COMPUTER PROGRAMMING I

INFOSYS 112

NOR-AINE M. CORPUZ
DANILYN A. FLORES

2021

UNIVERSITY OF SOUTHERN MINDANAO
Kabacan, Cotabato

COMPUTER PROGRAMMING I

INFOSYS 112

NOR-AINE M. CORPUZ
DANILYN A. FLORES

2021

UNIVERSITY OF SOUTHERN MINDANAO
Kabacan, Cotabato

Author's Declaration

Ideas, concepts, diagrams and/or illustrations depicted in this learning material are excerpts from established references and properly noted in the list of literatures cited herein. The author in this learning material does not claim full and authentic ownership of all the contents of this material, nor in any manner willfully infringe the copyright law and other existing provisions appertaining thereto.

This learning material is printed for the sole use of classroom or distance/remote learning of USM and is not intended for commercial purposes. Any use or reproduction in part or in full, whether electronic or mechanical, photocopying or recording in any information storage and retrieval system, other than what it is intended for requires the consent of authorized and competent authority of the University of Southern Mindanao.

USM VISION

Quality and relevant education for its clientele to be globally competitive, culture sensitive and morally responsive human resources for sustainable development

USM MISSION

Help accelerate socio-economic development, promote harmony among diverse communities and improve quality of life through instruction, research, extension and resource generation in Southern Philippines.

UNIVERSITY QUALITY POLICY STATEMENT

The University of Southern Mindanao, as a premier university, is committed to provide quality instruction, research development and extension services and resource generation that exceed stakeholders’ expectations through the management of continual improvement efforts on the following initiatives.

1. Establish Key Result Areas and performance indicators across all mandated functions;
2. Implement quality educational programs;
3. Guarantee competent educational service providers;
4. Spearhead need-based research outputs for commercialization, publication, patenting, and develop technologies for food security, climate change mitigation and improvement in the quality of life;
5. Facilitate transfer of technologies generated from research to the community for sustainable development;
6. Strengthen relationship with stakeholders;
7. Sustain good governance and culture sensitivity; and
8. Comply to customer, regulatory and statutory requirements.

PREFACE

In the field of science and engineering, skill in computer programming is a must-have. This book is an introductory computer programming textbook that uses Java as the language of instruction. This is written to aid students in their journey in learning the basics of programming.

This module is divided into five chapters.

- Chapter 1 talks about Introduction to Computer programming. In this chapter computer programming is defined. Traits of a good programmer is discussed. Good programming practices and qualities of a good program is elaborated.
- Chapter 2 discusses Programming Fundamentals. In this chapter Program Development Life Cycle, Comments, Tokens, Separators, Identifiers, Keywords, Literals, Data Types, Variables, Operators, Expressions, Statements and Blocks is explained.
- Chapter 3 users of this module are introduced to Inputs ang Outputs. It is in this chapter where they are going to learn how to create simple programs, displaying output in console, and displaying output and getting input using graphical user interface.
- Chapter 4 is about Control structures, decision and repetition control structures.
- The last chapter talks about Arrays. In this chapter topics including Introduction to Arrays, Declaring Arrays, Accessing Array Element, Array Length and Multidimensional Arrays are explained.

Every chapter of this module have intended learning outcomes for students to be well guided to what they are going to learn. A few example programs written in Java programing language are presented in every chapter. Sample programs used in this book are edited and compiled using Jcreator Pro 4.5. At the end of each chapter, the authors included two summative tests, an assignment, and a quiz. This is to quantify the level of learning each student acquired in every chapter of this module.

TABLE OF CONTENTS

COURSE GUIDE

Course Information

Course Title	**Computer Programming I**
Course Code	**InfoSys 112**
Pre-requisite/Co-requisite	**None**

Course Description

This course covers the use of general purpose programming language to solve problems. The emphasis is to train students to design, implement, test, and debug programs intended to solve computing problems using fundamental computing constructs.

Computer Programming 1 enables students to learn the important basic programming concepts like variables, data types, operators, and control structures, and apply these using the Java programming language. As an introduction, this course gives an overview of the different problem-solving strategies that will help them analyze problems to create programs throughout the course.

The course helps students understand the processes and the significance of making programs, its implementation, application, and significanceto the industry, and the community. Also, this course helps give students a deep understanding and developed skills of all the basic concepts of programming and effectively utilize the language and programming structures in program-making that will helpthem learn other and new programming languages for them to be competent individuals.

Course Objectives/Outcomes

Upon passing the course, you must be able to:

1. Discuss the different concepts and techniques used in creating computer programs.
2. Use appropriate programming components in designing a computer program for a given problem.
3. Create computer programs using the Java programming language for given problems.

Course Learning/Study Plan/Schedule

Week/Date (Deadlines)	Topic	Teaching and Learning Activities	Learning Materials	Assessment
1	**Introduction to Computer Programming** • What is Computer Programming?	Interactive Learning Brainstorming Independent Learning	Downloadable Slides/Videos Computer/Mobile Phone	Case Study Quiz

	• Traits of a Good Programmer • Good Programming Practices • Qualities of a Good Program		Module	
2 - 5	**Programming Fundamentals** • Program Development Life Cycle • Comments • Tokens • Separators • Identifiers • Keywords • Literals • Data Types • Variables • Operators • Expressions • Statements • Blocks	Interactive Learning Brainstorming Independent Learning	Downloadable Slides/Videos Computer/Mobile Phone Module	Problem Solving Quiz
6 - 8	**Inputs and Outputs** • Getting to Know the Software • Creating Simple Programs • Displaying Output on Console • Getting Input and Displaying Output using Graphical user Interface	Interactive Learning Brainstorming Independent Learning	Downloadable Slides/Videos Computer/Mobile Phone Module	Problem Solving Quiz
9	**MIDTEM EXAM**			
10 - 12	**Control Structures**	Interactive Learning	Downloadable Slides/Videos	Problem Solving Quiz

	Decision Control Structures • if • if-else • if-else-if • switch	Brainstorming Independent Learning	Computer/Mobile Phone Module	
13 - 15	**Control Structures** Repetition Control Structure • while • do-while • for	Interactive Learning Brainstorming Independent Learning	Downloadable Slides/Videos Computer/Mobile Phone Module	Problem Solving Quiz
16 – 17	**Arrays** • Introduction to Arrays • Declaring Arrays • Accessing Array Element • Array Length • Multidimensional Arrays	Interactive Learning Brainstorming Independent Learning	Downloadable Slides/Videos Computer/Mobile Phone Module	Problem Solving Quiz
18	**FINAL EXAM**			

Course Requirements/Assessment and Evaluation Scheme/Grading System

MIDTERM	
Requirement/ Assessment Task	**Percentage**
Attendance	10
Quizzes	25
Summative Assessments	30
Midterm Exam	35
TOTAL	**100%**

FINAL TERM	
Requirement/ Assessment Task	**Percentage**
Attendance	10
Quizzes	25
Summative Assessment	30
Final Exam/Project	35
TOTAL	**100%**

FINAL GRADE	
Midterm	50%
Final Term	50%
TOTAL	**100%**

House Rules/Class Policies

1. Students are REQUIRED to enroll in the course VLE where all course materials are uploaded and course requirements are submitted.

2. Students are also REQUIRED to be in the course Facebook Group Chat where all announcements and reminders will be posted.
3. Each student is encouraged to have a computer, either a desktop or a laptop, and install the required or related software needed for convenience in doing tasks required for the course.
4. Students are enjoined to attend the online live lectures. If a student cannot attend the lectures, the recording will be posted on the course VLE for download and viewing offline.
5. Students should wear appropriate clothing while attending the online live lecture.
6. Plagiarism is highly discouraged and will not be taken lightly. Tasks are required to be submitted together with a plagiarism receipt using any online plagiarism checker. The instructor will also manually check the authenticity of submissions.
7. Tasks should be submitted on time. Submissions have extended deadlines considering resources especially hardware and internet speed.
8. Students should inform their instructor on the time allocated or within 5 days after the time allocated for a quiz/exam to explain why they cannot take or have not taken the quiz/exam for them to be given considerations.
9. Students are encouraged to communicate with their instructor for consultations via any media the instructor will provide from Monday-Friday between 8AM-5PM only.

CHAPTER 1
Introduction to Computer Programming

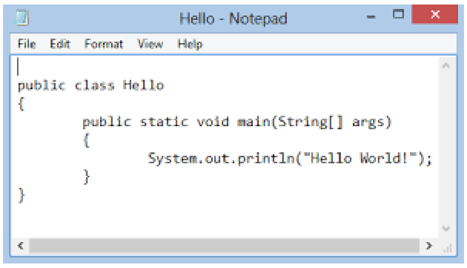
Intended Learning Outcomes

By the end of this topic/chapter, you must be able to:

1. Describe computer programming terms
2. Discuss the traits of a good programmer
3. Discuss the good programming practices
4. Discuss the qualities of a good program

What is Computer Programming?

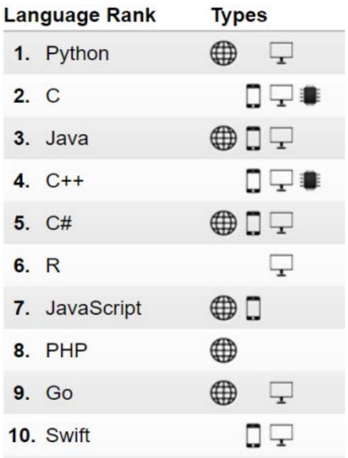
A **program** is a set of instructions written in a language understandable by the computer to perform a particular function on the computer. Fig. 1 shows a screenshot of a simple code in Java that displays *Hello World!*.



```
public class Hello
{
        public static void main(String[] args)
        {
                System.out.println("Hello World!");
        }
}
```

Fig. 1. Sample Program
Image Source:
http://javalanguageprogramming.blogspot.com/2013/03/first-java-program-hello-world.html

A **computer programmer** is a professional skilled in using constructs of programming languages to develop executable and acceptable computer programs. A computer programmer can be as young as 7 years old (as of 2016) or as old as 82 years old (as of 2017).



Language Rank	Types
1. Python	🌐 🖥
2. C	📱 🖥 ▦
3. Java	🌐 📱 🖥
4. C++	📱 🖥 ▦
5. C#	🌐 📱 🖥
6. R	🖥
7. JavaScript	🌐 📱
8. PHP	🌐
9. Go	🌐 🖥
10. Swift	📱 🖥

Fig. 2. Top 10 Programming Languages 2017
Image Source:
http://blog.revolutionanalytics.com/2017/07/ieee-spectrum-2017-top-programming-languages.html

A **Programming Language** is an artificial language created to be used in preparing coded instruction on the computer for later execution by the computer. It is a standardized communication technique for expressing instructions to a computer. It enables a programmer to precisely specify what data a computer will act upon, how these data will be stored or transmitted and precisely what actions to take under various circumstances. It is usually composed of a series of usage rules (syntax) that determine the meaning (semantics) of expressions written in the language. Each programming language comes hand in hand with its own translator like an interpreter or compiler as the case may be. Fig. 2 shows the 2017 Top 10 Programming

Languages of IEEE Spectrum, a magazine edited by the Institute of Electrical and Electronics Engineers.

Programming is the art of developing computer programs with the aid of selected programming language by a computer programmer to enable a computer to do a certain task. In programming, programming stages must be properly followed (Usman, Owoade, Abimbola, and Ogunsanwo, 2016).

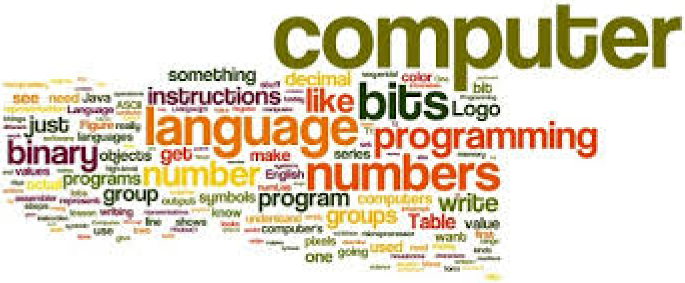


Fig. 3. Computer Programming
Image Source: http://guyhaas.com/bfoit/itp/Programming.html

Brief History of Programming

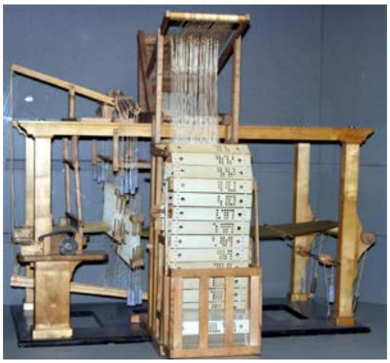


Fig. 4. Jacquard Loom
Image Source:
http://addiator.blogspot.com/2011/10/jacquards-loom-and-stored-programme.html

Many people would agree that computers have a huge impact on our lives, and computer programs tell those computers what to do and how to do it. With both good and bad effects, computer programs have altered our existence. So it is only right to at least know a little something on the history of programming. This brief history of programming is based from the illustration of Ellie Koning (image not included here because it is too long it won't fit the page but you can view it at https://visual.ly/community/infographic/technology/history-computer-programming).

You might not be aware that "pre-computers" did not use electricity back then or that the first computer programmer was a woman. Pre-computers existed in as early as 1801. An example is the **Jacquard loom** (Fig. 4), invented by Joseph Marie Jacquard, which is a mechanical automated loom that worked by creating different patterns using changeable punched cards that controlled its operation. These punch cards was later used by **Charles Babbage** (Fig. 5), considered the "Father of Computer", as a method for storing programs for his Analytic Engine.

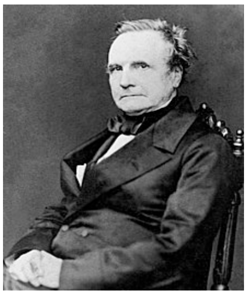


Fig. 5. Charles Babbage
Image Source:
https://en.wikipedia.org/wiki/Charles_Babbage

In 1842-1843, a woman translated an article about Charles Babbage's proposed Analytic Engine and described an algorithm that was cited as the first computer program making her the first computer programmer. She also theorized that computers could one day play music and chess. The United States Department of Defense named a computer language in her honor. Her name is **Augusta Ada Byron** (Fig. 6), Countess of Lovelace.

Fig. 6. Ada Byron
Image Source: https://en.wikipedia.org/wiki/Ada_Lovelace

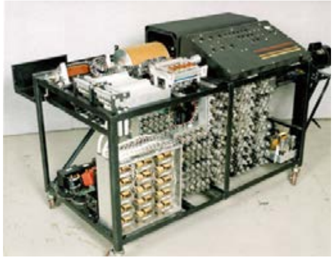


Fig. 7. The ABC
Image Source:
https://fahmirahman.wordpress.com/2011/04/19/inventors-of-the-modern-computer-2/

In 1889, **Herman Hollerith** used punched cards, known as Hollerith cards, after experimenting with paper tape. He also invented the tabular and keypunch machines along with the Hollerith cards which formed the basis for information processing. In 1896, he founded the Tabulating Machine Company which later became IBM. In 1906 he developed a plugboard or control panel that allowed the machine to perform different tasks without being reconstructed.

Developed in 1937 and tested in 1942, the first electronic digital computer, the **Atanasoff-Berry Computer** or ABC (Fig. 7) was designed to solve linear equations. It was not programmable but it did use binary arithmetic.

In World War II, the Colossus machines were the **first programmable electronic digital computer**. They were designed to break and read encrypted German messages.

In 1947, **Grace Murray Hopper**, a computer programming pioneer, documented the first actual computer '**bug**' (Fig. 8) when a moth got trapped in the Mark II Alken Relay Calculator. They removed the moth and 'debugged' the computer.

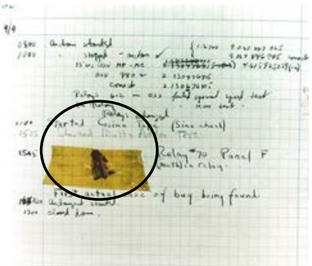


Fig. 8. 1st Computer Bug
Image Source:
https://www.atlasobscura.com/places/grace-hoppers-bug

The Electronic Delay Storage Automatic Calculator, or **EDSAC**, was the first practical stored-program electronic computer. On May 6, 1949, EDSAC ran its first program: calculating a table of squares and a list of prime numbers.

Fortran, the first high level computer programming language, was invented by John Backus in 1954 and was released commercially in 1957.

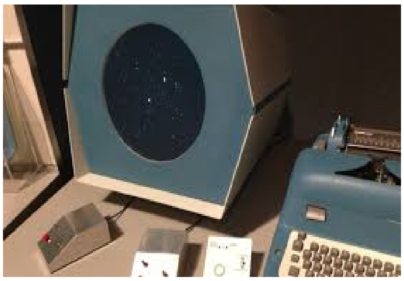


Fig. 9. SpaceWar!
Image Source:
http://www.syfy.com/syfywire/firsts-spacewar-was-the-worlds-first-video-game

The first computer game, **Spacewar!** (Fig. 9), was programmed by Steve Russell in 1961. It is a two-player game where two spaceships, affected by the gravity of a star, fire missiles, unaffected by gravity, at each other. Each spaceship has limited ammunition and fuel. It took roughly 200 man-hours to complete the game which was written on a DEC interactive minicomputer. At Stanford University, Russel introduced Nolan Bushnell to Spacewar! where Bushnell later on went to program the first coin-operated arcade game.

