Introduction to Java

Faculty of Information Technology, Hanoi University

What You Are About To Achieve

❖ By the end of this lecture, we will have gained essential insights and skills to elevate our coding journey:

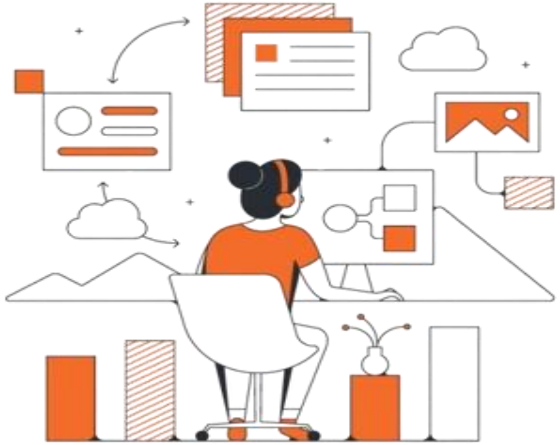
- ❑ **Grasp the fundamentals of algorithms.**
- ❑ **Master the essentials of source code management.**
- ❑ **Set up your development environment.**
- ❑ **Launch your very first program.**
- ❑ **Understand the structure of a basic Java program.**

- ❖ **Introduction to Programming**
- ❖ **Algorithm**
- ❖ Version Control System
- ❖ Launch First Program
- ❖ Program Structure
- ❖ Environment Setup
- ❖ Final Touches

Welcome to the world of programming!

❖ From now on, you're stepping into a domain where you’ll wear many hats. You will be:

A problem solver, identifying challenges and devising the most efficient solutions.



An artist, shaping your code to be not just functional, but elegant and intuitive.

An architect, designing and building robust software structures.

Introduction to *Java Programming*

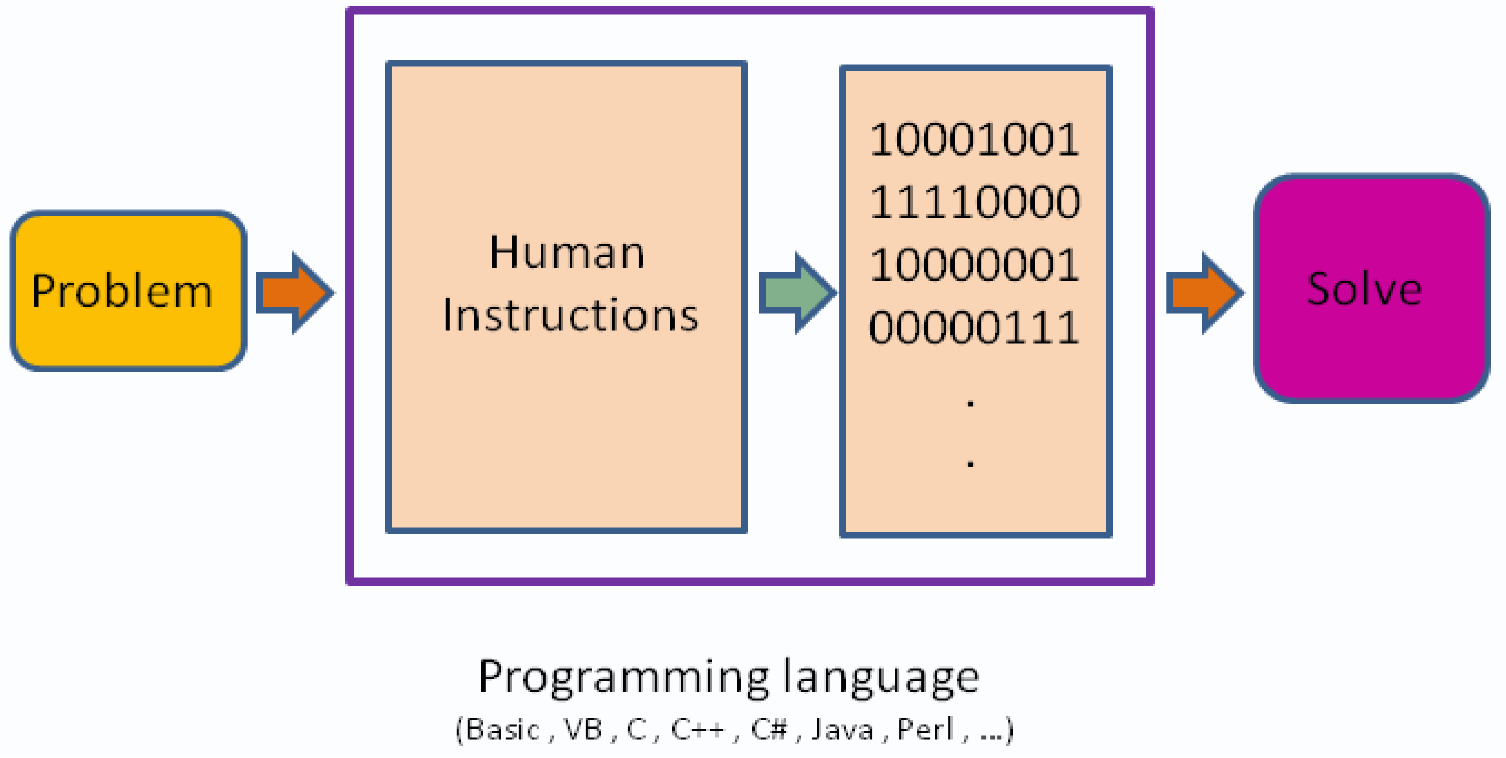
Sound interesting?
Wanna start the journey?

But before we dive deeper, let's understand what programming is all about.

Introduction to *Java Programming*

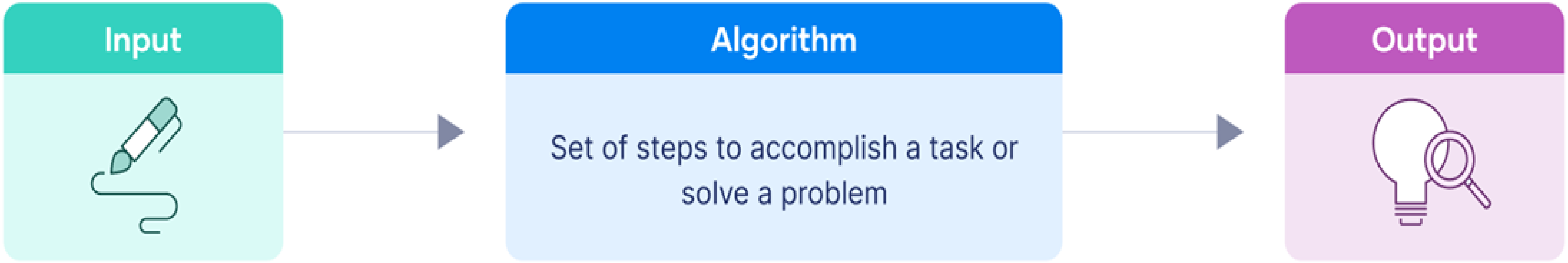
- Programming refers to a *technological process* for telling a computer *which tasks to perform* in order to *solve problems*.

  (You can think of programming as a collaboration between humans and computers, in which *humans create instructions* for a computer to follow *in a language* computers can understand.)



Refs: https://en.wikiversity.org/wiki/Introduction_to_Programming/About_Programming

- Meanwhile, a specific set of step-by-step instructions used to accomplish a particular task is known as ***an algorithm***. They are the *building blocks* for programming, and they allow things like computers, smartphones, and websites to function and make decisions.



Input

Algorithm

Set of steps to accomplish a task or solve a problem

Output

Refs: https://www.scribbr.co.uk/using-ai-tools/algorithms/

Introduction to Algorithm

❖ In addition to being used by technology, a lot of things we do on a daily basis are similar to algorithms. Let's say you want to make **Vietnamese Egg Coffee**. In order to do this successfully, there's a certain **set of steps** you need to follow in a **particular order:**

Refs: https://ethnicspoon.com/vietnamese-egg-coffee-ca-phe-trung/

1 First, you need to **prepare the ingredients**:

- 1 tablespoon of Vietnamese ground coffee
- 1 egg yolk
- 2 tablespoons of sweetened condensed milk
- 1 cup of boiling water

2 After that, you need to **prepare the coffee**:

- Use a Vietnamese drip filter (phin) and place 1 tablespoon of Vietnamese ground coffee into it.
- Place the phin on top of a cup.
- Pour a small amount of boiling water into the phin to let the coffee bloom for 30 seconds.
- Then, fill the phin with boiling water and let the coffee drip into the cup. This will take about 3-5 minutes.

Introduction to Algorithm

- ❖ In addition to being used by technology, a lot of things we do on a daily basis are similar to algorithms. Let's say you want to make **Vietnamese Egg Coffee**. In order to do this successfully, there's a certain **set of steps** you need to follow in a **particular order:**

Refs: https://ethnicspoon.com/vietnamese-egg-coffee-ca-phe-trung/

3 The third step is to **prepare the Egg Mixture**:

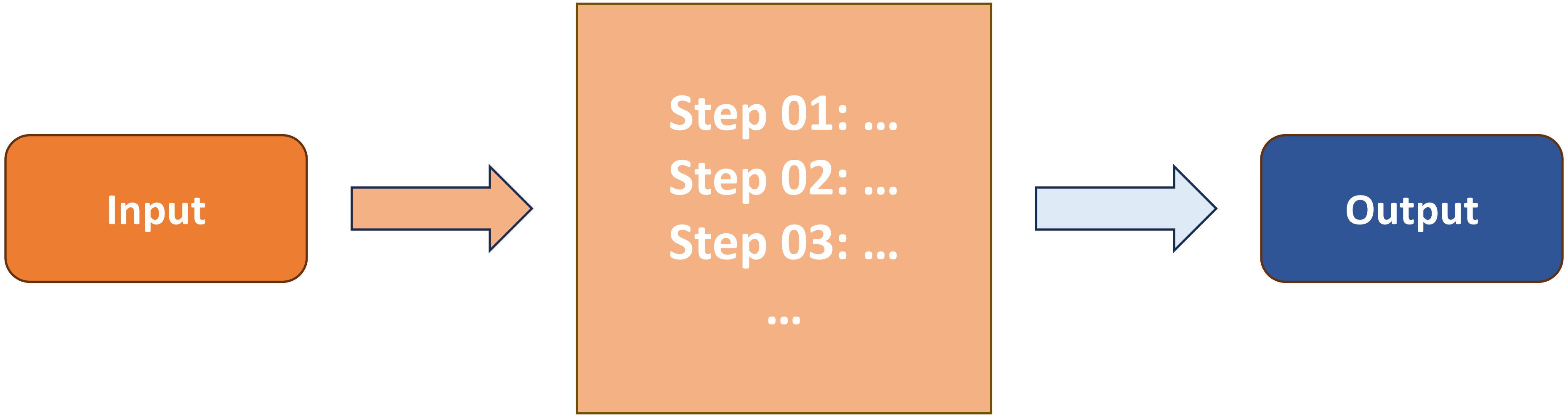
- ➢ Separate the egg yolk from the white.
- ➢ In a bowl, whisk the egg yolk until it becomes frothy and light.
- ➢ Add 2 tablespoons of sweetened condensed milk to the egg yolk and continue whisking until the mixture is creamy and smooth.

4 Finally, you need to **combine Coffee and Egg Mixture**:

- ➢ Pour the freshly brewed hot coffee into a cup.
- ➢ Gently spoon the egg mixture on top of the hot coffee.
- ➢ The egg mixture will float on top, creating a layered effect.

Introduction to Algorithm

- This process is actually **an algorithm**. Because you ***followed these steps in a particular order***, you reached your ***desired outcome***: a delicious cup of Vietnamese Egg Coffee. But if you were to make a mistake like under-whipping the egg yolks or using too much water for the coffee, it probably wouldn't be as good.
- Programs work in a *similar way*. Their ***source code is made up of algorithms*** telling them what to do.



Algorithm

Nobita

So, was that easy to make an egg coffee? Did you remember all the steps?

In some cases, for some reason, you might forget something like preparing the coffee first. Then you have to wait for the coffee!

Delivered

Nobita

Sure! But I think a person wouldn't be that dumb. When they know the receipt, they will understand these steps, right?

Yes. But a computer is that dumb. A computer will only do what you tell it to do. This might make programming frustrating at first, but it's relieving in a way: if you do everything right, you know exactly what the computer is going to do because you told it.

Delivered

Nobita

So, what’s the solution?

To avoid such mistakes and ensure clarity, we can translate our logic into an illustrated form. This method ensures that your algorithm is well-organized, understandable, and ready for implementation.

Delivered

In some cases, for some reason, you might forget something like preparing the coffee first. Then you have to wait for the coffee!

Delivered

Nobita

Sure! But I think a person wouldn't be that dumb. When they know the receipt, they will understand these steps, right?

Yes. But a computer is that dumb. A computer will only do what you tell it to do. This might make programming frustrating at first, but it's relieving in a way: if you do everything right, you know exactly what the computer is going to do because you told it.

Delivered

Nobita

So, what's the solution?

To avoid such mistakes and ensure clarity, we can translate our logic into an illustrated form. This method ensures that your algorithm is well-organized, understandable, and ready for implementation.

Delivered

Nobita

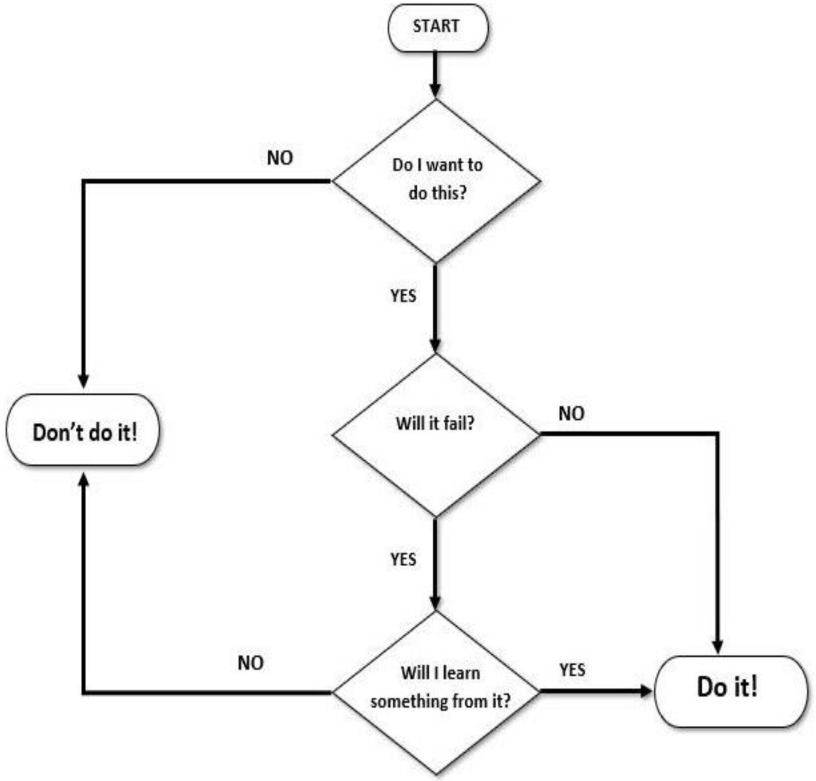
Really? Can you show me how to do this?

Introduction to Algorithm

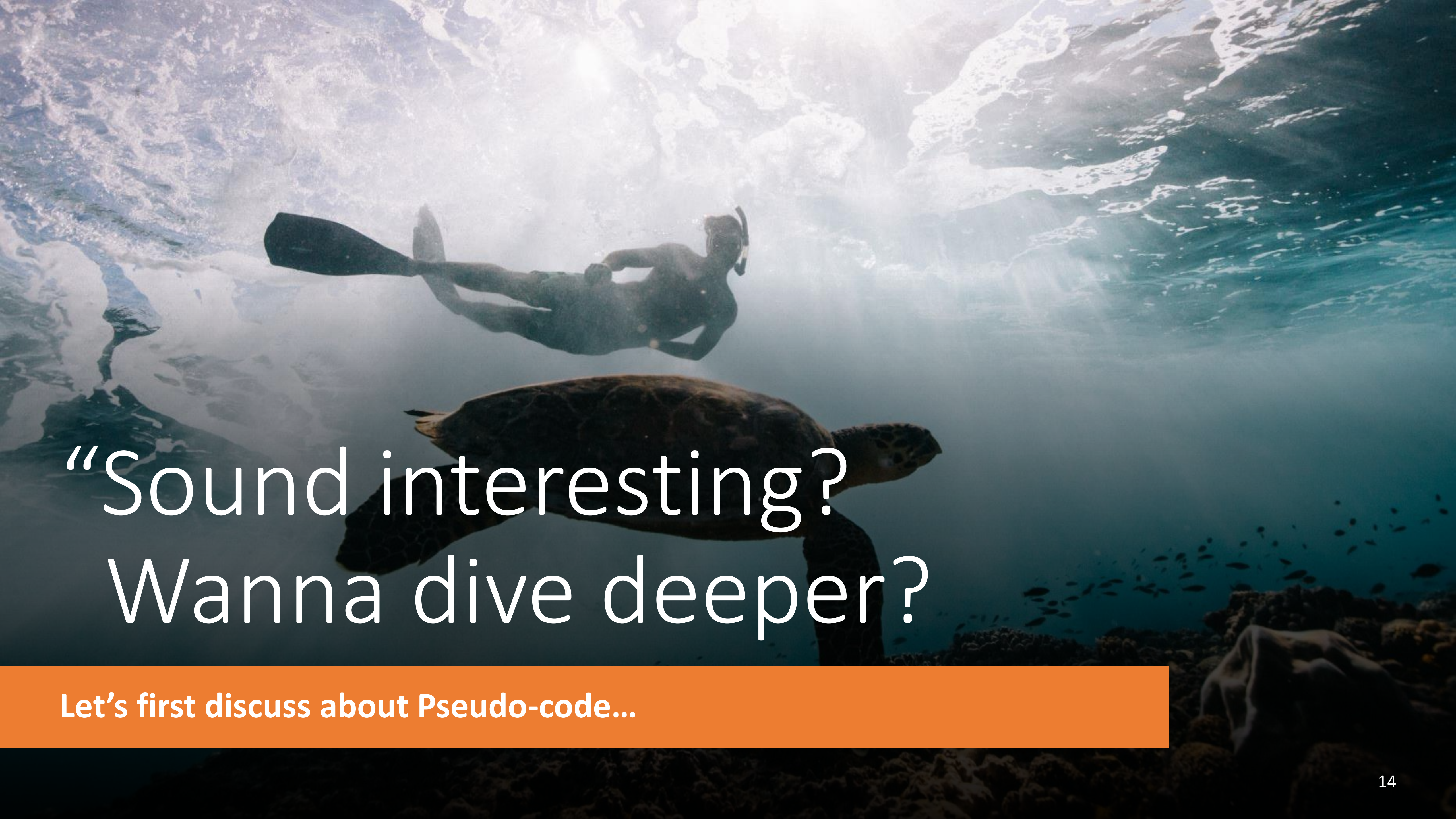
- ❖ To represent our algorithms, there are 2 main methods:
  - ❑ Using **Pseudo-code**
  - ❑ Using **Flowchart**

```
BEGIN
 DISPLAY "Welcome to the Guessing Game!"
 SET secretNumber TO RANDOM NUMBER BETWEEN 1 AND 100
 SET guess TO 0
 SET attempts TO 0
 WHILE guess IS NOT EQUAL TO secretNumber
  DISPLAY "Enter your guess (between 1 and 100):"
  INPUT guess
  INCREMENT attempts BY 1
  IF guess < secretNumber THEN
   DISPLAY "Too low! Try again."
  ELSE IF guess > secretNumber THEN
   DISPLAY "Too high! Try again."
  ELSE
   DISPLAY "Congratulations! You've guessed the right number."
   DISPLAY "It took you " + attempts + " attempts."
  END IF
 END WHILE
 DISPLAY "Thank you for playing!"
END
```

Pseudo-code



Flowchart



“Sound interesting? Wanna dive deeper?

