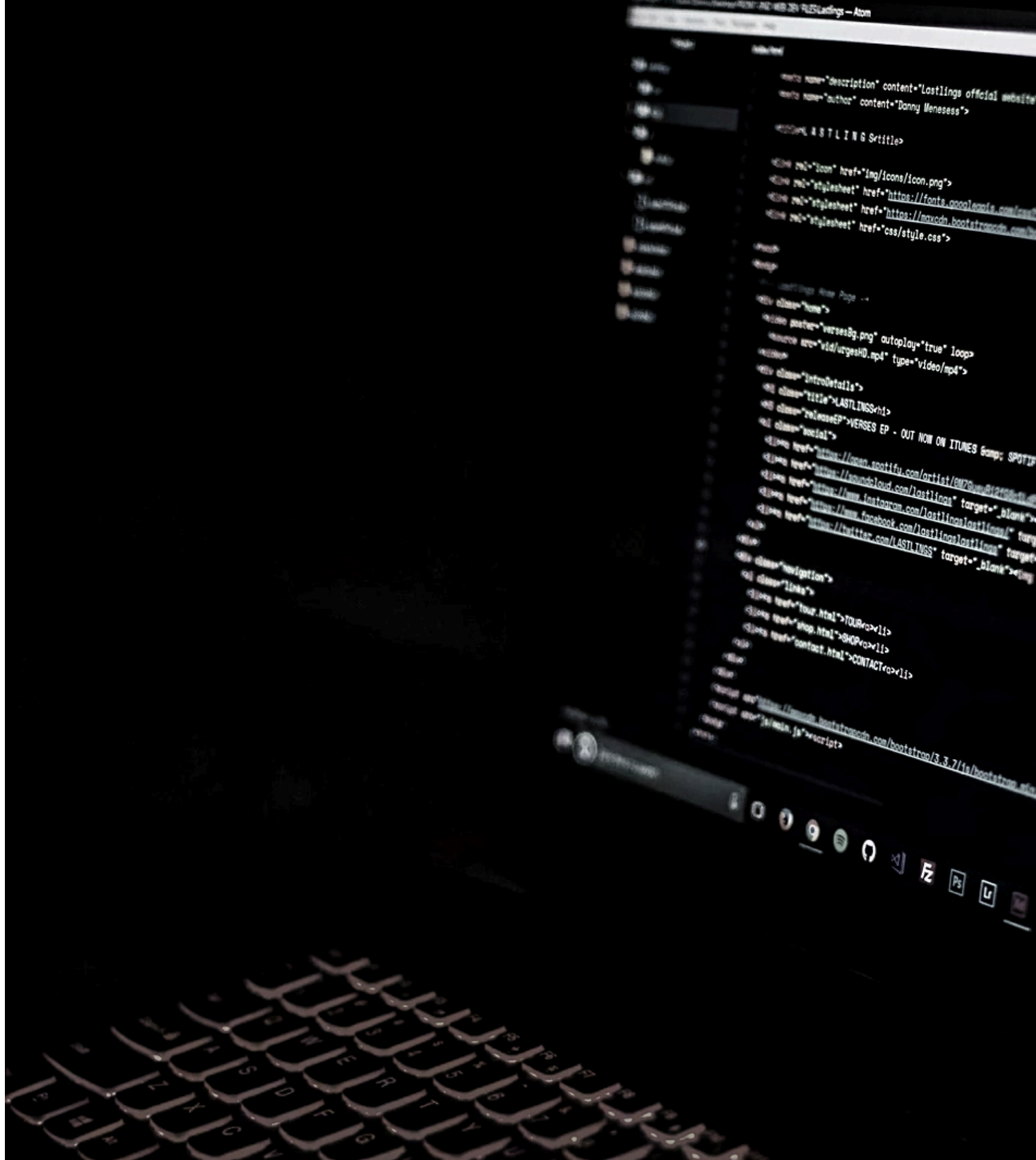
UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

PRIMITIVE AND REFERENCE DATA TYPES

PRIMITIVE DATA TYPES

Primitive data types are basic data types that are not objects and do not need to be instantiated to be used. Some examples of these primitive data types include:

byte, short, int, long, float, double, char, boolean.



