DATA STRUCTURES & ALGORITHMS

MICHAEL T. GOODRICH
ROBERTO TAMASSIA
MICHAEL H. GOLDWASSER

in JAVA™

WILEY

DATA STRUCTURES & ALGORITHMS

Chapter 1 Java Primer

Contents

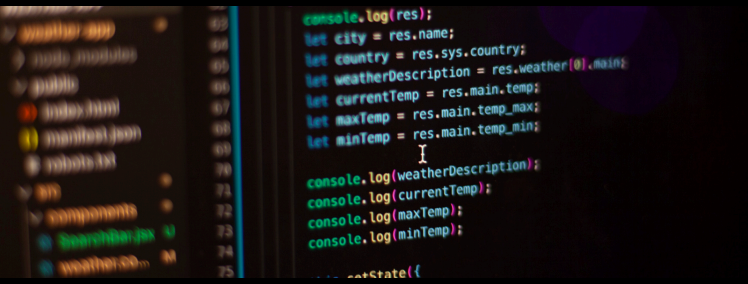
Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

💻 Getting Started with Java

✅ *What is Java?*

- Java is a programming language used to tell the computer what to do.
- It's a high-level language, meaning it's easier for humans to read and write compared to machine code.



🧠 *Key Java Concepts (for Beginners)*

- **1. Your First Program – Hello Universe!**

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

- ✅ This prints a message to the screen.
- 📌 Breakdown:
  - **public class Universe:** Defines a class (like a container for code).
  - **public static void main(String[] args):** The starting point of every Java program.
  - **System.out.println("...");**: Displays a message.

- **2. Blocks and Braces {}**

```java
if (x > 5) {
    System.out.println("x is big!");
}
```

- Code that belongs together goes inside { }.

- **3. Identifiers (Names)**
  - Used to name classes, methods, and variables.
  - Rules:
    - Must start with a letter.
    - Can include letters, numbers, and _ (underscores).
    - ❌ Can't use Java's reserved words (see below).

⚠ Reserved Words (Cannot Be Used as Names)
- Examples: class, public, if, int, true, null, etc.

💬 Comments (Non-code notes)

- Used to explain code. Ignored by the computer.
- Inline Comment: **// this is a comment**
- Block Comment:

```java
/* This is
   a block comment */
```

- Javadoc Comment (used for documentation):

```java
/** This is a Javadoc comment */
```

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

🔢 *Base (Primitive) Types in Java*

Type	Description	Example
`boolean`	true or false	`boolean flag = true;`
`char`	A single letter or symbol	`char grade = 'A';`
`byte`	Small integer (-128 to 127)	`byte b = 12;`
`short`	Small integer (-32,768 to 32,767)	`short s = 24;`
`int`	Regular integer	`int i = 100;`
`long`	Large integer (use `L` at the end)	`long l = 1000L;`
`float`	Decimal number (use `F` at the end)	`float pi = 3.14F;`
`double`	More precise decimal number	`double e = 2.718;`

📝 *Variable Tips*

- You can declare and initialize like this:

```java
int a = 5;
int b = 10, c = 15;
```

- Unused variables must be initialized before using them.
- Inside a class (but outside a method), Java gives default values:
  - 0 for numbers
  - false for booleans
  - '\u0000' (null char) for characters

! Common Mistakes for Beginners

- ✕ Forgetting main method – your program won't run!
- ✕ Using reserved words as names (e.g., int class = 5;)
- ✕ Forgetting to initialize variables before using them
- ✕ Forgetting semicolons (;) at the end of statements

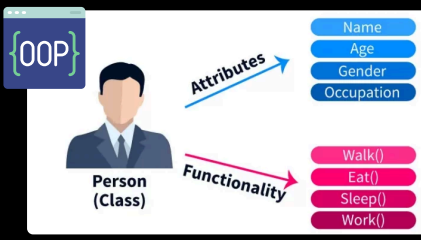
🎯 Recap

- Java programs start in the main method inside a class.
- Comments help explain code.
- Base types store basic values like numbers, letters, and true/false.
- Use correct syntax and naming rules to avoid errors.

Classes and Objects

🧠 *What Are Classes and Objects?*

- **Class =** A blueprint (template) for creating objects. It defines what data (fields) and actions (methods) the object will have.
- **Object =** An actual thing made from the class. It holds real values and can perform actions.
- 🍪 Think of a class like a mold for cookies, and an object as a cookie made from that mold.



Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

Key Parts of a Class

```java
public class Counter {
    private int count; // field (data)

    public Counter() { } // default constructor
    public Counter(int initial) { count = initial; } // custom constructor

    public int getCount() { return count; } // accessor method
    public void increment() { count++; } // update method
    public void increment(int delta) { count += delta; } // update method
    public void reset() { count = 0; } // update method
}
```

- **Definitions**
  - Field / Instance Variable: A variable inside the class that holds data (like int count).
  - Constructor: A special method that runs when you create a new object. It sets up initial values.
  - Accessor Method: A method that reads data (getCount()).
  - Update Method: A method that changes data (increment() or reset()).

How to Create and Use an Object

1. Declare a variable (just the name, no object yet):

```java
Counter c;
```

2. Create the object using `new`:

```java
c = new Counter(); // uses default constructor
```

Or:

```java
Counter d = new Counter(5); // starts with count = 5
```

3. Use the object with the dot (`.`) operator:

```java
c.increment();       // adds 1
c.increment(3);      // adds 3
int num = c.getCount();  // gets the count
c.reset();           // sets count to 0
```

Example Program

```java
public class CounterDemo {
    public static void main(String[] args) {
        Counter c = new Counter();  // count = 0
        c.increment();              // count = 1
        c.increment(3);             // count = 4
        int value = c.getCount(); // value = 4
        c.reset();                  // count = 0

        Counter d = new Counter(5); // count = 5
        d.increment();              // count = 6
        Counter e = d;              // e and d point to same object
        e.increment(2);             // count = 8
    }
}
```

Important Notes

- ✅ Objects are created with new ClassName()
  - new allocates memory and returns a reference to the object.
- ✅ Object variables are references (like remote controls)
  - You control the object with the variable.
  - Multiple variables can point to the same object (like d and e).
- 🚫 Using a variable without creating the object = NullPointerException

```java
Counter c;
c.increment(); // ❌ error! c is null
```

Common Beginner Mistakes

Mistake	Why it's wrong
Forgetting `new`	You only declare a reference, but no object exists yet.
Thinking two variables are separate when they point to the same object	If `e = d;`, both change the same object.
Using an object before it's created	Results in `NullPointerException`.
Assuming the method return type is part of the method signature	In Java, method overloading depends only on name + parameters, not return type.

Summary

- Use classes to define object structure.
- Use constructors to create objects.
- Use methods to work with object data.
- Use the dot . to call methods.
- Be careful with null references and object sharing.

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

📦 1. Defining a Class

```java
public class ClassName {
    // instance variables
    // methods
}
```

- A class is a code block that defines a type.
- Use {} to group all variables and methods.
- Class name = file name (ClassName.java).

🔑 2. Modifiers

- 🛡️ Access Control Modifiers

Modifier	Access Level
`public`	All classes can access
`private`	Only this class can access
`protected`	Same package and subclasses
(no keyword)	Only same package (package-private)

- ⚙️ Other Modifiers

Modifier	Meaning
`static`	Belongs to class, not to object
`final`	Cannot change or override
`abstract`	No body; subclass must implement

🔁 4. Methods

```java
public int getCount() {
    return count;
}
```

```java
public void increment(int delta) {
    count += delta;
}
```

- ✳️ Method Syntax

```java
[modifiers] returnType name(type1 param1, ..., typeN paramN) {
    // body
}
```

- **returnType** = value type (int, void, etc.).
- Use return to give a value back.
  - 🗣️ Methods can use instance variables and other methods.

📤 8. Returning Values

- Java methods return only one value.
- To return multiple: use an object or update parameters.

🧱 3. Instance Variables

```java
private int count;
```

- Stored **separately** for each object.
- Syntax:

```java
[modifiers] type name [= value];
```

🧱 5. Static Members

```java
public static int totalCount;
```

```java
Math.sqrt(4); // Static method
```

- Shared by all instances.
- Called like: ClassName.method().

🧱 6. Final

- **Variable**: constant, cannot change.

```java
final int MAX = 10;
```

- **Method**: cannot be overridden.
- **Class**: cannot be subclassed.

