Introduction to Algorithms

Department of Computer Science
College of Computing and Information Technologies (CCIT)
National University

Learning Objectives

At the end of the lesson, the student is expected to:

- **Remember** the vital concepts in the previous programming subjects;
- **Understand** how algorithm and data structure correlates with each other;
- **Highlight** key learnings on algorithmic operations, data structure, abstraction, parameters and arrays.
- **Apply** the concepts through test problems and programs.

Recap of your programming subjects

- Fundamentals of Programming
- Intermediate Programming
- Object Oriented Programming

What do they all have in common?

Q: How do you solve problems?

1. Do you just go at it?
2. Do you think about it first?
3. Do you follow a series of instructions?

What do we mean by **solving a problem**?

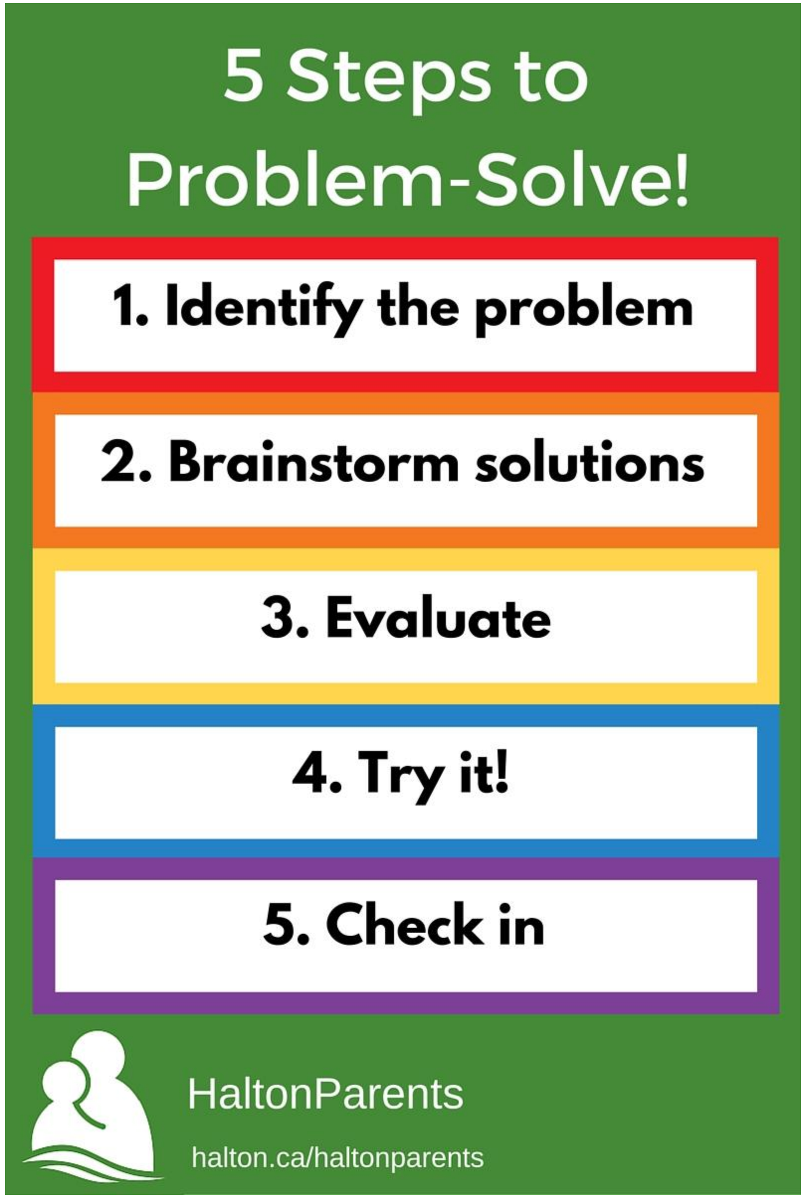
1. We can obtain a **correct answer** for any input.
    - Remember your Prog 1 memories. Test cases for each problem.
2. It can be obtained with **reasonable costs (time)**.
    - Fast. Does not take an infinite amount of time to output an answer. Should not result to an infinite loop.

What if a problem is **unsolvable**?

1. It takes a **long time** for some inputs.
2. It takes **so much memory** for some inputs.
3. We cannot make any dynamic, flexible algorithm for the problem.

Thus, to solve a problem, you need to consider two important factors.

1. The nature of the **problem**.
2. The nature of your **solution** / **algorithm**.



5 Steps to Problem-Solve!

1. Identify the problem
2. Brainstorm solutions
3. Evaluate
4. Try it!
5. Check in

HaltonParents

halton.ca/haltonparents

This Photo by Unknown Author is licensed under CC BY-SA-NC

Before anything, *know* first what to program!

Algorithms

- In mathematics and computer science, an algorithm is an unambiguous specification or **set of instructions** to solve a problem.
- Algorithms can perform **calculation**, **data processing** and **automation**.

Algorithms

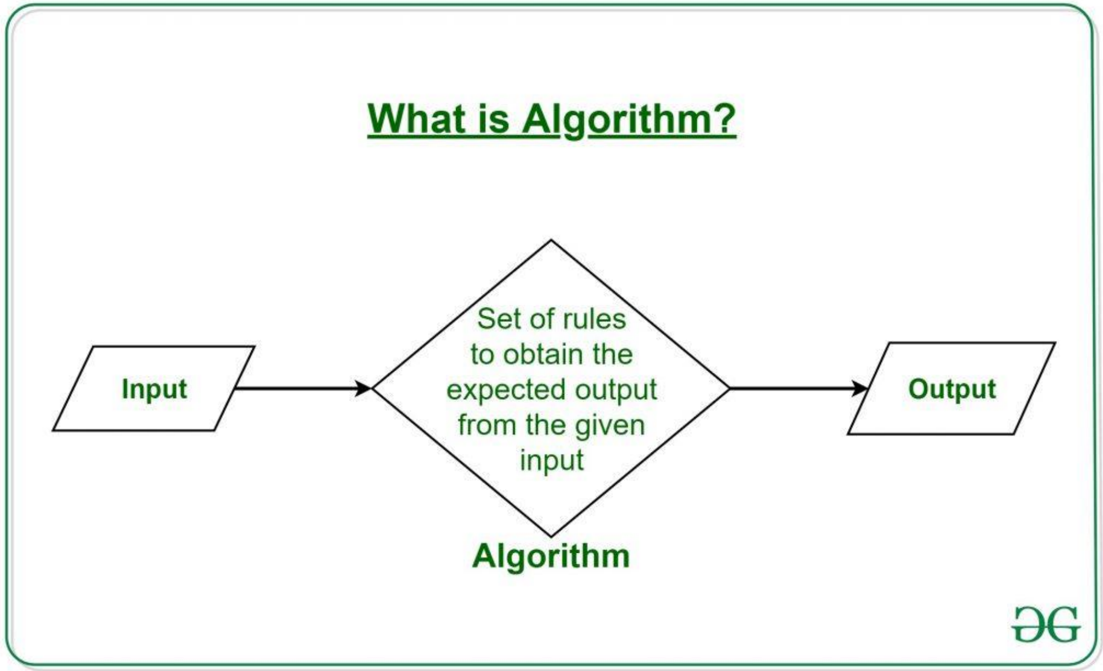

What is Algorithm?
Input
Set of rules to obtain the expected output from the given input
Output
Algorithm

Figure 1. Anatomy of an algorithm

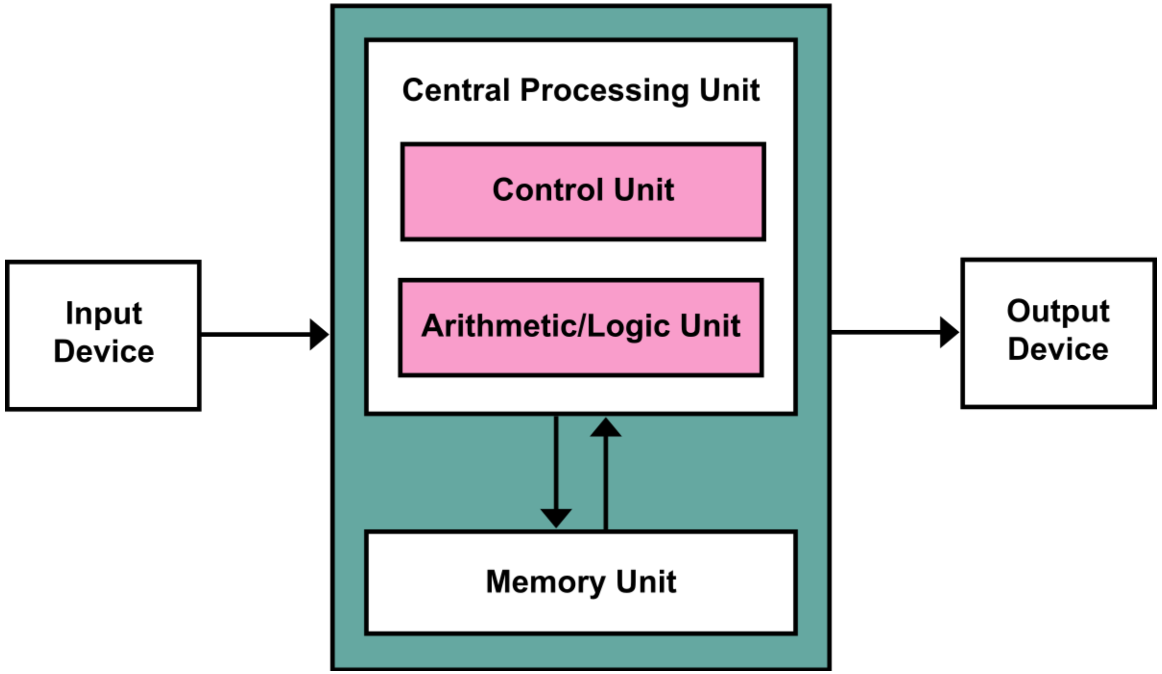


Figure 2. Von Neumann Architecture

Representations

Algorithms are independent from any programming language and can come in many forms such as:

- Recipes
- Flowcharts
- Manuals / instruction sets
- A series of verbal order

As long as each step is done systematically, it can be used to solve problems. In our case as programmers, our algorithms are in the form of **codes** written in various programming languages.

Problem-Solving Phase - Review

The student must read the problem statement several times to ensure that he/she understands what is asked before attempt to solve the problem.

The following steps need to be followed:

- Read the problem carefully.
- Understand what the problem entails.
- Only then, write down the steps to solve the problem.

Problem-Solving Phase - Review

Algorithm

The word is derived from the phonetic pronunciation of the last name of Abu Ja'far Mohammed ibn Musa al-Khwarizmi, who was an Arabic mathematician who invented a set of rules for performing the four basic arithmetic operations (**addition, subtraction, multiplication and division)** on decimal numbers.

Problem-Solving Phase - Review

Pseudocode

Pseudocode is a logic development tool that uses English statements or clauses to present the logical steps necessary to solve a problem. **“Pseudo”** technically means **“false”,** so pseudocode, taken literally, means **“false code”.** This is used in program development, pseudocode is made up statements written to depict steps, in the correct sequence, required to solve a specific problem.

References

Minhas, S. (2021). Pseudo Code. Retrieved from PDFCOFEE: https://pdfcoffee.com/pseudo-code-4-pdf-free.html

Problem-Solving Phase - Review

Pseudocode

It is composed of words, clauses and sentences. The following rules should be followed:

- **No actual programming code** should appear n the pseudocode. You just need to write down the logical steps at this stage, not coding the program
- All statements should be presented in **enough detail** so the reader can clearly understand the activity or action being described in each statement.

References

Minhas, S. (2021). Pseudo Code. Retrieved from PDFCOFEE: https://pdfcoffee.com/pseudo-code-4-pdf-free.html

Problem-Solving Phase - Review

Pseudocode

- The names of **"variables"** being used to solve the problem should be **totally descriptive** of what they represent, not shortened versions you use in your source code.
- Each activity or action being depicted should be **presented in a single line.** Appropriate **indentation** should be used if the activity or action being described cannot fit on one line.
- **Indentation** should be used where appropriate to show the logical grouping of related activities or actions.