B Y T E

Size: 8 bits

Purpose: Stores small integer values. Useful for saving memory in large arrays.

Range: -128 to 127

Syntax:

```
byte age = 25;
```

S H O R T

Size: 16 bits

Purpose: Stores medium integer values. Rarely used due to the availability of int.

Range: -32,768 to 32,767

Syntax:

```
short distance = 1500;
```

I N T

Size: 32 bits

Purpose: Most commonly used for storing integers.

Range: -2,147,483,648 to 2,147,483,647

Syntax:

```
int salary = 50000;
```

L O N G

Size: 64 bits

Purpose: Used for very large integer values.

Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Syntax:

```
long distanceToSun = 150000000000L;
```

F L O A T

Size: 32 bits
Purpose: Stores decimal numbers with single precision. Useful for saving memory when high precision is not required.
Range: Approximately ±3.40282347E+38F. Syntax:

```
float pi = 3.14f;
```

D O U B L E

Size: 64 bits
Purpose: Stores decimal numbers with double precision. Commonly used for precise calculations.
Range: Approximately ±1.79769313486231570E+308. Syntax

```
double largeDecimal = 123456.789;
```

C H A R

Size: 16 bits
Purpose: Stores a single Unicode character (letters, digits, symbols).
Range: '\u0000' (0) to '\uffff' (65,535)
Syntax:

```
char grade = 'A';
```