🌐 7. Abstract

```java
abstract class Shape {
    abstract double area();
}
```

- No objects of abstract class.
- Must be completed by subclass.

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

📩 9. Method Parameters

```java
public void sayHello(String name) {
    System.out.println("Hello, " + name);
}
```

- Declared in parentheses: (type name, type name, ...)
  - Example: (int x, double y)

- 🔁 Pass-by-Value
  - Java always uses pass-by-value:
    - For primitives: method gets a copy.
    - For objects: method gets a copy of the reference.

```java
public static void badReset(Counter c) {
    c = new Counter(); // does NOT affect original
}

public static void goodReset(Counter c) {
    c.reset(); // does affect original
}
```

✅ Changing an object's internal state works.
❌ Reassigning the object inside method does NOT affect original.

🏗️ 10. Constructors

```java
public class Counter {
    public Counter() { }
    public Counter(int value) {
        count = value;
    }
}
```

- A special method to initialize objects.
- Has same name as class.
- No return type (not even void).
- Called automatically with new:

```java
Counter c = new Counter(5);
```

🛠️ **Notes:**

- Cannot be static, abstract, or final.
- Java provides a default constructor (no parameters) only if no other constructor is defined.
- You can define multiple constructors with different parameters (called overloading).

🧍 11. The Keyword this

- Used to refer to the current object inside a method or constructor.
- ✅ Use Cases:
  - Differentiate local and instance variable

```java
public Counter(int count) {
    this.count = count; // left = instance, right = parameter
}
```

  - Call another constructor in the same class

```java
public Counter() {
    this(0); // calls the constructor with int parameter
}
```

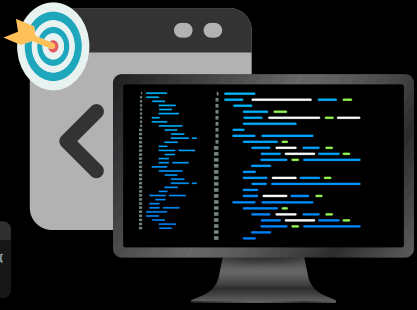
  - Pass the current object

```java
someMethod(this); // send current object as argument
```

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

What is the main Method?



- It's where your Java program starts running.
- Required in any class that you want to run as a full program.
- Without it, Java won't know where to begin.
  - ✅ Syntax

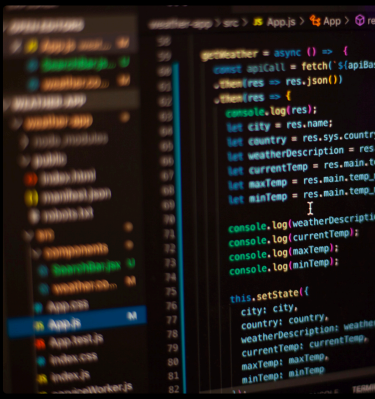
```java
public static void main(String[] args) {
    // your code goes here
}
```

🔍 **Breakdown**:
- **public**: Anyone can access this method.
- **static**: Runs without creating an object of the class.
- **void**: It doesn't return any value.
- **String**[] **args**: It holds extra input from the user (called command-line arguments).

💡 Example: Running a Java Program

- Suppose you have this class:

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



📄 *What are args[]?*

- It's a list of strings users can type when they run your program.
- Example:

```nginx
java Aquarium 45
```

- args[0] → "45" (a string!)
- You can use it in your code, like:

```java
int numberOfFish = Integer.parseInt(args[0]);  // convert string to number
System.out.println("You chose " + numberOfFish + " fish.");
```

⚠️ Common Mistakes

- Forgetting the exact method signature (must be public static void main(String[] args)).
- Trying to use args[0] without checking if it exists → can crash your program.
- Not converting strings to numbers when needed.

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

🩹 What if my class is NOT a full program?

- If you're just making a class to be used inside another program (like a helper class), you don't need a main method.
- But...
  - ✅ You can add a main method just for testing!

```java
public class Counter {
    public void increase() {
        // some logic
    }

    public static void main(String[] args) {
        Counter c = new Counter();
        c.increase();
        System.out.println("Counter tested!");
    }
}
```

- Running this helps test if your code works before using it in a bigger program.