Let’s first discuss about Pseudo-code…

*Introduction to **Algorithm** – Pseudo-code*

❖ **Let’s start with the logic of a game called Guessing game:**

1. Tell the user to pick a secret number between 1 and 100.
2. The smallest possible number is 1; the largest possible is 100.
3. Make a guess that is halfway between the smallest and largest (round down if necessary).
4. Ask the user if your guess is too large, too small or correct.
5. If they say you’re correct, you win and the game is over.
6. If they say your guess is too small, the smallest possible number is now the guess plus one.
7. If they say your guess is too large, the largest possible number is now the guess minus one.
8. Unless you guessed correctly, go back to step 3.

Nobita

I’ve got this Guessing Game logic written out, but it feels like something’s missing.

Yeah, writing out the full logic like that is a start, but it can get confusing, especially for complex problems.

Delivered

Nobita

Exactly! It’s easy to lose track of what’s happening

That’s where pseudo-code comes in. It’s like a simplified version of the program, focusing on the steps without worrying about syntax.

Delivered

Nobita

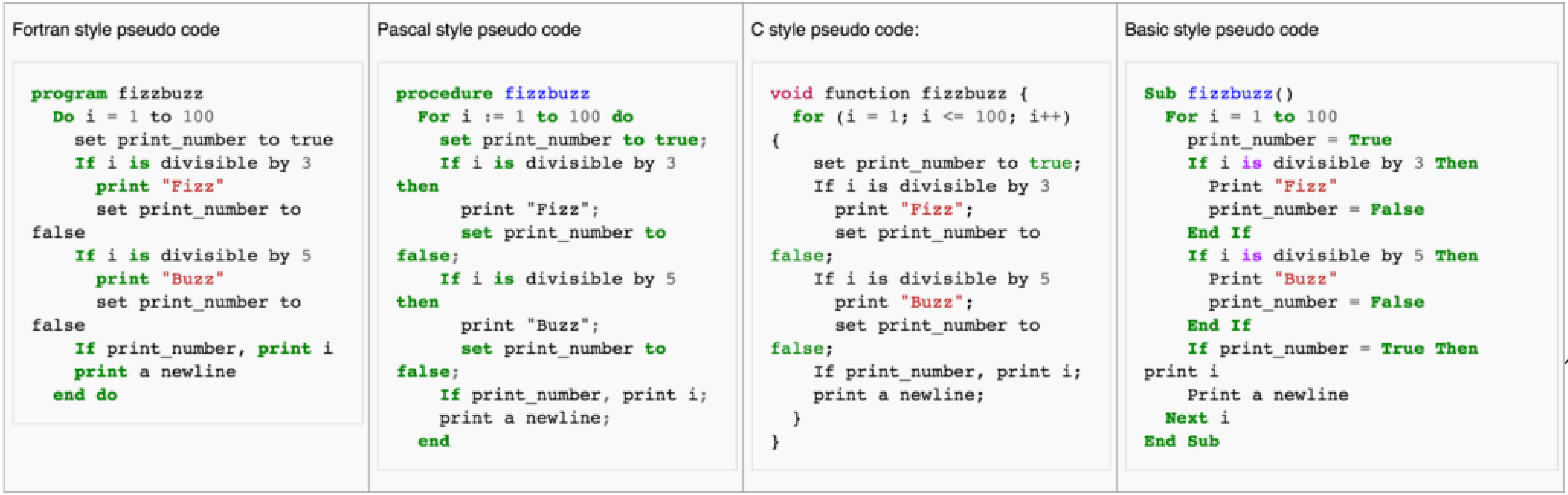
So, it’s like a bridge between the idea and the actual code?

Exactly! Let’s explore what is Pseudo-code and how to use it.

Delivered

Introduction to Algorithm – Pseudo-code

- ❖ “Pseudo” means “almost” or “imitation”!
- ❖ “Code” refers to the instructions written in a programming language.
- ❖ Pseudo code, therefore, means almost code or an imitation of actual computer instructions. These instructions are phrases written in ordinary natural language (e.g., English).



Fortran style pseudo code

```
program fizzbuzz
  Do i = 1 to 100
    set print_number to true
    If i is divisible by 3
      print "Fizz"
      set print_number to
false
    If i is divisible by 5
      print "Buzz"
      set print_number to
false
    If print_number, print i
    print a newline
  end do
```

Pascal style pseudo code

```
procedure fizzbuzz
  For i := 1 to 100 do
    set print_number to true;
    If i is divisible by 3
then
        print "Fizz";
        set print_number to
false;
    If i is divisible by 5
then
        print "Buzz";
        set print_number to
false;
    If print_number, print i;
    print a newline;
  end
```

C style pseudo code:

```
void function fizzbuzz {
  for (i = 1; i <= 100; i++)
{
    set print_number to true;
    If i is divisible by 3
      print "Fizz";
      set print_number to
false;
    If i is divisible by 5
      print "Buzz";
      set print_number to
false;
    If print_number, print i;
    print a newline;
  }
}
```

Basic style pseudo code

```
Sub fizzbuzz()
  For i = 1 to 100
    print_number = True
    If i is divisible by 3 Then
      Print "Fizz"
      print_number = False
    End If
    If i is divisible by 5 Then
      Print "Buzz"
      print_number = False
    End If
    If print_number = True Then
print i
    Print a newline
  Next i
End Sub
```

*Introduction to **Algorithm** – **Pseudo-code***

- ❖ Pseudo code is made up of the following basic logic structures that have been proved to be sufficient for writing any computer program.
- ❖ There are 3 programming/pseudocode constructs:
  1. **Sequence:** It refers that instructions should be executed one after another.
  2. **Selection:** This construct is used to make a decision in choosing an option from many available options on the basis of a condition. So, if a condition is true then one option would be chosen while if a condition is false then another option will be chosen.
  3. **Repetition:** This construct is used to repeat a block of code as per the given condition.

For more details on how to write these, please read materials from Resources on the LMS.

*Introduction to **Algorithm** – **Pseudo-code***

Despite Pseudo-code’s lack of a defined syntax, there are several common programming constructs that developers often use when writing Pseudo-code. Let’s take a look at each.

Pseudo-code - Sequence

- A **sequence** is a group of statements that are executed in a specific order. They're used to perform or repeat a series of simple actions. Some familiar sequence commands commonly used in pseudocode include **INPUT**, **SET**, **PRINT**, **READ**, **DISPLAY**, **SHOW**, and **CALCULATE**.
- Here's an example that uses some of these commands:

```
BEGIN
   DISPLAY "Welcome to the Guessing Game!"
   SET secretNumber TO RANDOM NUMBER BETWEEN 1 AND 100
   SET guess TO 0
   SET attempts TO 0
<< other code >>
DISPLAY "Thank you for playing!"
END
```

Pseudo-code - Selection

- **Selection** statements allow a program to make decisions based on certain conditions, then direct the program to execute certain statements if a condition is met (or not met). **IF-ELSE**, **IF-ELSE IF-ELSE**, and **CASE** statements are frequently utilized in pseudocode.
- Here’s an example showing an IF-ELSE script in Pseudo-code:

```
BEGIN
<< other code >>
  IF guess < secretNumber THEN
      DISPLAY "Too low! Try again."
    ELSE IF guess > secretNumber THEN
      DISPLAY "Too high! Try again."
    ELSE
      DISPLAY "Congratulations! You've guessed the right number."
      DISPLAY "It took you " + attempts + " attempts."
    END IF
<< other code >>
END
```

Pseudo-code - Repetition

- **Repetition** statements repeat a set of steps within a larger function or process. They're often tasked to perform the same operation on multiple items in a list or to repeat a process until certain conditions are met.
- Iterations are useful for repeating a set of steps multiple times and can be implemented using various types of loops, including **FOR**, **WHILE**, and **DO-WHILE** loops.
- Let's look at some Pseudo-code that uses a **WHILE** loop:

```
BEGIN
<< other code >>
   WHILE guess IS NOT EQUAL TO secretNumber
      DISPLAY "Enter your guess (between 1 and 100):"
      INPUT guess
      INCREMENT attempts BY 1
      << SELECTION statements >>
END WHILE
<< other code >>
END
```

Introduction to ***Algorithm*** ***– Pseudo-code***

This is just one demonstration of pseudocode's flexibility. The key is to ensure that the logic is stable while using names that are popular enough to be read by anyone.

Introduction to Algorithm – Pseudo-code

❖ Let see how the Guessing game represented using Pseudo-code!

1. Tell the user to pick a secret number between 1 and 100.
2. The smallest possible number is 1; the largest possible is 100.
3. Make a guess that is halfway between the smallest and largest (round down if necessary).
4. Ask the user if your guess is too large, too small or correct.
5. If they say you're correct, you win and the game is over.
6. If they say your guess is too small, the smallest possible number is now the guess plus one.
7. If they say your guess is too large, the largest possible number is now the guess minus one.
8. Unless you guessed correctly, go back to step 3.

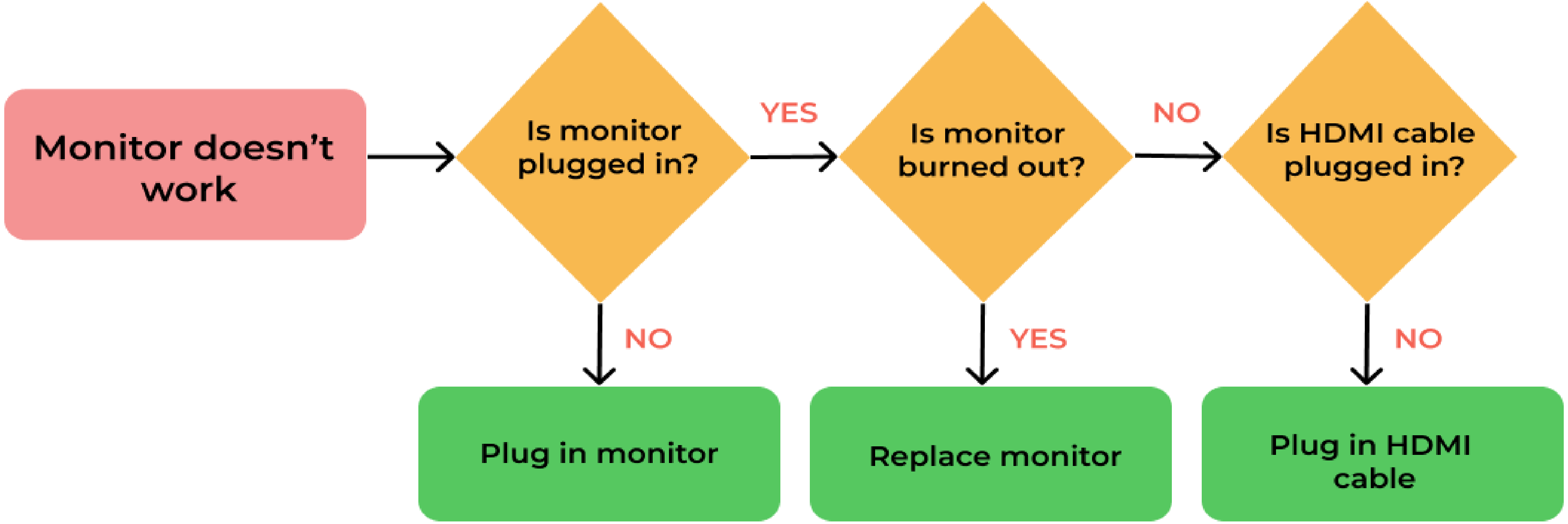
```
BEGIN
    DISPLAY "Pick a secret number between 1 and 100."
    SET smallest TO 1
    SET largest TO 100
    SET guess TO 0
    WHILE True
        SET guess TO (smallest + largest) / 2
        DISPLAY "My guess is " + guess
        DISPLAY "Is my guess too large, too small, or correct?"
        INPUT userResponse
        IF userResponse IS "correct" THEN
            DISPLAY "I win! The number is " + guess
            BREAK
        ELSE IF userResponse IS "too small" THEN
            SET smallest TO guess + 1
        ELSE IF userResponse IS "too large" THEN
            SET largest TO guess - 1
        END IF
    END WHILE
    DISPLAY "Thank you for playing!"
END
```

Introduction to Algorithm – Pseudo-code

- Here you can see that Pseudo-code requires less time and effort to represent and is easier to convert into a programming language. However, there are some limitations:
  - ❑ In case of Pseudo-code, a graphic representation of program logic is not available.
  - ❑ There are no standard rules to follow in using Pseudo-code.
  - ❑ Different programmers use their own style of writing Pseudo-code communication problems occur due to lack of standardization.
  - ❑ For a beginner, it is more difficult to follow the logic or write the Pseudo-code.

Introduction to Algorithm – Flowchart

- The flowchart is a diagram which visually presents the flow of data through processing systems. This means by seeing a flow chart one can know the operations performed and the sequence of these operations in a system. Algorithms are nothing but sequence of steps for solving problems. So a flowchart can be used for representing an algorithm. A flowchart, will describe the operations (and in what sequence) are required to solve a given problem.



Monitor doesn't work

Is monitor plugged in?

YES

Is monitor burned out?

NO

Is HDMI cable plugged in?

NO

Plug in monitor

YES

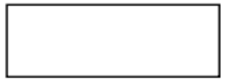
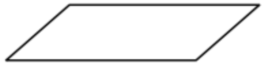
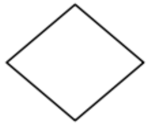
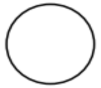
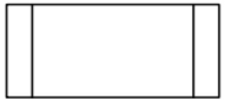
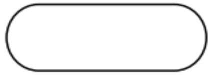
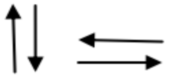
Replace monitor

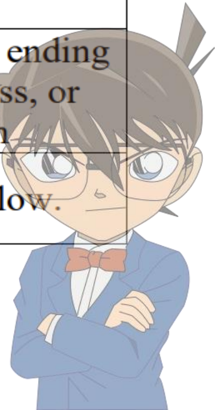
NO

Plug in HDMI cable

Introduction to Algorithm – Flowchart

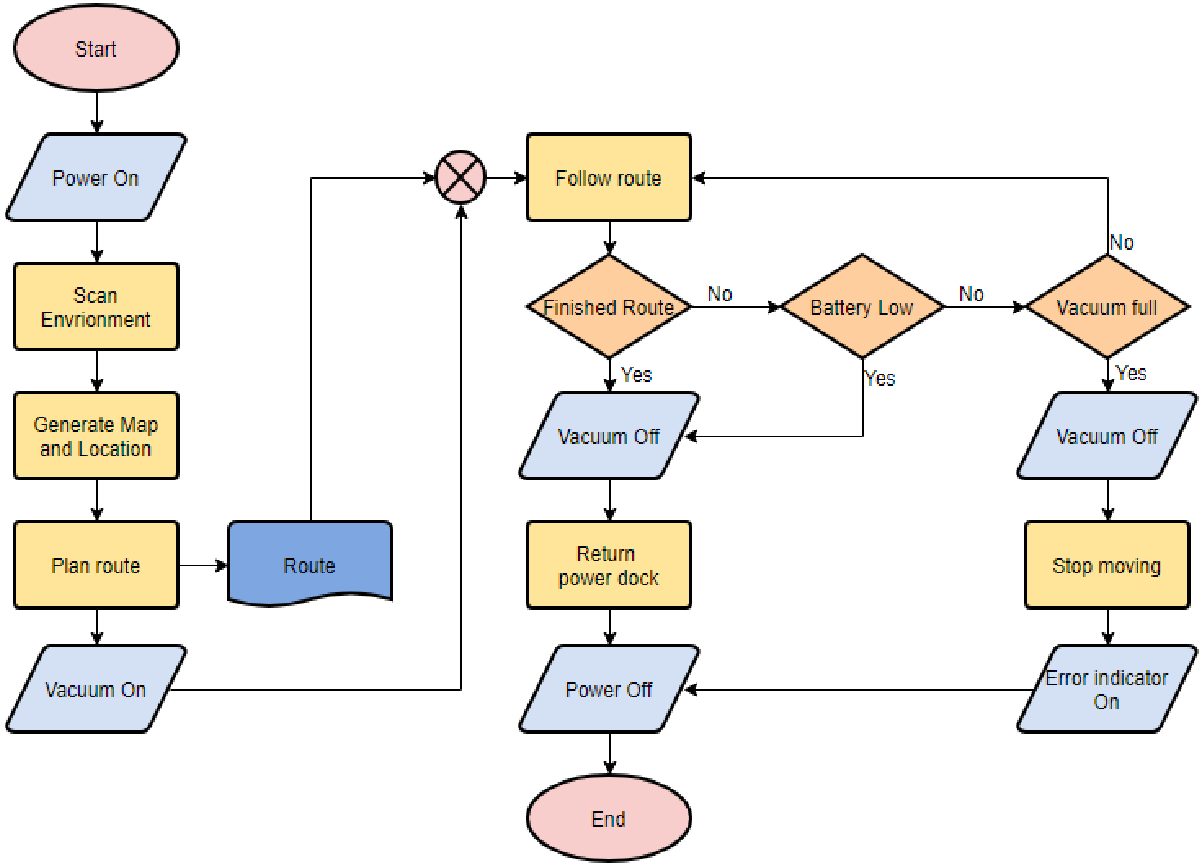
- ❖ You'll notice that the flowchart has different shapes.
- ❖ In this case, there are **two shapes**: those with *rounded ends* represent the start and end points of the process and rectangles are used to show the interim steps. These shapes are known as flowchart symbols. There are dozens of symbols that can be used in a flowchart.
- ❖ In this course, we will use *only some symbols commonly used* in flowcharting of Assembly language programs.