References
Minhas, S. (2021). Pseudo Code. Retrieved from PDFCOFEE: https://pdfcoffee.com/pseudo-code-4-pdf-free.html

Problem-Solving Phase - Review

Pseudocode

- Most of your pseudocode should not be done with capital letters, but certain key structures should be capitalized. They are:

IF, THEN, ELSE and **ENDIF** in decision structures.

DO WHILE and **ENDDO** in pre-test loop structures.

DO UNTIL and **ENDDO** in post-test loop structures.

CASE and **ENDCASE** is a case structures.

References

Minhas, S. (2021). Pseudo Code. Retrieved from PDFCOFEE: https://pdfcoffee.com/pseudo-code-4-pdf-free.html

Problem-Solving Phase - Review

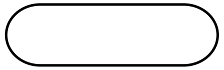
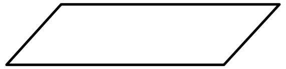
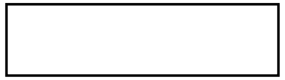
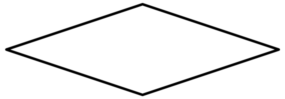
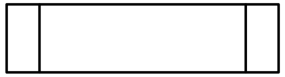
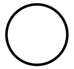
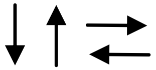
Flowchart

A flowchart uses special shapes to represent different types of actions or steps in process. These are known as flowchart symbols. This is a type of diagram **(graphical or symbolic)** that represents an algorithm or a process. It is typically showing the flow of data in a process, detailing the operations/steps in a pictorial format which is easier to understand than reading it in a textual format.

References
smartdraw. (n.d.). Flowchart Symbols. Retrieved from smartdraw: https://www.smartdraw.com/flowchart/flowchart-symbols.htm

Problem-Solving Phase - Review

Common Flowchart Symbols

Symbol	Name	Functions
	Terminal	Indicates the starting or ending of a program, process, function, or interrupt program.
	Input/Output	Used for any input or output operation. Indicates that the computer is to obtain data or output results.
	Process	Indicates any type of internal operation inside the Processor or Memory.
	Decision	Used to ask a question that can be answered in a binary format (yes/no, true/false).
	Predefined Process	Used to invoked subroutine, function, or an interrupt program.
	Connector	Allows flowchart to be drawn without intersecting lines or without a reverse flow.
	Flow Lines	Shows direction of flow

Problem-Solving Phase - Review

General Rules for flowcharting

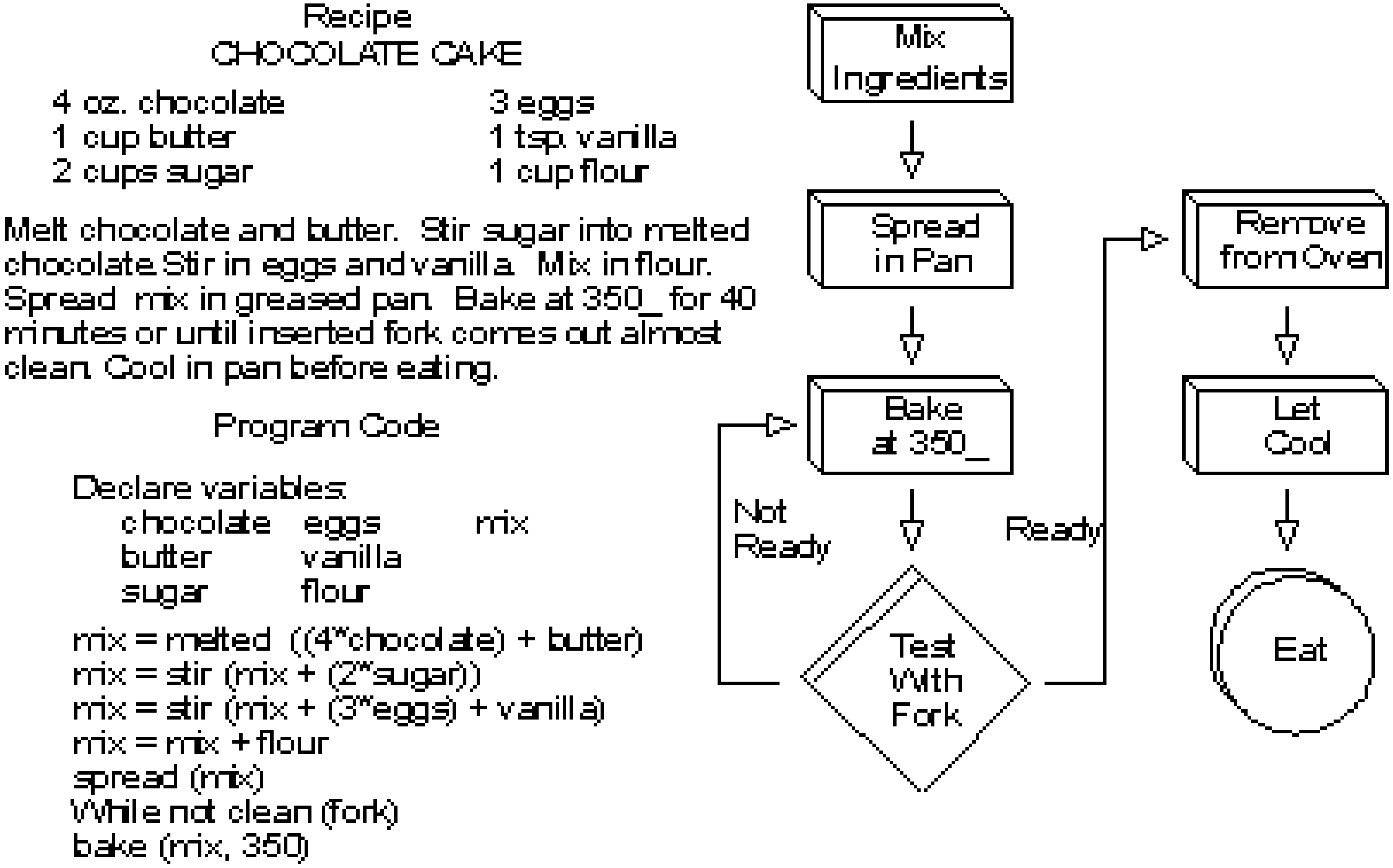
- All boxes of the flowchart are connected with Arrows. (Not lines)
- Flowchart symbols have an entry point on the top of the symbol with no other entry points. The exit point for all flowchart symbols is on the bottom except for Decision symbol.
- The Decision symbol has two exit points; these can be on the sides or the bottom and one side.
- Generally, a flowchart will flow from top to bottom.
- Connectors are used to connect breaks in the flowchart.

Problem-Solving Phase - Review

General Rules for flowcharting

Connectors are used to connect breaks in the flowchart. Examples are the following:

- From one page to another page.
- From the bottom of the page to the top of the same page.
- An upward flow or more than (3) symbols.
- Subroutines and interrupt programs have their own and independent flowcharts.
- All flow charts starts with a Terminal or Predefined Process.
- All flowcharts end with a terminal or a contentious loop.



Recipe
CHOCOLATE CAKE

4 oz. chocolate	3 eggs
1 cup butter	1 tsp. vanilla
2 cups sugar	1 cup flour

Melt chocolate and butter. Stir sugar into melted chocolate.Stir in eggs and vanilla. Mix in flour. Spread mix in greased pan. Bake at 350_ for 40 minutes or until inserted fork comes out almost clean. Cool in pan before eating.

Program Code

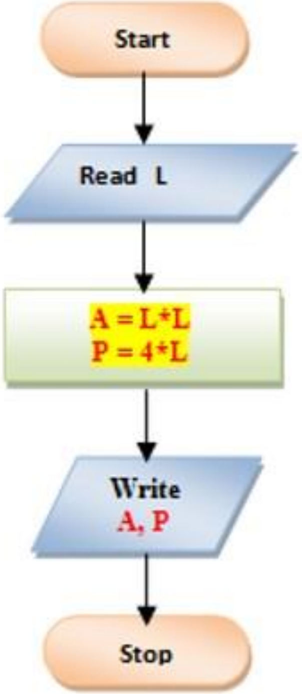
```
Declare variables:
     chocolate   eggs        mix
     butter      vanilla
     sugar       flour
mix = melted ((4*chocolate) + butter)
mix = stir (mix + (2*sugar))
mix = stir (mix + (3*eggs) + vanilla)
mix = mix + flour
spread (mix)
While not clean (fork)
bake (mix, 350)
```

Find the area & perimeter of a square

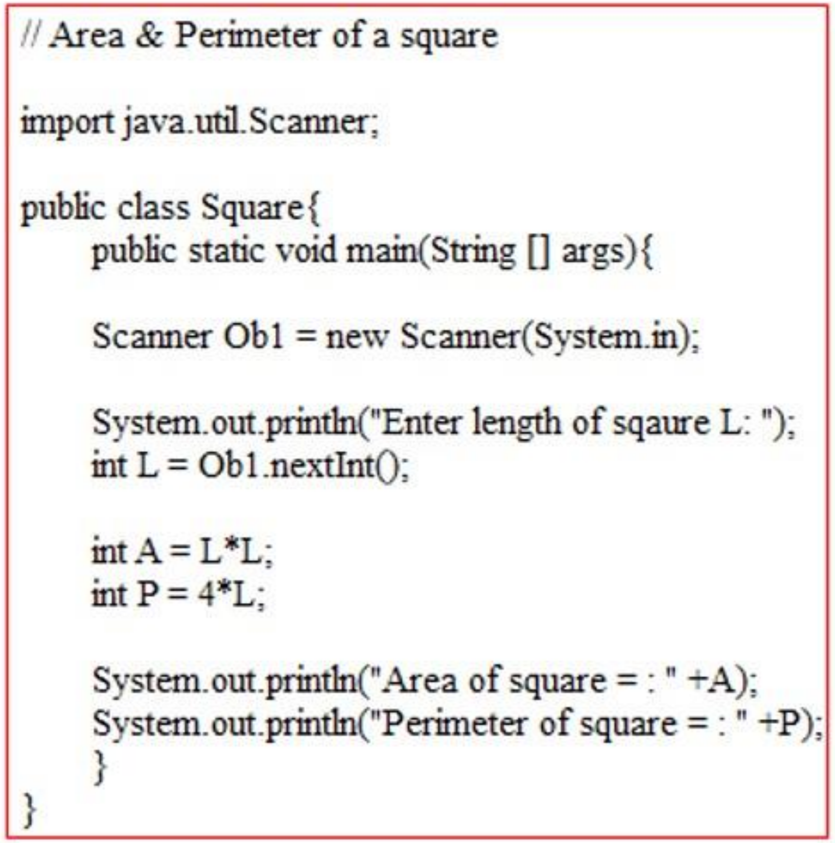
Algorithm

1. Start
2. Read length L
3. Area A = L*L
4. Perimeter P = 4*L
5. Print or display A,P
6. Stop

Flowchart



Program



```
// Area & Perimeter of a square

import java.util.Scanner;

public class Square{
        public static void main(String [] args){

        Scanner Ob1 = new Scanner(System.in);

        System.out.println("Enter length of sqaure L: ");
        int L = Ob1.nextInt();

        int A = L*L;
        int P = 4*L;

        System.out.println("Area of square = : " +A);
        System.out.println("Perimeter of square = : " +P);
        }
}
```

Nature of the **problem**

- Also called **computation complexity**.
- This is the nature of a problem being easy or difficult measured by how much **time** and **space** (memory) it takes to be solved.
- There are problems that are **inherently hard or difficult**. No existing algorithms can solve this problem dynamically.

Nature of the **algorithm**

- Also called **algorithmic complexity.**
- Concerned about how fast or slow particular algorithm performs with respect to a given dataset to process.
- Usually, algorithms are measured in terms of **best case**, **average case**, and **worst case.**

Operations of an algorithm (DS)

The most basic algorithmic operations that you need to know how are:

- Insert a new data item.
- Search for a specified item.
- Delete a specified item.
- Iterate over a set of values.