The first computer virus programmer was **Fred Cohen** in 1983. He designed a hidden program that could infect a computer, copy itself, and then infect other computers through the use of a floppy disk. The program was benign, meant only to prove that it was possible.

And here ends the story of computer programming's humble beginning. Though it can be realized that a lot has changed since then, it can only be imagined what more our current and future technology can offer in the field of programming.

Traits of a Good Programmer

So you know how to code. That does not make you a good programmer; that is just one of the traits of a good programmer. Programmers aren't born in a day. It is a product of constant learning and practice. Being a good programmer is not just all coding. To understand this, here is a list of what a good programmer's traits should be by Katie Bouwkamp, currently a Sr. Communications Manager at Smartsheet, in 2017.

1. **Be well rounded.** It's great to know one technology in depth, but problems in the real world are never solved with just a single technology. Even if you're hired as a specialist, you still need to understand how your tech interacts with the other software, hardware, and network that make up the application's ecosystem. You'll also be able to contribute to your project in multiple ways, helping out wherever more assistance is needed.
2. **Enjoy solving puzzles.** Building applications is not a straightforward process. Figuring out why code isn't compiling, what's causing bugs, and how to solve production problems requires puzzle-solving skills, as well as the belief that there's always a solution and not giving up until it's found. If you can solve puzzles under pressure, that's even better—when the system's down, you can expect management to be breathing down your neck while you figure it out.
3. **Love learning.** Technology is constantly changing. The tools and languages you work with today are not the tools you'll be working with next year, let alone next decade. You need to always be developing new skills to be able to contribute to upcoming projects. Your employer may provide ongoing training, but the best developers take time to learn on their own.
4. **Good communication skills.** Working as a developer isn't just about technology. Developers need to talk with business users to understand what they need from the application. Developers also often need to

generate technical documents, so being able to write clearly is also important—even if it's just to produce a status report.

5. **Confidence.** There's never just one way to build a system. No matter how good your ideas are, they won't have value if you keep them to yourself. The best developers have confidence in their ideas and speak up in design discussions to help shape the application architecture. To boost your confidence, start with a small suggestion, rather than proposing an entire application redesign.
6. **Be interested in the business.** Businesses use technology to solve business problems. The more you understand about your company's business, the better prepared you are to understand their problems and build solutions that help them grow. You should take advantage of opportunities to talk to the business users and ask them questions about the challenges they face in their work. If you get really interested in understanding the business, you can take courses or even work towards certifications in the business domain.
7. **Be a team player.** Movies often glorify a solo coder, and students usually work on assignments on their own, but real-world projects are team efforts. It's important for developers to be able to get along with co-workers. You need to be able to deal with people with varying abilities and respond to differences of opinion respectfully. If you can, get to know your teammates as people, not just technical staff. Having conversations about other things than the project helps form relationships that make working together easier.
8. **Understand the importance of deadlines.** The best project managers will get their developers' input when coming up with project deadlines, though sometimes external factors drive the schedule. In either case, once you've agreed to do a task, do your best to meet the deadline, even if it means a few late nights. You don't have to give up your whole personal life for the organization, but demonstrating commitment to completing the project and understanding its value to the business make a positive impression at work.
9. **Be adaptable.** Projects and priorities change for many reasons, and developers need to be able to context-switch to focus on what's most important right now. The changes may be small and temporary or major and permanent. If they're temporary, make sure you have good notes that will help you get back to your regular work when things settle down. If the changes are permanent, allow time to understand what the new situation is and how you fit into it. It may present new opportunities to

help you achieve your goals. In any case, it’s important to respond professionally and not lash out in frustration.

10. **Own your product.** Technically, your job may be done once you’ve written code that compiles cleanly and passes its test cases. Stand out by following it through the rest of its lifecycle—be willing to help with testing, deployment, training users, and solving production problems. While this helps your business, it actually helps you more, because you see and understand the real-world effectiveness of the code you wrote. Then take that understanding and let it help you make your next application even better.

Good Programming Practices

Programmers do not code for themselves. And though end users are the ones who actually use the software that programmers code, there are still cases that the software needs maintenance and modification. With this, codes should be easy to read and understand so that it will be easy to modify, debug, and maintain, especially when the original programmer is not around anymore. Below is a list of 10 good programming practices any programmer can follow from Burak Guzel (2011) who is a full time PHP Web Developer and Andrey Nikishaev (2017) who is a software developer for 15 years.

1. **Commenting and Documentation**
   Write simple inline documentation describing what and how the code works. This will save much time for understanding and even more.
2. **Consistent Indentation**
   Codes should be indented and it is a good idea to keep indentation style consistent.
3. **Avoid Obvious Comments**
   Commenting code is fantastic but it can sometimes be overdone or become redundant. When the text is that obvious, it is not productive to repeat it within comments.
4. **Code Grouping**
   Certain tasks require a few lines of code. It is a good idea to keep these tasks within separate blocks of code, with some spaces between them. Adding a comment at the beginning of each block of code emphasizes the visual separation.
5. **Consistent Naming Scheme**
   Names should have word boundaries either using **camelCase** where the first letter of a word is capitalized (e.g. helloWorld) or **underscores** where underscores are placed between words (e.g. hello_world). Whatever the case may be, consistency is the key.
6. **DRY Principle**
   Don’t Repeat Yourself also known as DIE: Duplication is Evil. The same piece of code should not be repeated over and over again.
7. **Avoid Deep Nesting**
   Too many levels of nesting can make code harder to read and follow.

8. **Limit Line Length**
   Our eyes are more comfortable when reading tall narrow columns of text like in the newspaper. It a good practice to avoid writing horizontally long lines of code.
9. **File and Folder Organization**
   An entire application code can be written within a single file. But that would prove to be a nightmare to read and maintain. Imitating a folder structure can be one approach.
10. **Keep It Simple, Stupid (KISS Principle)**
    Do not write complex code. The simpler the code the less bugs it may have and the less time will be needed to debug them.

In addition to the above mentioned, Nikishaev (2017) also suggested for programmers to **go out and get hobbies** since work differentiation increases mental abilities and gives new fresh ideas, and **learn new things as you get free time** because when people stop learning they start to degrade.

Qualities of a Good Program

Whatever the approach to development may be, the final program must satisfy some fundamental properties. Just because the code works does not mean it is a good program. The following properties are among the most important.

1. **Readability:** how well the code can be understood by other programmers
2. **Reliability:** how often the results of a program are correct. This depends on conceptual correctness of algorithms, and minimization of programming mistakes, such as mistakes in resource management (e.g., buffer overflows and race conditions) and logic errors (such as division by zero or off-by-one errors).
3. **Robustness:** how well a program anticipates problems due to errors (not bugs). This includes situations such as incorrect, inappropriate or corrupt data, unavailability of needed resources such as memory, operating system services and network connections, user error, and unexpected power outages.
4. **Usability:** the ergonomics of a program: the ease with which a person can use the program for its intended purpose or in some cases even unanticipated purposes. Such issues can make or break its success even regardless of other issues. This involves a wide range of textual, graphical and sometimes hardware elements that improve the clarity, intuitiveness, cohesiveness and completeness of a program's user interface.
5. **Portability:** the range of computer hardware and operating system platforms on which the source code of a program can be compiled/interpreted and run. This depends on differences in the programming facilities provided by the different platforms, including hardware and operating system resources, expected behavior of the hardware and operating system, and availability of platform specific compilers (and sometimes libraries) for the language of the source code.

6. **Maintainability:** the ease with which a program can be modified by its present or future developers in order to make improvements or customizations, fix bugs and security holes, or adapt it to new environments. Good practices during initial development make the difference in this regard. This quality may not be directly apparent to the end user but it can significantly affect the fate of a program over the long term.
7. **Efficiency/performance:** Measure of system resources a program consumes (processor time, memory space, slow devices such as disks, network bandwidth and to some extent even user interaction): the less, the better. This also includes careful management of resources, for example cleaning up temporary files and eliminating memory leaks.

Formative Assessment

1. List the Top 10 Programming Languages of 2021

Chapter 1 Summary

Programming is the art of writing a set of instructions called a program by a skilled person called a programmer using a particular programming language which is a technique to give instructions to a computer on what to do that does anything that can address a given problem.

Programming started way back 1801 with pre-computers and the first programmer who is a woman created an algorithm for the father of computer. Then the firsts happened; from the first computer game to the first computer bug to the first high level programming language and first programmable computer.

To understand the world, understand processes, understand the influence of computers to the world, use a new form of literacy, have a new way to learn science and mathematics, have an additional job skill, use computers better, and to learn problem solving are just some of the many reasons to learn programming and a good programmer should be well rounded, enjoys solving puzzles, loves learning, has good communication skills, is confident, interested in the business, a team player, understands the importance of deadlines, adaptable, and owns the product.

Good programming practices include commenting, indentation, code grouping, consistent name scheming, DRY principle, avoid deep nesting, limit line length, organize files and folders, and KISS principle which will in turn result in a good program that is readable, reliable, robust, usable, portable, maintainable, and efficient.

For now, these terms and concepts may not sink in at first. But as each of the next chapters are discussed, these terms and concepts will be slowly understood and applied.

Summative Assessment

1. Case Study (See VLE)
2. Chapter 1 Quiz (See VLE)

Chapter 1 References

[1] Guzel, B. (2021). *Top 18 Best Practices for Writing Super Readable Code*. Retrieved 21 July 2021, from https://code.tutsplus.com/tutorials/top-15-best-practices-for-writing-super-readable-code--net-8118

[2] Sharma, V. (2018). *Why is Programming Important* - KLIENT SOLUTECH. Retrieved 21 July 2021, from http://www.klientsolutech.com/why-is-programming-important/

[3] Woodward, J., & Fayed, M. (2016). *Why everyone should have to learn computer programming*. Retrieved 21 July 2021, from http://theconversation.com/why-everyone-should-have-to-learn-computer-programming-62328

CHAPTER 2
Programming Fundamentals

Intended Learning Outcomes

By the end of this topic/chapter, you must be able to:

1. Explain what happens at each step of the PDLC
2. Design a simple algorithm of a given problem
3. Describe the different ways to express an algorithm
4. Draw the different flowcharting symbols
5. Explain the different types of errors
6. Identify the different types of comments in a computer program
7. Explain the different types of tokens
8. Choose the proper data types for given values
9. Recall the rules in naming identifiers
10. Use different operators
11. Evaluate the result of expressions using different operators
12. Creating programming expressions from regular formulas
13. Create variables
14. Differentiate expressions, statements, and blocks

In order to create programs with ease, the basics must be fully understood. In this chapter, steps on how to develop programs are described and the different elements of a program are defined.

Program Development Life Cycle (PDLC)

Programmers do not just sit in front of their computers and write code. There has to be a problem that needs to be addressed using a program; it can be anything like process improvement, to understand the world more or to just help people use computers better. But even if there is a problem, the programmer cannot just go to writing codes just yet. Programmers have to follow a series of steps called the Program Development Life Cycle to ensure that the program written can indeed address the given problem. The steps of the PDLC are discussed below.

STEP 1. Problem Analysis

The programmer has to analyze the given problem first and here are the things that needs to be done:

- ✓identify the desired results (output),
- ✓determine what information (input) is needed to produce these results; and
- ✓figure out what must be done to proceed from the known data to the desired output (process).

STEP 2. Program Design

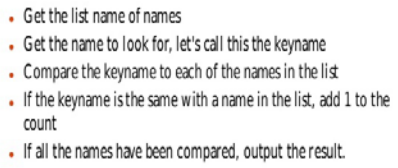
After identifying the required inputs, outputs, and processes, the programmer designs the algorithm by representing it in different ways.

An **algorithm** is a step by step process for solving a problem or doing a task. It can be represented in three (3) ways.

Ways to Represent an Algorithm

a) Human Language

When using human language, the programmer can use any language/dialect that can be understood by everyone. The algorithm can be represented using numbered description of the process or ordered description of a process such as in Fig. 10.



- Get the list name of names
- Get the name to look for, let's call this the keyname
- Compare the keyname to each of the names in the list
- If the keyname is the same with a name in the list, add 1 to the count
- If all the names have been compared, output the result.

Fig. 10. Algorithm in Human Language
Image Source:
https://www.slideshare.net/annisasabrinabatrisyia/programming-1-compatibility-mode

b) Pseudocode

A pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It is a combination of human language and programming language.

c) Flowchart

A flowchart, introduced by Frank and Lillian Gilberth in 1921, is a diagrammatic method of representing algorithms like what is shown in Fig. 11. They use an intuitive scheme of showing operations using different symbols shown in Table 1 that graphically show the flow of control in an algorithm.

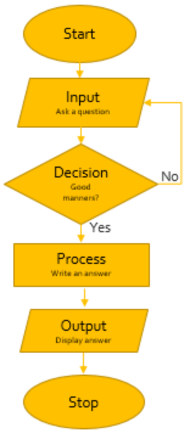


Fig. 11. Flowchart Sample

Table 1. Basic Commonly Used Flowcharting Symbols

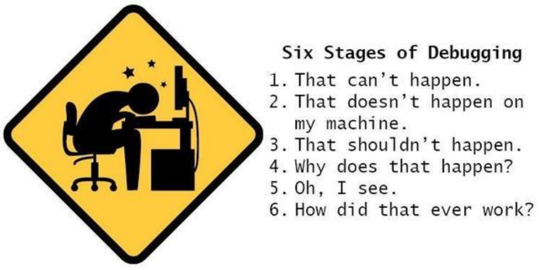
Symbol	Name	Description
→	Flowline	Shows the process's order of operation
⬭	Terminal	Indicates the beginning and ending of a program or sub-process
▭	Process	Represents a set of operations that changes values of data.
◇	Decision	Shows a conditional operation that determines which one of the two paths the program will take.
▱	Input/Output	Indicates the process of inputting data and displaying results.
○	Connector	Pairs of labelled connectors replace long or confusing lines on a flowchart page

STEP 3. Program Coding

Coding is when the programmer writes the source code or statements (instructions) of a program like in Fig. 12 in a particular programming language using different programming software after creating an algorithm. The programmer will have a hard time writing the code if the algorithm is not designed well but if the algorithm design is clear, then the programmer can code it with ease. Coding takes time; a lot of time especially for large programs. In this step, the programmer should keep in mind the good programming practices and the qualities of a good program discussed in the previous chapter.

Fig. 12. Coding
Image Source:
http://www.ghanalive.tv/2017/03/29/want-computer-programmer/

STEP 4. Program Testing



Six Stages of Debugging

1. That can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?

Fig. 13. Debugging
Image Source:
https://twitter.com/rbeenhakker/status/2926589350124134
40

Testing the program ensures that it is free of errors a.k.a **bugs** and that it does indeed solve the given problem.

Debugging is the term used for the process of fixing the errors in the source code of a program.

A programmer may encounter many bugs during debugging and can go through the stages of debugging shown in Fig. 13. Listed below are the different types of errors:

1. **Syntax Error**
   This type of error occurs from grammar errors in the use of the programming language such as a misspelled variable, missing semicolons, improperly matched parentheses, brackets, and braces and incorrect format in selection and loop statements

2. **Logic Error**
   This type of error occurs when there is a design flaw in the program where the output is not what is expected such as multiplying when it should be dividing, adding when it should be subtracting, opening a data from the wrong file, displaying the wrong message.

3. **Runtime Error**
   This type of error occurs when a program with no syntax error asks the computer to do something that the computer is unable to reliably do such as trying to divide by a variable that contains a value of zero or trying to open a file that doesn't exist.

After debugging, the programmer proceeds to testing the program further using different sets of inputs to ensure that the program produces correct and accurate output, functions as expected, and solves the problem. If not, the programmer goes back to Step 1 and repeats the process all over again until the problem is solved.

EXAMPLE 1
Problem: Solve for the sum of 2 numbers.

Solution (Following PDLC)
1. PROBLEM ANALYSIS

A problem can be addressed in many different ways. It just has to be analyzed properly. For this problem, a minimum of 2 programs can be created.