Symbol	Name	Function
	**Process**	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

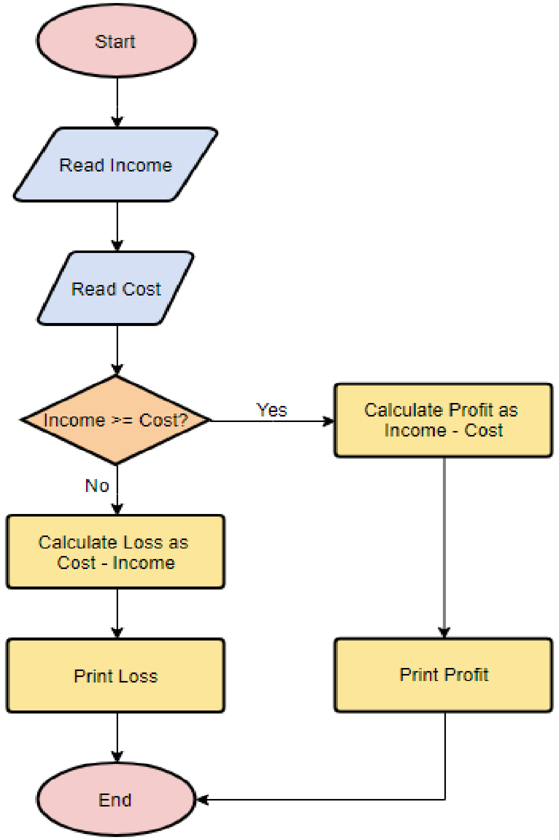


Introduction to Algorithm – Flowchart

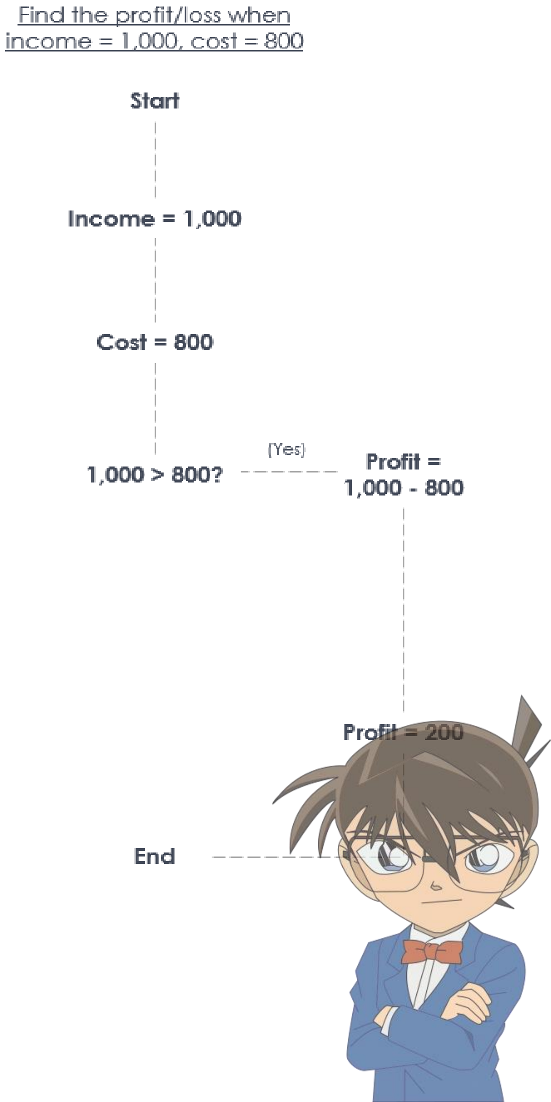
❖ Let’s see some examples of Flowchart:



Start
Power On
Scan Envrionment
Generate Map and Location
Plan route
Route
Vacuum On
Follow route
Finished Route
No
Yes
Battery Low
No
Yes
Vacuum full
No
Yes
Vacuum Off
Return power dock
Power Off
End
Vacuum Off
Stop moving
Error indicator On



Start
Read Income
Read Cost
Income >= Cost?
Yes
Calculate Profit as Income - Cost
No
Calculate Loss as Cost - Income
Print Loss
Print Profit
End



Find the profit/loss when income = 1,000, cost = 800

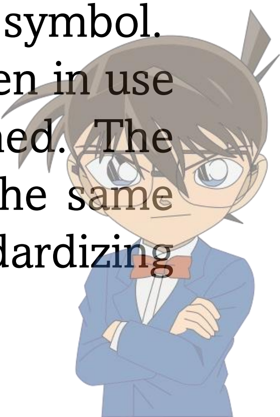
Start
Income = 1,000
Cost = 800
1,000 > 800?
(Yes)
Profit = 1,000 - 800
Profit = 200
End

Introduction to Algorithm – Flowchart

❖ Easy, isn't it? Let's try some and remember to follow some rules:

1. All boxes of the flowchart are connected with Arrows. (Not lines)
2. Flowchart symbols have an entry point on the top of the symbol with no other entry points. The exit point for all flowchart symbols is on the bottom except for the Decision symbol.
3. The Decision symbol has two exit points; these can be on the sides or the bottom and one side.
4. Generally a flowchart will flow from top to bottom. However, an upward flow can be shown as long as it does not exceed 3 symbols.
5. Connectors are used to connect breaks in the flowchart.

   Examples are:
   - From one page to another page.
   - From the bottom of the page to the top of the same page.
   - An upward flow of more then 3 symbols
6. Subroutines and Interrupt programs have their own and independent flowcharts.
7. All flow charts start with a Terminal or Predefined Process (for interrupt programs or subroutines) symbol.
8. All flowcharts end with a terminal or a contentious loop. Flowcharting uses symbols that have been in use for a number of years to represent the type of operations and/or processes being performed. The standardised format provides a common method for people to visualise problems together in the same manner. The use of standardised symbols makes the flow charts easier to interpret, however, standardizing symbols is not as important as the sequence of activities that make up the process.

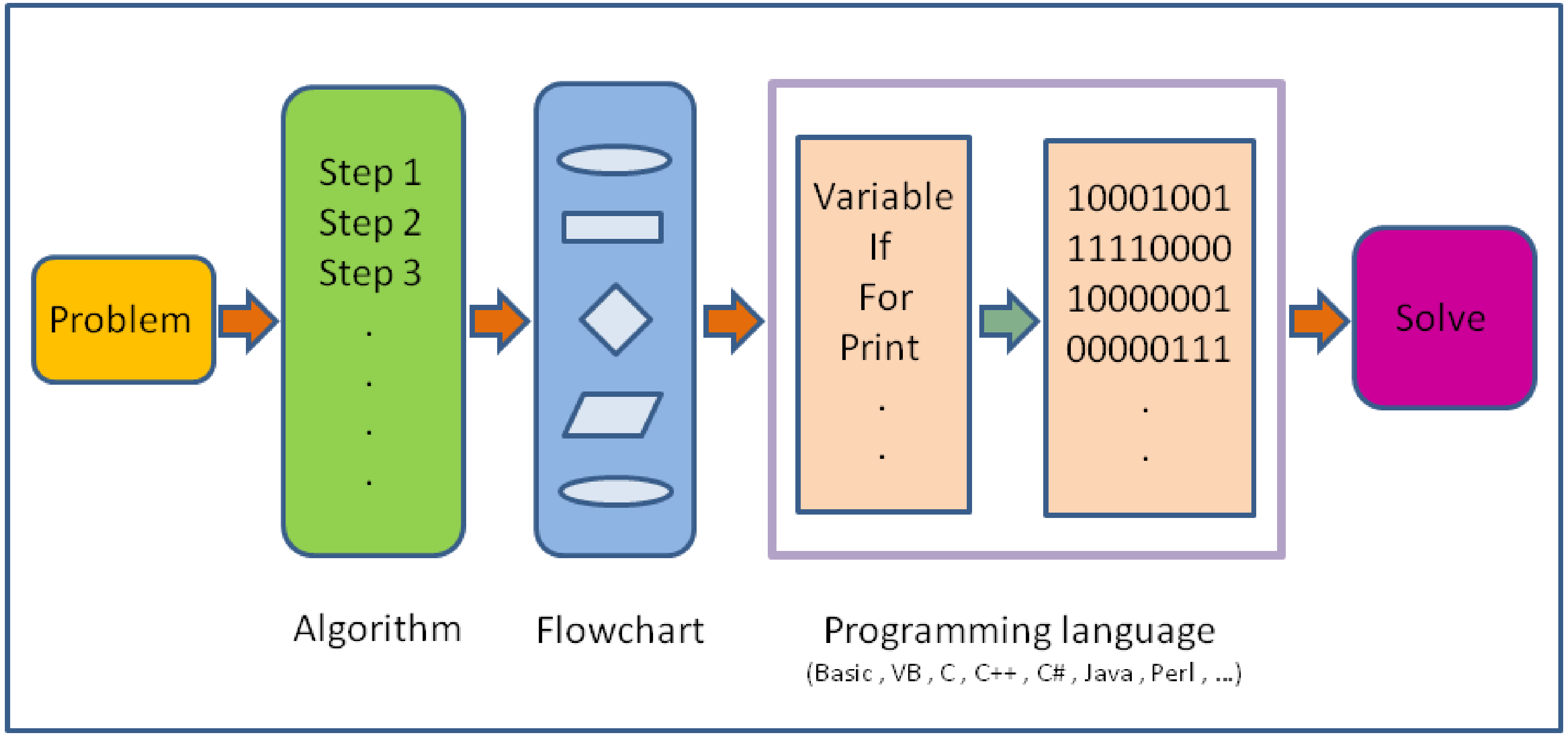


M-A Question:

What if we combine Programming and Flowchart?

Introduction to ***Algorithm***

When combining a flowchart with the process to solve a problem, we got something like this:



Refs: https://en.wikiversity.org/wiki/Introduction_to_Programming/About_Programming

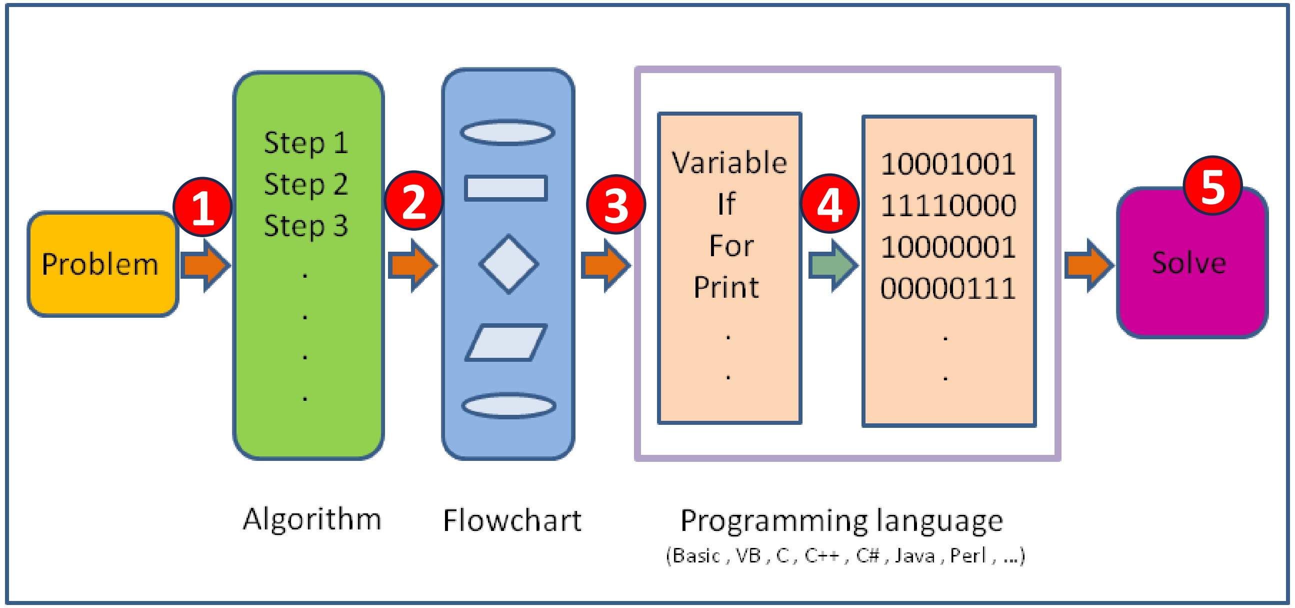
*Introduction to **Algorithm***

❖ And the workflow becomes…

● The process starts with a problem or a request that needs a solution.

1 First, we conceptualize the solution through algorithms - a step-by-step procedure to solve the problem.

2 Next, we translate these algorithms into a flowchart. A flowchart visually represents the sequence of steps and decisions using standardized symbols.



Refs: https://en.wikiversity.org/wiki/Introduction_to_Programming/About_Programming

3 The flowchart guides the development of actual code in a programming language (Java, in this case). This step translates the abstract solution into a structured, executable form.

4 The code is then translated by a compiler or interpreter into machine language (bytecode, in this case), which the computer can directly execute.

5 Finally, the computer executes the machine code, following the instructions laid out in the original algorithm, and then produces the desired result.

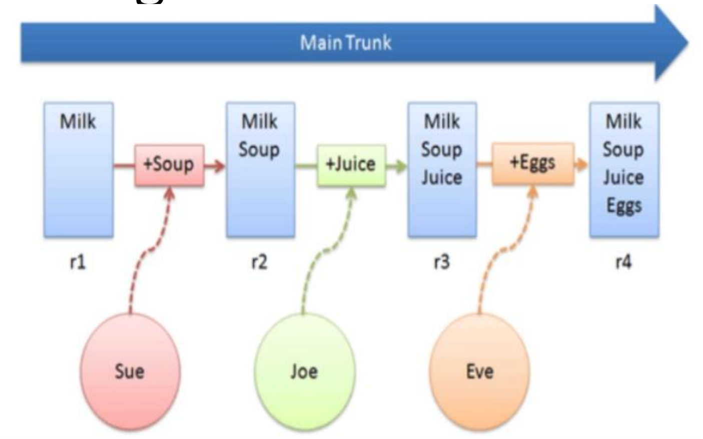
That’s all about Algorithms, do you have any question?

If not? Let’s discuss the following problem…

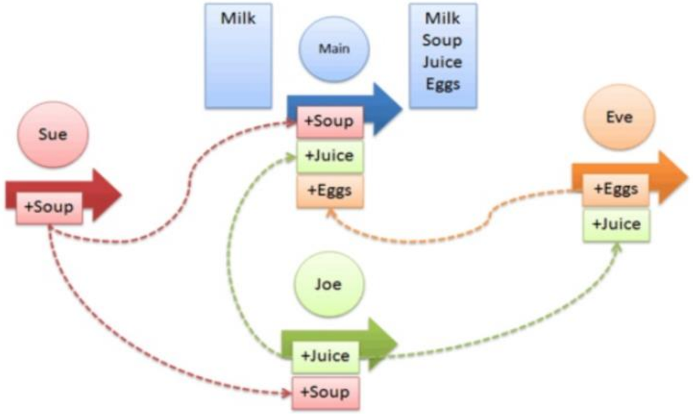
- Introduction to Programming
- Algorithm
- **Version Control System**
- Launch First Program
- Program Structure
- Environment Setup
- Final Touches

Version Control System

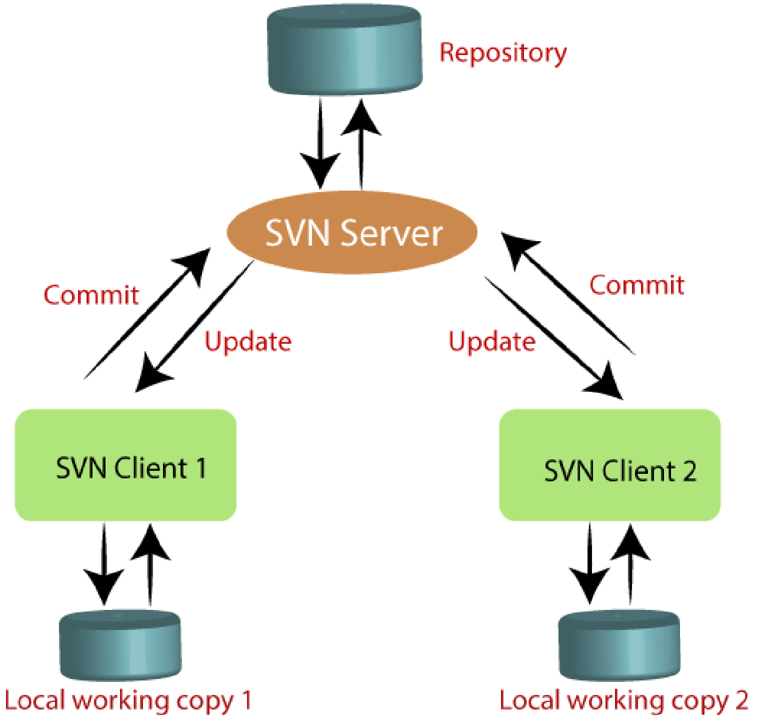
- **SVN (Subversion)** is an open-source **Version Control System** that helps track and manage changes to source code in software development projects, supporting team collaboration, version management, and merging changes from multiple developers.
- **Version Control System (VCS)** is a software that helps software developers to work together and maintain a complete history of their work. It is divided into two categories:



Centralized Version Control System (CVCS)



Distributed/Decentralized Version Control System (DVCS)

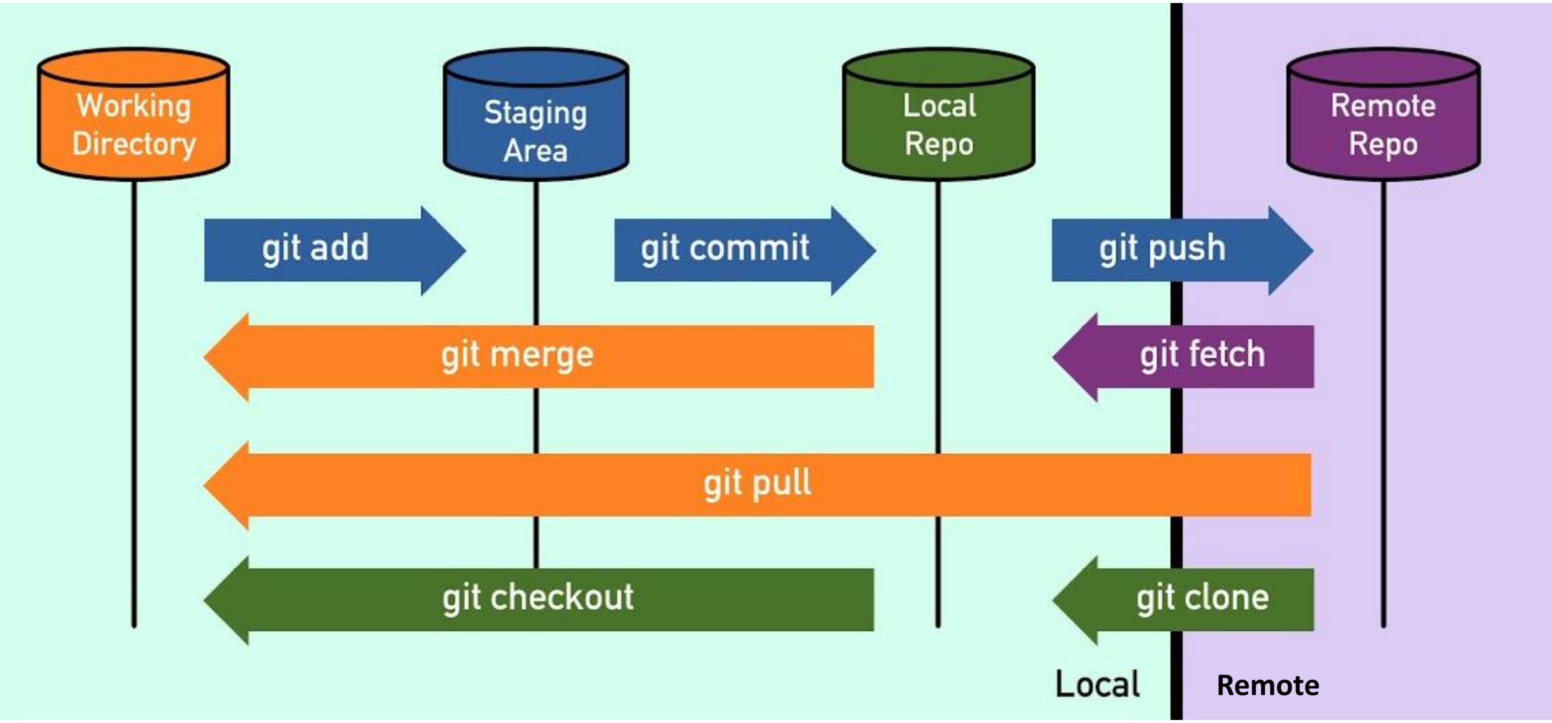


Ref: https://www.javatpoint.com/svn

Fun fact: Fsoft uses TortoiseSVN

Version Control System

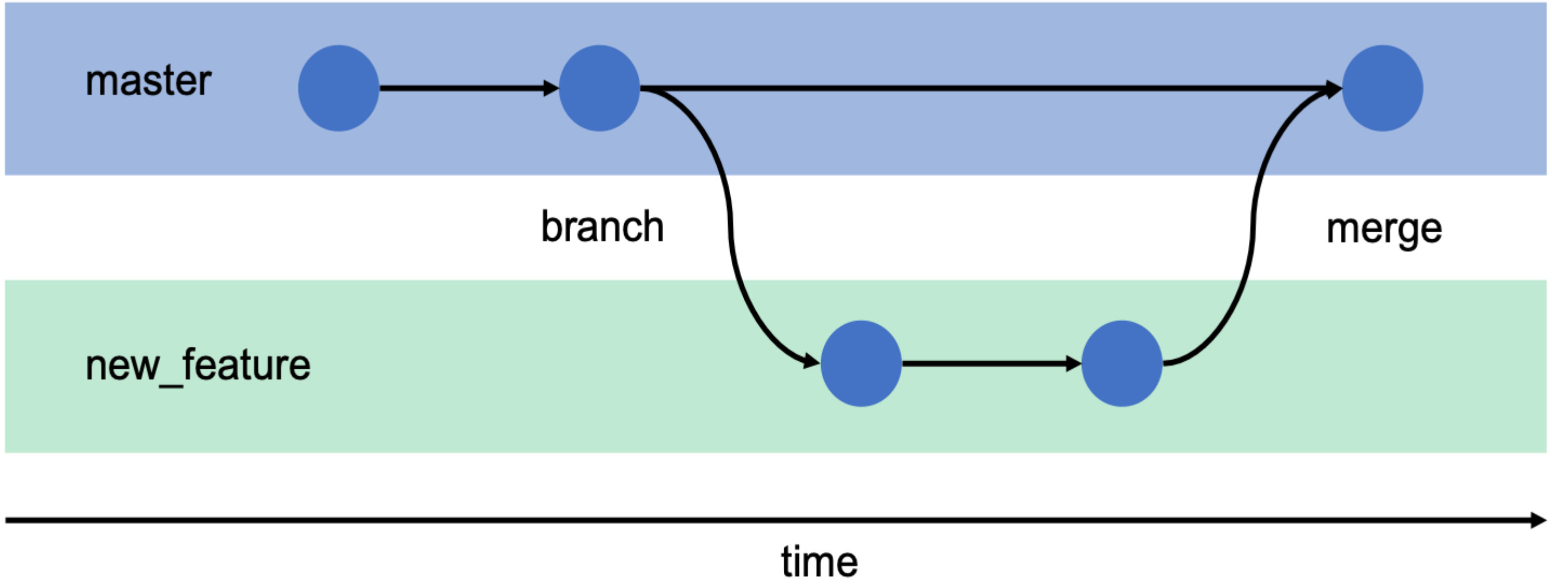
- ❖ **Git** is a **distributed version control system** which we'll focus on in this course.
- ❖ Unlike SVN, which depends on a central server, Git gives every developer their own full copy of the entire repository. This means you can track changes locally before sharing them with others, making collaboration smoother and enabling work even when you're offline.