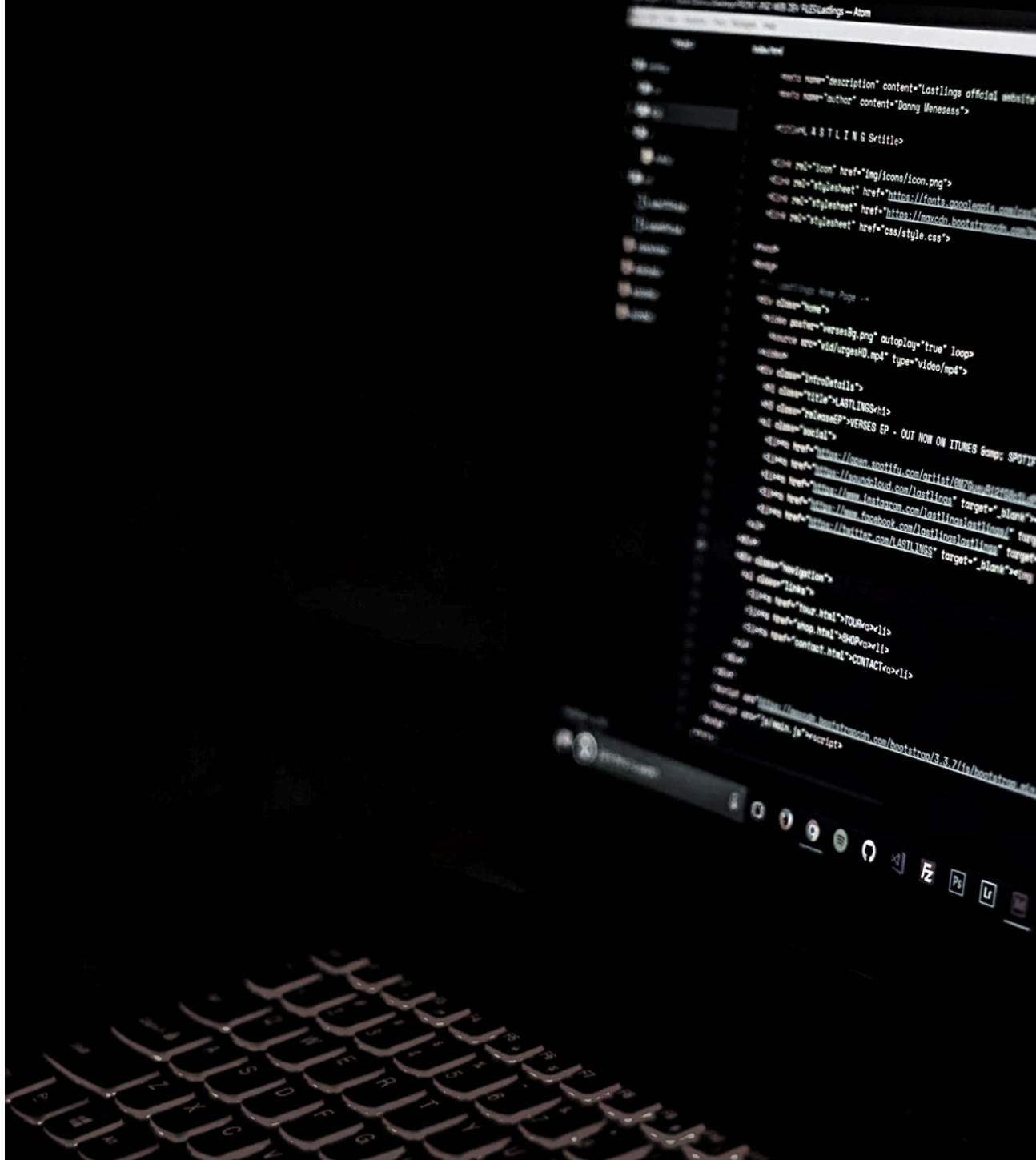
B O O L E A N

Size: 1 bit (conceptual, JVM uses larger blocks for efficiency)
Purpose: Stores true or false values. Often used in control statements.
Values: true or false
Syntax

```
boolean isJavaFun = true;
```