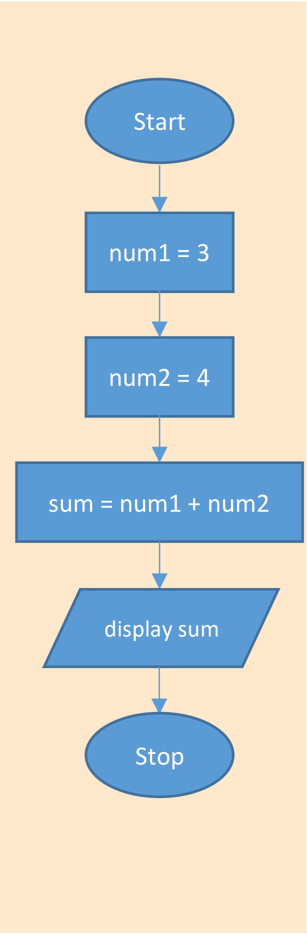
Analysis 1		Analysis 2	
Input	None	Input	num1, num2
Process	num1 = 3, num2 = 4, sum = num1+num2	Process	sum = num1 + num2
Output	sum	Output	sum

- For Analysis 1, the values of the 2 numbers to be added are determined while on Analysis 2, the numbers are not defined and will depend on user input.
- Programs have to be general (can solve a problem using any given values) so Analysis 2 is better. Analysis 1 is okay if the values are specified in the problem.
- Other analysis can also be created from of the problem like including num1 and num2 in the output.

2. PROGRAM DESIGN

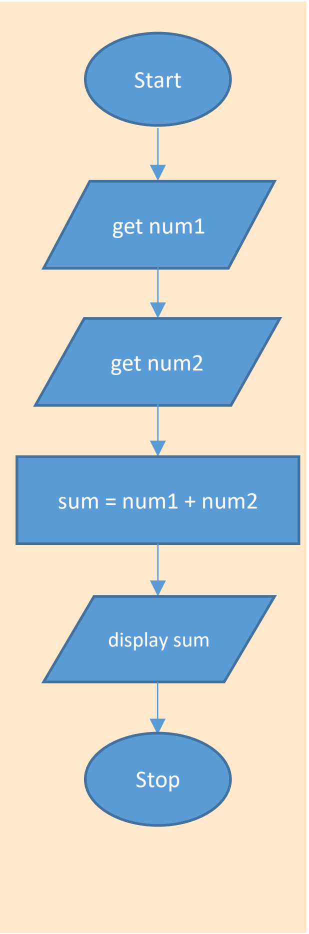
- Most people prefer visual stuff. So in programming, algorithm can be better understood when represented using a flowchart (Scanlan, 1987).
- A flowchart should have 1 start and 1 stop point only. For ease of understanding, add a "get" for inputs and add "display" for outputs since there is only 1 symbol for IO.

Flowchart 1



- When creating flowcharts, names of variables should match what is in the analysis.
- Assigning values to a variable is a process, thus, num1 = 3, num2 = 4, and sum = num1 + num2 are inside process symbols.
- We want to output the value of sum, thus, sum is in an IO symbol.
- Make sure the arrows of the flowline point towards the next symbol in the process.

Flowchart 2



- Since the 2 numbers are user inputs for this version, no values are initially assigned, num1 and num2 are inside IO symbols.
- Just include a "get" for inputs and "display" for outputs to lessen confusions.

Formative Assessment 1

1. Knowing that you know programming, your friend asked you if you could make him/her a program that will solve his/her average grade for the semester.
    a. What details will you ask him in order for you to analyze the problem?
    b. What method will you use to create the algorithm that your friend will easily understand too?
    c. How will you check if the output of your program is correct?
2. Do Step 1 and Step 2 of the PDLC for the following problems:
    a. Solving for the average of 3 input numbers

The Java code below is a simple **Hello World** program that will be used as an example in explaining the preceding parts of the chapter.

```
Line 1      //class declaration of a class named Hello
Line 2      public class Hello {
Line 3      //main method
Line 4        public static void main(String [ ]args) {
Line 5      //method that will display an output
Line 6                System.out.print("Hello World!"); } }
```

COMMENTS

Comments are notes written to a code for documentation purposes. Comments are not part of the program and do not affect the flow of the program. Different programming languages use different syntax for comments, though some are common.

Generally, there are two (2) types of comments:

1. **Block Comment** (multiline)
   Block comments are several lines long and usually enclosed in **/* */** for most programming languages and they can be placed at the beginning line, in between lines of code, or at the end line of the code.

2. **Line Comment** (end-of-line)
   Line comments are single line that normally begin with **//** for most programming languages and terminates at the end of the line. Like block comments, line comments can be placed at the beginning line, in between lines of code, or at the end line of the code.

Formative Assessment 2

1. What type of comment is used in the Hello World program?
2. Which lines in the program are comments?

TOKENS

Tokens are the smallest individual element of a program. Everything you see inside a program is a token. Generally there are 5 types of tokens: **Keywords**, **Identifiers**, **Operators**, **Separators**, and **Literals**.

SEPARATORS

Separators are used to separate different programming elements. The various types of separators used in programming are:
(Space) \t(Tab) \n(New Line) . , ( ) { } [ ]

Formative Assessment 3

What are the different separators in the Hello World program?

IDENTIFIERS

Identifiers are the name given to different programming elements. Either a name given to a variable such as *name*, *grade*, *average*, *salary*, *tuition,* or a function or any other programming element, all follow some basic naming conventions:

1. Keywords must not be used as an identifier
2. Identifiers must begin with an alphabet (a-z A-Z) or an underscore (_) symbol.
3. Identifiers can contain alphabets (a-z A-Z), digits (0-9), and underscore (_) symbol.
4. Identifiers must not contain any special characters such as **!** @ # *** . ‘ [ ]** etc. except for the Java programming language identifiers which can contain a dollar symbol ($).

Formative Assessment 4

1. Can you point out the different identifiers in the Hello World program? How many are there?

2. Write different identifiers following the naming conventions.

Identifiers with only letters	Identifiers with digits	Identifiers with allowed special characters

3. Identify whether the following are identifiers. Put a ✓in column 2 if it is an identifier or an × otherwise and briefly explain why it is × in column 3.

Activity 1			Num_1		
String			$var		

A			Ex@mple		
MyVariable			i-am-valid		
main			2ndNumber		
ChessTeam2			_crush		

KEYWORDS

Keywords are reserved words whose meaning is already defined by the programming language. Keywords cannot be used for any other purpose inside programming. Every programming language have their own set of keywords.

Example: public, class, int, do, while, if, void, return, static, etc.

Some keywords are common to some programming languages while others are not. (See https://techvidvan.com/tutorials/keywords-in-java/ for a complete list.)

Formative Assessment 5

How many and what are the keywords used in the Hello World program?

LITERALS

Literals are constant values that are used for performing various operations and calculations. The different types of literals are:

1. Integer Literals

Integer literals represent integer or numeric values.

Example: 1, 143, -830, 0

2. Floating Point Literals

Floating-point literals represent fractional values.

Example: 3.1416, -2.33

3. Boolean Literals

Boolean literals have two possible values: **true** or **false**.

4. Character Literals

Character literals represent single characters enclosed in single quotes (‘ ’).

Example: ‘A’, ‘z’, ‘9’

5. String Literals

String literals represent multiple characters enclosed in double quotes (““).

Example: “Error”, “Computer Programming”, “1 2 3 Go!”

Formative Assessment 6

1. Is there a literal in the Hello World program? What line is it in? What literal is it?
2. Give 5 data examples for each of the literals.

Integer	Floating-point	Character	String

DATA TYPES

A data type is a particular kind of data item, as defined by the values it can take, the programming language used, or the operations that can be performed on it.

Each literal mentioned in the previous section have different data types shown in Table 2.

Table 2. Data Types commonly used in Java

Literal	Data Type	Range
Integer	byte (1 byte)	-128 to 127
	short (2 bytes)	-32768 to 32767
	int (4 bytes)	-2147483648 to 2147483647
	long (8 bytes)	-9223372036854775808 to 9223372036854775807
Floating Point	float (4 bytes)	Approximately -/+ 3.40282347E+38 (6-7 significant decimal digits)
	double (8 bytes)	Approximately -/+ 1.79769313486231570E+308 (15 significant decimal digits)

Table 2. Data Types commonly used in Java continued…

Literal	Data Type	Range
Boolean	boolean	true or false
Character	char	0 to 65536 (unsigned)
String	String	2147483647 characters

In Java, there are eight (8) **primitive data types**: byte, short, int, long, float, double, boolean, and char. Other programming languages have different data types with different ranges. For now we will focus on Java data types.

String is not a primitive data type in Java; it is a class that can be used as a data type that contains multiple characters (numbers, letters and special characters).

Formative Assessment 7

1. Write the suitable data type for each data.

2.5	
Aspire 3	
E	
true	
99	

DDR4	
3.14344	
802.11ac	
555	
z	

2. Danielle, 15 years old, is the youngest 1-BSIT student at a university. Her average grade in high school is 99.50 that made her an automatic College Scholar which saved her from paying a tuition of 22314.77 pesos for that semester.

Given the statement above, identify the different data, the data types and identifiers that suite the data.

Data/Value	**Data Type**	**Identifier**

VARIABLES

A variable is used to store data. It has a **data type**, **variable name or identifier,** and an **initial value**. The data type indicates the type of value that the variable can hold. The variable name or identifier must follow rules for identifiers. The initial value is the data itself or initial data.

There are two (2) types of variables that Java programs have: **primitive variables** and **reference variables**. Primitive variables are variables with primitive data types while reference variables are variables that uses a class as data type.

To declare a variable:

<data type><space><identifier/variable name> [= initial value];

Note: Values enclosed in <> are required values, while those enclosed in [ ] are optional.

Example:

//declare a variable named result of data type boolean
boolean result;
//declare a variable named option of data type char
char option;
//assign the character ‘C’ to be the value of variable option
option = ‘C’;

```
//declare a variable named grade of data type double with an initial value of 0.0
double grade = 0.0;
```

Coding Guide:

1. It is always good to initialize variables as they are declared
2. Use descriptive names for variables. For example, use the variable name **grade** for a variable that contains a grade of a student.
3. Declare one variable per line of code.

Formative Assessment 8

1. Declare 5 different variables.

2. Declare a variable named ***age*** of type int with your age as the initial value.

3. Add variable x to variable y and assign to variable z of type double.

4. The value of a variable named ***ave*** of type int is the average of variables num1, num2, and num3.

5. Assign your full name as the initial value of a String variable named ***details***.

OPERATORS

Operators are special symbols performing specific operations on one, two, or three operands and then returning a result.

Basic operators are discussed below where Arithmetic operators are shown in Table 3, Relational Operators in Table 4, Logical Operators in Table 5, and Assignment Operators in Table 6.

Table 3. Arithmetic Operators

Assume integer variable A holds 10 and variable B holds 20, then −

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30

Table 3. Arithmetic Operators continued…

- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
***** (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

Table 4. Relational Operators
Assume variable A holds 10 and variable B holds 20, then −

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is false.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is false.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is false.

<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Table 5. Logical Operators

Assume Boolean variables A holds true and variable B holds false, then −

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are true, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are true, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

Example:

Assuming Expr, Expr1 and Expr2 are boolean expressions.

a. AND (&&)

Expr1		Expr2	Result
true	&&	true	true
true	&&	false	false
false	&&	true	false
false	&&	false	false

b. OR (||)

Expr1		Expr2	Result
true	||	true	true
true	||	false	true
false	||	true	true
false	||	false	false

d. NOT (!)

	Expr	Result
!	true	false
!	false	true

Table 6. Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
***=**	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A

Table 6. Assignment Operators continued…

Conditional Operator

The conditional operator **?:** is a ternary operator that means it takes in three (3) arguments that together form a conditional expression. The structure of an expression using a conditional operator is:

Expr1?Expr2:Expr3

where Expr1 is a boolean expression whose result must either be true or false. If Expr1 is true, the value of Expr2 is returned but if it is false, the value of Expr3 is returned. For example,

```
public class CondOp {
  public static void main(String [ ]args) {
          String status = “ ”;
          int grade = 59;
                  status = (grade>=60)?”Passed”:”Failed”;
                  System.out.print(status); } }
```

The output of this program will be:

```
Failed
```

Formative Assessment 9

Transform the following formulas into programming expressions using different operators:

	Formula	Expression
A	Length x Width x Height	
B	1/2Base x Height	
C	Pi x Raduis^2	
D	$\frac{b1+b2}{2}$ x (H)	
E	4Pi x Radius^3	
F	m(c^2)	
G	dailyRate x 22 – ((10% of dailyRate x 22) + sss + philhealth)	

Operator Precedence

Just like MDAS or PEMDAS in mathematics, operators in programming also follow precedence. Operator precedence is the order of operations and a collection of rules that reflect the conventions about which procedures to perform first in order to evaluate a given expression.

Table 7 presents the different programming operators and their precedence.

Table 7. Precedence of Common Programming Operators

Level	Operators	Level	Operators
1	( ) [ ] .	6	== !=
2	! ++ --	7	&&
3	* / %	8	||
4	+ -	9	?:
5	< <= > >=	10	= += -+ *= /= %= &= |=

In evaluating expressions, always solve those inside parenthesis first, followed by the operator in the highest precedence. If there are multiple operators of the same level, evaluate the operator that comes first from left to right.

Example: **8 ÷ 2 (2 + 2)**

For the expression above, some will say the answer is **1** some **16**. The correct answer is **16** because there can only be 1 correct answer.

If programming operators are used for the example above, it will look like this:

8 / 2 * (2 + 2)

/ is for division between 8 and 2, and * for multiplication because of the absence of an operator between 2 and ( ) which implies multiplication.

Solving it step by step would look like this:

8 / 2 8 (2 + 2)
8 / 2 * (4)	- + is done first since it is inside the ( )
4 * (4)	- / is done next since / and * are on the same level of precedence but / came first from the left
16	- * is done last

Formative Assessment 10

Evaluate the following expressions:

Provide the result of the following operation if int a = 5, int b = -2, int c = 2	
boolean exp1 = a + b * c <= a + c * b;	
boolean exp2 = !(c + a > b);	
boolean exp3 = b - a >= c;	
Evaluate the value of the following expressions:	
6 – 2 + 10 % 4 + 7	
int result = 4 + 2 / 1 * 3 – 1;	

EXPRESSIONS

An expression is made up of variables, operators and method invocations that evaluates to a single value.

Example:

sum = x + y

A **boolean expression** is an expression whose result is either true or false. It normally uses relational and/or logical operators but can also contain arithmetic operators.

Example:

z = x > y

Formative Assessment 11

1. Solve the following expressions using different operators:

Given **A = 1, B = 20, and C = 30**

EXPRESSION	RESULT
(B + C) / A	
A * B	
++C	
(B > A)	
A--	
Answer = A + B	
C!=B	
C – A	
A – (B*C)	
B==C	

2. Determine whether the following in column 1 is an **expression**, a **boolean expression** or **not an expression** and write your answer in column 2. If it is not an expression, briefly explain why in column 3.

A>=B		
A!C		
A + B + C;		
B<=C		
C==B*1.5		

STATEMENT

Statements are roughly equivalent to sentences in natural languages; it forms a complete unit of execution. Expressions can be made into a statement called **expression statements** by terminating the expression with a **semicolon (;)**.

- Assignment Statement
  Example: aValue = 89.33;
- Increment Statement
  Example: aValue++;
- Method invocation statement
  Example: System.out.println("Hello World!");

In addition to expression statements, there are two other kinds of statements: **declaration statements** and **control flow statements**.

```
//declaration statement
  double aValue = 123.456;
```

Control flow statements will be discussed in the Chapter 4 - Control Structures.

Formative Assessment 12

Determine whether the following is a **statement** or **not a statement**.

System.out.print("I am not a statement");	
Sum = A + B	
int num1, num2;	
String FamName;	
public class SampleStatement {	

BLOCKS

A block is one or more statements bounded by an opening and closing curly braces { } that groups statements as one unit. A block can be placed inside another block. For example,

```
public class Factorial {
public static void main(String[] args)
{           final int NUM_FACTS = 100;
            for(int i = 0; i < NUM_FACTS; i++) {
                        System.out.println( i + "! is " + factorial(i));   }   }
public static int factorial(int n)
{           int result = 1;
            for(int i = 2; i <= n; i++)
                        result *= i;
            return result;
}           }
```

Formative Assessment 13

1. How many blocks are there in the Hello World program?

2. How many blocks are there in the sample program given?

Continuation of EXAMPLE 1 on page 13

3. PROGRAM CODING

Before coding, the programmer must determine the data types of each variable based on the values. There are 2 versions of the code, 1 for each version of the program design.

Source Code 1

```
import javax.swing.JOptionPane;
public class Sum {
public static void main(String [ ]args) {
int num1 = 3;
int num2 = 4;
int sum = num1 + num2;
System.out.println(sum);
}   }
```

Source Code 2

```
import javax.swing.JOptionPane;
public class Sum {
public static void main(String [ ]args) {
int num1=Integer.parseInt(JOptionPane.showInputDialog(“Enter first
number:”);
int num2 = Integer.parseInt(JOptionPane.showInputDialog(“Enter second
number:”);
int sum = num1 + num2;
System.out.println(sum);
}   }
```

- The code in the previous slide can be modified by changing the data type of the variables to accommodate other types of values.

- The int data type can only accommodate whole number values. If the inputs are decimal numbers, change the data type to float or double.

4. PROGRAM TESTING

- The code is not always 100% bug free after writing. The trick is solving the topmost or the first error first because usually the last error is just a side effect of the first.
- It helps to be observant of formats when debugging code.
- Check also the accuracy of the output using different inputs when there are no more syntax errors.