Ref: How Git Works: Explained in 4 Minutes **(Youtube)**

Version Control System

- ❖ One of Git's powerful features is its branching and merging capabilities. You can create branches to experiment with new features without disturbing the main codebase. Once you're satisfied, merging those changes back is straightforward. This flexibility is why Git is the tool of choice in the industry, with platforms like GitHub and GitLab built around it.

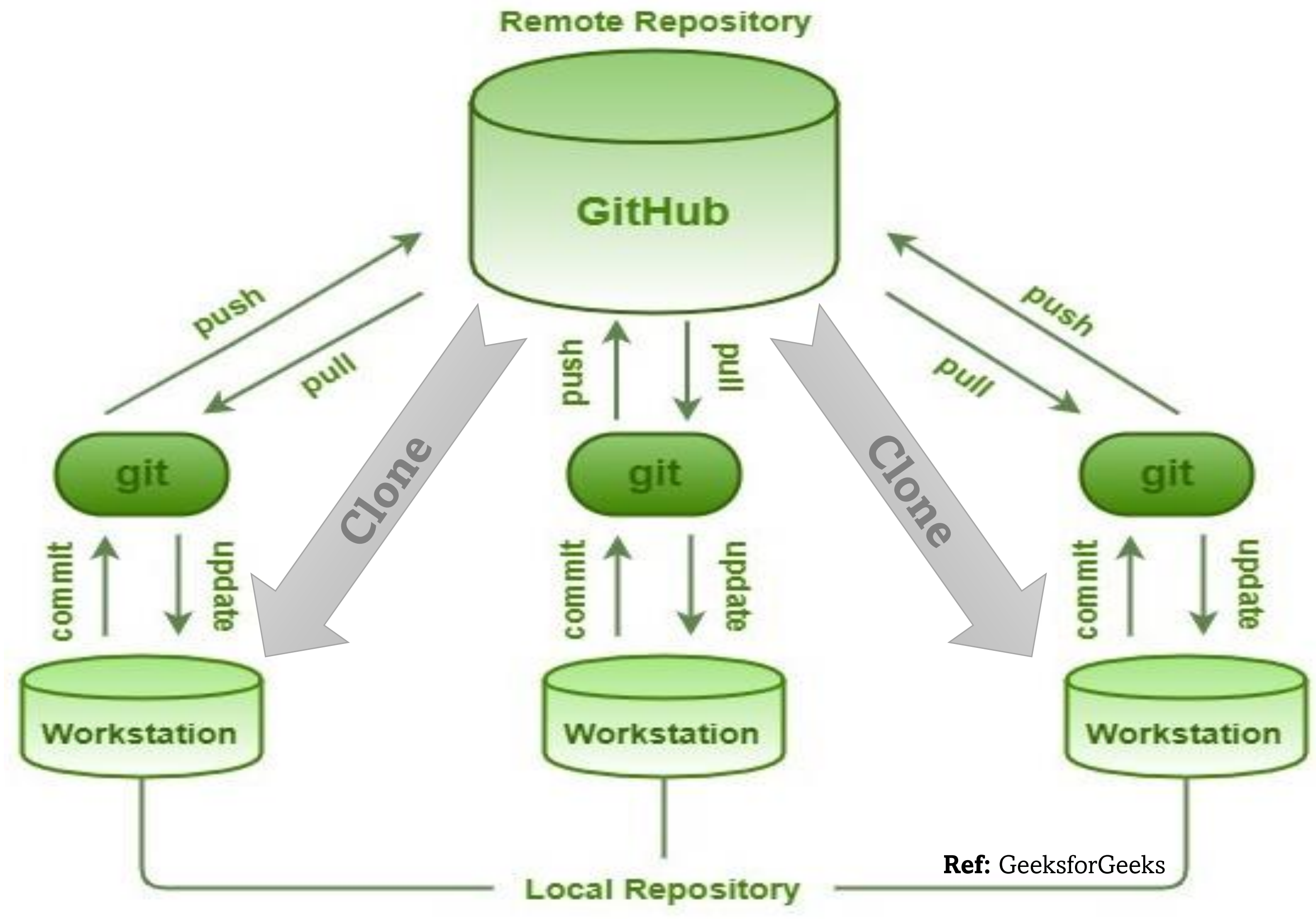


Ref: https://gitbookdown.dallasdatascience.com/branching-git-branch.html

In the upcoming slides, we'll dive into how Git works and how you can leverage its features to manage your projects efficiently.

Version Control System - Git

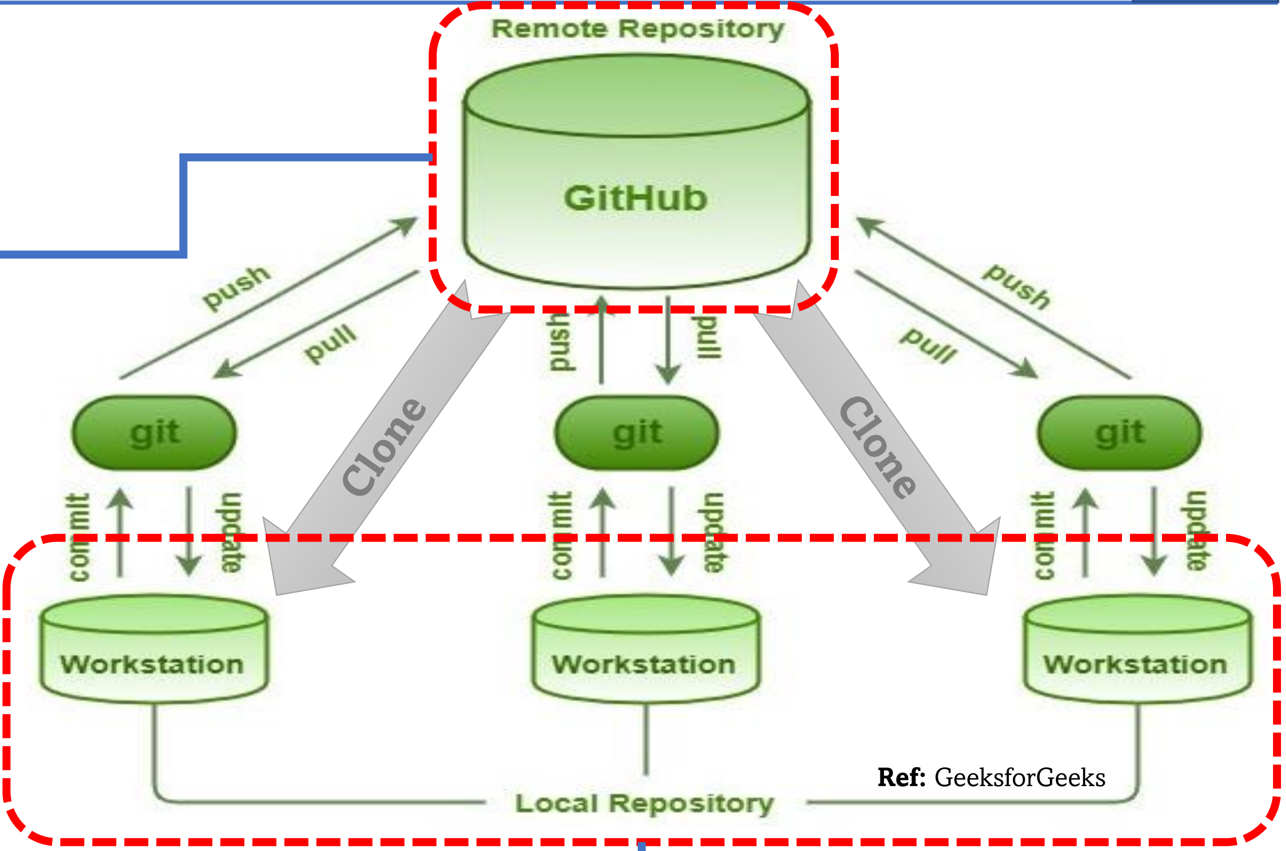
- Next, we'll explore **key terminologies** and the essential **Git commands** associated with them. Understanding these will help you navigate Git more effectively.



Ref: GeeksforGeeks

Version Control System - Git

Remote: A remote repository is a version of your project that is hosted on the internet or another network. It allows multiple collaborators to work on the same project.



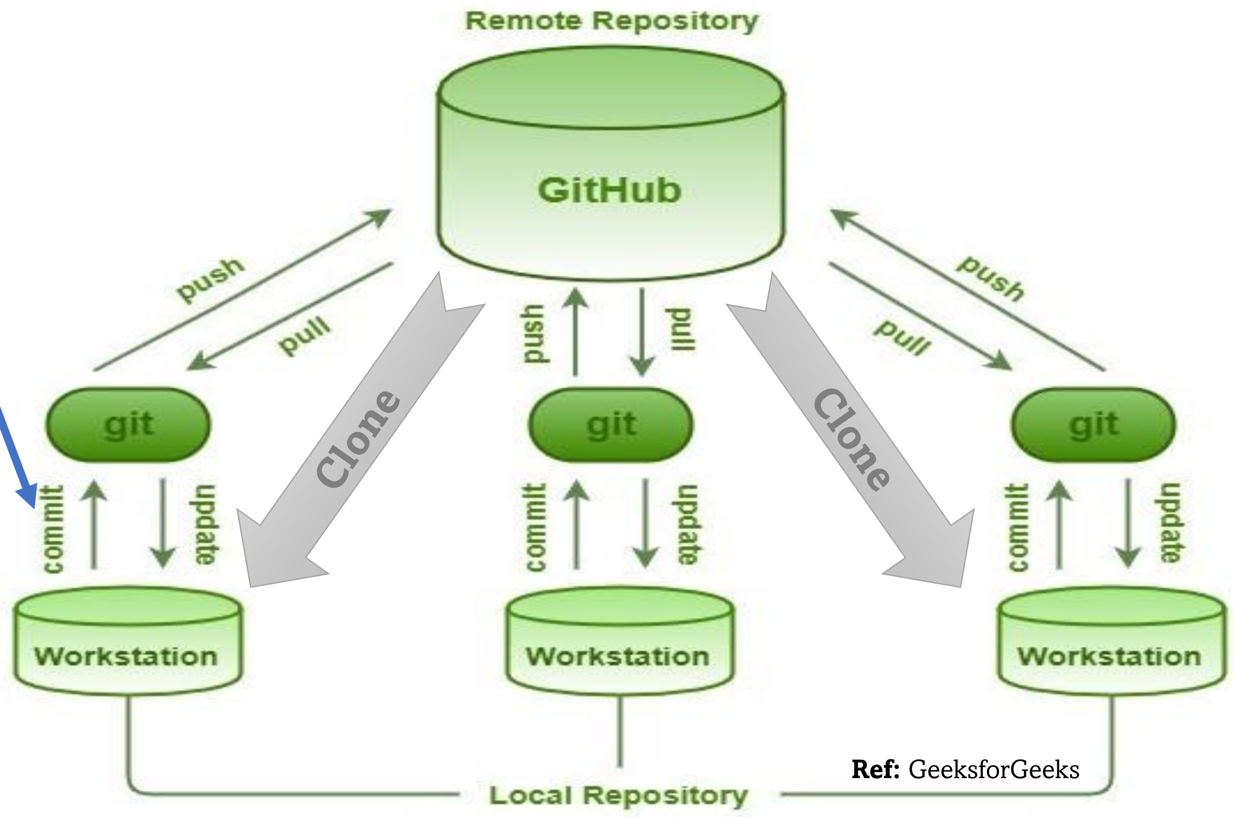
Ref: GeeksforGeeks

Repository (Repo): A directory or storage space where your project's files and the entire history of their changes are stored. There are 2 kinds of repo: **Local** Repo and **Remote** Repo.

Local: This refers to the version of your repository that is on your own machine. It includes your working directory, staging area, and local repository.

Version Control System - Git

Commit: A snapshot of changes made to the files in your repository. Each commit has a unique ID and includes a message describing what was changed.



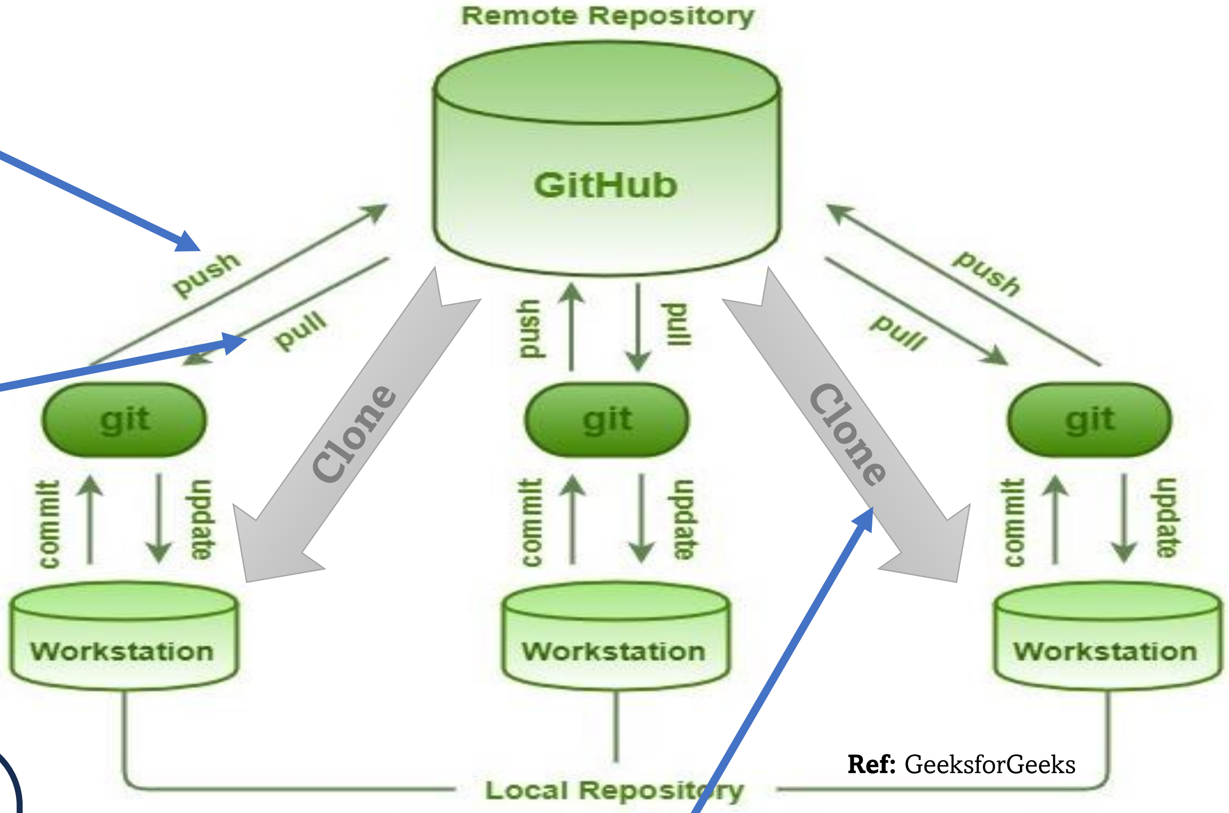
Ref: GeeksforGeeks

Snapshot: In Git, a snapshot is a record of the entire working directory at a particular point in time. Each **commit** in Git creates a snapshot of all the files in your project, as opposed to storing differences between file versions.

Version Control System - Git

Push: The act of sending your committed changes to a remote repo.

Pull: The act of fetching changes from a remote repo and merging them into your local repo.



Ref: GeeksforGeeks

Clone: A copy of a repository that you can work on locally. When you clone a repository, you download all its files, branches, and history.

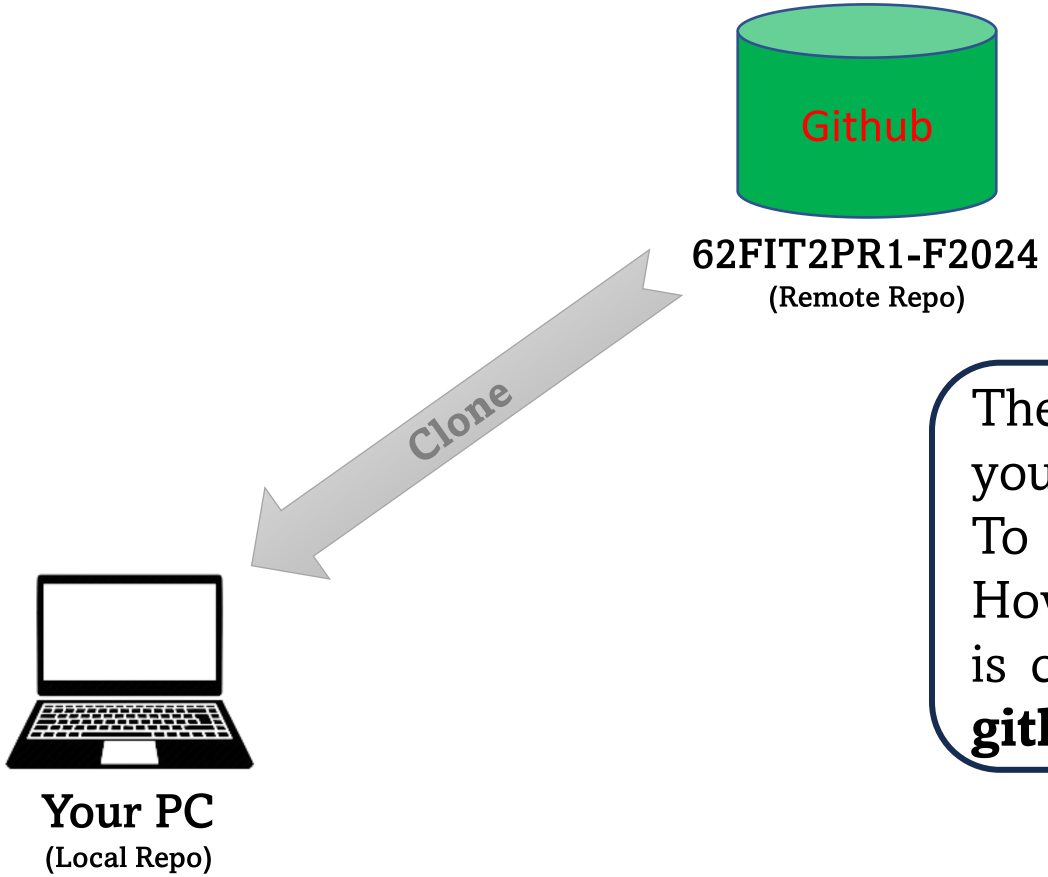
Note to avoid cloning a remote repository multiple times into the same local directory. Each clone operation should be directed to a separate folder. Cloning the same repository into the same location can lead to conflicts and overwriting of files, potentially causing loss of data or corruption of the local repository.

