-phrase (e.g., "write number", "display name") indicates that data is given to an external entity. For example, output data may be displayed to a user or written to a data file. • An input/output symbol can have one or more inbound flows but must have only one outbound flow. • After the verb-noun-phrase is performed, processing continues to the symbol pointed to by the outbound flow.
(diamond)	A *decision* is a diamond labeled with a question (aka: test, condition) that must result in a yes/no or true/false answer. • A decision symbol can have one or more inbound flows but must have exactly two outbound flows. • One outbound flow is labeled with "yes" or "true" while the other outbound flow is labeled with "no" or "false". • As a result of processing a decision symbol, flow continues to one of the two outbound flows based on the answer to the question.
(subroutine)	A *subroutine call* is a rectangle with rectangular-sides labeled with a subroutine name that identifies a separate set of processing steps to be performed. • A subroutine call symbol can have one or more inbound flows but must have only one outbound flow. • After the subroutine is performed, processing continues to the symbol pointed to by the outbound flow.
(connector)	A *connector* is a circle or boxed-arrow labeled with a letter that identifies a separate set of processing steps to be performed. • A connector symbol can either have one or more inbound flows or one outbound flow.

Nassi–Shneiderman diagram

A Nassi-Shneiderman Diagram (NSD) models the processing flow of a module, program, or system. It is a graphical representation of the steps that describe the processing being performed.

Symbols and Semantics

A NSD is a series of nested sequence, selection, and iteration symbols within a rectangle that describe the processing flow of one module, program or system. The nested symbol at the top of the rectangle is processed (executed) first, followed by the nested symbol immediately below the top symbol, and so on, until the symbol at the bottom of the rectangle has been processed, at which point the algorithm ends.

Symbol	Semantics
	A *sequence* is a rectangle that contains a verb-phrase. After the verb-phrase is performed, flow continues to the next nested symbol in the NSD.
	A *selection* is a "back-of-an-envelope" shape used to represent a question (aka: test, condition) that usually result in a yes/no or true/false answer. - When the answer to the question is yes/true, the nested symbols underneath the yes/true are processed. - When the answer to the question is no/false, the nested symbols underneath the no/false are processed. - Alternatively, a selection may include more than two answers; in which case only one of the answers can be true at a given time (the shape looks like a lopsided back-of-an-envelope). The nested symbols underneath the answer-that-is-true are processed. - After the nested symbols underneath the answer-that-is-true are performed, flow continues with the next nested symbol contained within the NSD.
	An *iteration* is a "carpenter's square" shape used to represent a loop statement. - There are two iteration symbols in NSD: - A loop where the question (aka: test, condition) is tested before the loop body - A loop where the question is tested after the loop body. - When the question results in yes/true the nested symbols inside the "carpenter's square" (i.e., the loop body) are executed again. - When the question results in no/false the iteration ends and the nested symbol that follows the iteration symbol is then processed.