Formative Assessment 14

Write the code for the Program Design in Formative Assessment 1 #2.

Chapter 2 Summary

In order for programmers to develop effective programs, they have to follow the four steps of the PDLC which are problem analysis, program design, program coding, and program testing. The programmer first determines the inputs, outputs, and processes. The programmer then designs the algorithm using either human language, pseudocode, or flowchart. After designing, program coding will take place. This is when the programmer writes the source code of a program. The last part is the program testing, it is in this phase when the programmer ensures that the program is free of errors and that it is functioning properly. When an error exists, the programmer will fix the error, technically, fixing of errors is called debugging. There are different types of errors; syntax error, logic error and runtime error.

A Java program is made up of different parts starting with comments, a **comment** is generally used for documentation. There are two types of comments: a block comment, and a line comment. And then there is **data type**, in Java programming, there are eight primitive data types: byte, short, int, long, float, double, boolean, and char. A String is not a primitive data type in Java programming, it is a class that can be used as a data type that contains multiple characters (numbers, letters and special characters). Next is **variable,** this is used to store data. A variable has a data type, a variable name, and an initial value. There are two types of variables in Java programming, primitive and reference variables.

One relevant element of a Java program is the token. Generally, there are 5 types of tokens; separators, identifiers, keywords, literals and operators. A literal has different types including Integer Literal, Floating Point Literal, Boolean Literal, Character Literal, and String Literal. An operator has different types as well: Arithmetic, Relational, Logical, Assignment and Conditional Operators.

Understanding these parts and elements of a Java program will definitely help you go through your programming experience smoothly. ☺

Summative Assessment

1. Chapter 2 Activity (See VLE)
2. Chapter 2 Quiz (see VLE)

Chapter 2 References

[1] Burd, B. (2017). *Beginning Programming with Java For Dummies (5th Ed)*. New Jersey: John Wiley and Sons, Inc.

[2] Cavida, D.G, Frio, F.M., Flores, D. (2010). *Computer Programming I*. Unpublished Workbook, University of Southern Mindanao, Kabacan, Cotabato

[3] Flask, R. (ND). *Java for Beginners 2nd Edition* [PDF File]. Retrieved from http://staff.um.edu.mt/__data/assets/pdf_file/0010/57169/jn.pdf

[4] Mayfield, B. (2016). *From Problem Analysis to Program Design Lab Manual (3rd ed.).* Philippines: Cengage Learning Asia Pte Ltd.

[5] *Module 3: Algorithm Representation*. Dragonwins.com. (2010). Retrieved 23 July 2021, from http://www.dragonwins.com/courses/ECE1021/STATIC/LESSONS/Algorithm Representation.htm.

[6] Prakash, P. (2015). codeforwin.org. Retrieved 23 July 2021, from https://codeforwin.org/2015/05/introduction-to-programming-tokens.html.

[7] Usman, O., Owoade, A., Abimbola, B., & Ogunsanwo, G. (2016). *(PDF) INTRODUCTION TO COMPUTER PROGRAMMING (BASIC)*. ResearchGate. Retrieved 23 July 2021, from https://www.researchgate.net/publication/317182495_INTRODUCTION_TO_COMPUTER_PROGRAMMING_BASIC.

[8] Fulmanski.pl. (1995). Retrieved 23 July 2021, from https://fulmanski.pl/zajecia/ics/materialy/compalgorithm.pdf.

[9] Nishadha (2021). Retrieved 23 July 2021, from https://www.google.com.ph/amp/s/creately.com/blog/diagrams/flowchart-guide-flowchart-tutorial/amp/.

[10] *Flowchart - Wikipedia*. En.m.wikipedia.org. (2021). Retrieved 23 July 2021, from https://en.m.wikipedia.org/wiki/Flowchart#History.

[11] Baldwin, R. (2007). *... in Java and C++ by Richard G Baldwin*. Dickbaldwin.com. Retrieved 23 July 2021, from http://www.dickbaldwin.com/Cosc1315/Pf00120.htm.

[12] *Comment (computer programming) - Wikipedia*. En.m.wikipedia.org. (2021). Retrieved 23 July 2021, from https://en.m.wikipedia.org/wiki/Comment_(computer_programming).

[13] Van Tassel, D. (2004). *Comments in programming languages*. Gavilan.edu. Retrieved 23 July 2021, from http://www.gavilan.edu/csis/languages/comments.html.

[14] Kumar, K. (2021). *Java Primitive Data Types. Size, Range and Default Value of Basic Data Types*. cs-fundamentals.com. Retrieved 23 July 2021, from http://cs-fundamentals.com/java-programming/java-primitive-data-types.php.

CHAPTER 3
Inputs and Outputs

Intended Learning Outcomes

By the end of this topic/chapter, you must be able to:

1. Create simple programs following the steps of the PDLC with different programming elements using a specific software
2. Develop interactive programs that gets input and displays output on console
3. Develop interactive programs that gets input and displays output on GUI
4. Debug a given erroneous computer program
5. Determine the output of given computer programs

Getting to know the Software Interface

There are a lot of available software that can be used to develop programs in computers as well as in mobile devices. But whatever that software may be, to develop programs with ease, one must be familiar with the user interface, and aware of the different functions and features of that specific software.

Fig. 14. Shortcut Icon of JCreator

Here are some of the software that can be used for Java programming:

Desktop/Laptop:

1. NetBeans
2. Eclipse
3. JCreator
4. Notepad++

Smartphone:

1. Anacode
2. Java N-IDE
3. AIDE
4. DCoder

Since we will only focus on the Java programming language for this chapter onwards, though the common software that can be used are NetBeans, Eclipse and JCreator, we will use JCreator (Fig. 14).

Just like any other software, the user interface of JCreator have some common features to any other window like the title bar, menu bar, toolbars, maximize, minimize, close buttons, and some software specific parts as shown in Fig. 15.

The **Title Bar** is just usually the name of the application, in this case JCreator. The **Menu Bar** is where you can find common functions such as creating a new file, opening an existing file, running your code and the like. The **Standard Toolbar** consists of icons that are shortcuts to some functions that you can find in the menu bar. The **Active Files Window** is like your taskbar; open files appear there as tabs. The **Build Output Window** is where your bugs, specifically syntax errors, will appear; their description and location. Though it is not 100% accurate at times depending on the bug, but it is pretty useful. The **General Output Window** is where the console output of your program, logic, and runtime errors will appear. Other windows together with the Build Output and General Output can be found in the **View→Other Windows** menu. You will discover other functions by clicking menus and icons with the help of curiosity and some basic exploration.

Once you are familiar with the user interface, you can now start creating simple programs. Of course, you have learned from the previous chapter that before you proceed with program coding, you should have analyzed the problem first by identifying inputs, processes and outputs, and created an algorithm for it like a flowchart. And also do not forget the good programming practices and the qualities of a good program.

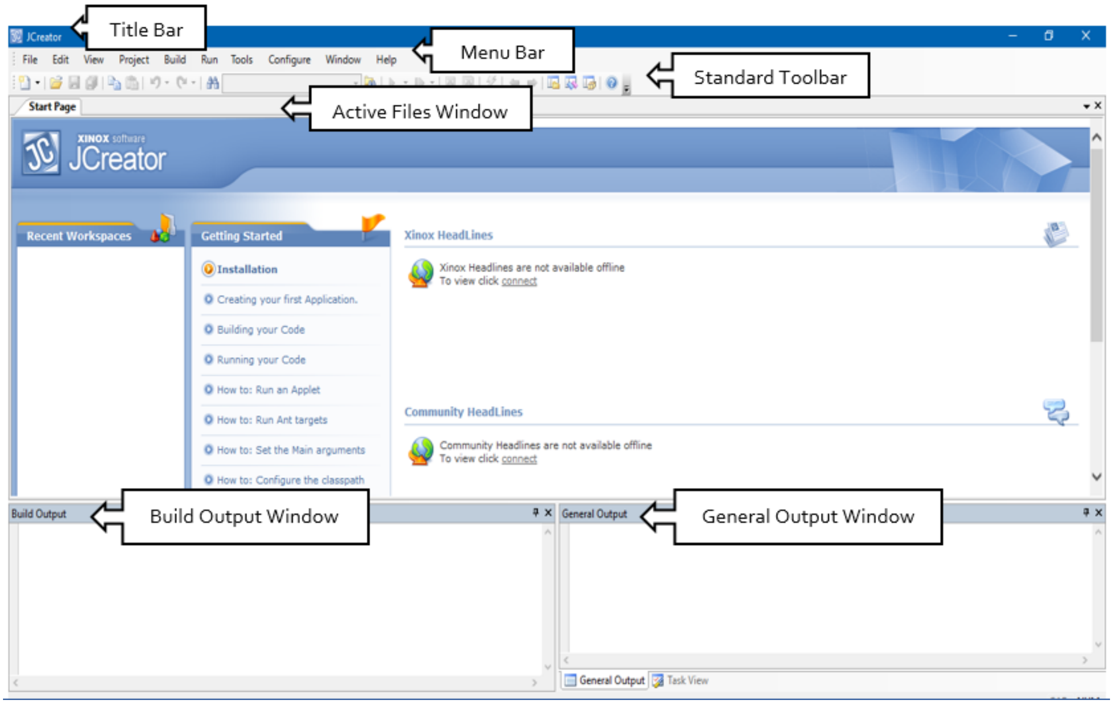


Fig. 15. UI of JCreator

Creating Simple Programs

Given you have analyzed a problem and created an algorithm for it, the next step of course is to code it. To create a program using JCreator is simple. Just follow these steps:

Step 1: Create a new file

Click **File→New→File** on the menu or press Ctrl + N on the keyboard.

Step 2: Select file type

The File Wizard window (Fig. 16) will appear. Since we will be creating a simple Java program, click **Java Classes** and then **Empty Java File** and click **<u>N</u>ext**.

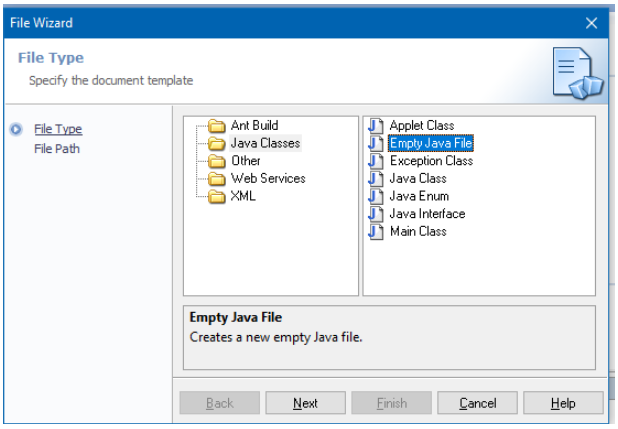
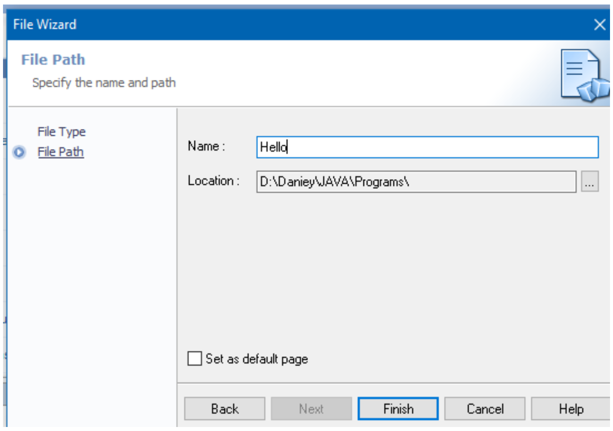


Fig. 16. File Wizard Window

Step 3: Name your program and specify location

Before you start coding, the software asks you to name your file first and specify where you want to save it (Fig. 17). The name here is an identifier which means it should follow the naming conventions. Click **Finish** when you are done.



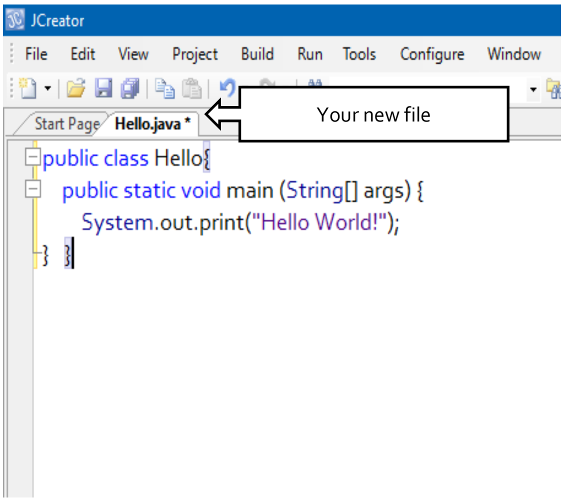
Name : Hello

Location : D:\Daniey\JAVA\Programs\

Fig. 17. Setting name and location

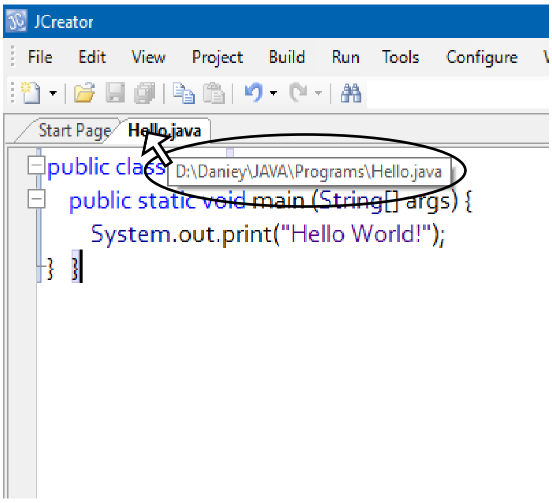
Step 4: Start Coding

A new active file with the name you specified in Step 3 will appear (Fig. 18). **Hello.java** means the name of your file is **Hello** and the file format is **.java**. Notice the *****? That means the file or whatever revision you made to the current file is not yet saved. You can save it through the menu bar or toolbar or Ctrl+S and the * will disappear. So where is your file? It is in the location that you specified in Step 3. You can double check the location of your file by placing your mouse pointer on the file's tab (Fig. 19).



```
public class Hello{
  public static void main (String[] args) {
    System.out.print("Hello World!");
} }
```

Fig. 18. Writing source code



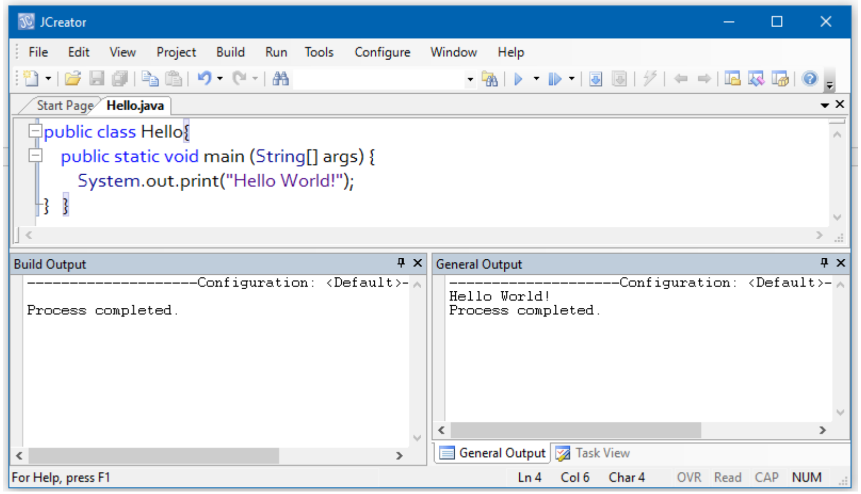
D:\Daniey\JAVA\Programs\Hello.java

```
public class Hello{
  public static void main (String[] args) {
    System.out.print("Hello World!");
} }
```

Fig. 19. Double checking file location

Step 5: Run your code

After writing your source code, you can now run it. You can run your code by pressing the **F5** function key, or **Run→Run File** menu or the ► button on the toolbar. If your code has no bugs, the output will appear on the General Output window (Fig. 20) otherwise, a list of bugs will appear on the Build Output window (Fig. 21).



```
public class Hello{
   public static void main (String[] args) {
      System.out.print("Hello World!");
} }
```

Build Output:

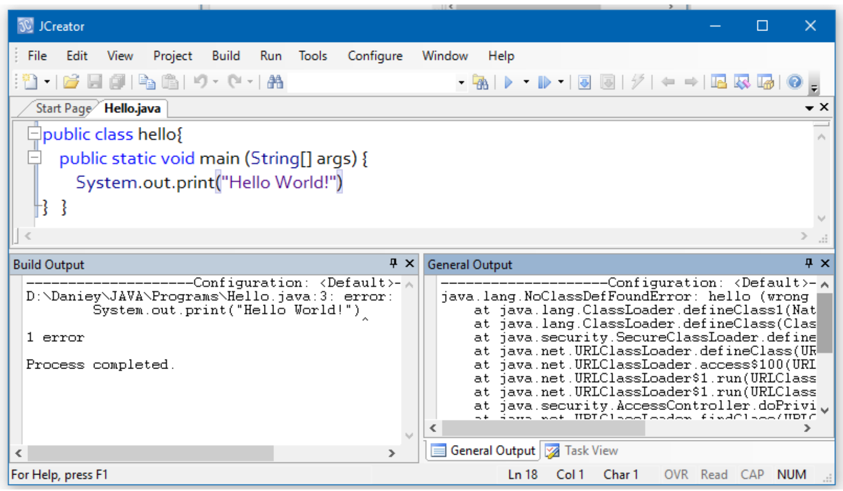
```
--------------------Configuration: <Default>-

Process completed.
```

General Output:

```
--------------------Configuration: <Default>-
Hello World!
Process completed.
```

JCreator
File Edit View Project Build Run Tools Configure Window Help
Start Page
Hello.java
General Output
Task View
For Help, press F1
Ln 4 Col 6 Char 4 OVR Read CAP NUM

Fig. 20. Code without bug



```
public class hello{
   public static void main (String[] args) {
      System.out.print("Hello World!")
} }
```

Build Output:

```
--------------------Configuration: <Default>-
D:\Daniey\JAVA\Programs\Hello.java:3: error:
        System.out.print("Hello World!")
                                        ^

1 error

Process completed.
```

General Output:

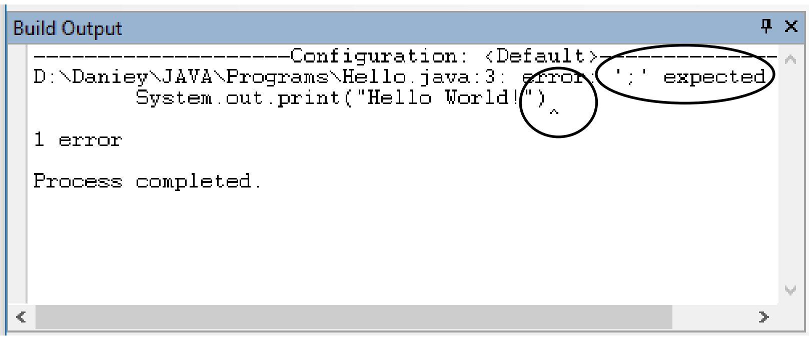
```
--------------------Configuration: <Default>-
java.lang.NoClassDefFoundError: hello (wrong
    at java.lang.ClassLoader.defineClass1(Nat
    at java.lang.ClassLoader.defineClass(Clas
    at java.security.SecureClassLoader.define
    at java.net.URLClassLoader.defineClass(UR
    at java.net.URLClassLoader.access$100(URI
    at java.net.URLClassLoader$1.run(URLClass
    at java.net.URLClassLoader$1.run(URLClass
    at java.security.AccessController.doPrivi
```

JCreator
File Edit View Project Build Run Tools Configure Window Help
Start Page
Hello.java
General Output
Task View
For Help, press F1
Ln 18 Col 1 Char 1 OVR Read CAP NUM

Fig. 21. Code with bugs

Step 6: Debug your code

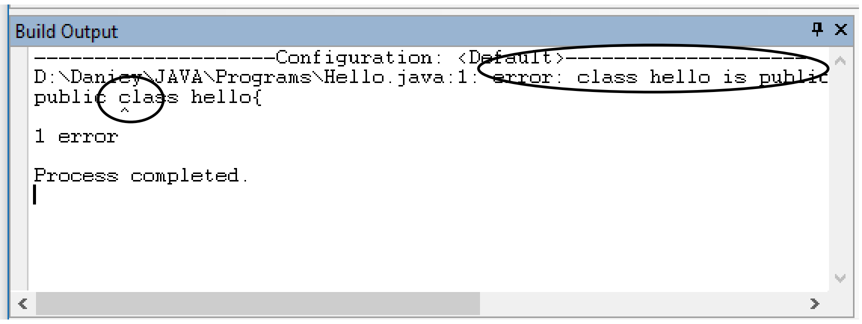
Notice in Fig. 21 the "**1 error**" in the Build Output window. That code actually has 2 errors but the software detected only 1 at first. For example you failed to place a '**;**' at the end of your statement. The bug that will be detected is a missing '**;**' (Fig. 22). Notice the **^**? It points to the line or where the error is. Though sometimes the error is above the line but sometimes it is correct in detecting where the actual error is. Also notice the description "**';' expected**". It actually gives you a hint on what the actual error is; something missing etc. BUT, again it is not that 100% accurate. You still have to manually identify the error. In this case, you have to put a **;** where the **^** points to solve the error. But, a new error will appear (Fig. 23).



```
--------------------Configuration: <Default>--------------
D:\Daniey\JAVA\Programs\Hello.java:3: error: ';' expected
        System.out.print("Hello World!")
                                        ^
1 error

Process completed.
```

Fig. 22. Identifying the bug

Sometimes, even if you have solved the error specified by the software, another error will appear. That is just some of the limitations of the software. It is not the software’s fault. Like for example you mistakenly typed a letter in the wrong case for your class name. In this case, the **^** is below the keyword class. The software is correct in identifying the line where the error is here but the actual error is the identifier **hello**. The identifier here should be **identical** to your filename which is **Hello**. Changing the ‘h’ to ‘H’ will solve the problem.



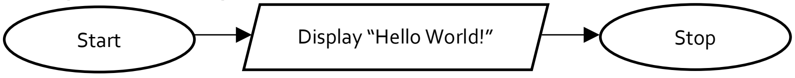
```
--------------------Configuration: <Default>--------------------
D:\Daniey\JAVA\Programs\Hello.java:1: error: class hello is public
public class hello{
       ^
1 error

Process completed.
```

Fig. 23. Identifying the other bug

When you have solved all the errors and all that is left in the Build Output window is **Process Completed** alone, then your code is error free and your output will appear on the General output window just like in Fig. 20.

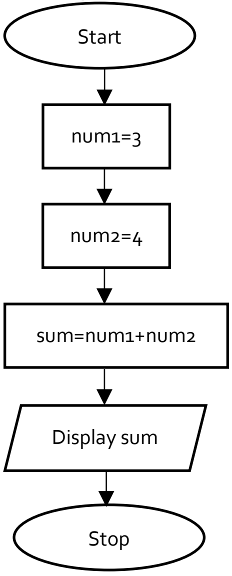
Displaying Output on Console

The program given in the previous example is just a simple program that will display an output. Following PDLC, if we are to create an analysis for that and a design it would look like this:

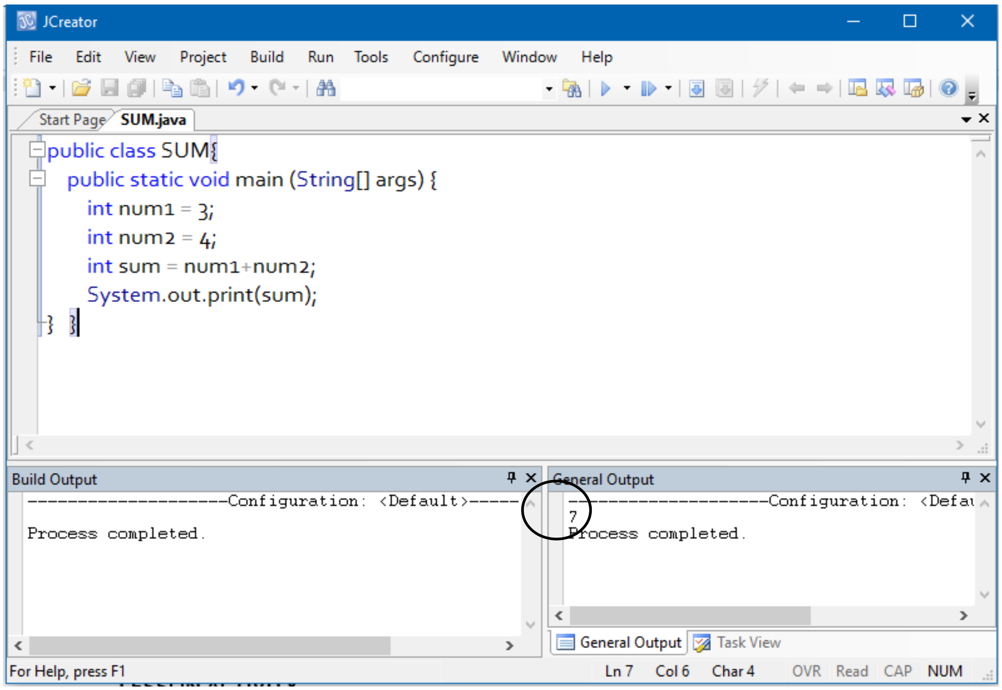
Problem Analysis: Input: None Process: None Output: “Hello World!”	**Program Design:** 

Now if you are to create a program requires a process it would look like this:

Problem: Create a program that will add any 2 whole numbers		

Problem Analysis:
Output: sum
Input: None
Process:
num1 = 3
num2 = 4
sum = num1+num2
 	
**Since the problem says "any 2 numbers" you can assign any number to be added. Notice that the numbers are assigned to identifiers. This is done so that it will be easy to track the different data used in a program.*
	Program Design:
	
	 Start
	num1=3
	num2=4
	sum=num1+num2
	Display sum
	Stop
		Program Coding:
		public class SUM{
		public static void main(String [ ]args){
		
		int num1 = 3;
		
		int num2 = 4;
		
		int sum = num1+num2;
		
		System.out.print(sum);
		
		} }

If you can notice, each step in the flowchart has an equivalent code. For the **Start**, the class is declared followed by the **main() method** which is the starting point of all Java programs. Without it, some codes will still compile but it will not display any output. Next, inside the main block, are the variable declarations where 2 variables of type **int** (since the problem clearly states whole numbers) num1 and num2 are declared and assigned 3 and 4 to be their initial values respectively, though those values can be changed anytime. Next



JCreator
File Edit View Project Build Run Tools Configure Window Help
Start Page
SUM.java
Build Output
General Output
General Output
Task View
For Help, press F1
Ln 7 Col 6 Char 4 OVR Read CAP NUM

```
public class SUM{
  public static void main (String[] args) {
    int num1 = 3;
    int num2 = 4;
    int sum = num1+num2;
    System.out.print(sum);
} }
```

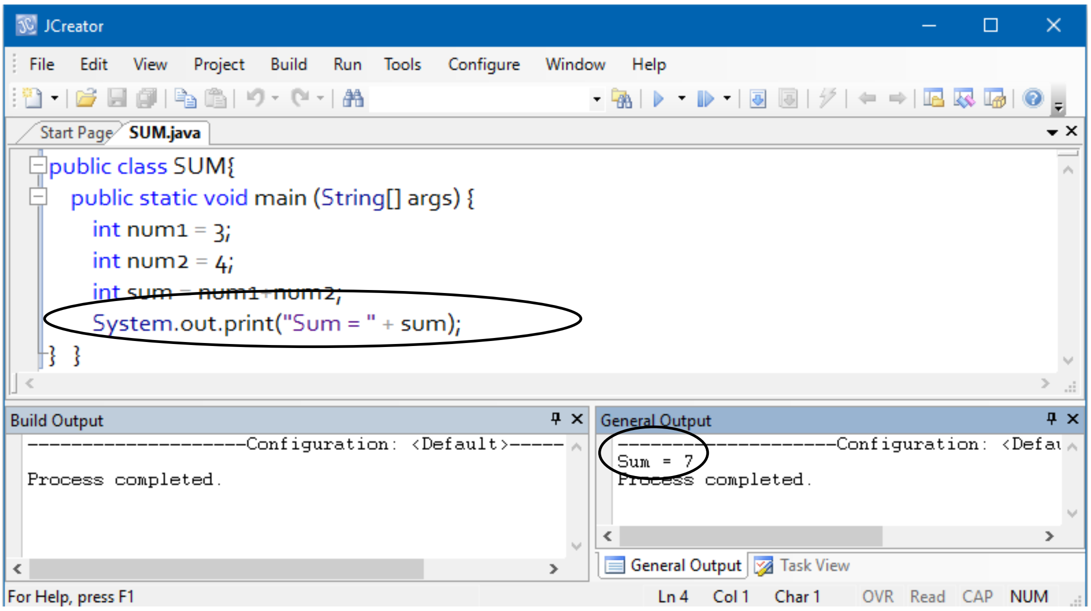
```
--------------------Configuration: <Default>------

Process completed.
```

```
7-------------------Configuration: <Defau
Process completed.
```

Fig. 24. Program for adding 2 whole numbers

is the process **sum** which is also declared as a variable of type int and assigned the value of adding num1 and num2 using the **+** arithmetic operator. Then the code for displaying output on the General Output window **System.out.print()** is next. Whatever that is needed to be displayed on the General Output window must be inside that **( )** of the **print()** method. Anything outside the **( )** will not be considered as output. Data of any literal, it can be an identifier/variable name or an actual data, can be placed for output in the code. In this case we placed a variable name/identifier which is **sum**. Only the value of sum will be displayed on the General Output window (Fig. 24). And last are the closing curly braces for the blocks which indicates the end of the code.

As you can see in the General Output window (Fig. 24) there is **7** which is the value of sum. Now, technically, outputs should have **labels** for them to be identified especially if you have a program with multiple outputs. To do that, we just have to modify the code for output by adding a string to label it (Fig. 25).


JCreator
File Edit View Project Build Run Tools Configure Window Help
Start Page
SUM.java
Build Output
General Output
Task View
For Help, press F1
Ln 4 Col 1 Char 1 OVR Read CAP NUM

```
public class SUM{
   public static void main (String[] args) {
      int num1 = 3;
      int num2 = 4;
      int sum = num1+num2;
      System.out.print("Sum = " + sum);
} }
```

```
--------------------Configuration: <Default>------
Process completed.
```

```
--------------------Configuration: <Defau
Sum = 7
Process completed.
```

Fig. 25. Program Output with label

By placing a string "Sum = " before the identifier sum separated by a **+** sign (Fig. 25), we have added a label which makes the output more comprehensive.

Do not be confused with the use of the **+** here. It can be an arithmetic operator if placed between 2 integer or floating point literals but it can act as a separator when placed between 2 different literals assigned as value to a string variable or an output.

Creating Simple Programs Using Java N-IDE

Installation

- ✓ For android users, open Play Store and search for JAVA N-IDE.
- ✓ Tap Install.
- ✓ Tap Open when done.

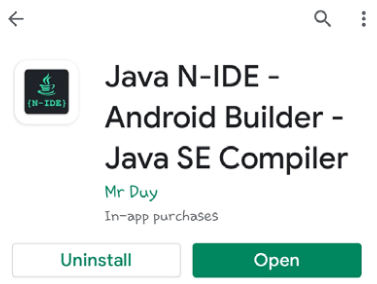
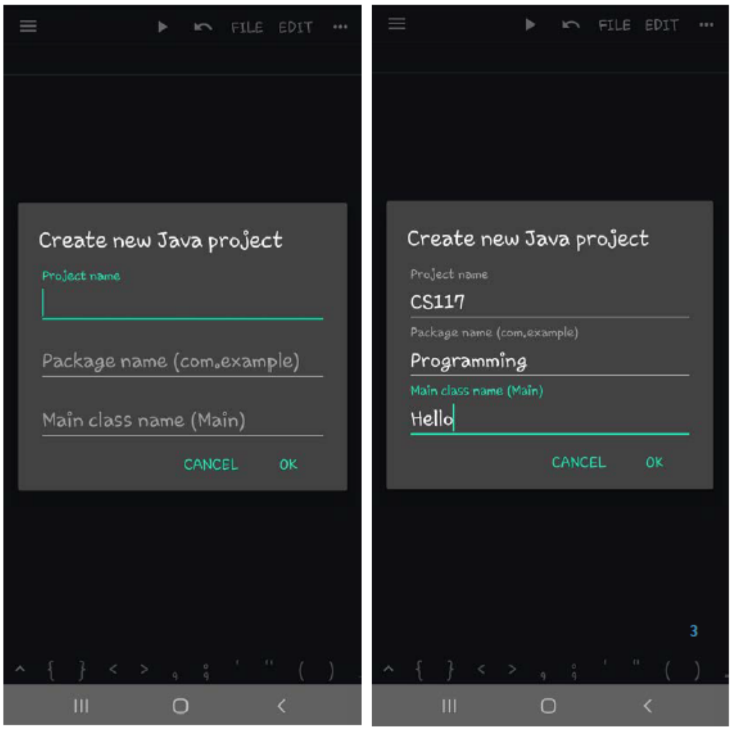


Fig. 26. JAVA N-IDE App

Step 1

- ✓ Type in a **Project Name** – It's like a folder name for your files. It **only** accepts letters and numbers.
- ✓ Type in a **Package Name** – Still like a folder to organize your programs.
- ✓ Type in a **Class Name** –The name of the program. Keep in mind the rules for naming identifiers here.
- ✓ Tap **OK** when done.