Feeling a bit challenged?

Let’s visualize them.

Version Control System - Git

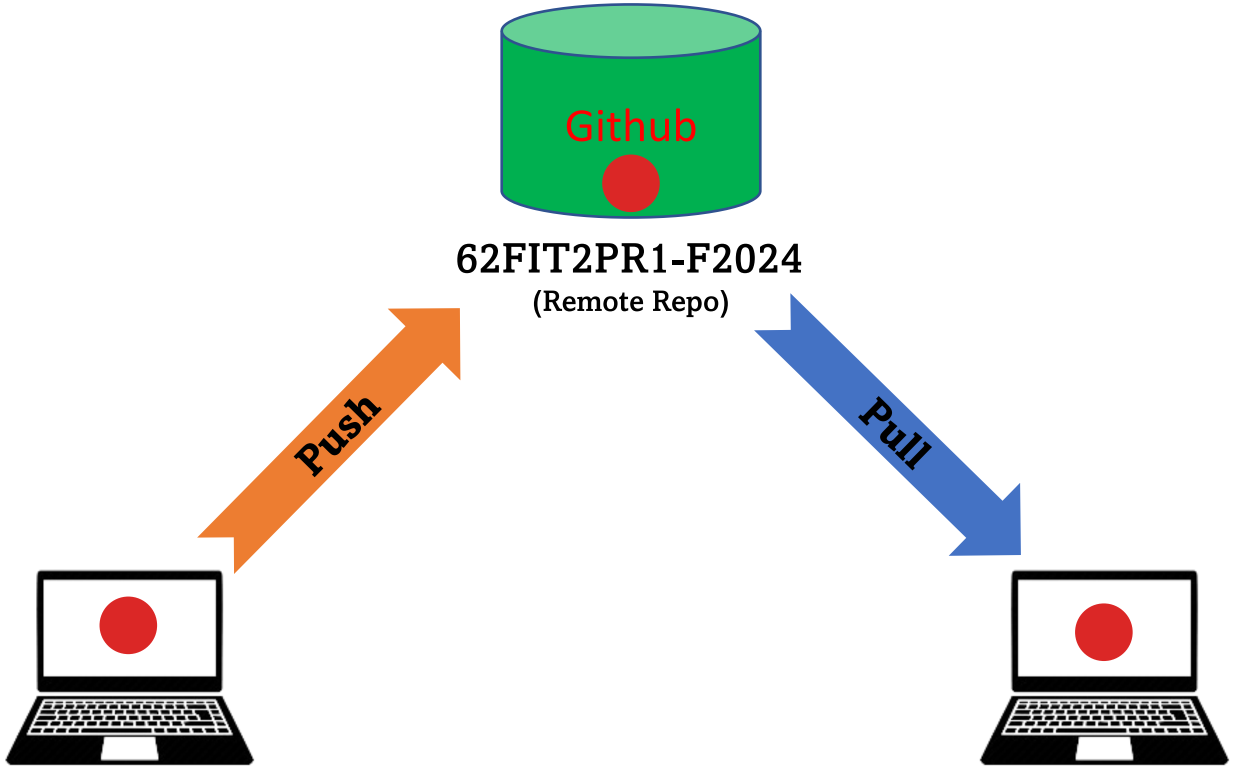
❖ Imagine you have created a repository on Github called **62FIT2PR1-F2024**.



The first time you access this repository, you won't have it locally on your machine. To get it, you need to download it. However, thanks to Git, all you need to do is clone it using the command "**git clone github.com/username/your_repo.git**"

Version Control System - Git

❖ Imagine you have created a repository on Github called **62FIT2PR1-F2024**.



Github

62FIT2PR1-F2024

(Remote Repo)

Push

Pull

Your PC

(Local Repo)

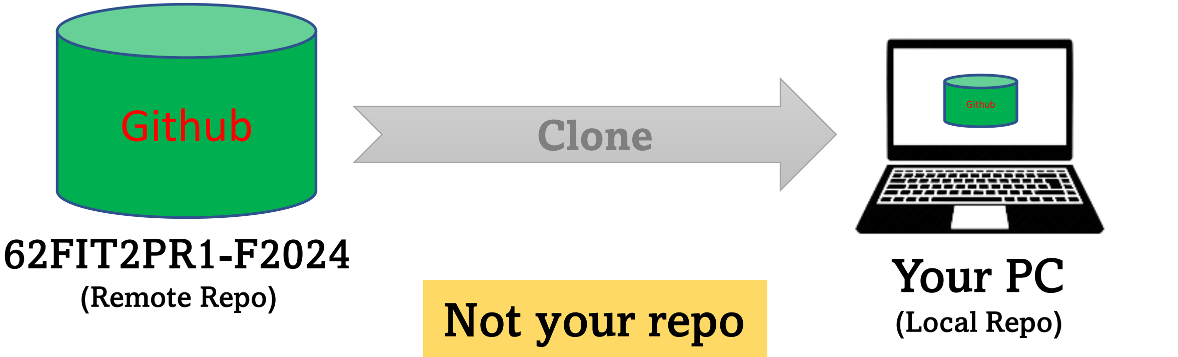
Another PC

(Local Repo)

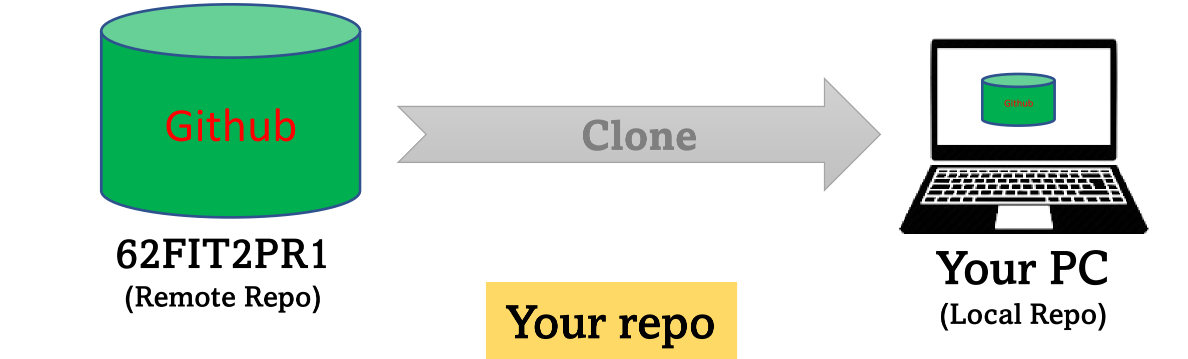
After cloning the repo, you won't have to clone it again, at least in the current folder. From now on, you can either **pull** or **push** the updated code.

Version Control System - Git

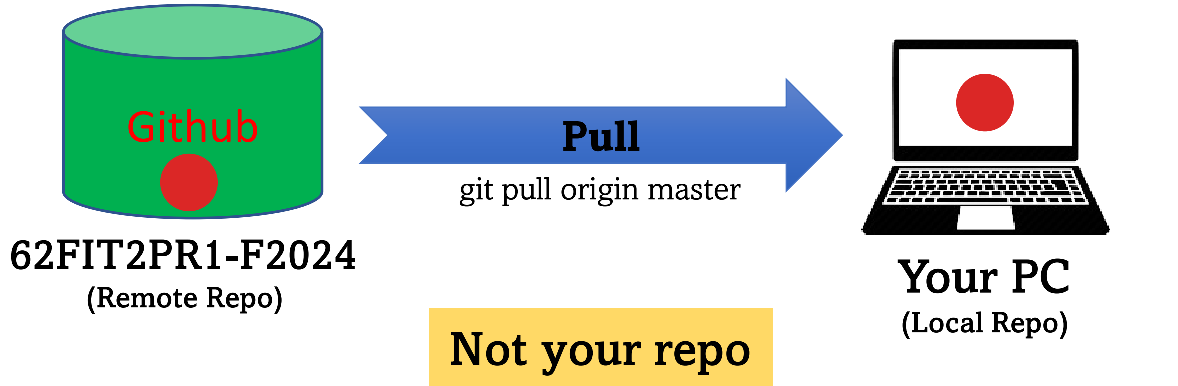
❖ Now, this is how you will use **git** in this course…



The first time you don't have this repo locally on your machine, run command: "**git clone https://github.com/QuyenNV-Hanu/62FIT2PR1-F2024.git**"

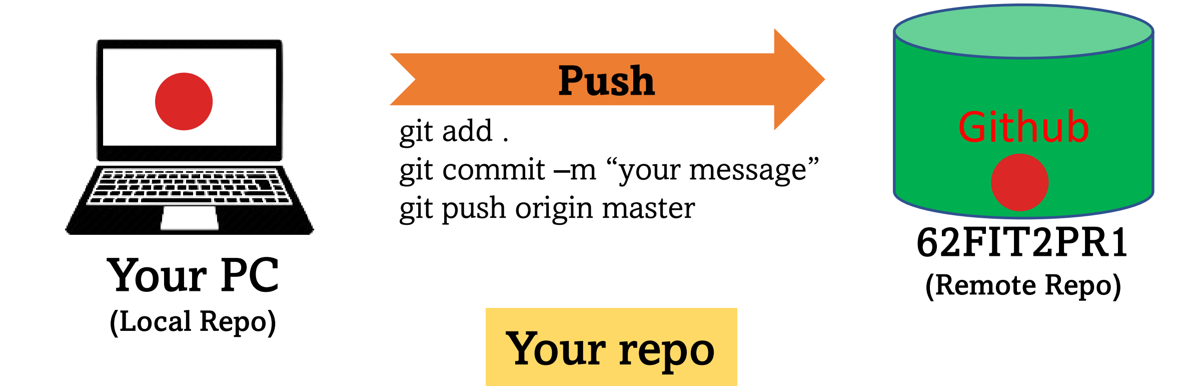


The first time you don't have this repo locally on your machine, run command: "**git clone github.com/username/your_repo.git**"



```
git pull origin master
```

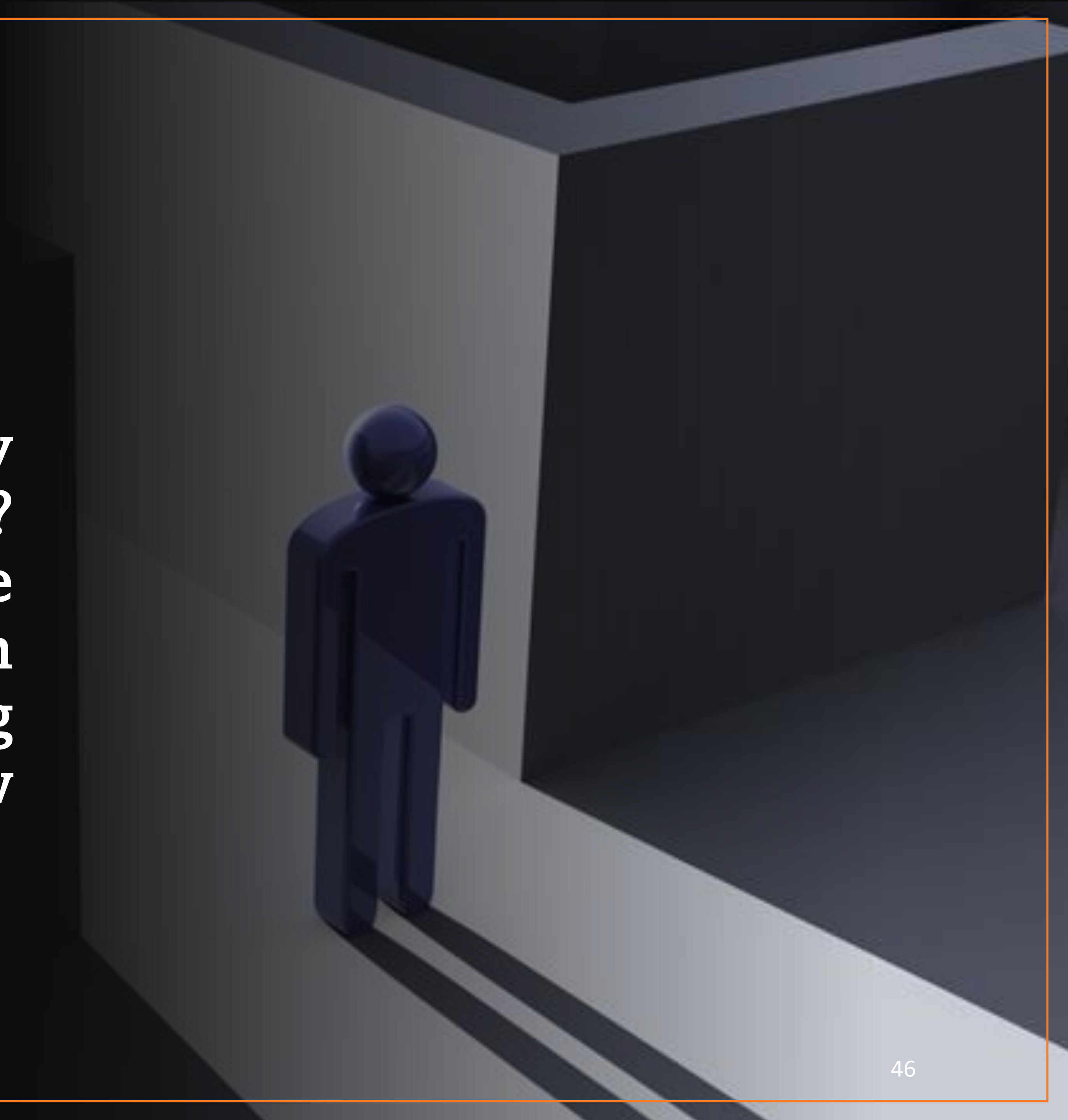
When you have already had the course repo locally on your machine, you want to get my demo code, run command: "**git pull origin master**"



```
git add .
git commit –m "your message"
git push origin master
```

When you have already had the course repo locally on your machine, you want to update your changes on the remote repo, just push it.

Feeling overloaded by too much information? No worries - I’ll guide you step by step through the first tutorial, making everything easy to follow and remember.

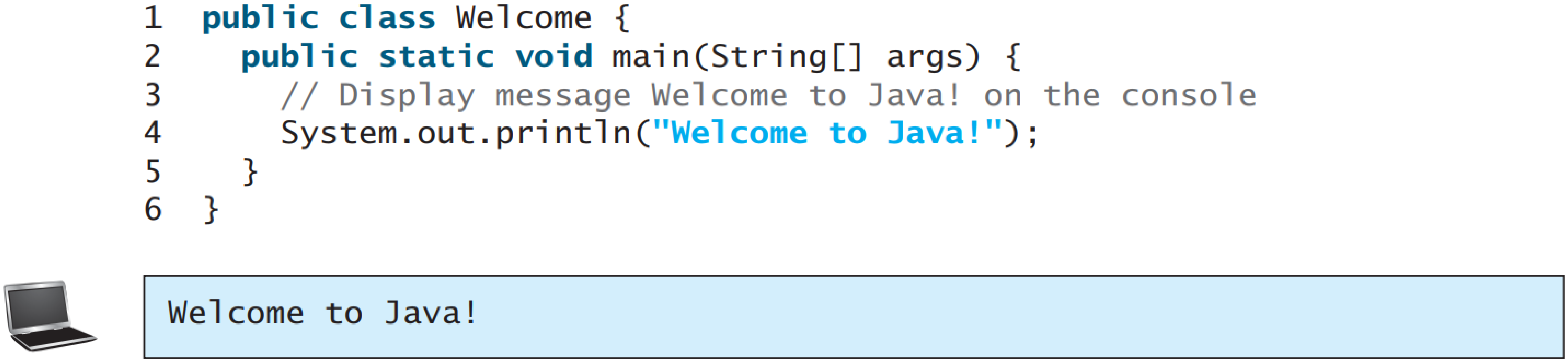


We've covered the essential theories that will guide our programming journey - algorithms for problem-solving and Git for version control. But understanding theory is just the beginning. Now, we'll bring these concepts into practice by writing our first program. This is where everything we've learned so far starts to make sense in the real world. Let's see how we can turn these ideas into a working Java program.