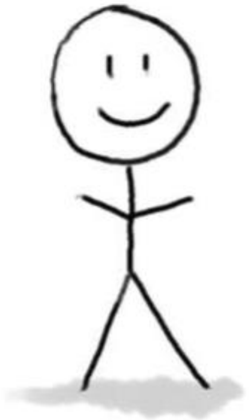
Properties of an algorithm

1. **Input** - An algorithm has input values from a specified set.
2. **Output** - From each set of input values an algorithm produces output values from a specified set. The output values are the solution to the problem.
3. **Definiteness** - The steps of an algorithm must be defined precisely.
4. **Correctness** - An algorithm should produce the correct output values for each set of input values.
5. **Finiteness** - An algorithm should produce the desired output after a finite (but perhaps large) number of steps for any input in the set.
6. **Effectiveness** - It must be possible to perform each step of an algorithm exactly and in a finite amount of time.
7. **Generality** - The procedure should be applicable for all problems of the desired form, not just for a particular set of input values.

You as a problem solver / algorithm designer

The algorithm designer’s question

Can I do better?



Algorithm designer

The algorithm designer's internal monologue…

What exactly do we mean by better? And what about that corner case? Shouldn't we be zero-indexing?

Plucky the Pedantic Penguin

Detail-oriented
Precise
Rigorous

Can I do better?

Algorithm designer

Dude, this is just like that other time. If you do the thing and the stuff like you did then, it'll totally work real fast!

Lucky the Lackadaisical Lemur

Big-picture
Intuitive
Hand-wavey

Both sides are necessary!

ACTIVITY 1: What are algorithms for you?

FOR OUR LIBRARY OR ONLINE SEARCH ACTIVITY LATER.

Data Structures

A computer is an electronic machine which is used for data processing and manipulation. When programmer collects such type of data for processing, he would require to **store** all of them in computer’s main memory.

This is where we use data structures.

What are Data Structures?

- A data structure is an **arrangement of data** in a computer’ memory.
- A method or way of organizing all data items that considers not only the elements stored but also their **relationship to each other**.

Data structure and algorithms go hand in hand. Algorithms **manipulate the data** in data structures in various ways (insertion, deletion, iteration, search).

Focus of the subject

How do we organize information so that we can find, update, add, and delete portions of it **efficiently**?

Given a problem, what data structure will allow us **to handle information efficiently** to create the solution?

Applications of Data Structures

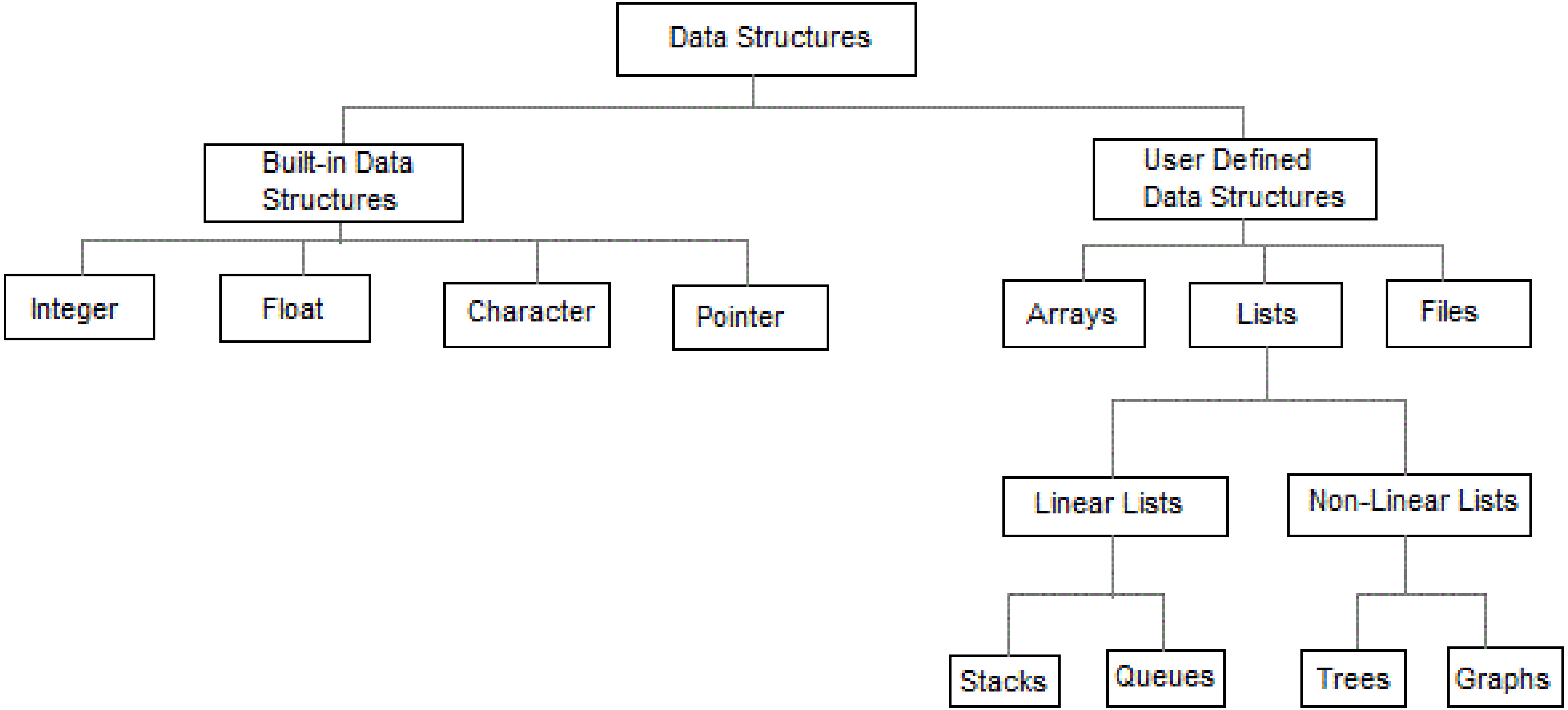
1. How does Google quickly find web pages that contain a search term?

2. What's the fastest way to broadcast a message to a network of computers?

3. How can a subsequence of DNA be quickly found within the genome?

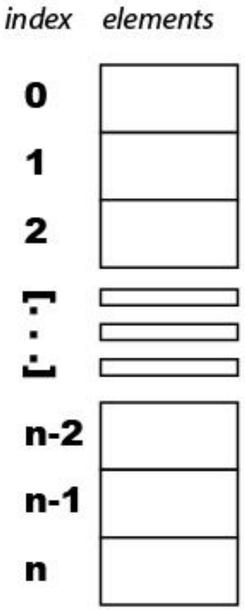
4. How does your operating system track which memory (disk or RAM) is free?

5. In the game Half-Life, how can the computer determine which parts of the scene are visible?



INTRODUCTION TO DATA STRUCTURES

An Array: array[n]



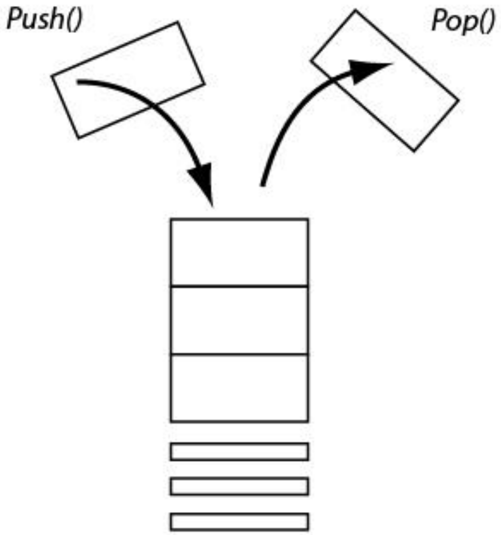
Typical features:

indexing
length/size
copying

Good For:

storing a fixed number of things and doing something to every one of those things

A Stack:



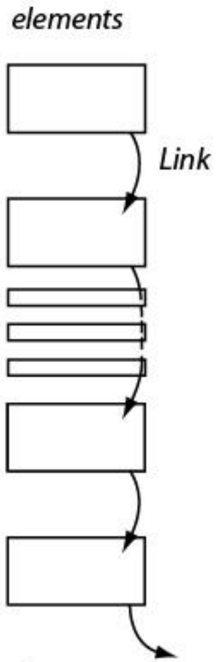
Typical features:

pushing
popping
size

Good For:

dealing with a flow of things that need to be handled in certain groups.

A Linked List:



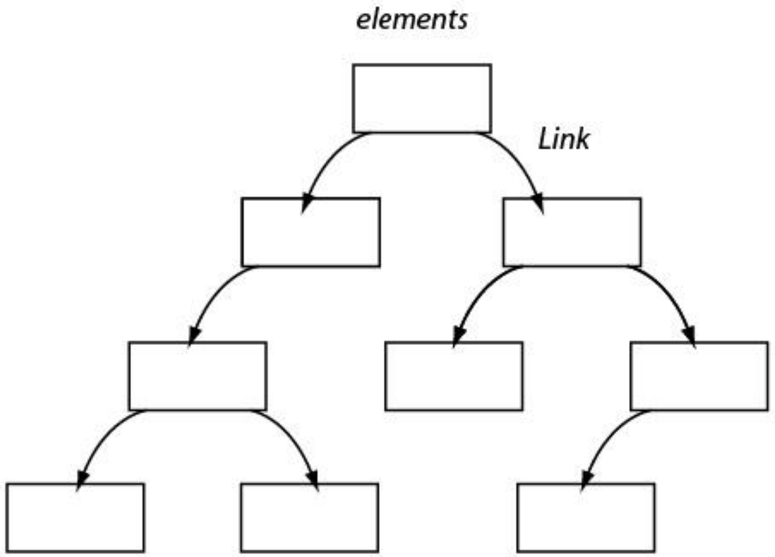
Typical features:

get "next" element
insert element
remove element

Good For:

dealing with a dynamically changing list -- i.e. where you may need to insert or remove elements from anywhere in the list

A Tree:



Typical features:

get "children"
insert child
remove element

Good For:

storing things that belong in trees creating rapidly search-able data sets (i.e. decision trees)

Overview

Characteristic	Description
Linear	In Linear data structures, the data items are arranged in a linear sequence. Example: **Array**
Non-Linear	In Non-Linear data structures, the data items are not in sequence. Example: **Tree, Graph**
Homogeneous	In homogeneous data structures, all the elements are of same type. Example: **Array**
Non-Homogeneous	In Non-Homogeneous data structure, the elements may or may not be of the same type. Example: **Generic (Object class)**
Static	Static data structures are those whose sizes and structures associated memory locations are fixed, at compile time. Example: **Array**
Dynamic	Dynamic structures are those which expands or shrinks depending upon the program need and its execution. Also, their associated memory locations changes. Example: **Linked Lists**