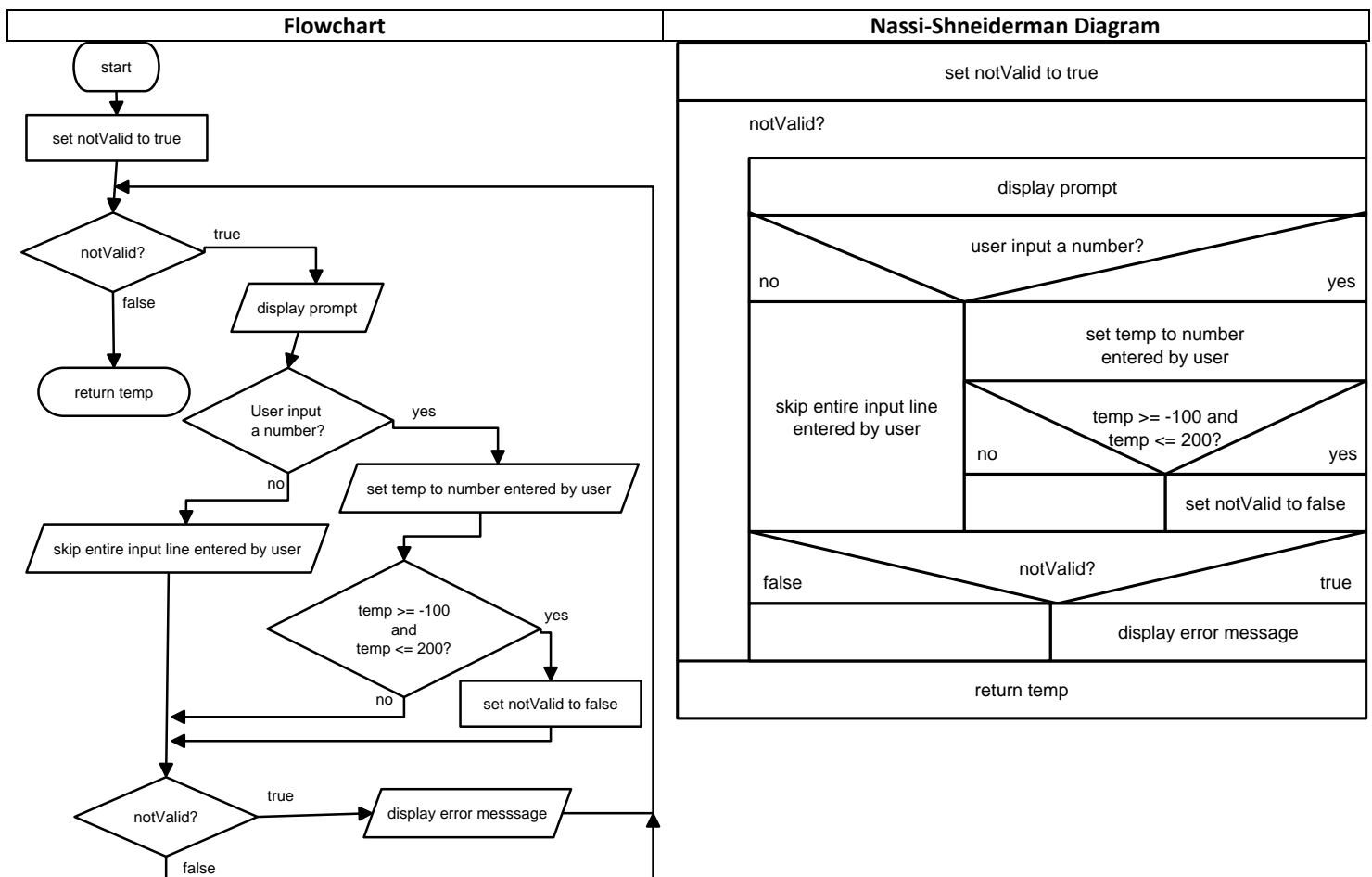## *Examples: Iteration and Selection Structures*

Version 1

Obtain a Fahrenheit temperature value from user, ensuring that the temperature is a number in the range [-100, 200].

```java
public double getFahrenheitTemp(Scanner Scan)
{
    final String prompt = "Enter a Fahrenheit temp: ";
    double temp;
    boolean notValid = true;

    while (notValid)
    {
        System.out.print(PROMPT);
        if (scan.hasNextDouble())
        {
            temp = scan.nextDouble();
            if (temp >= -100 && temp <= 200)
                //temp value entered is valid!
                notValid = false;
        }
        else
            //Skip entire line of input entered by user.
            scan.nextLine();

        if (notValid)
            System.out.println("Error! Must enter a numeric temperature in range [-100, 200].");
    }

    return temp;
}
```

Flowchart	Nassi-Shneiderman Diagram

Version 2

Obtain a Fahrenheit temperature value from user, ensuring that the temperature is a number in the range [-100, 200].

```java
public double getFahrenheitTemp(Scanner Scan)
{
    final String prompt = "Enter a Fahrenheit temp: ";
    double temp;
    boolean notValid = true;

    do
    {
        System.out.print(PROMPT);
        if (scan.hasNextDouble())
        {
            temp = scan.nextDouble();
            if (temp >= -100 && temp <= 200)
                //temp value entered is valid!
                notValid = false;
        }
        else
            //Skip entire line of input entered by user.
            scan.nextLine();

        if (notValid)
            System.out.println("Error! Must enter a numeric temperature in range [-100, 200].");
    }
    while (notValid);

    return temp;
}
```

Flowchart	Nassi-Shneiderman Diagram