🔧 Better Testing? Use JUnit

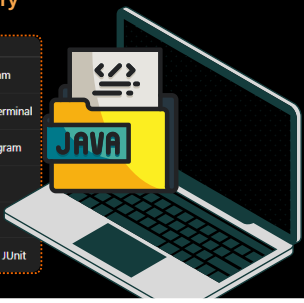
- JUnit is a special tool (called a testing framework).
- It helps write automated tests to check your code.
- It's better than using main just for testing.

🛠️ Tools Tip

- If you use an IDE like Eclipse or IntelliJ, you can:
  - Run your program by pressing a button.
  - Set command-line arguments in the run configuration.

📌 Summary

Concept	What It Means
`main` method	Entry point of your program
`String[] args`	Inputs from the user via terminal
`java ClassName`	Command to run the program
`args[0]`, `args[1]`	First, second arguments
No `main`?	Fine for helper classes
Test code?	Add a quick `main`, or use JUnit



Strings, Wrappers, Arrays, and Enum Types

🔤 Characters and Strings in Java

✅ **char – A Single Character**

- Stores just one text symbol (letter, digit, punctuation, etc.)
- Uses single quotes:

```
char letter = 'A';
```

🌍 **Unicode vs ASCII**

- Java uses Unicode – it can store characters from most world languages.
- ASCII is smaller (English only) and fits into Unicode.

Gabriel Miño V.

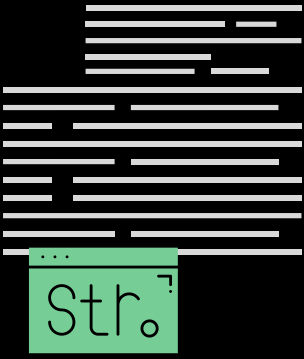
DATA STRUCTURES & ALGORITHMS

🧵 *The String Class – For Text*

✅ What is a String?

- A sequence of characters (could be letters, numbers, spaces, etc.).
- Uses double quotes:

```
String name = "Java";
```



🔍 Indexing Characters

- Java counts from 0 (zero-based index).
- You can get characters like this:

```
String text = "Hello";
text.charAt(1); // 'e'
```

📏 Get String Length

```
text.length(); // returns 5 for "Hello"
```

➕ *String Concatenation*

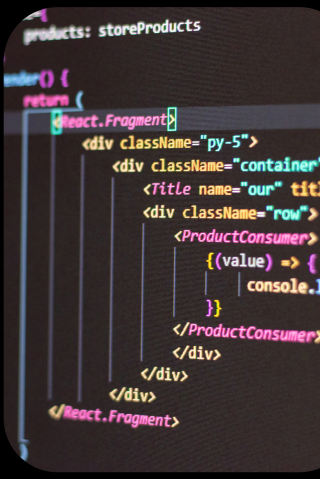
✅ Join Strings Using +

```
String word = "code" + "lab";  // "codelab"
```

🔁 Add to Existing String

```
String greet = "Hi";
greet = greet + "!";  // "Hi!"
```

💡 **Note**: Every + creates a new string in memory. This can be slow for big texts.



🚫 *String Immutability*

- Strings cannot be changed once created.
- But you can reassign:

```
String lang = "Java";
lang = "Python"; // now it points to a new string
```

🧱 *StringBuilder – A Better Way to Build Text*

- Use when you need to edit or build large strings efficiently.

🔧 Common StringBuilder Methods:

Method	What it does
`.append("text")`	Adds text at the end
`.insert(index, "text")`	Inserts text at a specific position
`.setCharAt(index, 'c')`	Changes a character
`.reverse()`	Reverses the text
`.toString()`	Converts back to a normal `String`

- ✅ **Example:**

```java
StringBuilder sb = new StringBuilder("Hi");
sb.append(" there");          // "Hi there"
sb.setCharAt(0, 'B');         // "Bi there"
String result = sb.toString(); // convert to String
```

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

⚠️ Common Mistakes to Avoid

❌ Using the wrong quotes: use 'A' for char, "A" for String

❌ Accessing out-of-range characters:

```
String s = "cat";
s.charAt(10); // ❌ Error!
```

❌ Expecting String to be mutable – use StringBuilder instead

🖍️ Quick Recap

Concept	Description	Example
`char`	One letter	`'X'`
`String`	Sequence of letters	`"Hello"`
`.charAt(index)`	Get letter by position	`"Hi".charAt(1) → 'i'`
`.length()`	Get string length	`"Java".length() → 4`
`+`	Join strings	`"a" + "b" → "ab"`
`StringBuilder`	Editable string	`sb.append("!")`