Create new Java project

Project name

Package name (com.example)

Main class name (Main)

CANCEL OK

Create new Java project

Project name: CS117

Package name (com.example): Programming

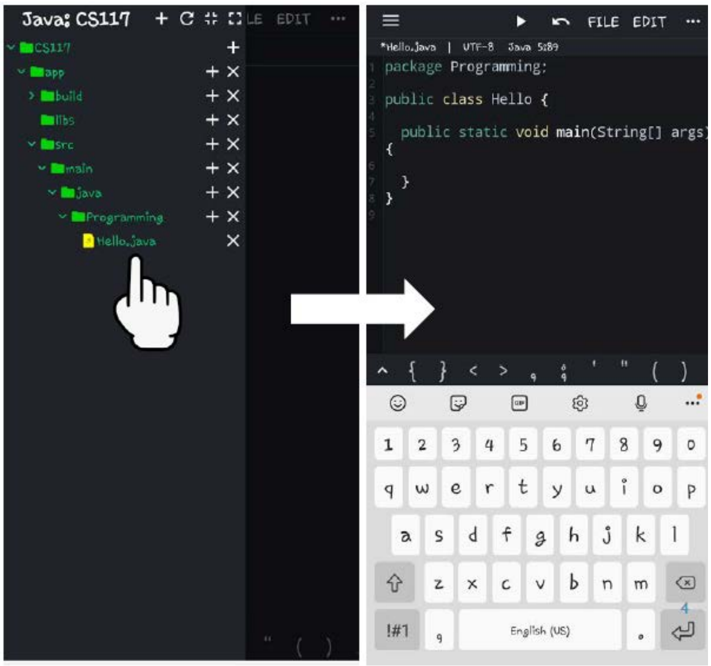
Main class name (Main): Hello

CANCEL OK

Fig. 27. Creating new Java N-IDE Project

Step 2

After setting up and tapping OK, a directory will appear.



```
CS117
  app
    build
    libs
    src
      main
        java
          Programming
            Hello.java
```

*Hello.java | UTF-8 Java 5:89

```
package Programming;

public class Hello {

    public static void main(String[] args)
  {

    }
}
```

Fig. 28. Creating simple program in Java N-IDE

✓ Tap your newly created class to write source code.

✓ The directory will disappear and a text editor will appear with the class name and main method already. All you have to do is edit and add your code.

✓ You can tap the directory button ☰ on the upper left corner to see the directory again.

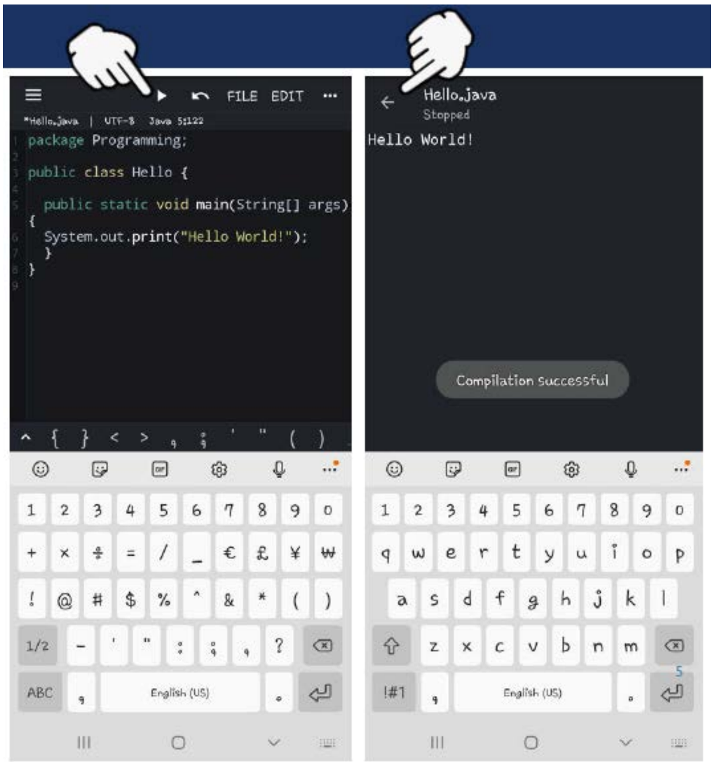
Step 3

✓ Type your code inside the main method block and make sure it has no bugs.

✓ Tap the play button ▶ to run your code.

✓ If the code has no errors, the output will appear.

✓ Tap the back button ← on the upper left to return to the editor.



```
package Programming;

public class Hello {

  public static void main(String[] args)
{
  System.out.print("Hello World!");
  }
}
```

```
Hello World!
```

Fig. 29. Running program in Java N-IDE

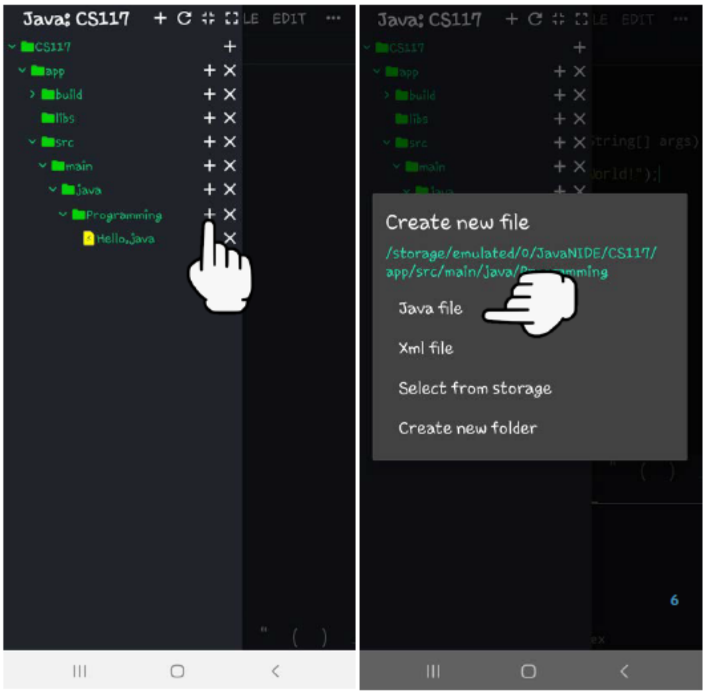


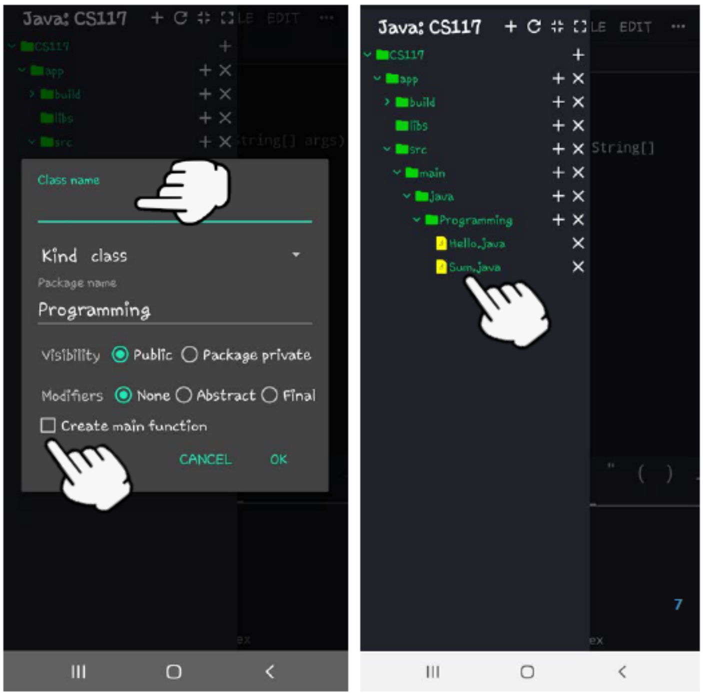
Fig. 30. Creating another program in Java N-IDE

If you are done creating your program and you want to create another:

✓ Tap the directory button on the upper left corner

✓ Tap the **+** button on the right side of the package name

✓ Tap **Java File** as shown in Fig. 30



Class name

Kind class

Package name
Programming

Visibility ● Public ○ Package private

Modifiers ● None ○ Abstract ○ Final

□ Create main function

CANCEL OK

Fig. 31. Creating another program in Java N-IDE

✓ Type in **class name**

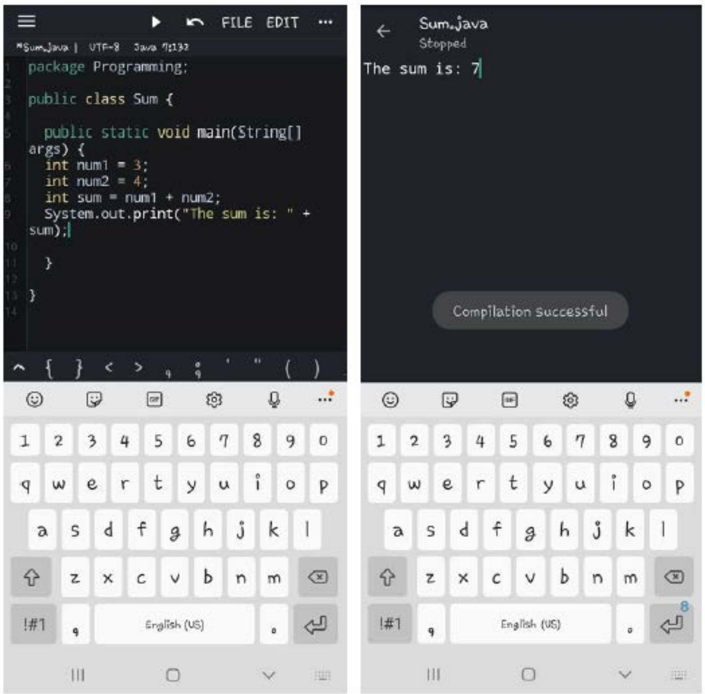
✓ Tap **Create main function** to check it (this includes the main method in the new class you are creating)

✓ Tap **OK**

✓ The new class will appear at the bottom of the directory. Tap the class to edit the code as shown in Fig. 31.

✓ Write the code as shown in Fig. 32.

✓ Tap the play button to run the code.



```
package Programming;

public class Sum {

    public static void main(String[] args) {
     int num1 = 3;
     int num2 = 4;
     int sum = num1 + num2;
     System.out.print("The sum is: " + sum);

    }

}
```

```
The sum is: 7
```

Compilation successful

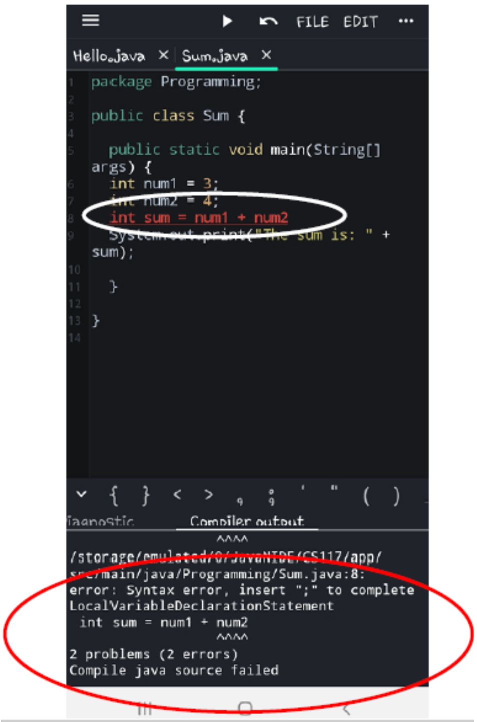
Fig. 32. Creating another program in Java N-IDE

If your code has errors as shown in Fig. 33, it will appear at the bottom of the screen.

✓ The line with error turns **red** when the program is run.

✓ The error description suggests what should be done to solve the bug.

✓ Correct the error and run again to check if the bug is solved.



```
package Programming;

public class Sum {

  public static void main(String[]
args) {
    int num1 = 3;
    int num2 = 4;
    int sum = num1 + num2
    System.out.print("The sum is: " +
sum);

  }

}
```

```
                    ^^^^
/storage/emulated/0/JavaNIDE/CS117/app/
src/main/java/Programming/Sum.java:8:
error: Syntax error, insert ";" to complete
LocalVariableDeclarationStatement
 int sum = num1 + num2
                    ^^^^
2 problems (2 errors)
Compile java source failed
```

Fig. 33. Program Errors in Java N-IDE

Formative Assessment 1

<table>
<tr><td colspan="3"><b>Problem:</b> Create a program that will solve for the average grade of a 1st year student enrolled in the 1st Year 1st semester.</td></tr>
<tr><td><b>Problem Analysis:</b></td><td><b>Program Design:</b></td><td><b>Program Coding:</b></td></tr>
</table>

See Appendix A for sample output screenshot.

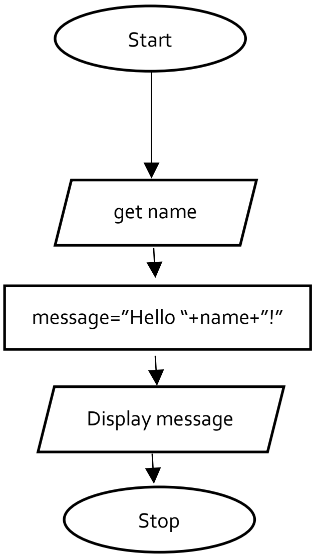
Getting Input and Displaying Output Using Graphical User Interface

Programs need to be more interactive by getting some input from users and not just display output upon running the code. There are a lot of ways to create interactive programs. Here we will discuss one (1) way to get input from users and another way to display output because the other way, System.out.print(), was already discussed in the previous section.

Using JOptionPane

JOptionPane is a pre-defined class from the **javax.swing** package that can be used to pop up standard dialog boxes that **prompt** users for a value or display outputs that tell users what to do.

For example:

<table>
<tr><td colspan="3"><b>Problem:</b> Create a program that asks the user for a name and display a message containing the name entered.</td></tr>
<tr>
<td><b>Problem Analysis:</b>
<b>Output:</b> message
<b>Input:</b> name
<b>Process:</b>
message = “Hello “ + name + “!”</td>
<td><b>Program Design:</b>


Start
get name
message=”Hello “+name+”!”
Display message
Stop

</td>
<td><b>Program Coding:</b>
import javax.swing.JOptionPane;
public class InputOutputName {
public static void main(String [ ]args)
{
String name = “”;
String message = “”;
name = JOptionPane.showInputDialog (“Enter name:”);
message = “Hello “ + name + “!”;
JOptionPane.showMessageDialog(null, message);
} }</td>
</tr>
</table>

The first statement

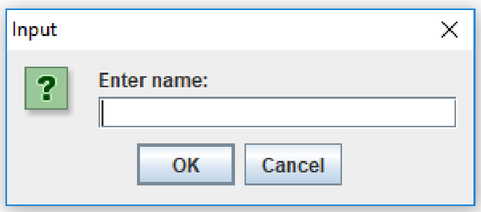
import javax.swing.JOptionPane;

indicates that the predefined class JOptionPane from the javax.swing package will be imported or used in the program.

The statement

name = JOptionPane.showInputDialog(“Enter name:”);

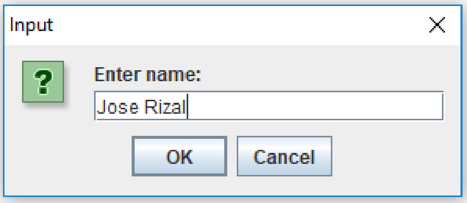
creates a JOptionPane input dialog (Fig. 37), which will display a dialog box with a prompt message “Enter name:”, a textbox, an OK, and a Cancel button.



Input

Enter name:

Fig. 37. Input Dialog Box



Input

Enter name:

Jose Rizal

Fig. 38. Assigning Input Value for name

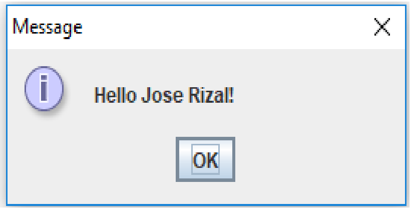
Whatever is typed in the textbox will be assigned as the value of the variable name. Like in Fig. 38, if “Jose Rizal” is typed in the textbox, it will be assigned as the value of **name**.

The statement,

message = “Hello “ + name + “!”;

assigns the string “Hello “, value of the variable **name** and an exclamation point as the value of variable **message**.

This line, **JOptionPane.showMessageDialog(null, message);** displays a dialog box which contains the value of the variable **message** and an OK button (Fig. 39).

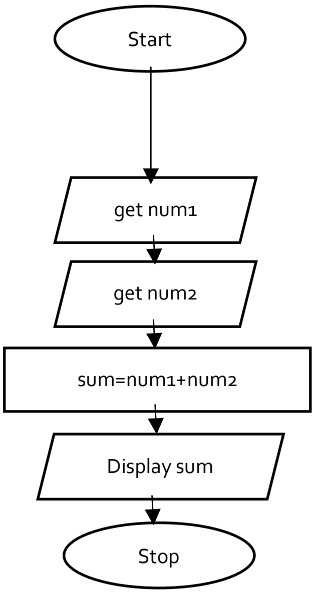


Hello Jose Rizal!