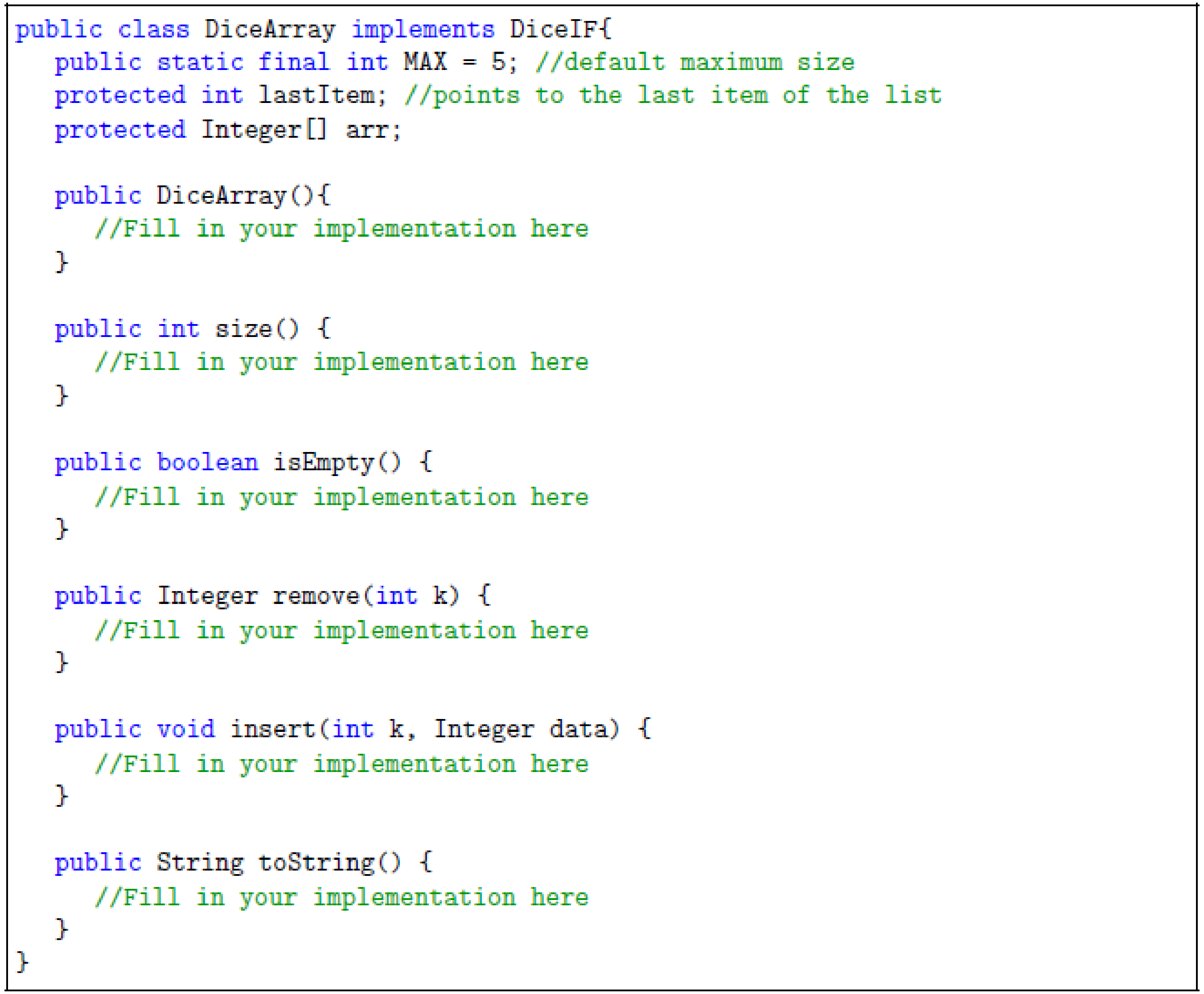
Implementation

Data structures are commonly implemented as **classes**. These classes are the model of you data structure.

On the other hand, algorithms such as (a) adding an element, (b) deleting an element, (c) updating the value of an element, and (d) iterating through the set of values are in the form of **functions/methods**.

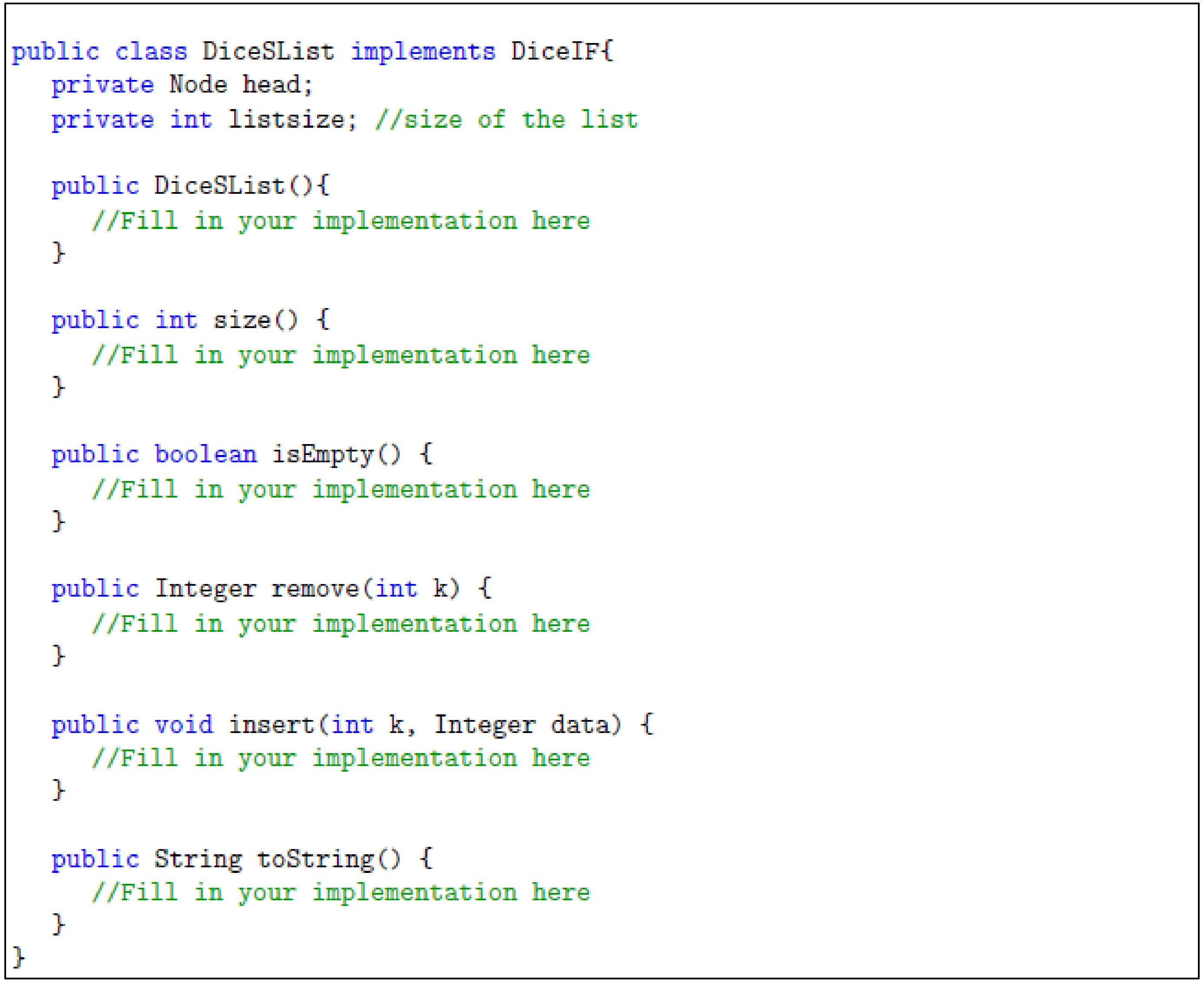
Instances of a data structure can be called through creating an **object** and performing actions can be done through calling the object's methods.

Example #1



```
public class DiceArray implements DiceIF{
   public static final int MAX = 5; //default maximum size
   protected int lastItem; //points to the last item of the list
   protected Integer[] arr;

   public DiceArray(){
      //Fill in your implementation here
   }

   public int size() {
      //Fill in your implementation here
   }

   public boolean isEmpty() {
      //Fill in your implementation here
   }

   public Integer remove(int k) {
      //Fill in your implementation here
   }

   public void insert(int k, Integer data) {
      //Fill in your implementation here
   }

   public String toString() {
      //Fill in your implementation here
   }
}
```

Example #2



```java
public class DiceSList implements DiceIF{
   private Node head;
   private int listsize; //size of the list

   public DiceSList(){
      //Fill in your implementation here
   }

   public int size() {
      //Fill in your implementation here
   }

   public boolean isEmpty() {
      //Fill in your implementation here
   }

   public Integer remove(int k) {
      //Fill in your implementation here
   }

   public void insert(int k, Integer data) {
      //Fill in your implementation here
   }

   public String toString() {
      //Fill in your implementation here
   }
}
```

Recap: What are data structures for you?

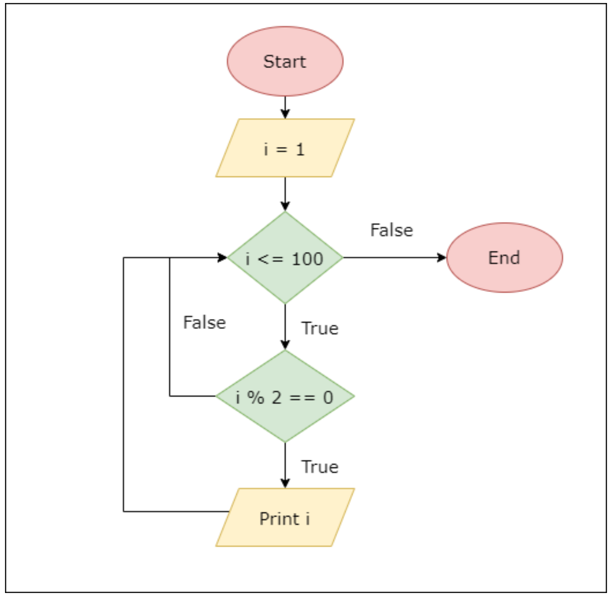
Example problems (Mental – WarmUP)

1. Finding even numbers from [1-100]
2. Finding numbers divisible by 3 and 5 from [1-100]
3. Identifying if a string is a palindrome or not.

Example problems (Mental – WarmUP)

Finding Even Numbers from [1-100]

FLOWCHART



PROGRAM CODE

```
public class EvenNumber {
    public static void main(String[] args) {

            for(int i = 1; I <= 100; i++) {
                if(i % 2 == 0) {
                    System.out.print(i);
                }
            }
        }
    }
```

Example problems (Mental – WarmUP)

Finding numbers divisible by 3 & 5 from [1-100]

ALGORITHM

```
1. Program Start
2. Initialize Variable i
3. Set i to 1
4. Is i divisible to 3 &
   5?
    IF YES: Print i then
    proceed count
    IF NO: Proceed count
5. Is i <= 100?
    IF YES: Proceed count
    IF NO: STOP PROGRAM
```

PROGRAM CODE

```
public class Divisible {
    public static void main(String[] args) {

            for(int i = 1; I <= 100; i++) {
                if(i % 3 == 0 && I % 5 == 0) {
                    System.out.print(i);
                }
            }
        }
    }
```

Example problems (Mental – WarmUP)

Identifying if a string is a Palindrome or Not

PSEUDOCODE

```
INPUT VALUE
Set left to index the leftmost
or first character

Set right to index the rightmost
or last character

while left is less than right
    compare left with right
    if not equal, then false
    increment left
    decrement right

end of loop
return true
```

PROGRAM CODE

```java
public class Palindrome {
    public static void main(String[] args) {
        System.out.print(“Enter String: ”);
        Scanner scan = new Scanner(System.in);
        String str = scan.nextLine();
        Char [] ch = str.toCharArray();
        int num = 0;

        for (int i = 0; i < ch.length/2; i++) {
            if (ch[i] != ch[ch.length-1-i]) {
                num = 1; break;
            }
        }

        if (num == 0) {
            System.out.println(str + “is Palindrome”);
        } else System.out.println (str + “is not Palindrome”);
    }
```