REFERENCE DATA TYPE

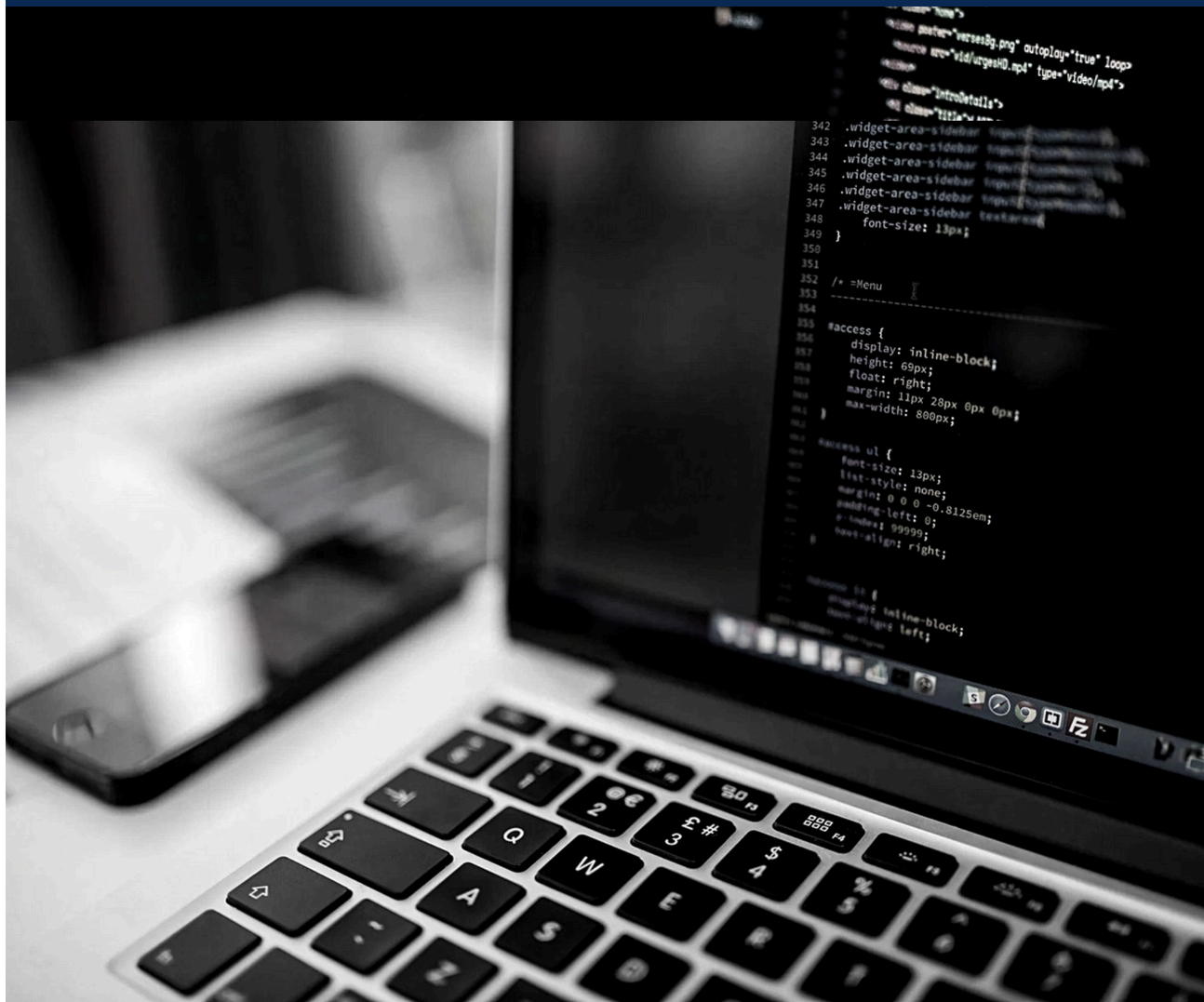
Reference types refer to objects or complex data structures. They store the memory address (reference) of the data, not the data itself, and are allocated on the heap. Classes, interfaces and arrys.



STATIC DATA

Static data belongs to the class instead of a specific instance of the class. This means you don't need to create an object of the class to access static data.

This is achieved using the static keyword, which indicates that a field, method, or block belongs to the class as a whole, not to individual objects created from the class.



CHARACTERISTICS OF STATIC DATA

Class-Level Ownership:
Static members do not require an instance of the class to access them; they are accessed directly using the class name.

Shared Memory:
Static members are stored in a single memory location (in the static memory area) and are shared among all instances of the class.

Long Lifetime:
Static members persist in memory for the entire duration of the program's execution.

Global Access:
Static data can be accessed by any object of the class or directly from a static context within the class.

STATIC VARIABLES:

Variables that belong to the class rather than to individual instances. They are used to store information share

```
class Example {
    static int counter = 0;

    public Example() {
        counter++;
    }
}

public class Main {
    public static void main(String[] args) {
        Example e1 = new Example();
        Ex                 xample();
        Sy                 (Example.counter); // Outputs: 2
    }
}
```

STATIC METHODS:

Methods that can be called without creating an instance of the class. These methods cannot directly access non-static instance variables.

```
class Calculator {
    static int add(int a, int b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        System.out.println(Calculator.add(5, 3));
    }
}
```

COMMON USES

Constants:
Variables that don't change, marked as static final.