Fig. 39. Output using a Dialog Box

Another example:.

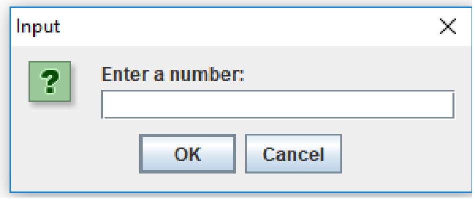
Problem: Create a program that asks the user for 2 whole numbers and display the sum		

Problem Analysis:
Output:
sum
Input:
num1 and num2
Process:
sum=num1+num2
	Program Design:
	
	 Start
	get num1
	get num2
	sum=num1+num2
	Display sum
	Stop
		Program Coding:
		import javax.swing.JOptionPane;
		public class InteractiveSum {
		public static void main(String [ ]args) {
		int num1=Integer.parseInt(JOptionPane.showInput Dialog(“Enter a number:”));
		int num2=Integer.parseInt(JOptionPane.show Input Dialog(“Enter another number:”));
		int sum = num1+num2;
		JOptionPane.showMessageDialog(null, “Sum: “ + sum);
		} }

The line,

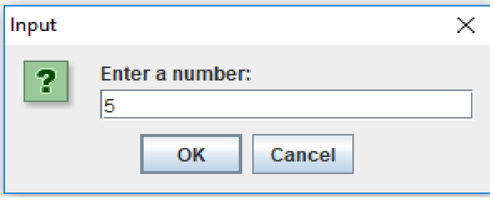
num1 = Integer.parseInt(JOptionPane.showInputDialog(“Enter a number:”));

prompts the user for an input (Fig. 40) that will be **converted** to an integer value through the **parseInt()** method, and will be assigned as the value of variable **num1**. After the user enters a data for num1 (Fig. 41), input for num2 will be prompted (Fig. 42) and after the user enters a data (Fig. 43) an output will be displayed (Fig. 33).



Enter a number:

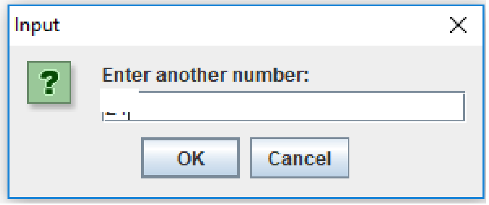
Fig. 40. Input Dialog Box for num1



Enter a number:

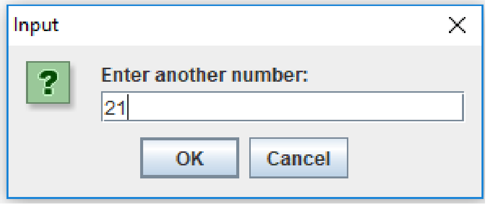
5

Fig. 41. Assigning Input Value for num1



Enter another number:

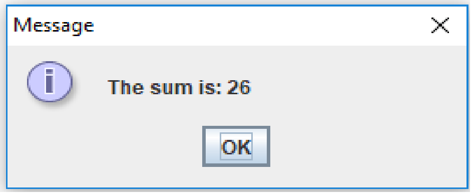
Fig. 42. Input Dialog Box for num2



Enter another number:

21

Fig. 43. Assigning Input Value for num2



The sum is: 26

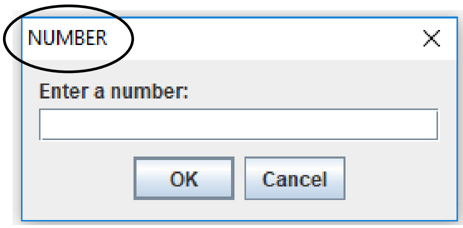
Fig. 44. Output using a Dialog Box

Message and input dialog boxes can be **customized** wherein the dialog title and message type can be changed.

For example:

```
import javax.swing.JOptionPane;
public class KeyboardIO3{
public static void main (String[] args) {
int num1 = 0;
int num2 = 0;
num1 = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter a number:
", "NUMBER", JOptionPane.PLAIN_MESSAGE));
num2 = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter another
number: ", "NUMBER", JOptionPane.PLAIN_MESSAGE));
int sum = num1+num2;
String msg = "The sum is: " + sum;
JOptionPane.showMessageDialog(null, msg, "OUTPUT", 1); }        }
```

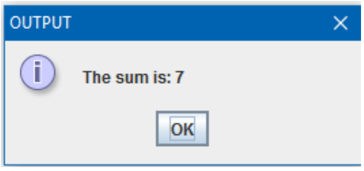
The line
num1 = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter a number:
", **"NUMBER"**, **JOptionPane.PLAIN_MESSAGE**));
customizes the input dialog box (Fig. 45) to have the title **NUMBER** instead of Input and plain message as its type.



Enter a number:

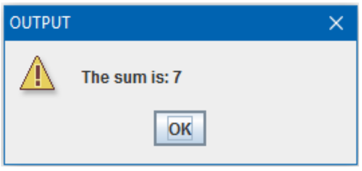
Fig. 45. Customized Input Dialog box

The line
JOptionPane.showMessageDialog(null, msg, **"OUTPUT"**, **1**);
customizes the output dialog box to have the title **OUTPUT** instead of Message and 1 as message type. There are 3 message types shown in Fig. 46, Fig. 47, and Fig. 48:



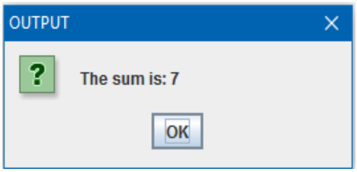
The sum is: 7

Fig. 46. Type 1 Output Dialog box



The sum is: 7

Fig. 47. Type 2 Output Dialog box



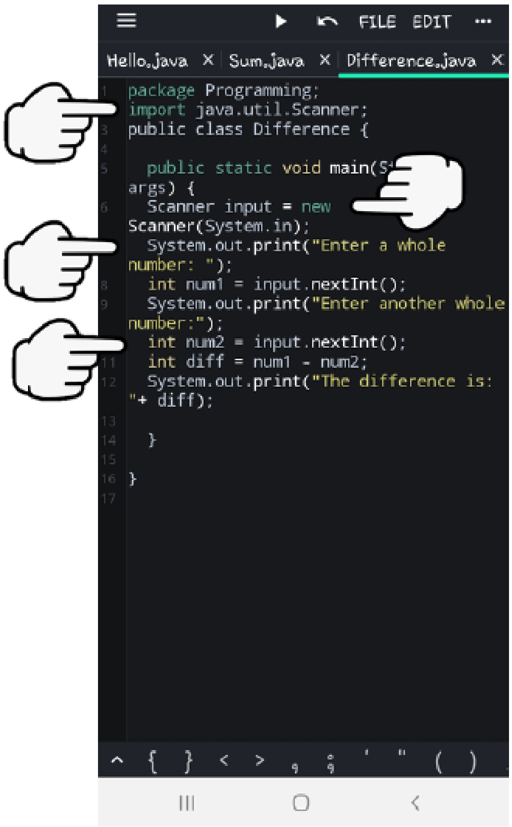
The sum is: 7

Fig. 48. Type 3 Output Dialog box

Using Java N-IDE

Java N-IDE does not support GUI such as JOptionPane. So for programs the need user inputs, Scanner can be used as shown in Fig. 34.

- ✓ Add an import statement **import java.util.Scanner;** before the class name.
- ✓ Declare a scanner variable after the main method that will be used to read the user input: **Scanner input = new Scanner(System.in);**
- ✓ Use System.out.print() to display prompt messages.
- ✓ To input integers use **.nextInt()**, **.nextDouble()** for floating-point, and **.next()** for String.



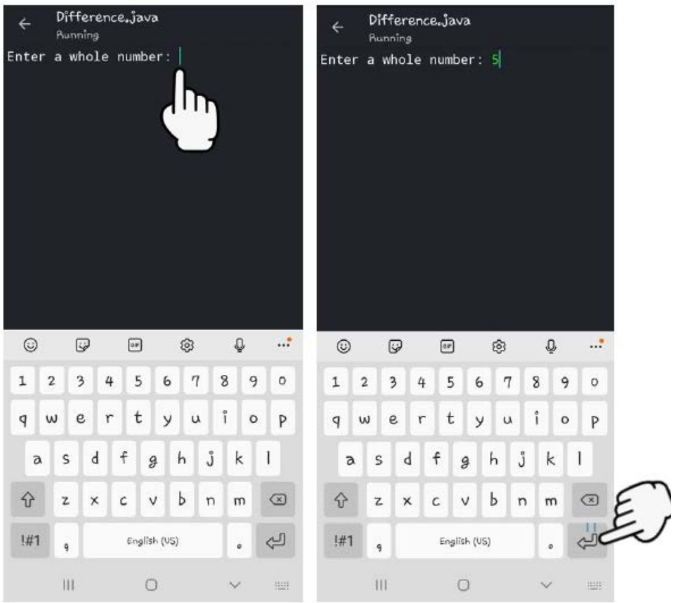
```
package Programming;
import java.util.Scanner;
public class Difference {

  public static void main(S
args) {
  Scanner input = new
Scanner(System.in);
  System.out.print("Enter a whole
number: ");
  int num1 = input.nextInt();
  System.out.print("Enter another whole
number:");
  int num2 = input.nextInt();
  int diff = num1 - num2;
  System.out.print("The difference is:
"+ diff);

  }

}
```

Fig. 34. Program Errors in Java N-IDE

Run the code.

- ✓ Type inputs when prompted as shown in Fig 35.

- ✓ Tap enter on keyboard to proceed.

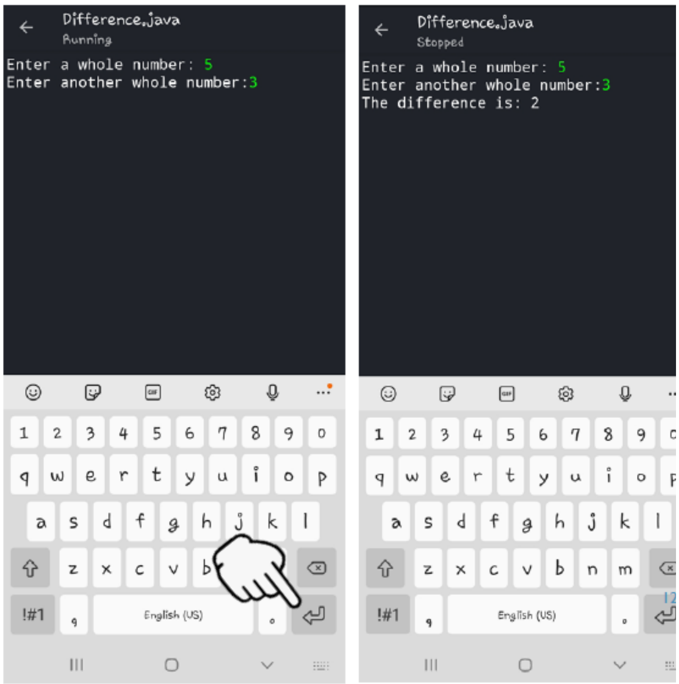


```
Difference.java
Running
Enter a whole number: 
```

```
Difference.java
Running
Enter a whole number: 5
```

Fig. 35. Program Inputs in Java N-IDE

- ✓ Type additional inputs when prompted.
- ✓ Tap enter on keyboard to proceed until the output is displayed as shown in Fig. 36.
- ✓ Run the code again using different inputs to check the accuracy of the output.
- ✓ Edit code as needed.



```
Difference.java
Running
Enter a whole number: 5
Enter another whole number:3
```

```
Difference.java
Stopped
Enter a whole number: 5
Enter another whole number:3
The difference is: 2
```

Fig. 36. Program Inputs and outputs in Java N-IDE

Formative Assessment 2

1. The clerk of the department wants to have a way to input the basic information of each new students such as full name, email address, contact number, and complete home address and will display those data

for information purposes. If you are tasked to create a simple solution, how will you do it? Fill up table below. See Appendix B for sample output screenshot.

Problem Analysis:	**Program Design:**	**Program Coding:**

2. A businessman who owns a small company wants to calculate the net monthly salary of each of his employees using only their names and number of days worked as entries. Since all his employees are minimum wage earners, their daily rate is 500 pesos. Each of them has a monthly deduction of 220 pesos for PhilHealth and 330 pesos for SSS. What will be your solution to this problem? Fill up table below.

Problem Analysis:	**Program Design:**	**Program Coding:**

See Appendix C for sample output screenshot.

CHAPTER 3 SUMMARY

To be able to create programs with ease, you must be familiar with the user interface of the software that you are using like JCreator that has common features of a regular window and some software specific features. It requires you to save your file first before you can start coding. It also helps you detect bugs and suggest solutions but you still have to spend time and effort in detecting bugs that the software cannot identify.

To help you do program coding faster, it is recommended you follow the PDLC since if you have analyzed the problem and created a design, all you have to do is convert it into codes just like in the examples given.

The main() method is the starting point of a Java program and in order to create programs that are interactive, where your program asks for inputs from users, you can use pre-defined classes such as JOptioPane. Always mind the indentation and code grouping so you won’t have a hard time debugging it in case you have a lot of error and tons of lines of code.

You can display output in 2 different ways: in the General Output window or in a dialog box. Your prompt messages should be as specific as possible so that users will not have a hard time figuring out what input is needed. Same goes to outputs, it should be well labeled. Dialog boxes can be customized as to the title and type for a more user-friendly look.

If you have familiarized the logic of creating programs in a certain programming language, it will be easier for you to use other programming languages since the logic is just the same though some keywords and syntax might change.

Programming is not easy. But it is also not that hard when you have constant practice and dedication to what you are doing. It is just like math or any other course where you have to understand a bunch of terms and memorize formulas or formats so you can apply it to a bunch of different situations so you can solve or create a solution for a given problem.

SUMMATIVE ASSESSMENT

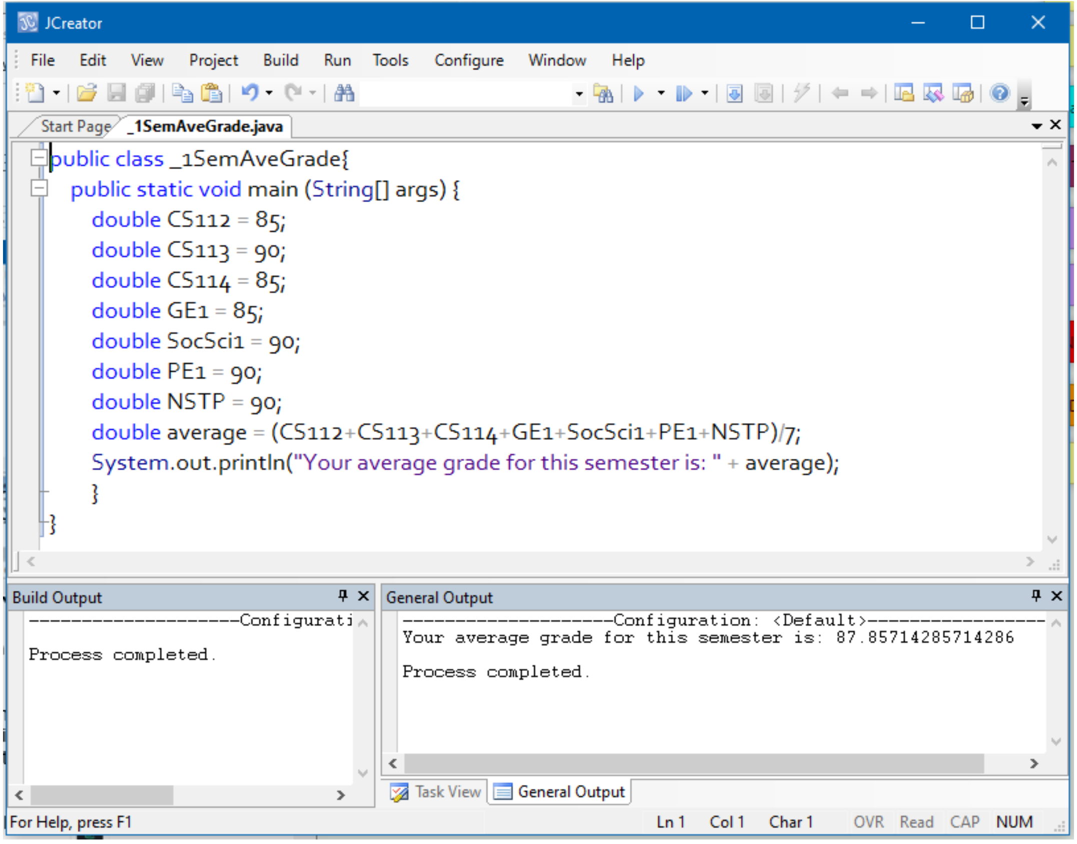
1. Chapter 3 Activity (See VLE)
2. Chapter 3 Quiz (See VLE)

CHAPTER 3 REFERENCES

[1] Burd, B. (2017). *Beginning Programming with Java For Dummies (5th Ed)*. New Jersey: John Wiley and Sons, Inc.
[2] Cavida, D.G, Frio, F.M., Flores, D. (2010). *Computer Programming I*. Unpublished Workbook, University of Southern Mindanao, Kabacan, Cotabato.
[3] Flask, R. (ND). *Java for Beginners 2nd Edition* [PDF File]. Retrieved from http://staff.um.edu.mt/__data/assets/pdf_file/0010/57169/jn.pdf
[4] Mayfield, B. (2016). *From Problem Analysis to Program Design Lab Manual (3rd ed.).* Philippines: Cengage Learning Asia Pte Ltd.