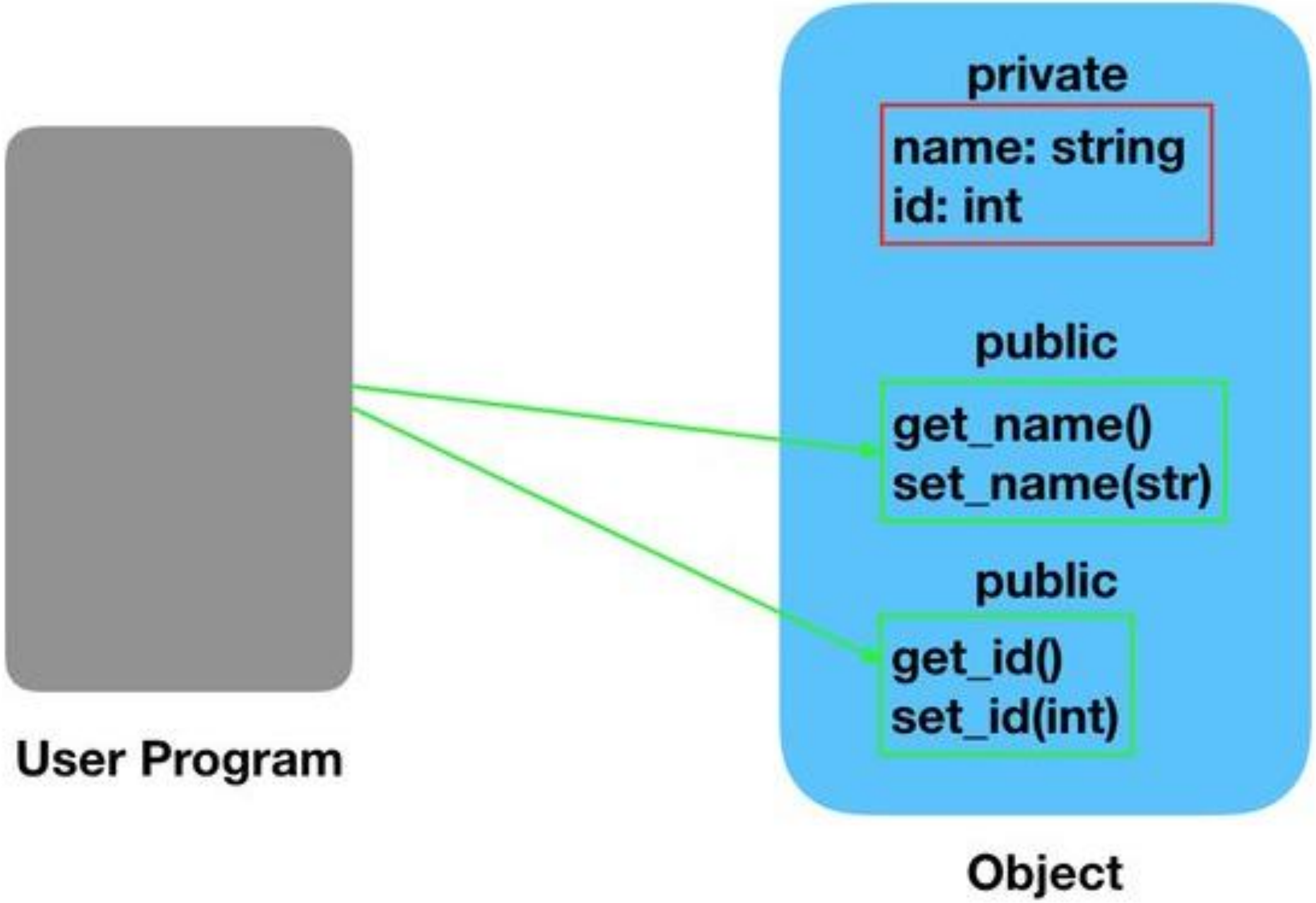
Abstraction

Abstraction is used to obscure background information or other extraneous data implementation so that consumers only see the relevant data. One of the most significant and fundamental aspects of programming is this.

Abstraction is linked with data structures as some users expect **security** and **anonymity** exclusively on some of the data types they construct and creating abstraction can attain this factors.

Abstraction in Laundry Machines

Abstraction in Data



Implementation

Abstraction is commonly implemented using **abstract classes**.

An abstract class is a class that is declared as abstract – it **may or may not** include abstract methods.

This class cannot be instantiated, but they can be subclassed.

An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon):

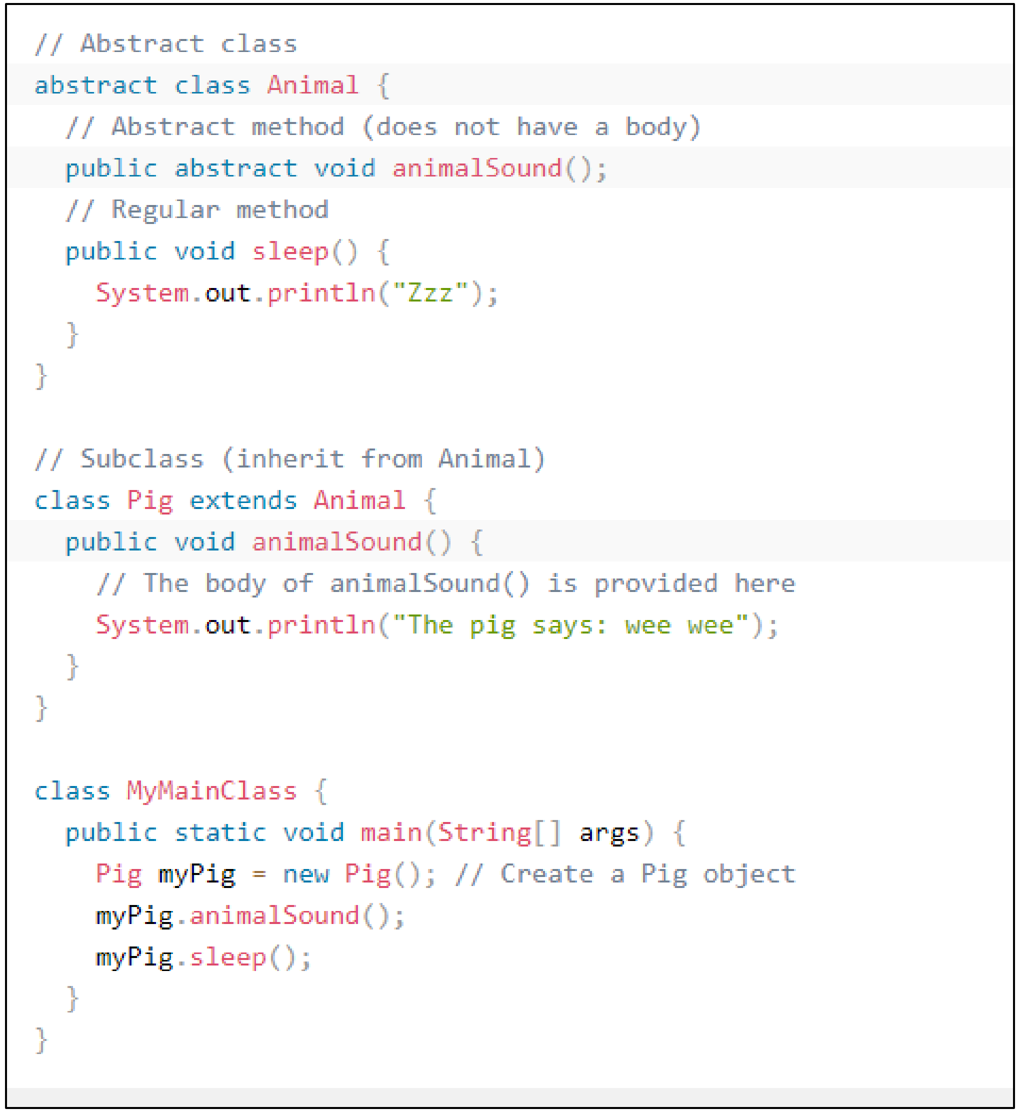
```
abstract void moveTo(double deltaX, double deltaY);
{
      //No body inside the abstract method
};
```

If a class includes abstract methods, then the class itself must be declared abstract, as in:

```
public abstract class GraphicObject {
   // declare fields
   // declare nonabstract methods
   abstract void draw();
}
```

Remember!