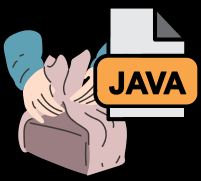
```
}
foreach( $this->rules as $key=>$rule ) {
    if ( $details['role_id'] == $rule['role_id'] &&
        if ( $access == false ) {
            unset( $this->rules[ $key ] );
        } else {
            $this->rules[ $key ]['access'] = $access;
        }
    }
```

🎁 *Wrapper Classes – Turning Primitives into Objects*

Why?

- Some Java structures (like ArrayList, HashMap) only work with objects.
- Java has Wrapper Classes to convert primitive types into objects.

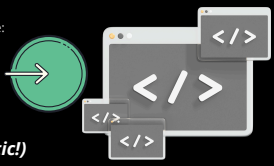
Primitive Type	Wrapper Class	Example
`int`	`Integer`	`Integer num = new Integer(10);`
`double`	`Double`	`Double pi = new Double(3.14);`
`boolean`	`Boolean`	`Boolean flag = new Boolean(true);`
`char`	`Character`	`Character c = new Character('A');`



✅ *Accessing Values*

You can get the original value using methods like:

```
Integer x = new Integer(42);
int num = x.intValue();
```



🔄 *Boxing and Unboxing (Automatic!)*

Java automatically converts between primitives and wrappers.

🔁 Auto-Boxing (primitive ➡️ object)

```java
int a = 5;
Integer b = a;  // auto-boxed
```

🔁 Auto-Unboxing (object ➡️ primitive)

```java
Integer x = new Integer(7);
int y = x;  // auto-unboxed
```

🔧 Other Useful Features

```java
Integer a = new Integer("-135"); // from string
int year = Integer.parseInt("2024"); // string to int
```

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

Arrays – Store Multiple Values

✅ **What's an Array?**
An array holds multiple values of the same type in a single variable.

```
int[] scores = {90, 85, 100};
```

🔢 **Indexing and Length**

- Arrays start at index 0
- Access value: scores[1] → 85
- Array length: scores.length → 3

⚠️ **Watch Out!**
Accessing an invalid index gives:

`ArrayIndexOutOfBoundsException`

🛠️ **Declaring Arrays**

Option 1: Declare & Initialize with Values

```
java
string[] colors = {"Red", "Green", "Blue"};
```

Option 2: Declare & Create with Size

```
java
int[] ages = new int[5];  // fills with 0s
```

Type	Default Value
int, double	0 / 0.0
boolean	false
Objects	null

Enum Types – Limited Value Variables

✅ **What is an enum?**
A type that can have only a specific set of values.

```
java
public enum Day {
  MON, TUE, WED, THU, FRI, SAT, SUN
}
```

🎯 **Why Use Enums?**

- Safer and clearer than just using numbers or strings.
- Avoids invalid values.

✅ enum is like a smart list of constants.

📄 **Using Enums**

```
Day today = Day.WED;
```

Common Mistakes to Avoid

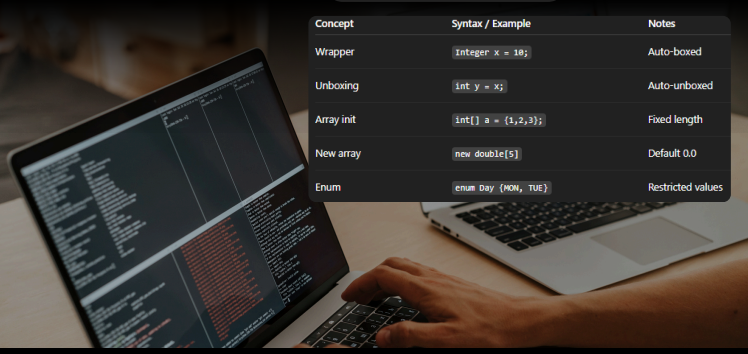
❌ Forgetting .length has no parentheses for arrays: array.length, not array.length()

❌ Thinking arrays grow dynamically – they don't! Use ArrayList for that.

❌ Using int when an enum is better for clarity.