APPENDICES

Appendix A



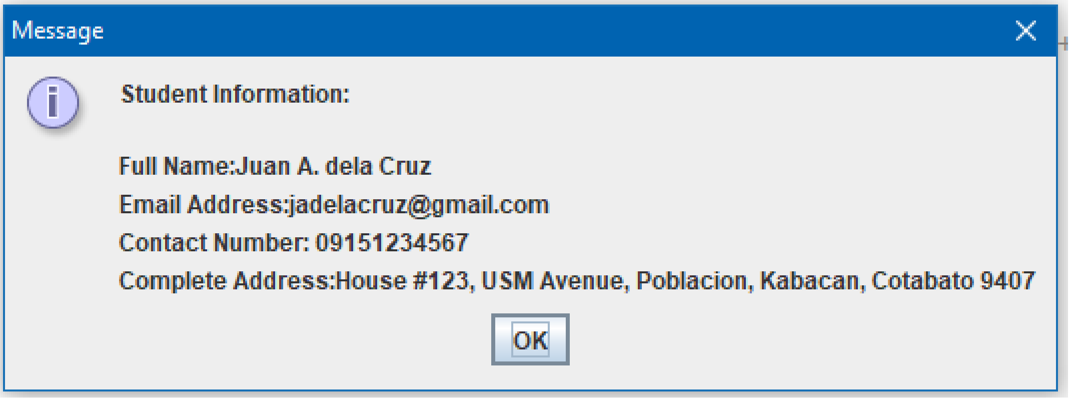
JCreator
File Edit View Project Build Run Tools Configure Window Help
Start Page _1SemAveGrade.java
Build Output
General Output
Task View
For Help, press F1
Ln 1 Col 1 Char 1 OVR Read CAP NUM

```
public class _1SemAveGrade{
  public static void main (String[] args) {
    double CS112 = 85;
    double CS113 = 90;
    double CS114 = 85;
    double GE1 = 85;
    double SocSci1 = 90;
    double PE1 = 90;
    double NSTP = 90;
    double average = (CS112+CS113+CS114+GE1+SocSci1+PE1+NSTP)/7;
    System.out.println("Your average grade for this semester is: " + average);
    }
}
```

```
--------------------Configurati
Process completed.
```

```
--------------------Configuration: <Default>-----------------
Your average grade for this semester is: 87.85714285714286

Process completed.
```

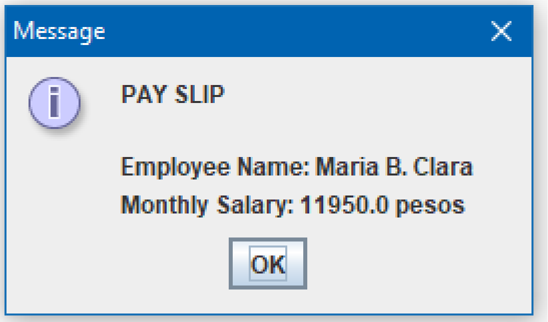
Appendix B



Student Information:

Full Name:Juan A. dela Cruz
Email Address:jadelacruz@gmail.com
Contact Number: 09151234567
Complete Address:House #123, USM Avenue, Poblacion, Kabacan, Cotabato 9407

Appendix C



PAY SLIP

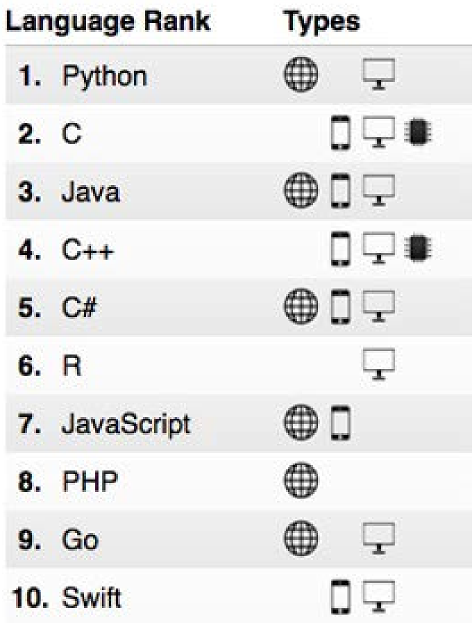
Employee Name: Maria B. Clara
Monthly Salary: 11950.0 pesos

ANSWER KEY

CHAPTER 1

Formative Assessment 1

1.



Language Rank	Types
1. Python	🌐 🖥
2. C	📱 🖥 ■
3. Java	🌐 📱 🖥
4. C++	📱 🖥 ■
5. C#	🌐 📱 🖥
6. R	🖥
7. JavaScript	🌐 📱
8. PHP	🌐
9. Go	🌐 🖥
10. Swift	📱 🖥

Image Source: https://blog.sagipl.com/top-programming-languages/

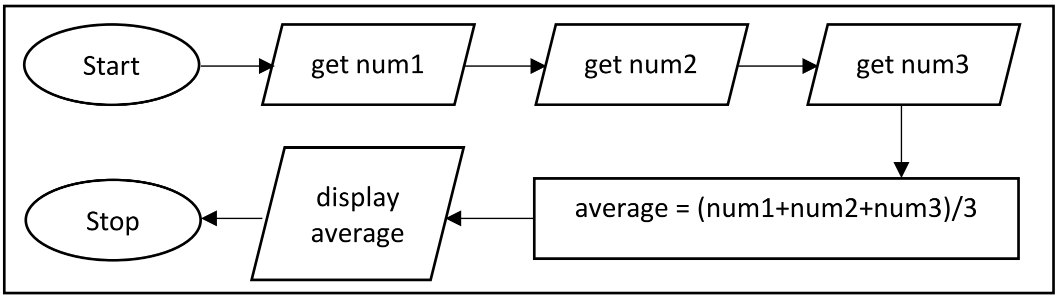
CHAPTER 2

Formative Assessment 1

1. a. Final grades for each subject for 1 semester
   b. Flowchart
   c. Manually compute the average and compare to the output of the program
2. Step 1

   Input: num1, num2, num3
   Process: average = (num1 + num2 + num3) /3
   Output: average

   Step 2



Formative Assessment 2

1. Line Comment
2. Line 1, 3, and 5

Formative Assessment 3

Space, { }, ( ), [ ], and .

Formative Assessment 4

1. Hello, main, args, out, print
2.

Identifiers with only letters	Identifiers with digits	Identifiers with allowed special characters
num	num1	num_1
average	average2	_average2
sum	sum3	sum3_
name	name4	name$4
grade	grade5	grade5$

3.

Activity 1	✗	There is a space	Num_1	✓	
String	✓		$var	✓	
A	✓		Ex@mple	✗	Special char @
MyVariable	✓		i-am-valid	✗	Special char -
main	✓		2ndNumber	✗	Began with digit
ChessTeam2	✓		_crush	✓	

Formative Assessment 5

4: public, class, static, void

Formative Assessment 6

1. Yes, at Line 6 is a string literal.
2.

Integer	Floating-point	Character	String
5	-143.4	‘A’	“USM”
10	3.1416	‘b’	“college”
200	1.00	‘8’	“same as”
350	-2.75	‘$’	“You passed!”
-30	99.99	‘*’	“Submit by 5PM”

Formative Assessment 7

1.

2.5	float
Aspire 3	String
E	char
true	boolean
99	byte

DDR4	String
3.14344	float
802.11ac	String
555	short
z	char

2.

Data/Value	Data Type	Identifier
Danielle	String	name
15	byte	age
1-BSIT	String	course
99.50	float	average
College Scholar	String	scholarship

22314.77	float	tuition

Formative Assessment 8

1.

String name = “Danielle”;
byte age = 15;
String course = “1-BSIT”
double average = 99.50;
double tuition = 22314.77;

2. int age = 20;
3. double z = x + y;
4. int ave = (num1+num2+num3) / 3;
5. String details = “Juan A. dela Cruz”;

Formative Assessment 9

	Formula	Expression
A	Length x Width x Height	Length * Width * Height
B	1/2Base x Height	(Base * Height) / 2
C	Pi x Raduis^2	Pi * Radius * Radius
D	$\frac{b1+b2}{2}$ x (H)	((b1 + b2) / 2) * H
E	4Pi x Radius^3	4*Pi * Radius * Radius * Radius
F	m(c^2)	m * c * c
G	dailyRate x 22 – ((10% of dailyRate x 22) + sss + philhealth)	dailyRate * 22 – (((dailyRate*.10) * 22) + sss + philHealth)

Formative Assessment 10

Provide the result of the following operation if int a = 5, int b = -2, int c = 2	
boolean exp1 = a + b * c <= a + c * b;	false
boolean exp2 = !(c + a > b);	true
boolean exp3 = b - a >= c;	false
Evaluate the value of the following expressions:	
6 – 2 + 10 % 4 + 7	13
int result = 4 + 2 / 1 * 3 – 1;	9

Formative Assessment 11

1.

EXPRESSION	RESULT
(B + C) / A	50
A * B	20
++C	31
(B > A)	true
A--	0
Answer = A + B	21
C!=B	true
C – A	29
A – (B*C)	-599
B==C	false

2.

A>=B	Boolean Expression	
A!C	Not an Expression	Operator placement invalid
A + B + C;	Expression	
B<=C	Boolean Expression	
C==B*1.5	Boolean Expression	

Formative Assessment 12

System.out.print(“I am not a statement”);	Statement
Sum = A + B	Not a Statement
int num1, num2;	Statement
String FamName;	Statement
public class SampleStatement {	Not a Statement

Formative Assessment 13

1. 2 blocks
2. 4 bolcks

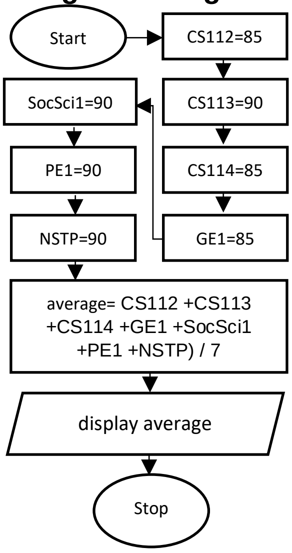
Formative Assessment 14

```
import javax.swing.JOptionPane;
public class average {
    public static void main (String [ ]args) {
int num1=Integer.parseInt(JOptionPane.showInputDialog(“Enter a number: “));
int num2=Integer.parseInt(JOptionPane.showInputDialog(“Enter a number: “));
int num3=Integer.parseInt(JOptionPane.showInputDialog(“Enter a number: “));
int average = (num1+num2+num3)/3;
System.out.print(average);
} }
```

CHAPTER 3

Formative Assessment 1

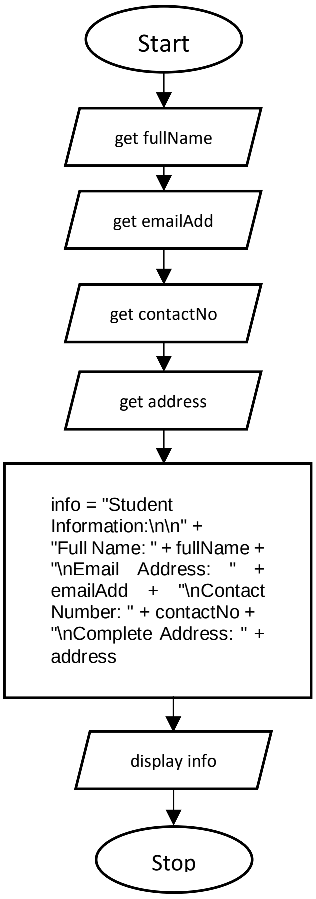
Problem Analysis:	**Program Design:**	**Program Coding:**

Input: None
 	
Process:
CS112 = 85
CS113 = 90
CS114 = 85
GE1 = 85
SocSci1 = 90
PE1 = 90
NSTP = 90
 	
average = (CS112+CS113+CS114+GE1+SocSci1+PE1+NSTP)/7;
 	
Output: average
	 Start
	CS112=85
	CS113=90
	CS114=85
	GE1=85
	SocSci1=90
	PE1=90
	NSTP=90
	average= CS112 +CS113 +CS114 +GE1 +SocSci1 +PE1 +NSTP) / 7
	display average
	Stop
		public class _1SemAveGrade{
		public static void main (String[] args) {
		double CS112 = 85;
		double CS113 = 90;
		double CS114 = 85;
		double GE1 = 85;
		double SocSci1 = 90;
		double PE1 = 90;
		double NSTP = 90;
		double average = (CS112 +CS113 +CS114 +GE1 +SocSci1 +PE1 +NSTP) / 7;
		System.out.println("Your average grade for this semester is: " + average);
		}
		}

Formative Assessment 2

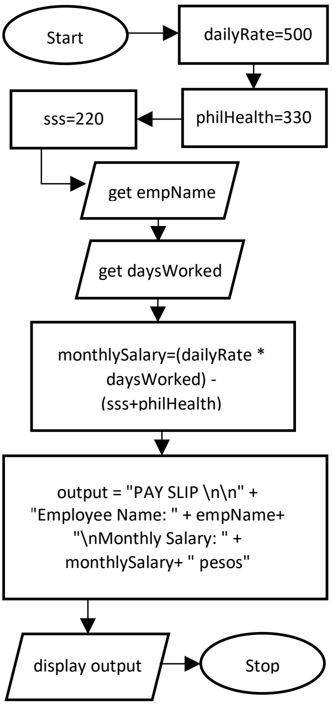
1.

Problem Analysis:	**Program Design:**	**Program Coding:**

Input: fullName. emailAdd, contactNo, address
 	
Process:
info = "Student Information:\n\n" + "Full Name: " + fullName + "\nEmail Address: " + emailAdd + "\nContact Number: " + contactNo + "\nComplete Address: " + address
 	
Output:
info
	 Start
	get fullName
	get emailAdd
	get contactNo
	get address
	info = "Student Information:\n\n" + "Full Name: " + fullName + "\nEmail Address: " + emailAdd + "\nContact Number: " + contactNo + "\nComplete Address: " + address
	display info
	Stop
		import javax.swing.JOptionPane;
		public class Student_Info{
		public static void main (String[] args)
		{
		String fullName =
		JOptionPane.showInputDialog
		("Enter Full Name:");
		String emailAdd =
		JOptionPane.showInputDialog
		("Enter Email Address:");
		String contactNo =
		JOptionPane.showInputDialog
		("Enter 11 digit Contact Number:");
		String address =
		JOptionPane.showInputDialog
		("Enter Complete Address:");
		
		String info = "Student
		Information:\n\n" +
		"Full Name: " +fullName +
		"\nEmail Address: " +emailAdd +
		"\nContact Number: " + contactNo +
		"\nComplete Address: " + address;
		
		JOptionPane.showMessageDialog
		(null, info);
		}
		}

2.

Problem Analysis:	**Program Design:**	**Program Coding:**

Input:
emName, daysWorked
 	
Process:
dailyRate = 500
philHealth = 330
sss = 220
monthlySalary = (dailyRate * daysWorked) - (sss + philHealth)
output = "PAY SLIP \n\n" + "Employee Name: " + empName+ "\nMonthly Salary: " + monthlySalary+ " pesos"
 	
Output:
output
	 Start
	dailyRate=500
	philHealth=330
	sss=220
	get empName
	get daysWorked
	monthlySalary=(dailyRate * daysWorked) - (sss+philHealth)
	output = "PAY SLIP \n\n" + "Employee Name: " + empName+ "\nMonthly Salary: " + monthlySalary+ " pesos"
	display output
	Stop
		import javax.swing.JOptionPane;
		public class SalaryCalculator{
		public static void main (String[] args) {
		int dailyRate = 500;
		int philHealth = 330;
		int sss = 220;
		String empName = JOptionPane.showInputDialog ("Enter Employee's Full Name: ");
		int daysWorked = Integer.parseInt (JOptionPane.showInputDialog("Enter Number of Days Worked for the month: "));
		double monthlySalary = (dailyRate * daysWorked) - (sss+philHealth);
		String output = "PAY SLIP \n\n" +
		"Employee Name: " + empName+
		"\nMonthly Salary: " + monthlySalary+ " pesos";
		JOptionPane.showMessageDialog(null, output);
		}
		}