- When an abstract class is *subclassed*, the subclass usually provides **implementations** for **all of the abstract methods** in its parent class. However, if it does not, then the **subclass must also be declared abstract**.



```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class MyMainClass {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

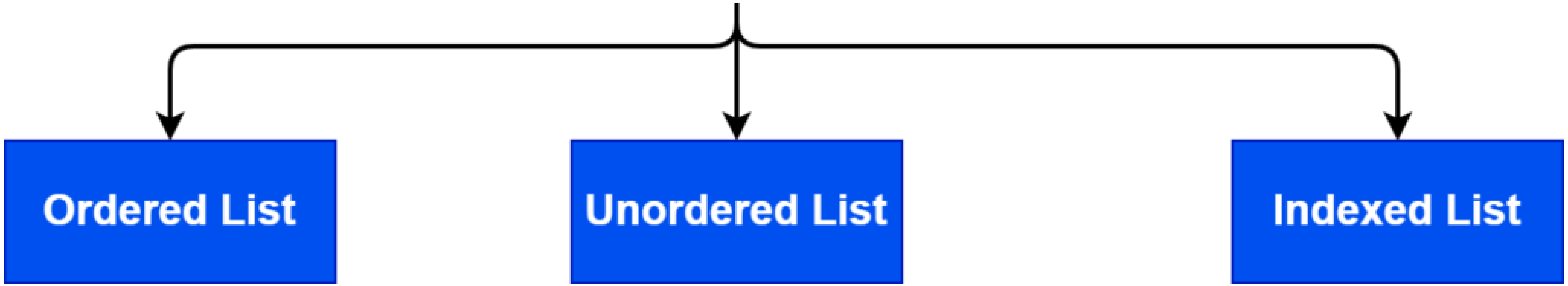
Abstract Data Types

An idea or model of a data type is an Abstract Data Type (ADT). ADT **eliminates the need for users to worry** about how a data type has been implemented. The implementation of the functions on a data type is **likewise handled by ADT**. Here, the user will **get ready-to-use preset functions** for any data type for every action.

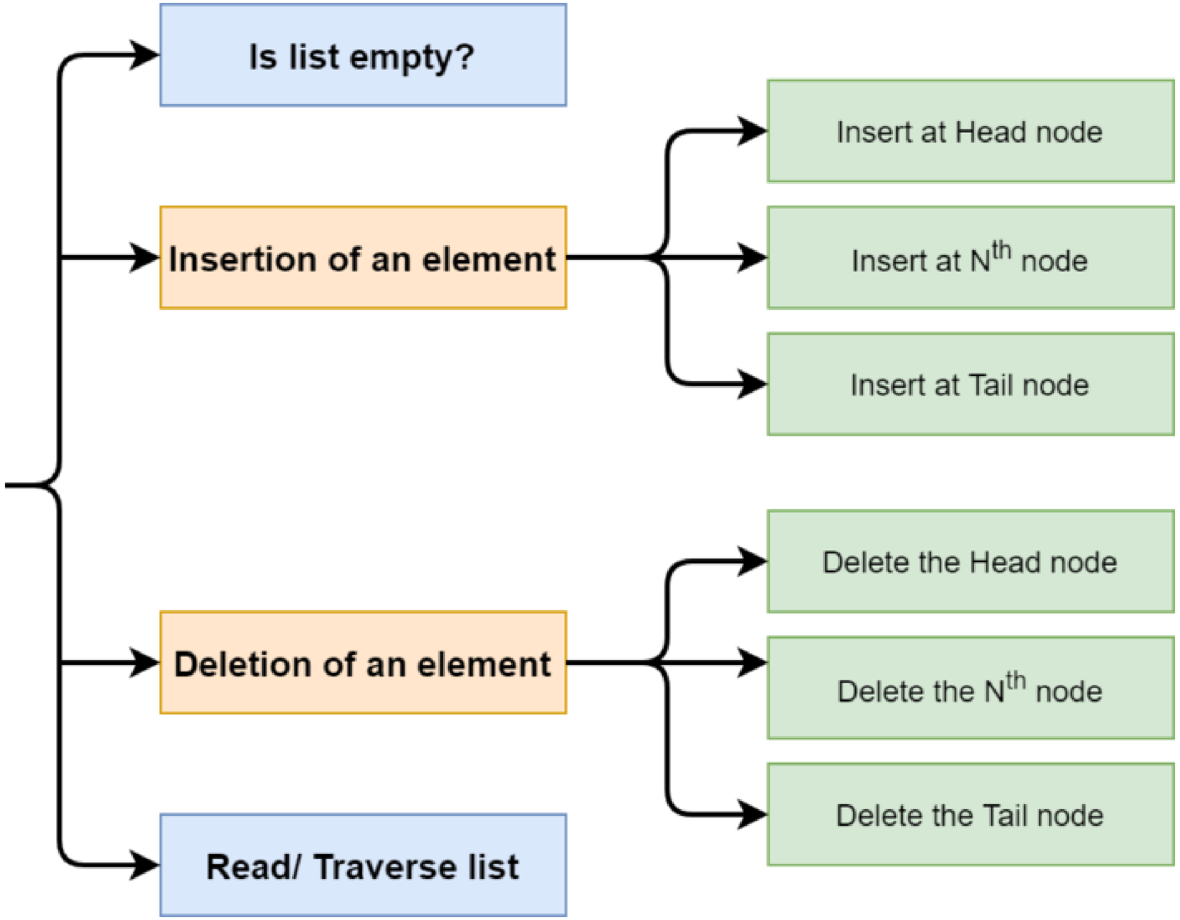
Generally, in ADT, a user knows what to do without disclosing how to do it. These kinds of models are defined in terms of their data items and associated operations.

Lists

A list is an ordered collection of the same data type. Moreover, a list contains a finite number of values. We can't store different data types in the same list. Here, ordered doesn't mean that the list is sorted, but they're properly indexed. Hence, if we know the location of the first element of a list, we can perform any operations on the whole list.



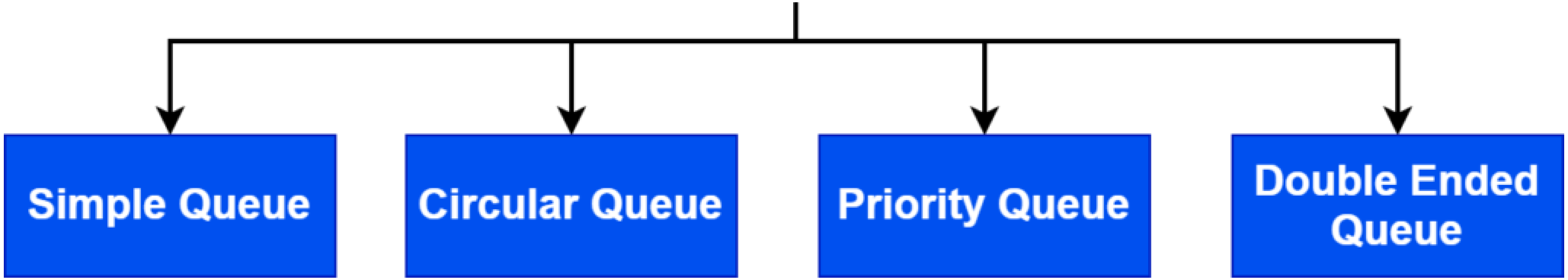
Operations in Lists



In a list, the basic operations are **checking if a list is empty or not, inserting an element** in some position, **deleting an element** from some position, and **read the whole list**.

Queues

A queue is a linear ADT with the restriction that insertion can be performed at one end and deletion at another. It works on the principle of FIFO (first-in, first-out). Hence, the first element to be removed from the queue is the element added first. We can store only one data type in a queue.



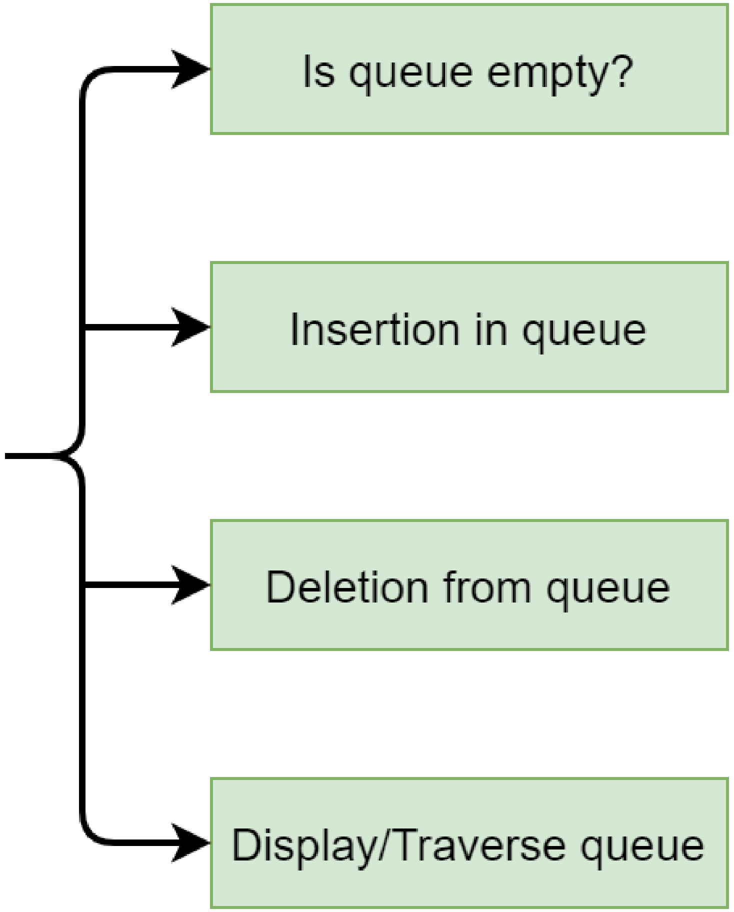
Simple Queue

Circular Queue

Priority Queue

Double Ended Queue

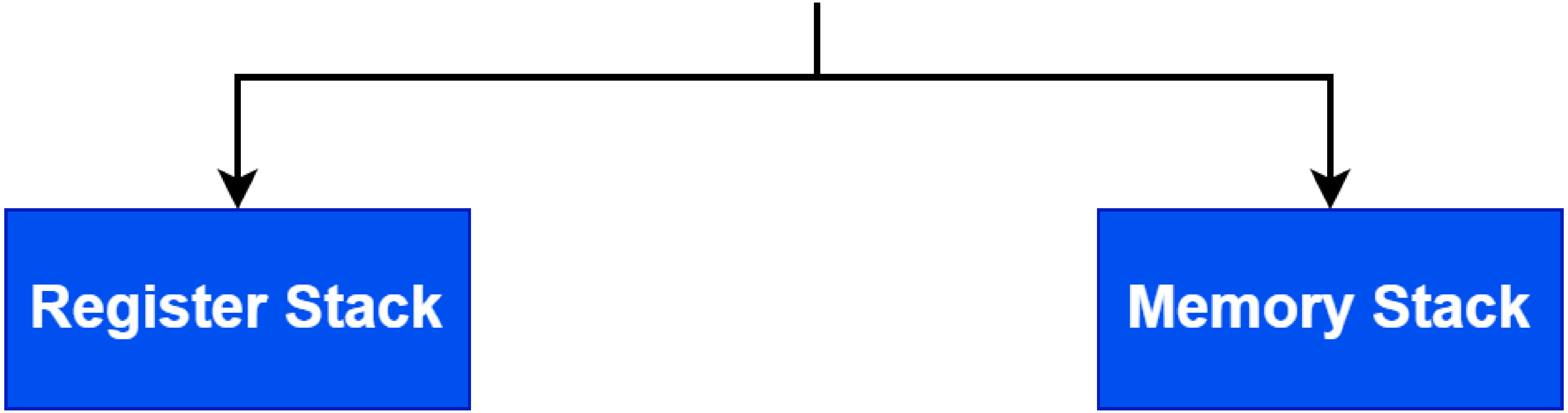
Operations in Queues



We use operations in queues to check if the queue is **empty** or not (is_Queue_Empty()). Or to **insert a value** within a queue (insertion_in_Queue(Value)). We can also **delete some item** in queue (deletion_in_Queue()) and to **read all the elements** in a queue (traverse_Queue).

Stacks

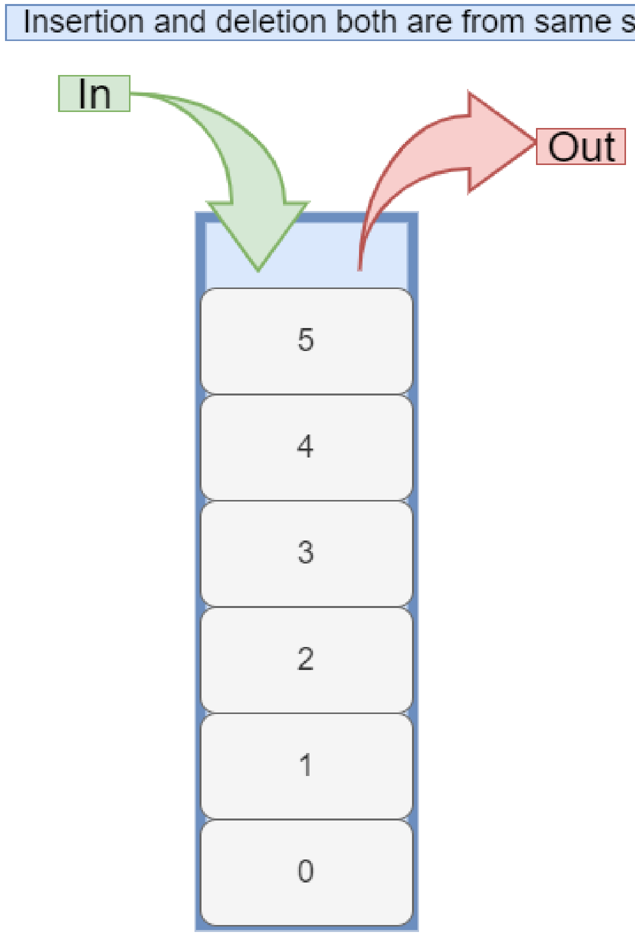
Stack is a linear ADT, with restrictions in inserting and deleting elements from the same end. It’s like making a pile of plates in which the first plate will be the last plate to be taken. It works on the principle of LIFO (last-in, first-out). It stores only one type of data



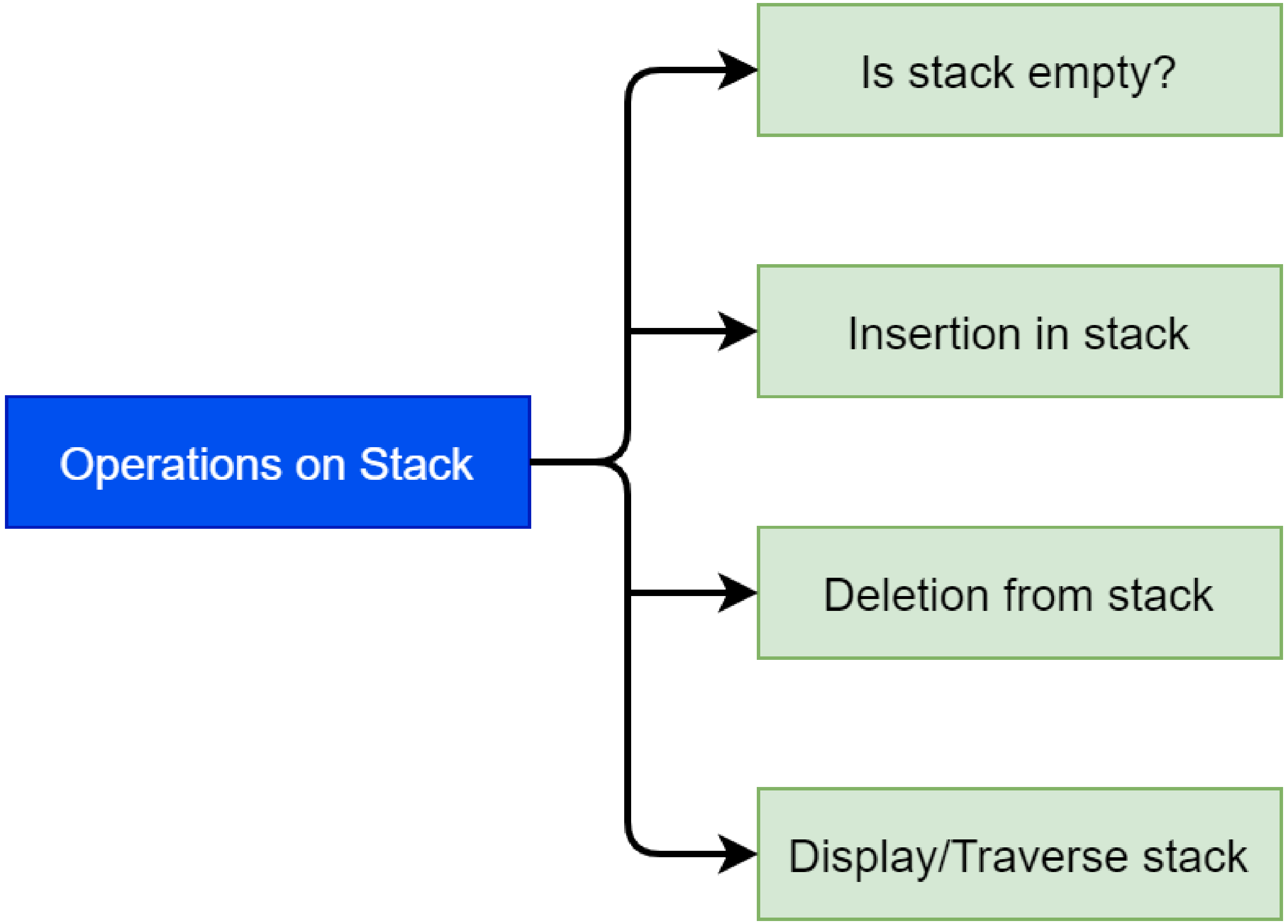
Register Stack

Memory Stack

Operations in Stacks



Insertion and deletion both are from same side



Operations on Stack

- Is stack empty?
- Insertion in stack
- Deletion from stack
- Display/Traverse stack

Recap: What did you learn about Abstraction and Abstract Data Types?

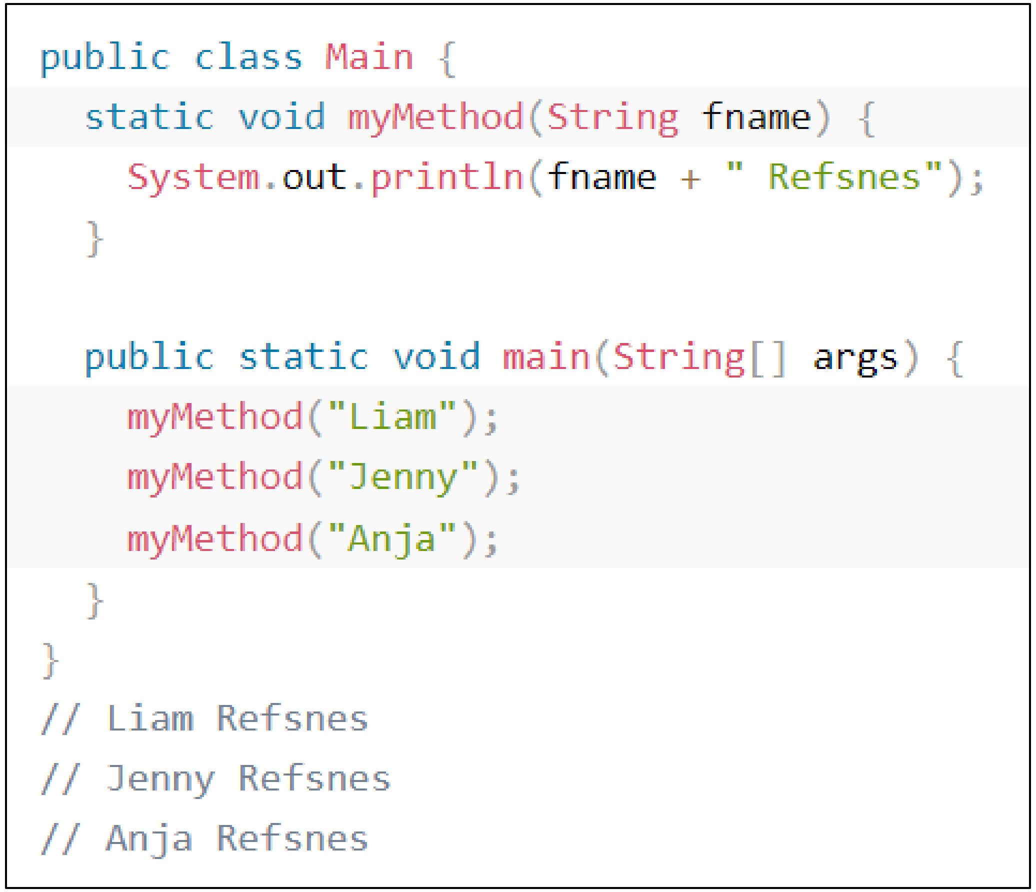
Parameters

Parameters are the variables that are listed as part of a method declaration. Each parameter must have a unique name and a defined data type.

In other words, Information can be passed to methods as parameter. **Parameters act as variables inside the method.**

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

Example of Parameter