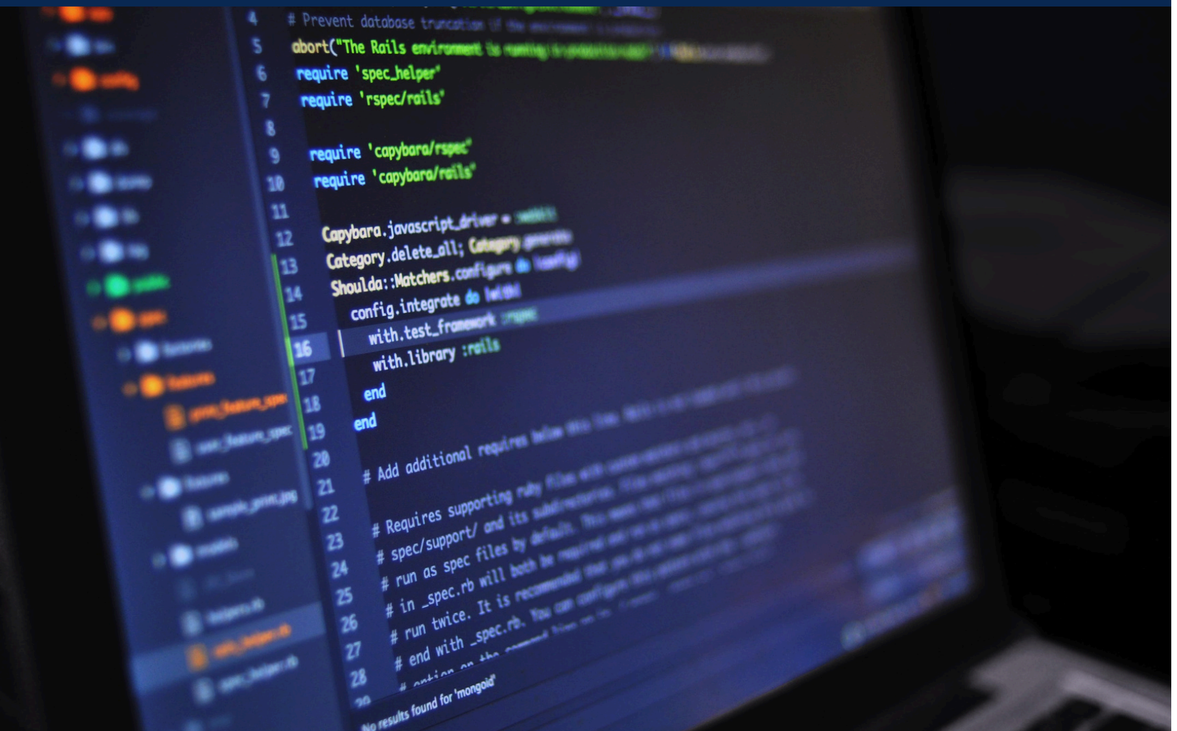
```
class Config {
    static final double PI = 3.14159;
}
```

Advantages of Static Data
Allows sharing information between all instances of a class.
Reduces memory consumption by avoiding duplication of common data

UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

DYNAMIC DATA STRUCTURES IN JAVA: STACKS AND QUEUES

INDICE DE CONTENIDOS

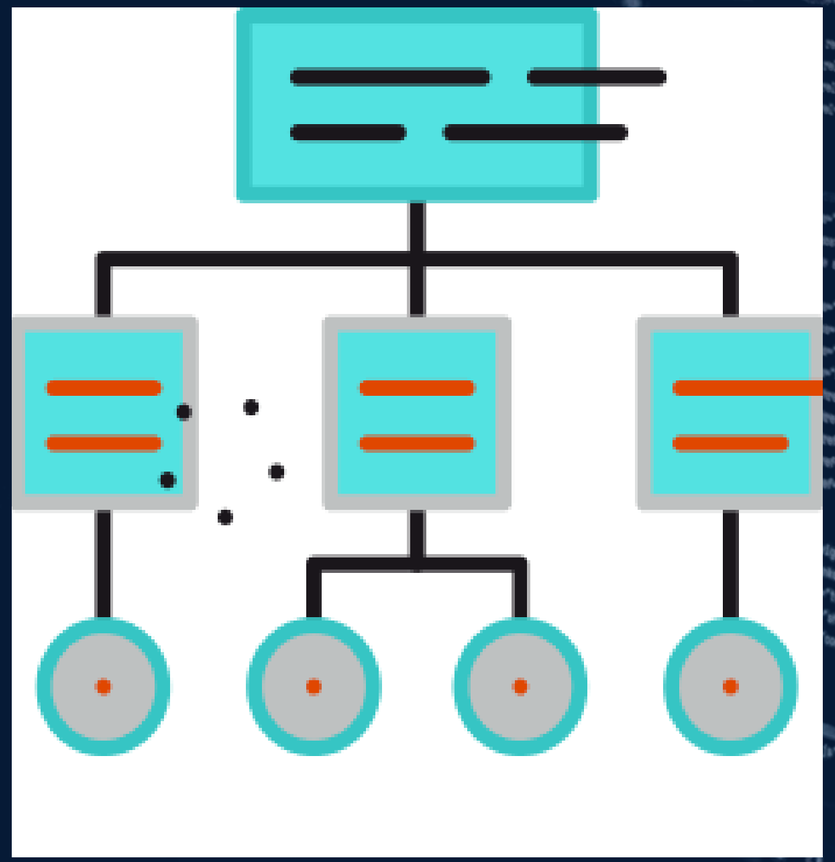


PROYECTO FIN DE GRADO

1.-INTRODUCTION

1. INTRODUCTION

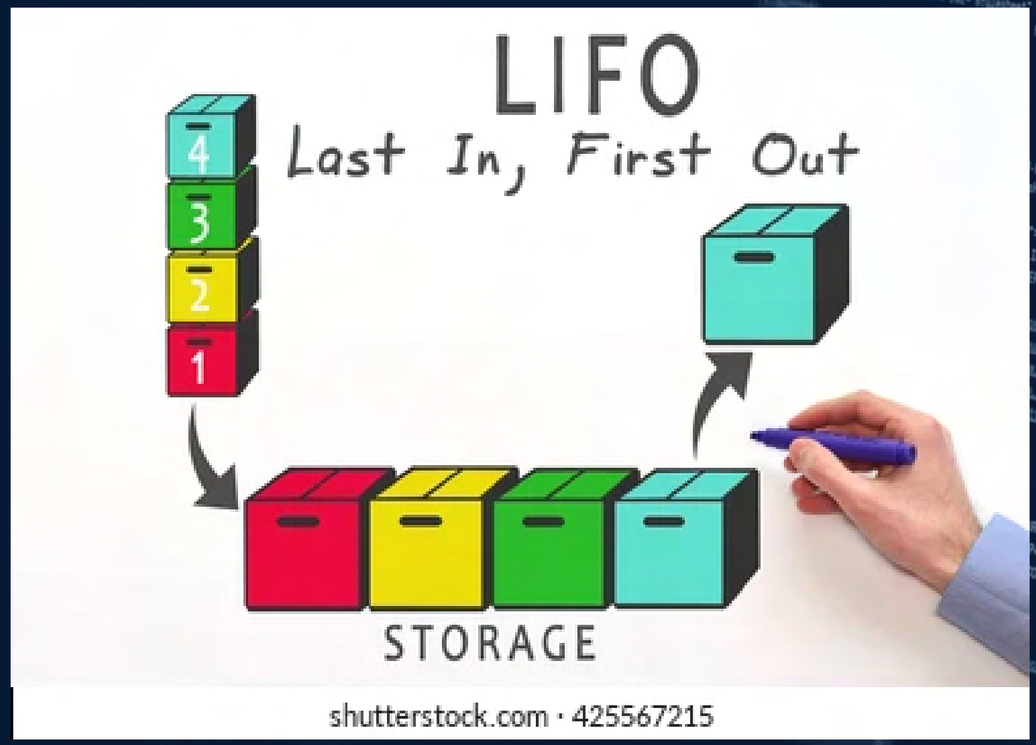
General definition: Stacks and queues are dynamic data structures that restrict how elements are inserted and removed. These structures are essential for solving problems that require sequential or hierarchical handling of data.