- Introduction to Programming
- Algorithm
- Version Control System
- **Launch the First Program**
- Program Structure
- Environment Setup
- Final Touches

Launch the First Program

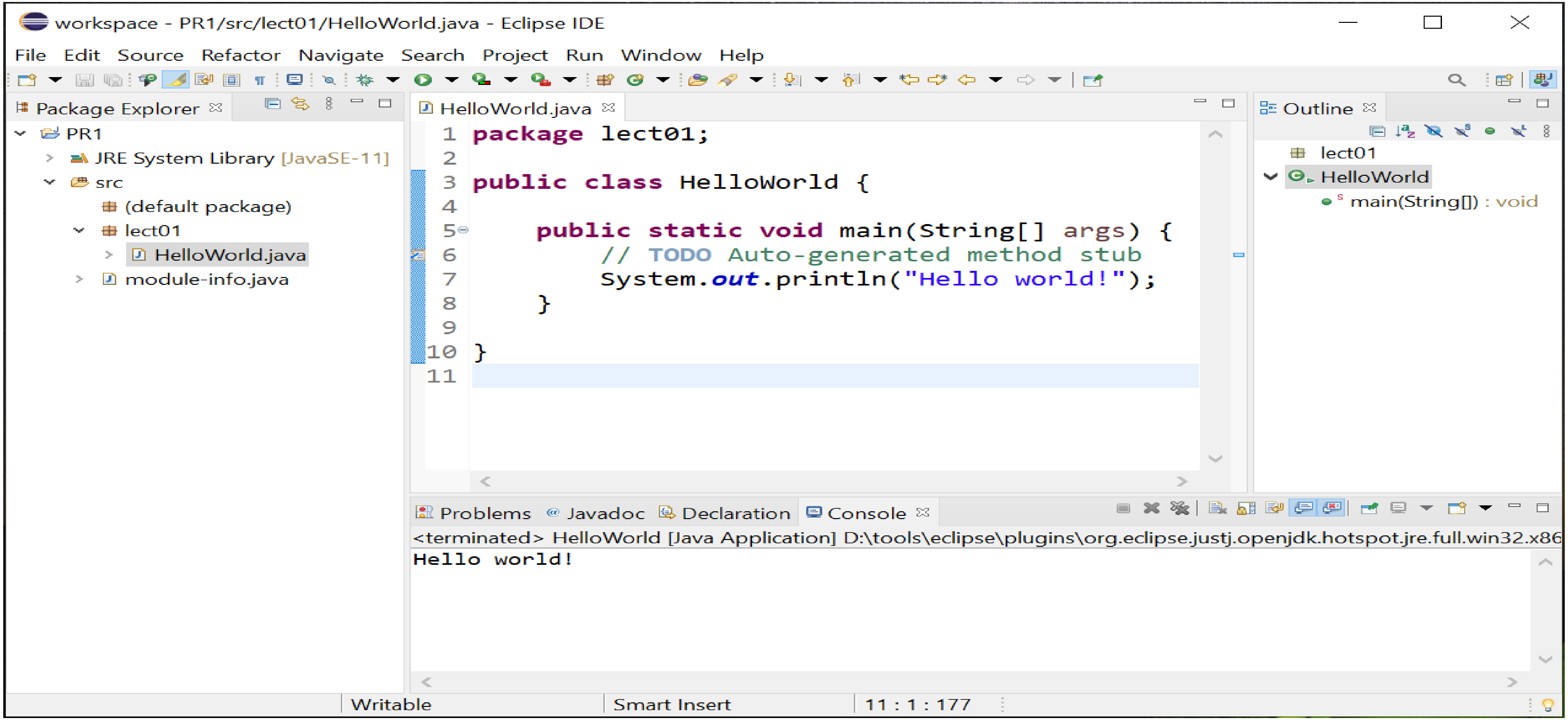
- Let’s begin with a simple Java program that displays the message **Welcome to Java!** on the console. (The word *console* is an old computer term that refers to the text entry and display device of a computer. *Console input* means to receive input from the keyboard, and *console output* means to display output on the monitor.)



```java
public class Welcome {
  public static void main(String[] args) {
    // Display message Welcome to Java! on the console
    System.out.println("Welcome to Java!");
  }
}
```

```
Welcome to Java!
```

Live instructions: Eclipse



```java
package lect01;

public class HelloWorld {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hello world!");
    }

}

```

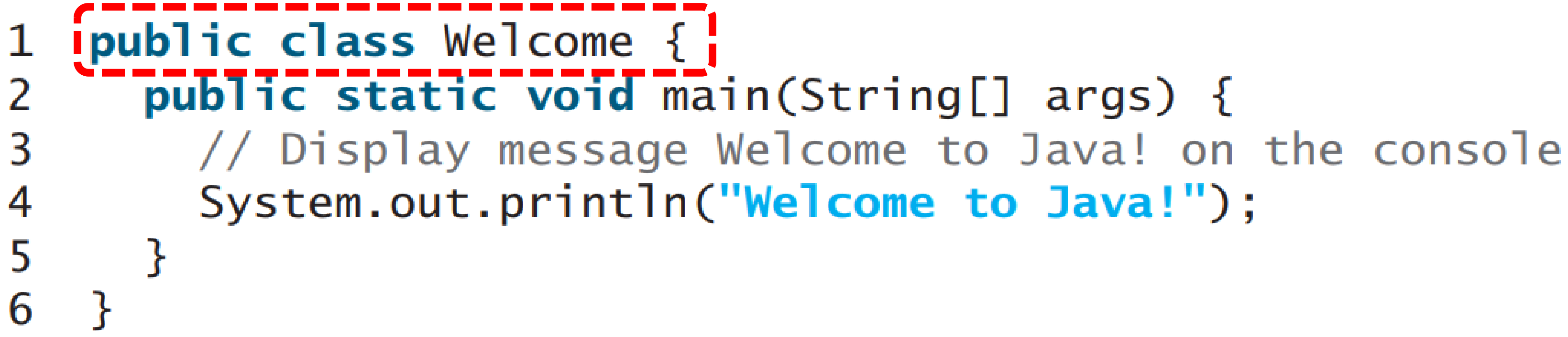
```
<terminated> HelloWorld [Java Application] D:\tools\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86
Hello world!
```

After creating and executing the program, have you ever wondered what goes on behind the scenes?

- Introduction to Programming
- Algorithm
- Version Control System
- Launch the First Program
- **Program Structure**
- Environment Setup
- Final Touches

Program Structure

❖ Back to the demo program, let’s discuss this!

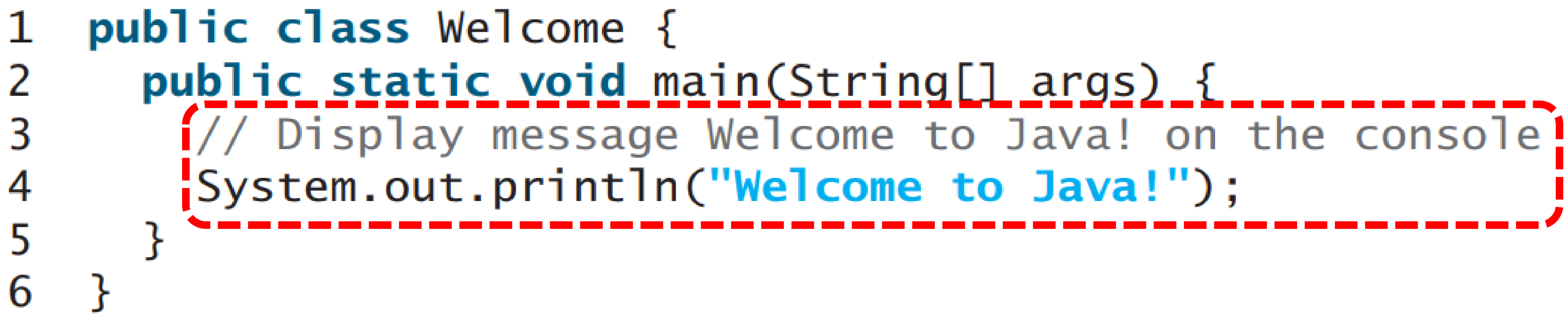


```
1  public class Welcome {
2    public static void main(String[] args) {
3      // Display message Welcome to Java! on the console
4      System.out.println("Welcome to Java!");
5    }
6  }
```

❑ **Line 1** defines a class. Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter. In this example, the class name is **Welcome**.

Program Structure

❖ Back to the demo program, let’s discuss this!



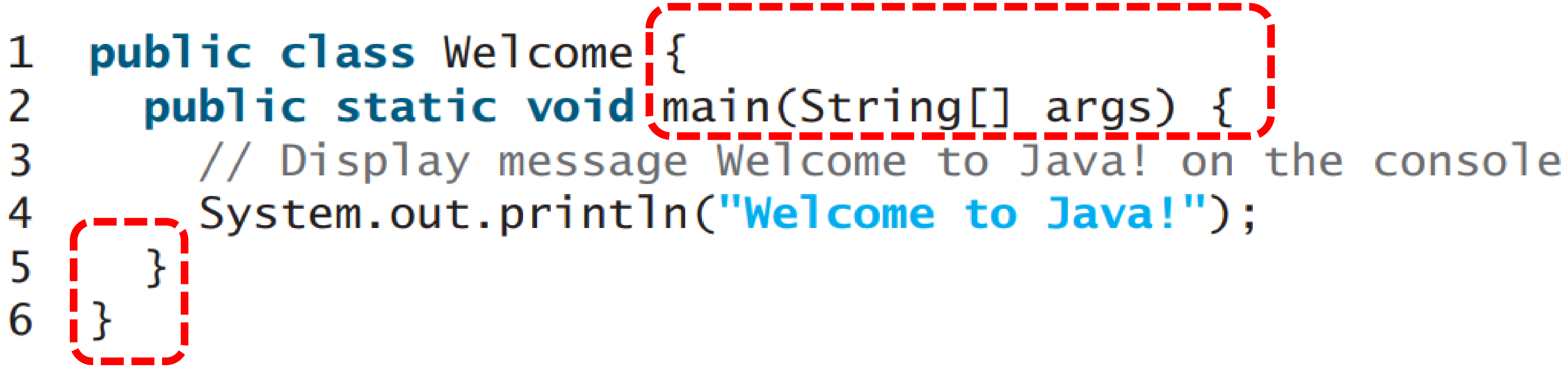
```
public class Welcome {
  public static void main(String[] args) {
    // Display message Welcome to Java! on the console
    System.out.println("Welcome to Java!");
  }
}
```

❑ The program is executed from the *main* method. A class may contain several methods. The main method is the entry point where the program begins execution.

❑ The main method in this program contains the ***System.out.println*** statement. This statement **displays** the string **Welcome to Java!** on the console. String is a programming term meaning a sequence of characters. A string must be enclosed in double quotation marks. Every statement in Java ends with a semicolon (**;**), known as the statement terminator.

Program Structure

❖ Back to the demo program, let’s discuss this!

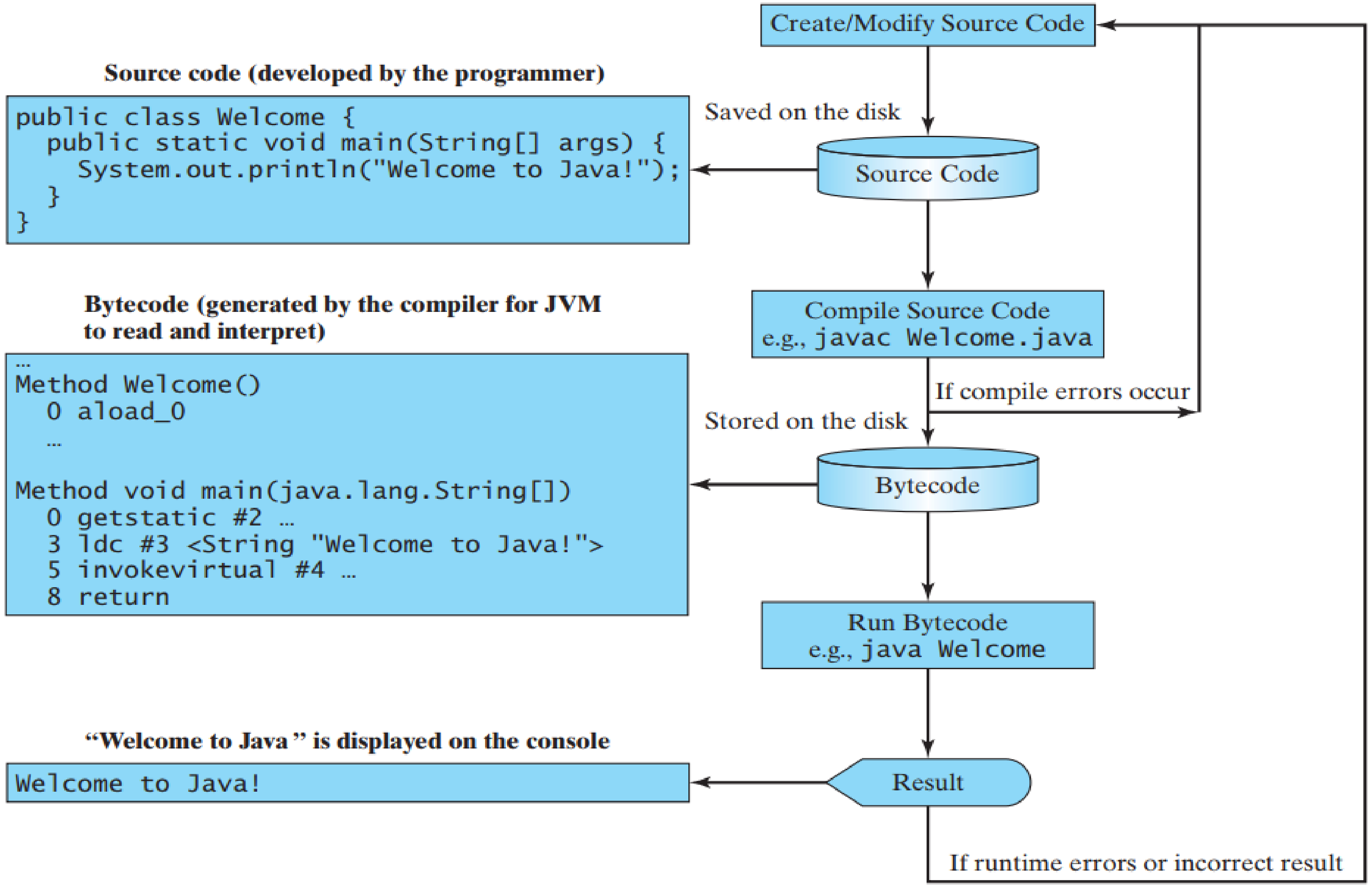


```
public class Welcome {
  public static void main(String[] args) {
    // Display message Welcome to Java! on the console
    System.out.println("Welcome to Java!");
  }
}
```

❑ Also, A pair of curly braces in a program forms a block that groups the program’s components. In Java, each block begins with an opening brace ({) and ends with a closing brace (}). Every class has a class block that groups the data and methods of the class. Similarly, every method has a method block that groups the statements in the method. Blocks can be nested, meaning that one block can be placed within another.

Program Structure

❖ Let’s dive deeper on how to create, compile, and execute a program.



Source code (developed by the programmer)

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

Bytecode (generated by the compiler for JVM to read and interpret)

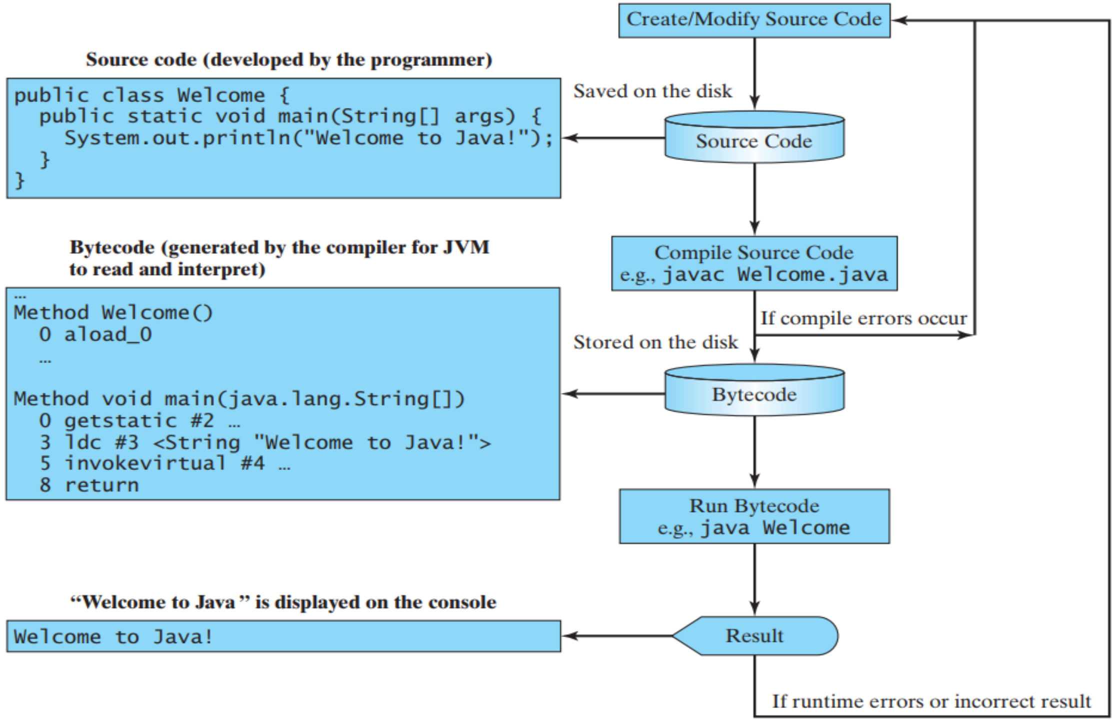
```
…
Method Welcome()
  0 aload_0
  …

Method void main(java.lang.String[])
  0 getstatic #2 …
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 …
  8 return
```

“Welcome to Java ” is displayed on the console

```
Welcome to Java!
```

Program Structure

- As can be seen that you have to create your program and compile it before it can be executed. This process is repetitive.
- If your program has compile errors, you have to modify the program to fix them, and then recompile it.
- If your program has runtime errors or does not produce the correct result, you have to modify the program, recompile it, and execute it again.



Source code (developed by the programmer)

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

Bytecode (generated by the compiler for JVM to read and interpret)

```
…
Method Welcome()
  0 aload_0
  …

Method void main(java.lang.String[])
  0 getstatic #2 …
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 …
  8 return
```

“Welcome to Java” is displayed on the console

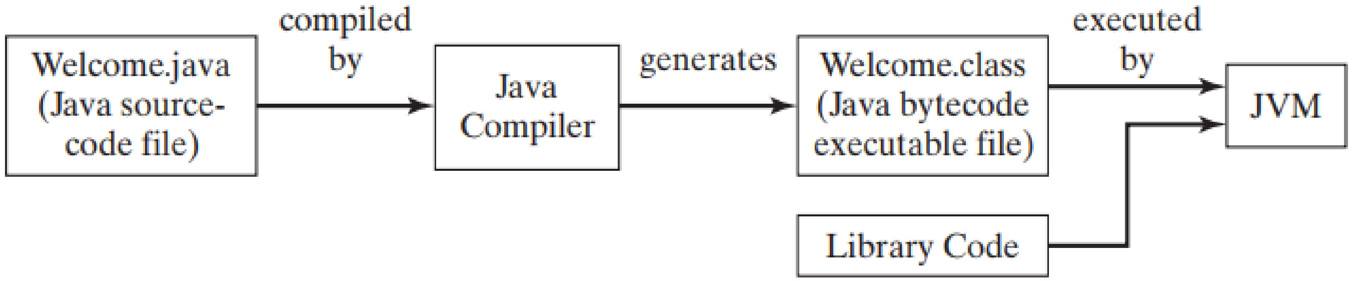
```
Welcome to Java!
```

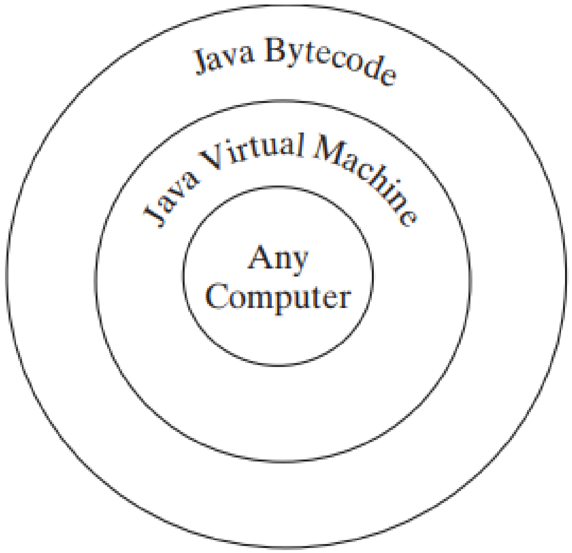
Note

The source file must end with the extension .java and must have the same exact name as the public class name. For example, the file for the source code in this example should be named **Welcome.java**, since the public class name is **Welcome**.

Program Structure

- You can write and edit Java code using various text editors or **Integrated Development Environments (IDEs)**. Popular IDEs like ***NetBeans*** and ***Eclipse*** offer comprehensive tools for coding, compiling, and running Java programs. Alternatively, you can use text editors like ***Notepad++*** or ***Sublime Text*** and compile your Java code via the command line or an IDE.
- A Java compiler translates a Java source file into a Java bytecode file. If there aren't any syntax errors, the compiler generates a bytecode file with a *.class* extension. Thus, the preceding command generates a file named **Welcome.class**.