```
public class Main {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

When a **parameter** is passed to the method, it is called an **argument**. So, from the example:

fname is the **parameter**; while…

Liam, Jenny and Anja are the **arguments**

Note: You can have many parameters as you like, but the method have the same number of arguments within execution

Parameter Passing

Parameter passing convention in a programming language, is the method used to pass one or more values (called actual parameters) to a function by means of a function call.

It is also a technique in programming, in which an actual parameter is passed to a function; this happens when a function calls another function.

Parameter Types

Formal Parameter: Formal parameters are usually written in the function prototype, and function header of the definition, i.e., they appear in function declarations.

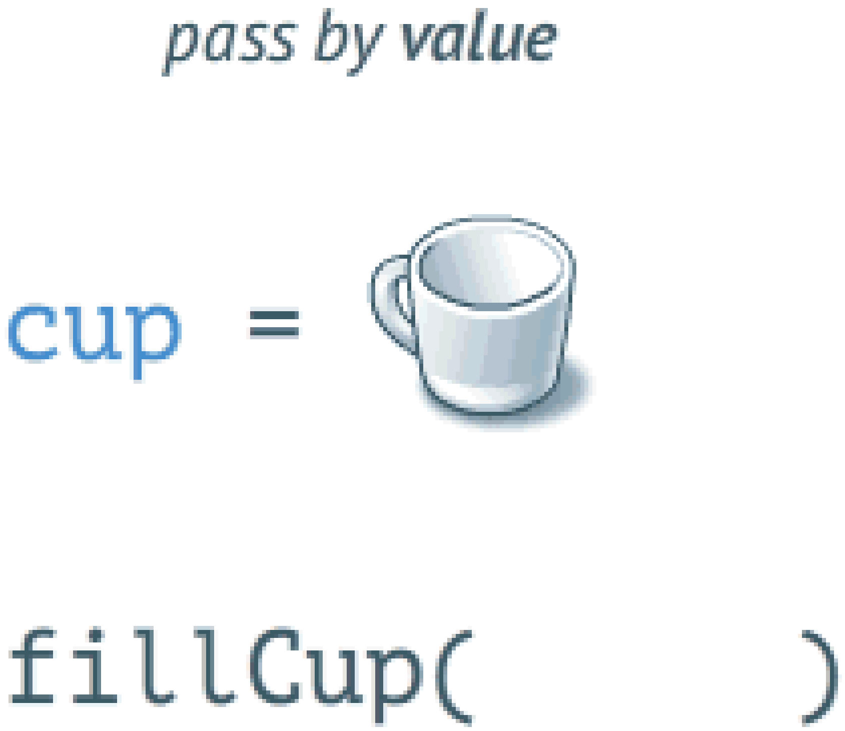
```
function_name(datatype variable_name)
```

Actual Parameter: An actual parameter is the values passed in the call of a function, i.e., they appear in the function call.

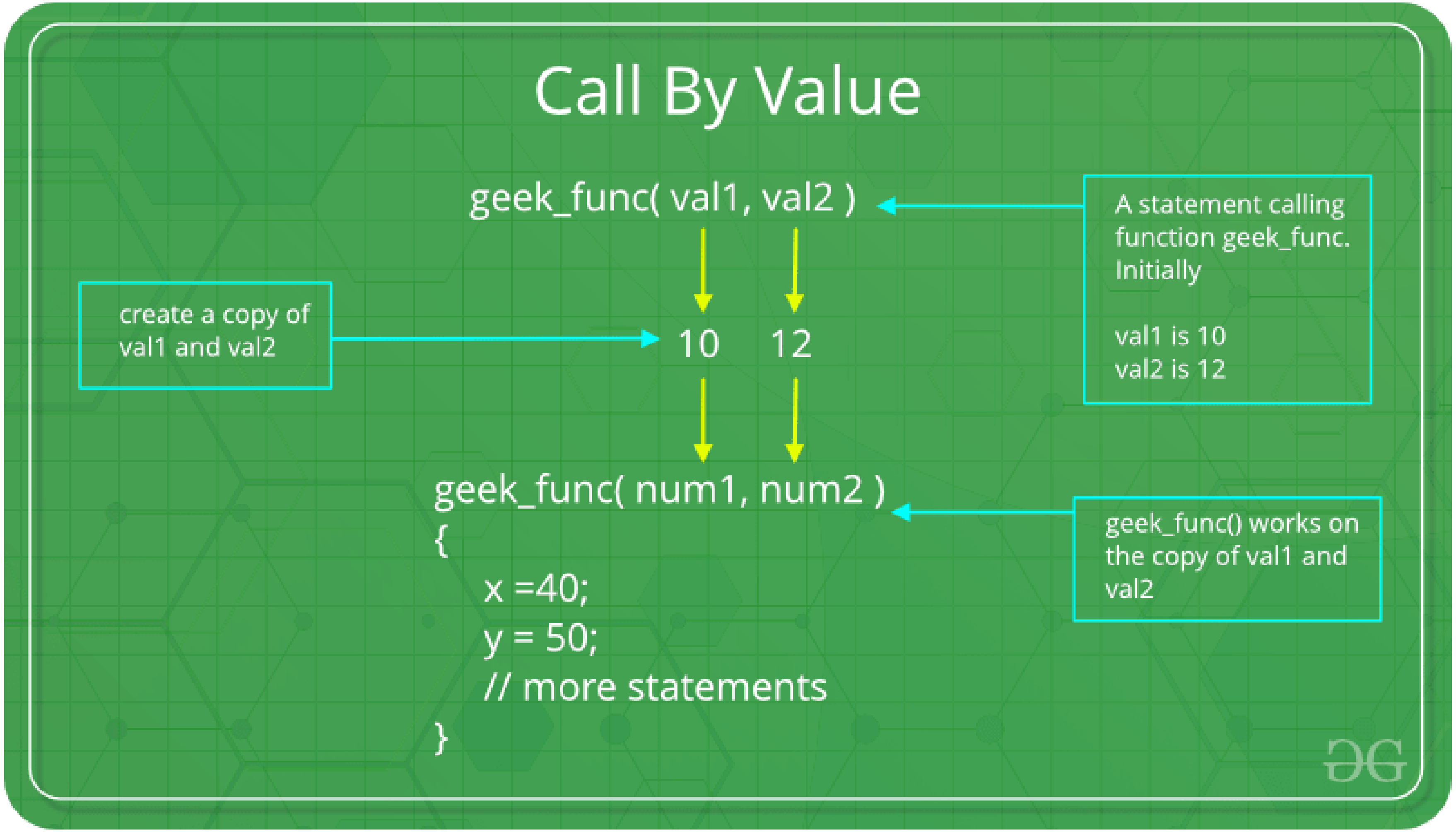
```
func_name(variable name(s));
```

Pass by Value

Formal parameter changes are not sent back to the caller. Any changes to the formal parameter variable made inside the called function or method will only have an impact on the separate storage location and not the real parameter in the calling environment.