2.-STACKS

2. STACKS

Concept: A stack follows the LIFO (Last In, First Out) principle. Elements are added and removed from the same end, called the top. Usage in Java: The Stack class of the standard library implements abstract stacks. Main operations: push: Adds an element to the top. pop: Removes and returns the top element. peek: Returns the top element without removing it



EXAMPLE

```java
import java.util.Stack;

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

        Stack<String> pila = new Stack<>();
        pila.push("rojo");
        pila.push("blanco");
        pila.push("azul");

        System.out.println("Contenido de la pila: " + pila);
        String elementoEliminado = pila.pop();
        System.out.println("Elemento eliminado: " + elementoEliminado);
        System.out.println("Contenido de la pila tras el pop: " + pila);

        String elementoSuperior = pila.peek();
        System.out.println("Elemento en la cima de la pila: " + elementoSuperior);
        System.out.println("Contenido final de la pila: " + pila);
    }
}

```

input

```
Contenido de la pila: [rojo, blanco, azul]
Elemento eliminado: azul
Contenido de la pila tras el pop: [rojo, blanco]
Elemento en la cima de la pila: blanco
Contenido final de la pila: [rojo, blanco]


...Program finished with exit code 0
```

3.-QUEUES

3. QUEUES

A queue follows the FIFO (First In, First Out) principle. Elements are added to the end of the queue and removed from the beginning. Usage in Java: The Queue interface defines the behavior of a queue. LinkedList is a common implementation for dynamic queues. Main operations: add: Adds an element to the end of the queue. remove: Removes and returns the first element. peek: Returns the first element without removing it. Practical example: Code that enqueues and dequeues colors: Operations: Add colors to the end and remove them from the beginning. Output: Shows how the elements are processed in the order they entered.

EXAMPLE

```java
import java.util.Queue;

public class EjemploCola {
    public static void main(String[] args) {
        Queue<String> cola = new LinkedList<>();

        cola.add("rojo");
        cola.add("blanco");
        cola.add("azul");

        System.out.println("Contenido de la cola: " + cola);

        String elementoEliminado = cola.remove();
        System.out.println("Elemento eliminado: " + elementoEliminado);
        System.out.println("Contenido de la cola tras el remove: " + cola);

        String primerElemento = cola.peek();
        System.out.println("Primer elemento de la cola: " + primerElemento);
        System.out.println("Contenido final de la cola: " + cola);
    }
}
```