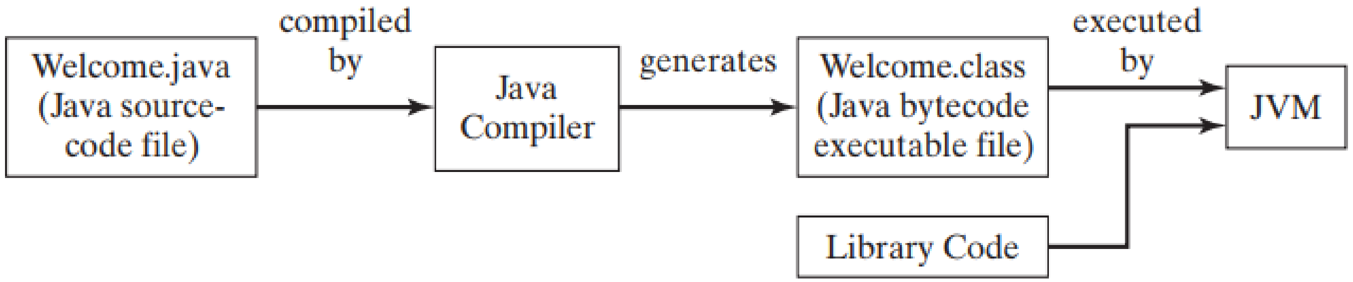


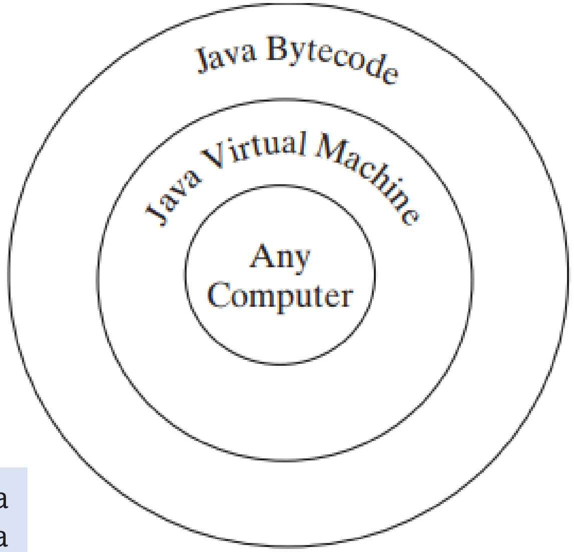


Note: You must first install and configure the JDK before you can compile and run programs.

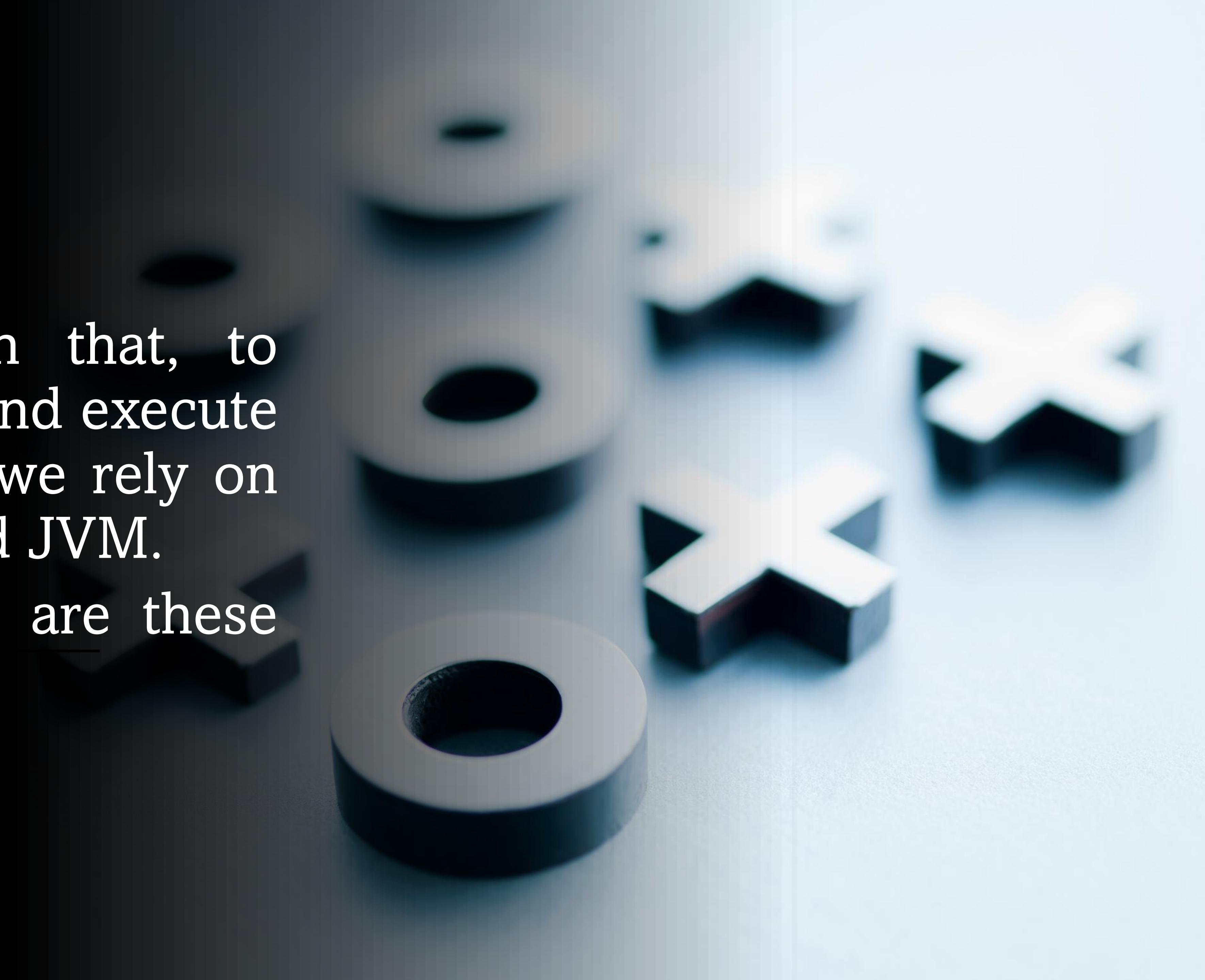
Program Structure

- You can write and edit Java code using various text editors or **Integrated Development Environments (IDEs)**. Popular IDEs like ***NetBeans*** and ***Eclipse*** offer comprehensive tools for coding, compiling, and running Java programs. Alternatively, you can use text editors like ***Notepad++*** or ***Sublime Text*** and compile your Java code via the command line or an IDE.
- A Java compiler translates a Java source file into a Java bytecode file. If there aren't any syntax errors, the compiler generates a bytecode file with a *.class* extension. Thus, the preceding command generates a file named **Welcome.class**.


Welcome.java (Java source-code file)
compiled by
Java Compiler
generates
Welcome.class (Java bytecode executable file)
executed by
JVM
Library Code



Note: You must first install and configure the JDK before you can compile and run programs. Java source code is compiled into Java bytecode and Java bytecode is interpreted by the JVM. Your Java code may use the code in the Java library (JRE). The JVM executes your code along with the code in the library.

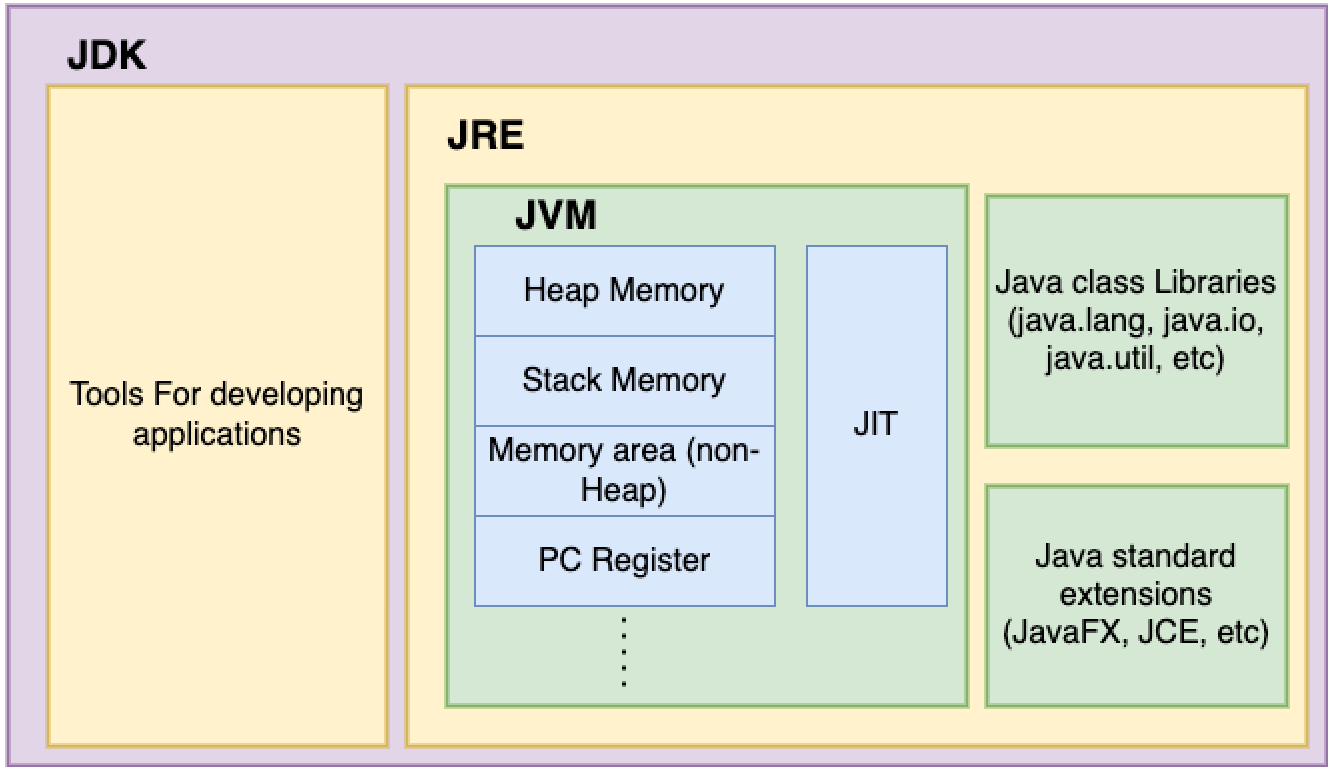


As can be seen that, to create, compile, and execute a Java program, we rely on the JDK, JRE, and JVM.

But what exactly are these components?

Introduction to Environment

- **JVM** (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.



JDK

Tools For developing applications

JRE

JVM

Heap Memory

Stack Memory

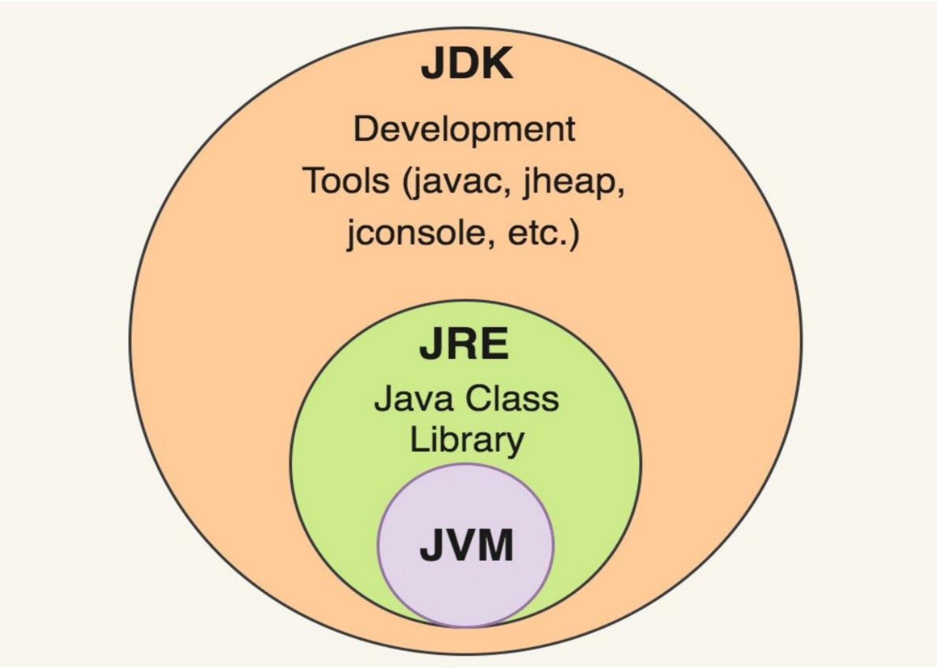
Memory area (non-Heap)

PC Register

JIT

Java class Libraries (java.lang, java.io, java.util, etc)

Java standard extensions (JavaFX, JCE, etc)



JDK

Development Tools (javac, jheap, jconsole, etc.)

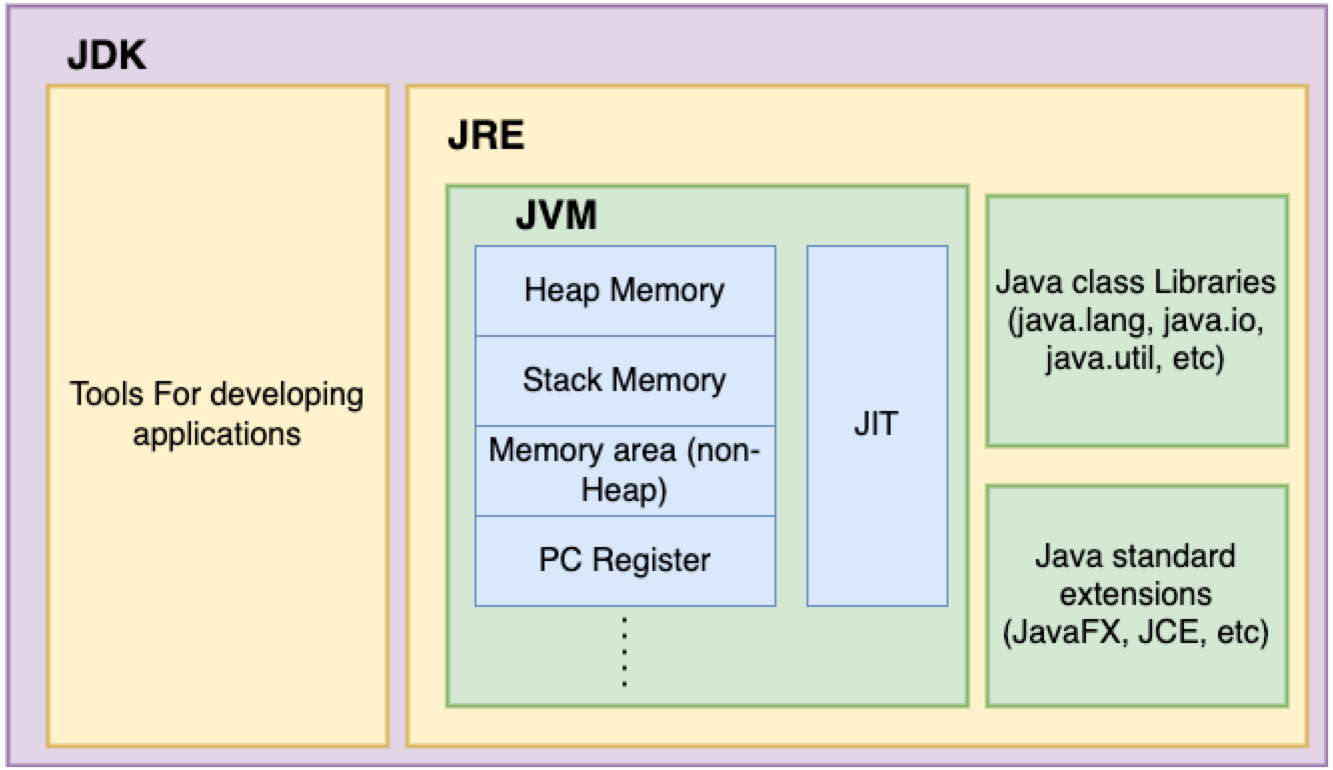
JRE

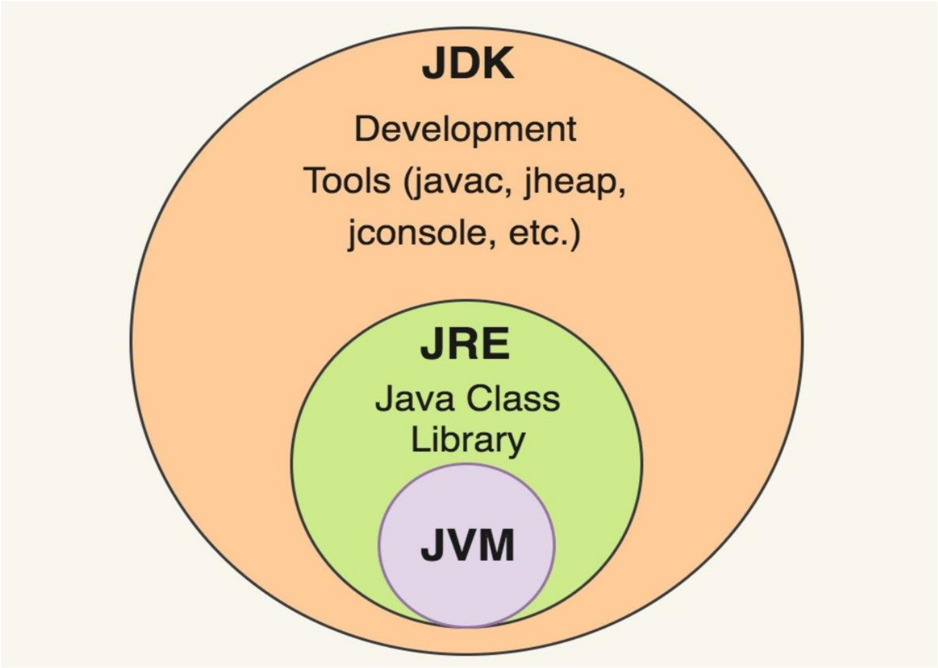
Java Class Library

JVM

Introduction to Environment

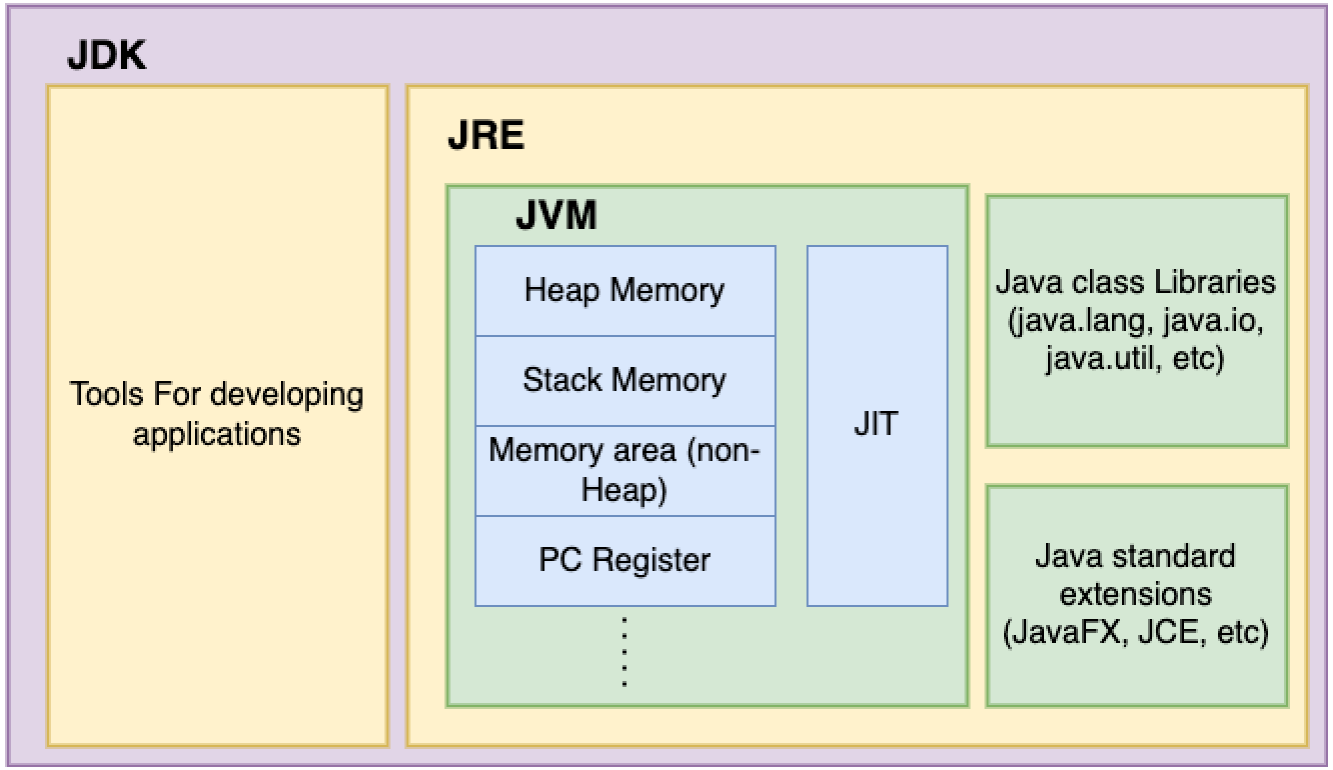
- **JDK** is an acronym for **Java Development Kit**. The **Java Development Kit (JDK)** is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