Example of Pass by Value



Call By Value

```
geek_func( val1, val2 )

10    12

geek_func( num1, num2 )
{
    x =40;
    y = 50;
    // more statements
}
```

A statement calling function geek_func. Initially

val1 is 10
val2 is 12

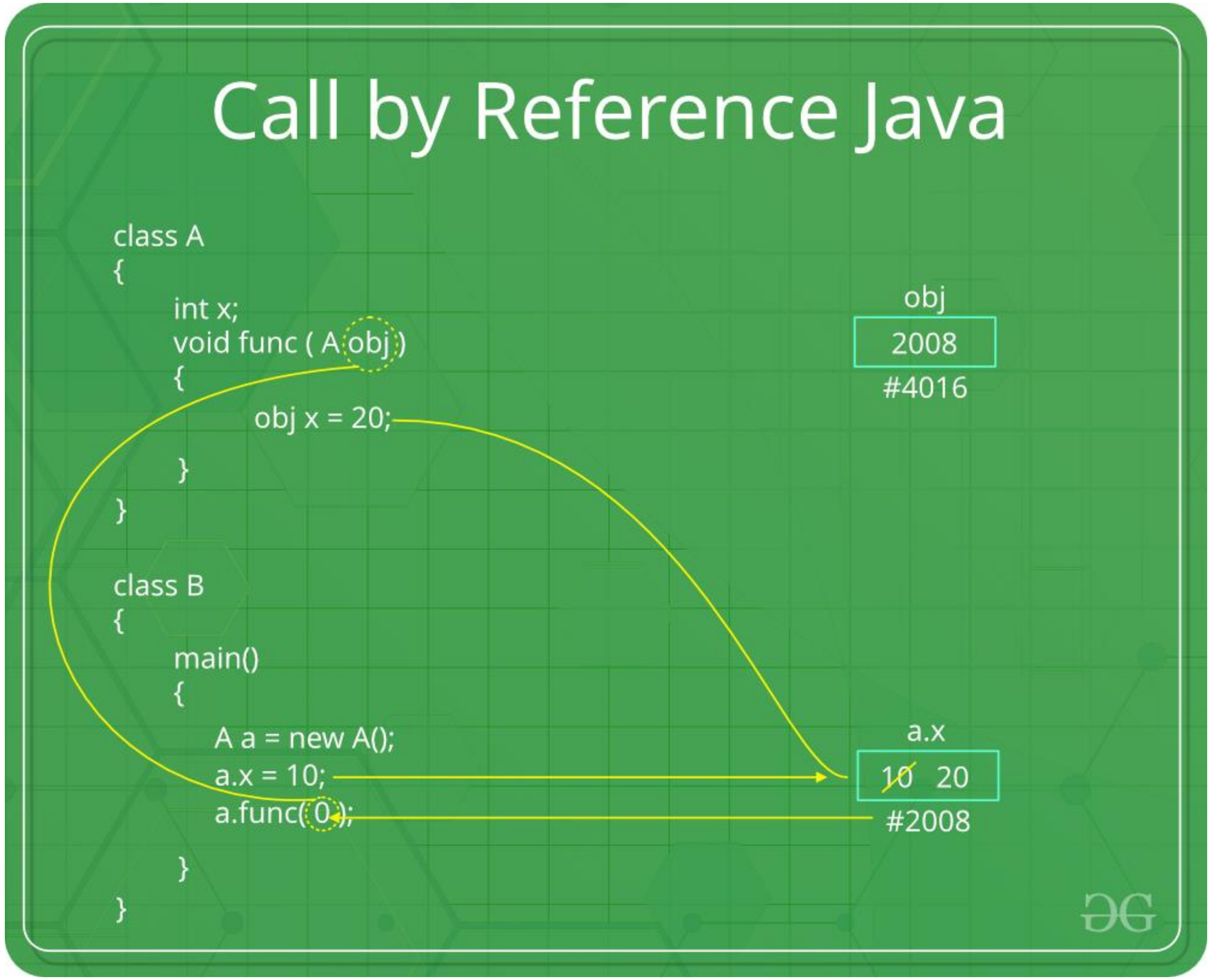
create a copy of val1 and val2

geek_func() works on the copy of val1 and val2

Pass by Reference

Changes made to formal parameter do get transmitted back to the caller through parameter passing. Any changes to the formal parameter are reflected in the actual parameter in the calling environment as formal parameter receives a reference (or pointer) to the actual data.

Example of Pass by Reference



Call by Reference Java

```
class A
{
     int x;
     void func ( A obj )
     {
          obj x = 20;
     }
}

class B
{
     main()
     {
        A a = new A();
        a.x = 10;
        a.func(0);
     }
}
```

Recap: What is parameter and what are the types of parameter passing?

Array Processing

```
nums = new
etup() {
(500, 400)
(int i = 0
ms[i] = n
```

SOURCE: Arrays in Processing 3: https://www.youtube.com/watch?v=5tSBbWySCC0

Array Processing

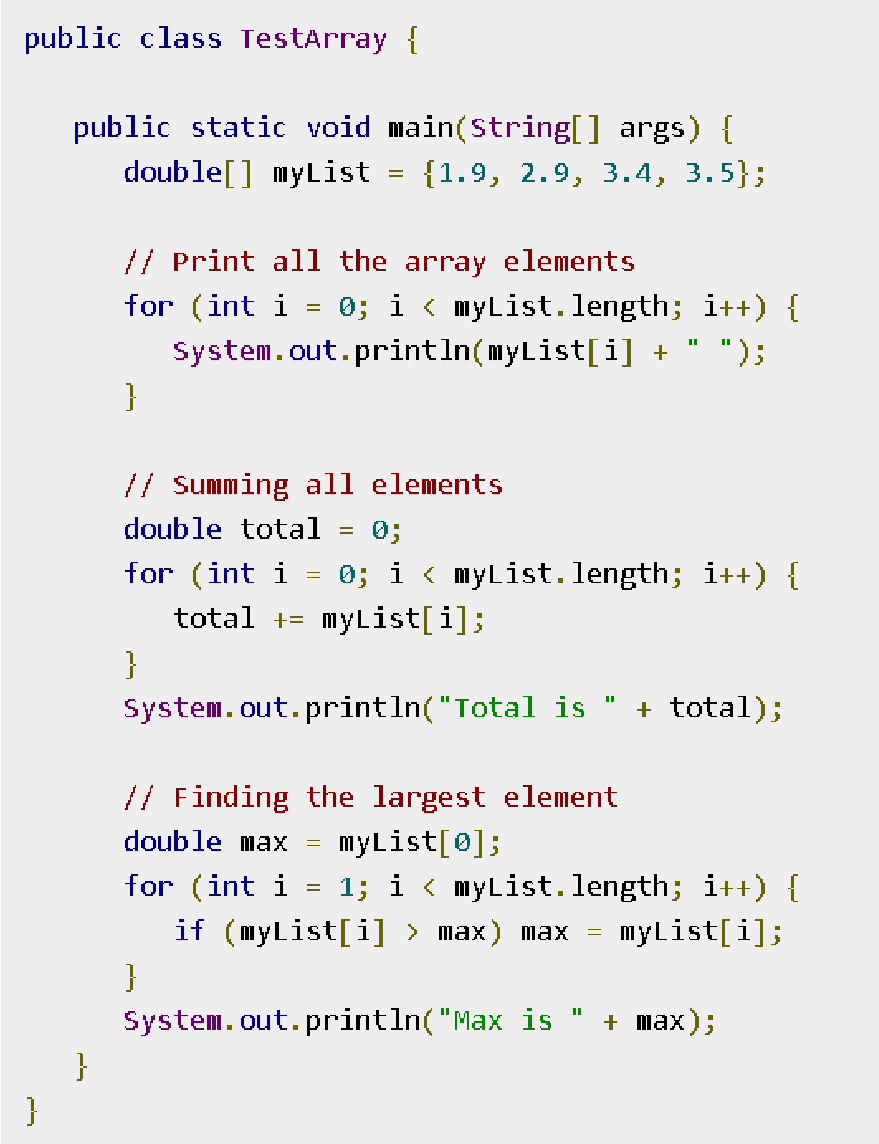
A fixed-size sequential collection of elements of the same type is stored in an array, a data structure that Java offers. It is important to think of an array as a collection of variables of the same type even though it is used to store a collection of data.

Instead of declaring individual variables, such as number0, number1, ..., and number99, we **process** one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

Processing Techniques

When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

foreach loop - which enables you to traverse the complete array sequentially without using an index variable.



```java
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (int i = 0; i < myList.length; i++) {
         System.out.println(myList[i] + " ");
      }

      // Summing all elements
      double total = 0;
      for (int i = 0; i < myList.length; i++) {
         total += myList[i];
      }
      System.out.println("Total is " + total);

      // Finding the largest element
      double max = myList[0];
      for (int i = 1; i < myList.length; i++) {
         if (myList[i] > max) max = myList[i];
      }
      System.out.println("Max is " + max);
   }
}
```

Array Manipulation

Arrays can be manipulated by using several actions known as **methods**. Some of these methods allow us to add, remove, modify and do lots more to arrays.

- **toString**() converts an array to a string separated by a comma.
- **join**() combines all array elements into a string.
- **concat** combines two arrays together or add more items to an array and then return a new array.
- **push**() adds item(s) to the end of an array and changes the original array.
- **pop**() removes the last item of an array and returns it

Array Manipulation

- **shift**() removes the first item of an array and returns it
- **unshift**() adds an item(s) to the beginning of an array and changes the original array.
- **splice**() changes an array, by adding, removing and inserting elements.
- **slice**() copies a given part of an array and returns that copied part as a new array. It does not change the original array.
- **split**() divides a string into substrings and returns them as an array.

Array Manipulation

- **indexOf**() looks for an item in an array and returns the index where it was found else it returns -1
- **lastIndexOf**() looks for an item from right to left and returns the last index where the item was found.
- **filter**() creates a new array if the items of an array pass a certain condition.
- **map**() creates a new array by manipulating the values in an array.
- **reduce**() calculates a single value based on an array.
- **forEach**() iterates through an array, it applies a function on all items in an array

Array Manipulation

- **every**() checks if all items in an array pass the specified condition and return true if passed, else false.
- **some**() checks if an item (one or more) in an array pass the specified condition and return true if passed, else false.
- **includes**() checks if an array contains a certain item.

Recap: What is array manipulation methods did you used before?

References

[1] Medium. 2020. An Intro To Algorithms: Searching And Sorting Algorithms. [online] Available at: <https://codeburst.io/algorithms-i-searching-and-sorting-algorithms-56497dbaef20> [Accessed 6 July 2020].

[2] GeeksforGeeks. 2020. Introduction To Algorithms - Geeksforgeeks. [online] Available at: <https://www.geeksforgeeks.org/introduction-to-algorithms/> [Accessed 6 July 2020].

[3] “Introduction to Data Structures and Algorithms.” Studytonight.com, www.studytonight.com/data-structures/introduction-to-data-structures.

[4] “What is abstraction in Programming?.” Education.IO, https://www.educative.io/answers/what-is-abstraction-in-programming

[5] “What is Abstract Data Type?” Baeldung. https://www.baeldung.com/cs/adt#:~:text=Abstract%20data%20type%20(ADT)%20is,functions%20on%20a%20data%20type.

[6] “Java Method Parameters” W3Schools. https://www.w3schools.com/java/java_methods_param.asp#:~:text=Parameters%20act%20as%20variables%20inside,String%20called%20fname%20as%20parameter.

[7] “Parameter Passing Convention.” Technipages. https://www.technipages.com/definition/parameter-passing-convention

[8] “Java – Arrays” TutorialsPoint. https://www.tutorialspoint.com/java/java_arrays.htm#:~:text=%20Java%20-%20Arrays%20%201%20Declaring%20Array,foreach%20loop%20because%20all%20of%20the...%20More%20

[9] “How to Manipulate Arrays in JavaScript” FreeCodeCamp. https://www.freecodecamp.org/news/manipulating-arrays-in-javascript/#:~:text=Summary%201%20toString%20%28%29%20converts%20an%20array%20to,an%20array%20and%20returns%20it%20More%20items...%20