input

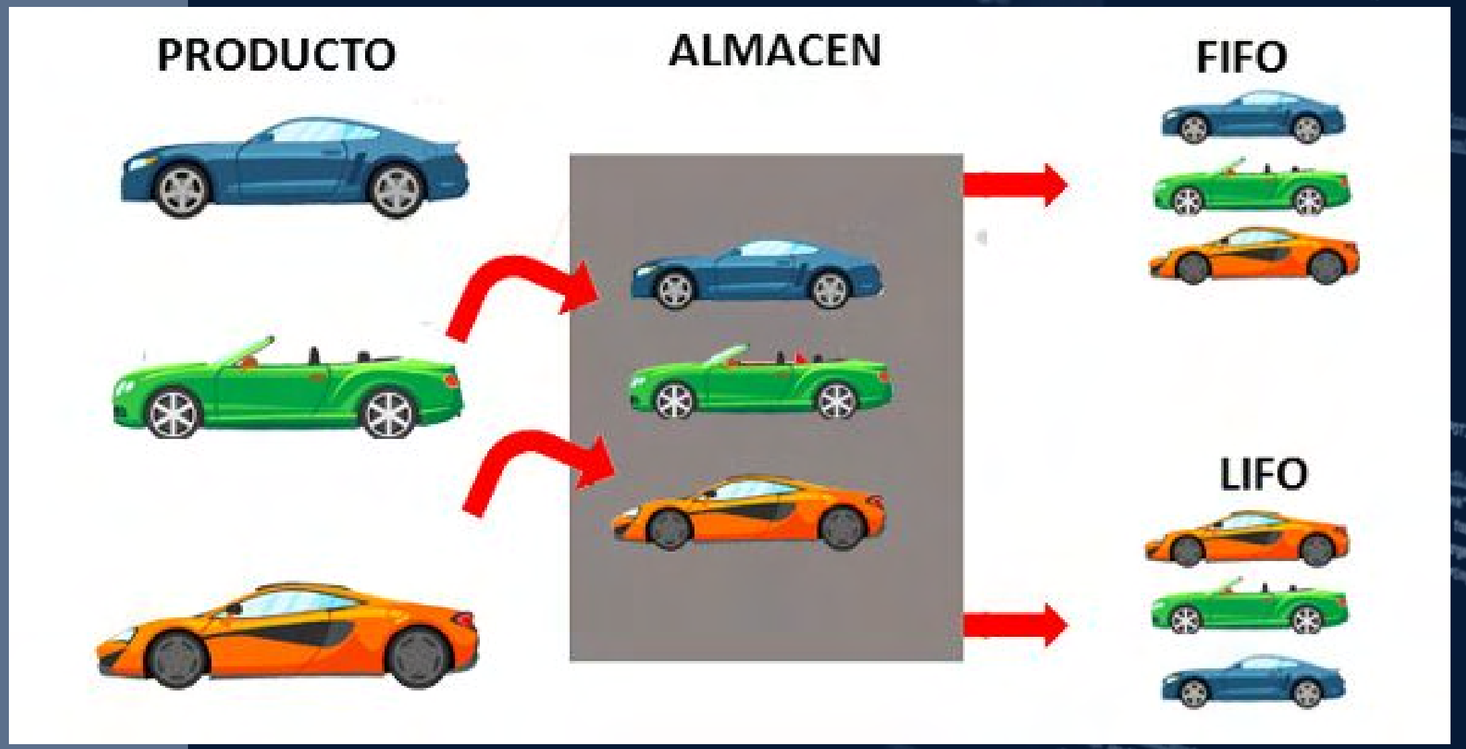
```
Elemento eliminado: azul
Contenido de la pila tras el pop: [rojo, blanco]
Elemento en la cima de la pila: blanco
Contenido final de la pila: [rojo, blanco]


...Program finished with exit code 0
Press ENTER to exit console.
```

4.-DIFFERENCES BETWEEN STACKS AND QUEUES, , USE CASES

4.-DIFFERENCES BETWEEN STACKS AND QUEUES, , USE CASES

Differences between Stacks and Queues Characteristic Stacks Queues LIFO Principle FIFO Insertion Top End of queue Deletion Top Start of queue Common use Backtracking, recursion Task management, processes.
Use Cases Stacks: Recursive algorithms. Navigation in browsers ("back" button). Resolution of mathematical expressions (postfix notation). Queues: Printing systems. Processing in multithreaded systems. Task management in priority queues.



BIBLIOGRAPHY:

BLOCH, J. (2018). EFFECTIVE JAVA (3RD ED.). ADDISON-WESLEY.

LAFORE, R. (2002). DATA STRUCTURES AND ALGORITHMS IN JAVA (2ND ED.). SAMS PUBLISHING.

ORACLE. (N.D.). THE JAVA™ TUTORIALS - COLLECTIONS. ORACLE. RETRIEVED FROM HTTPS://DOCS.ORACLE.COM/JAVASE/TUTORIAL/COLLECTIONS/

WEISS, M. A. (2013). DATA STRUCTURES AND ALGORITHM ANALYSIS IN JAVA (4TH ED.). PEARSON.

GEEKSFORGEEKS. (N.D.). STACK DATA STRUCTURE. RETRIEVED FROM HTTPS://WWW.GEEKSFORGEEKS.ORG/STACK-DATA-STRUCTURE/


THANKS