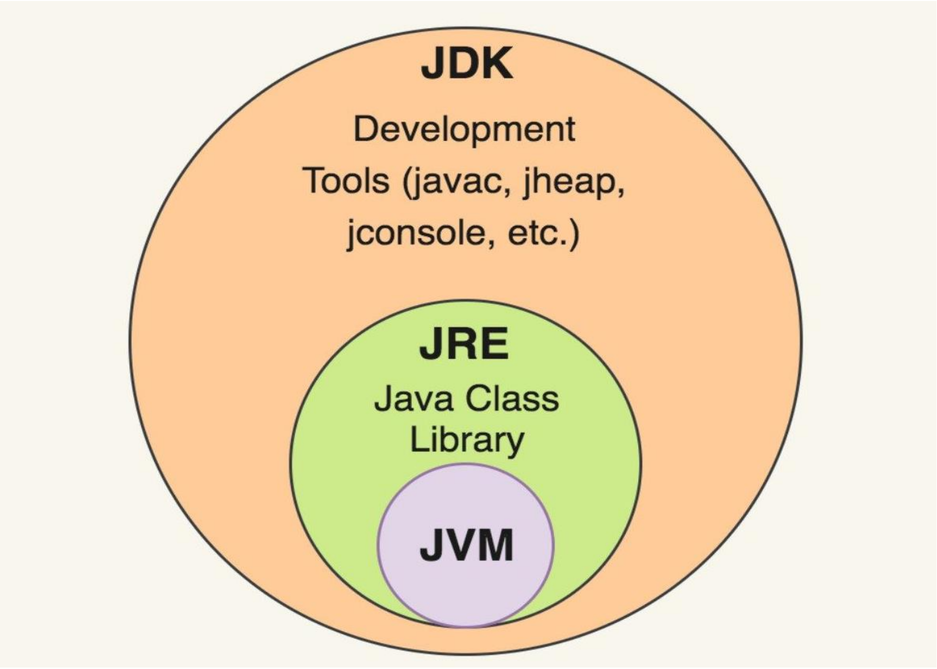

JDK
Tools For developing applications
JRE
JVM
Heap Memory
Stack Memory
Memory area (non-Heap)
PC Register
JIT
Java class Libraries (java.lang, java.io, java.util, etc)
Java standard extensions (JavaFX, JCE, etc)



Introduction to Environment

- **JRE** is an acronym for **Java Runtime Environment**. It is also written as **Java RTE**. The **Java Runtime Environment** is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of **JVM**. It physically exists. It contains a set of libraries + other files that **JVM** uses at runtime.


JDK
Tools For developing applications
JRE
JVM
Heap Memory
Stack Memory
Memory area (non-Heap)
PC Register
JIT
Java class Libraries (java.lang, java.io, java.util, etc)
Java standard extensions (JavaFX, JCE, etc)

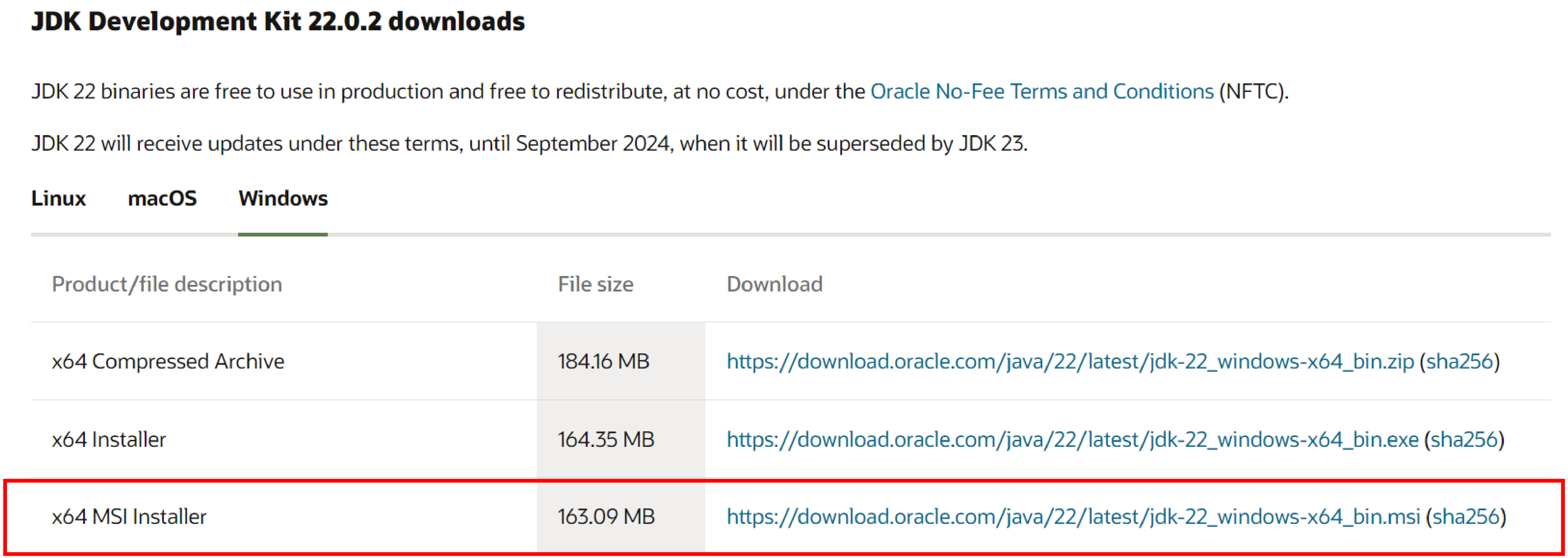


Okie! That’s enough! Let’s install necessary tools and libraries…

- Introduction to Programming
- Algorithm
- Version Control System
- Launch the First Program
- Program Structure
- **Environment Setup**
- Final Touches

Setting up JDK

❖ Install JDK: https://www.oracle.com/java/technologies/downloads/#jdk22-windows



JDK Development Kit 22.0.2 downloads

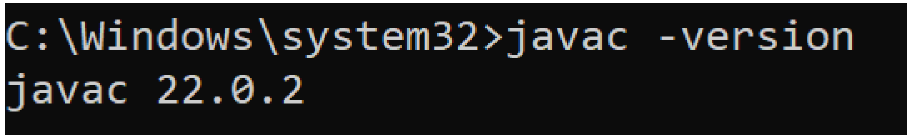
JDK 22 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions (NFTC).

JDK 22 will receive updates under these terms, until September 2024, when it will be superseded by JDK 23.

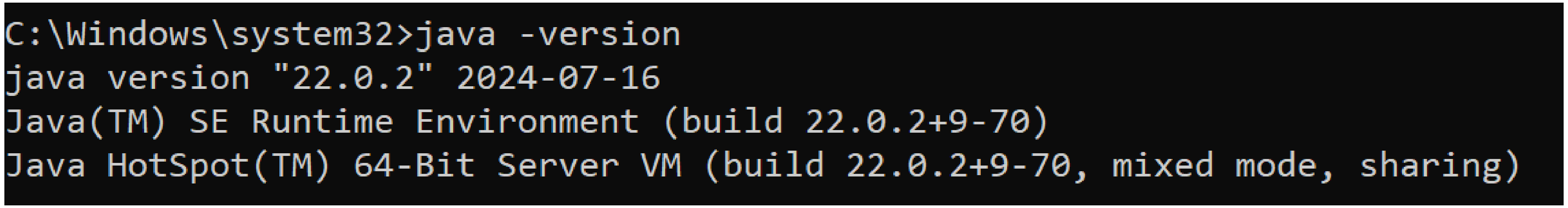
Product/file description	File size	Download
x64 Compressed Archive	184.16 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.zip (sha256)
x64 Installer	164.35 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.exe (sha256)
x64 MSI Installer	163.09 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.msi (sha256)

Setting up JDK

- ❖ After installing the JDK, make sure that the JDK works well by running these following commands:



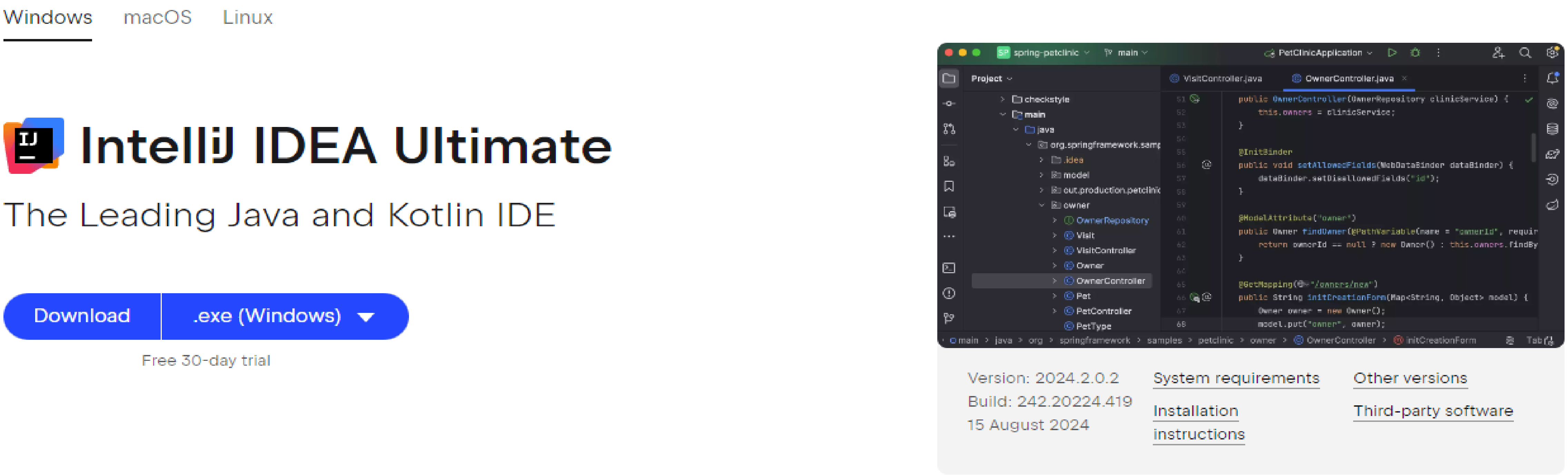
```
C:\Windows\system32>javac -version
javac 22.0.2
```



```
C:\Windows\system32>java -version
java version "22.0.2" 2024-07-16
Java(TM) SE Runtime Environment (build 22.0.2+9-70)
Java HotSpot(TM) 64-Bit Server VM (build 22.0.2+9-70, mixed mode, sharing)
```

Setting up IntelliJ

- Install Eclipse: https://www.jetbrains.com/idea/download/?section=windows



Windows macOS Linux

IntelliJ IDEA Ultimate

The Leading Java and Kotlin IDE

Download .exe (Windows)

Free 30-day trial

```
public OwnerController(OwnerRepository clinicService) {
    this.owners = clinicService;
}

@InitBinder
public void setAllowedFields(WebDataBinder dataBinder) {
    dataBinder.setDisallowedFields("id");
}

@ModelAttribute("owner")
public Owner findOwner(@PathVariable(name = "ownerId", requir
    return ownerId == null ? new Owner() : this.owners.findBy
}

@GetMapping(@~"/owners/new")
public String initCreationForm(Map<String, Object> model) {
    Owner owner = new Owner();
    model.put("owner", owner);
```

Version: 2024.2.0.2
Build: 242.20224.419
15 August 2024

System requirements

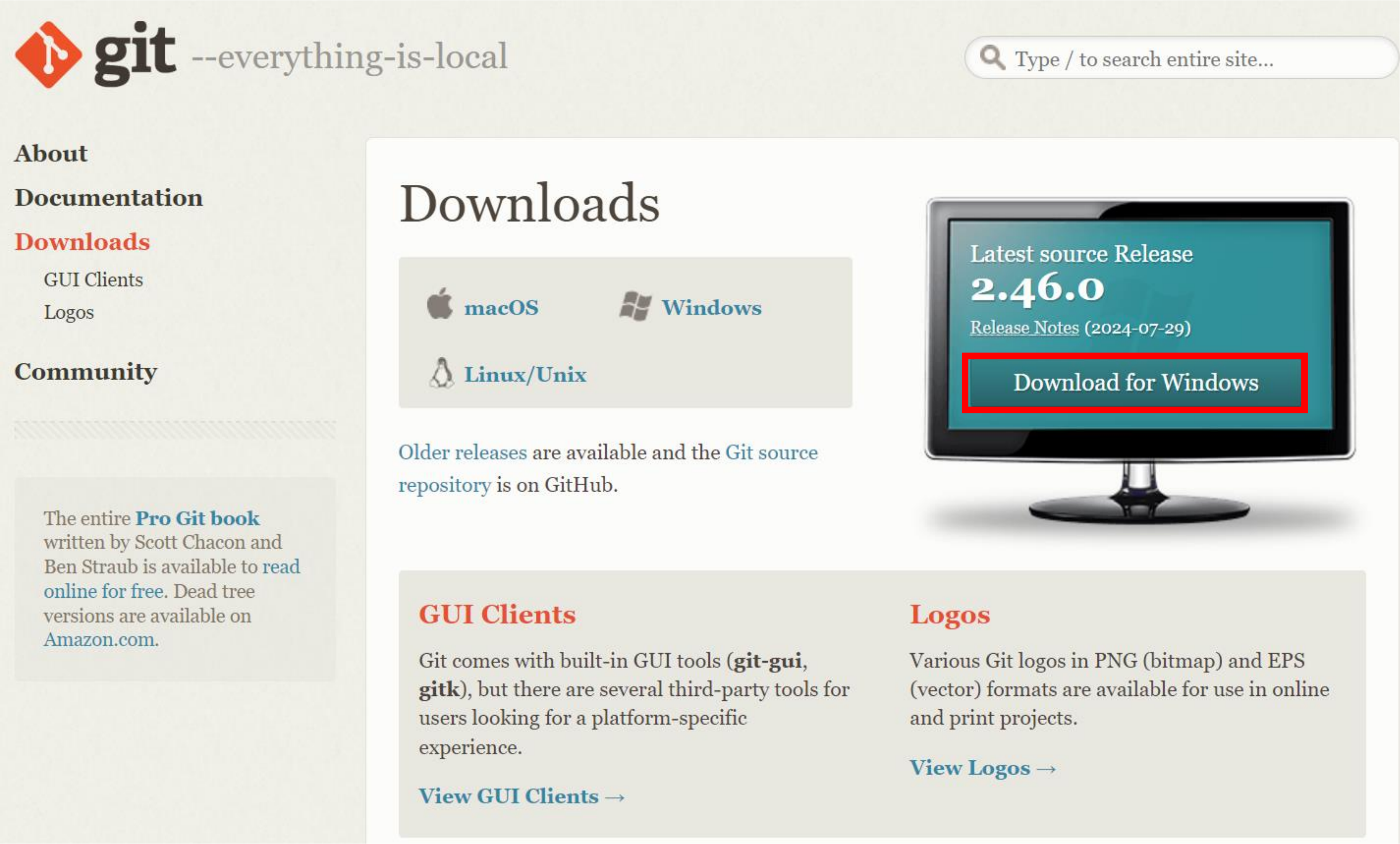
Other versions

Installation instructions

Third-party software

Setting up Git GUI

- Install Git Gui: https://git-scm.com/downloads
- If you are using Window, just download then install it like any other programs.



git --everything-is-local

Type / to search entire site...

About

Documentation

Downloads

GUI Clients

Logos

Community

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Downloads

macOS

Windows

Linux/Unix

Older releases are available and the Git source repository is on GitHub.

Latest source Release

2.46.0

Release Notes (2024-07-29)

Download for Windows

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

View GUI Clients →

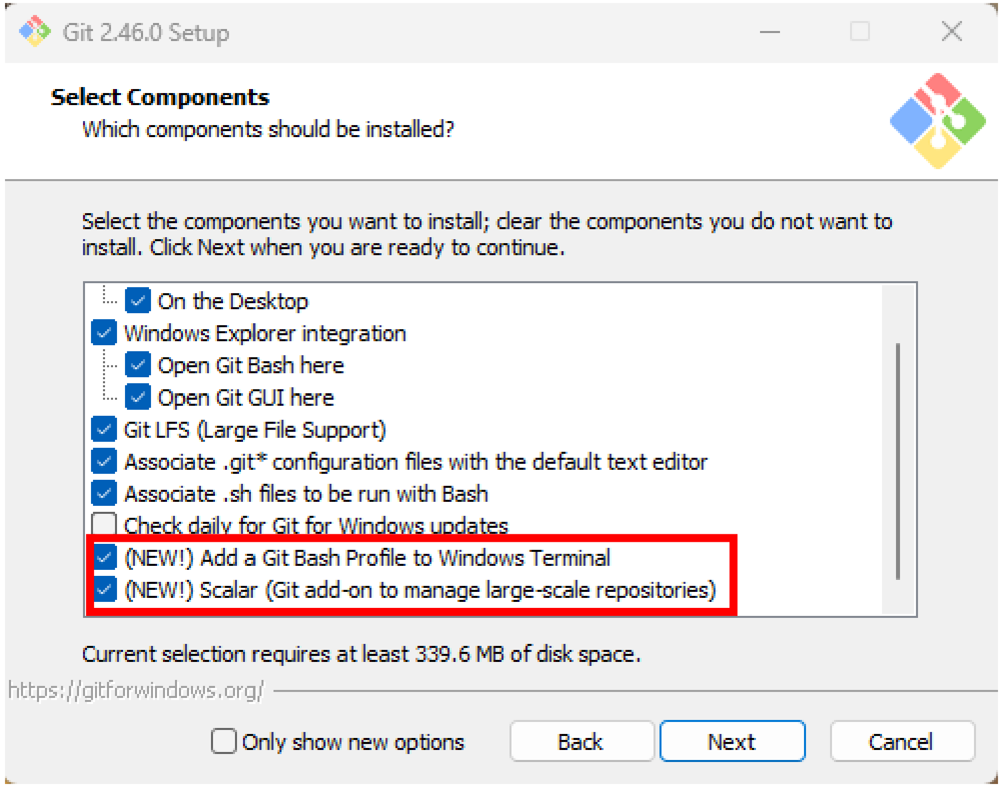
Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

View Logos →

Setting up Git GUI

- Note to check these new features…



Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- ☑ On the Desktop
- ☑ Windows Explorer integration
  - ☑ Open Git Bash here
  - ☑ Open Git GUI here
- ☑ Git LFS (Large File Support)
- ☑ Associate .git* configuration files with the default text editor
- ☑ Associate .sh files to be run with Bash
- □ Check daily for Git for Windows updates
- ☑ (NEW!) Add a Git Bash Profile to Windows Terminal
- ☑ (NEW!) Scalar (Git add-on to manage large-scale repositories)

Current selection requires at least 339.6 MB of disk space.

Thanks!

Any questions?

For an in-depth understanding of Java, I highly recommend referring to the textbooks. This slide provides a brief overview and may not cover all the details you're eager to explore!