Quick Summary Table

Concept	Syntax / Example	Notes
Wrapper	`Integer x = 10;`	Auto-boxed
Unboxing	`int y = x;`	Auto-unboxed
Array init	`int[] a = {1,2,3};`	Fixed length
New array	`new double[5]`	Default 0.0
Enum	`enum Day {MON, TUE}`	Restricted values



Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

Expressions

🔑 What is an Expression?

An expression in Java is a piece of code that combines values, variables, and operators to produce a new value.
Example:

```
int x = 5 + 3; // expression is 5 + 3
```

🧱 Building Blocks of Expressions

📌 1. Literals (Fixed values written directly in the code)

- Boolean: true, false
- Integer: 100, -20 → Default type: int
  - Add L for long: 100L
- Floating Point: 3.14, 0.5 → Default: double
  - Add F for float: 2.5F
- Character: 'a', '%', '\n'
  - Special characters:
  - '\n' (newline), '\t' (tab), '\\' (backslash), '\"' (double quote), etc.
- String: "Hello World"

➕ Operators in Java

🧮 2. Arithmetic Operators

- + (add), - (subtract), * (multiply), / (divide), % (modulo = remainder)

```
int result = 7 % 3; // result is 1
```

🔗 3. String Concatenation

- Use + to combine strings or strings with numbers.
- Example:

java

```
String msg = "Hi " + "there!"; // "Hi there!"
```

If you use a number in the operation, it will be turned into a string automatically.

java

```
String time = "It took " + 5 + " hours."; // "It took 5 hours."
```

🔍 5. Comparison Operators

Used to compare values (results are true or false):

Operator	Meaning
<	less than
<=	less than or equal
==	equal
!=	not equal
>=	greater or equal
>	greater than

Note:

- For objects, use .equals() to check content.
- Use == only to check if two references point to the same object.

🔁 4. Increment & Decrement

- **++ adds 1**
- **-- subtracts 1**

java

```
int i = 8;
int j = i++; // j = 8, i = 9 (post-increment)
int k = ++i; // i = 10, k = 10 (pre-increment)
```

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

🔵 6. Logical Operators (used with booleans)

Operator	Meaning
`!`	NOT
`&&`	AND

Short-circuiting:

- && stops if the first condition is false.
- || stops if the first condition is true.

🧠 7. Bitwise Operators (advanced, used with integers)

Operator	Meaning
`~`	NOT (invert bits)
`&`	AND (bitwise)
`|`	|
`^`	XOR (exclusive OR)
`<<`	shift left
`>>`	shift right
`>>>`	shift right (unsigned)

📝 8. Assignment Operators

- Basic: =
- Compound: Combine assignment and operation

Operator	Meaning
`+=`	x = x + value
`-=`	x = x - value
`*=`	x = x * value
`/=`	x = x / value
`%=`	x = x % value

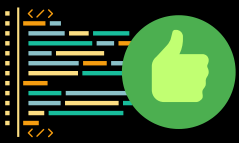
```java
int x = 4;
x *= 2; // x becomes 8
```

❗ Common Beginner Mistakes

- Forgetting to end lines with ;
- Using == instead of .equals() to compare strings or objects
- Using = (assignment) instead of == (comparison)
- Misusing ++i vs i++
- Mixing up float and double without F suffix

✅ Quick Tips

- Always use () to control the order of operations.
- String + anything = String
- Use comments (//) to remind yourself what your expressions do.



Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

Control Flow

🔄 *What is Control Flow?*

Control flow decides which parts of your code run, and when. It's how you make decisions and repeat actions in your program.

✅ *1. If Statements (Making decisions)*

Basic Syntax:

```java
if (condition) {
  // code runs if condition is true
} else {
  // code runs if condition is false
}
```

✅ Conditions must be boolean (`true` or `false`) — not numbers.

EXAMPLE


```java
if (age >= 18) {
  System.out.println("You're an adult.");
} else {
  System.out.println("You're a minor.");
}
```

Else If (Multiple Choices):

```java
if (score >= 90) {
  System.out.println("A");
} else if (score >= 80) {
  System.out.println("B");
} else {
  System.out.println("Try harder!");
}
```

✅ Braces `{}` are optional if only 1 line of code follows, but always using them is safer.

🔁 *2. Nested Ifs (Ifs inside Ifs)*

You can put an `if` inside another `if` to add more checks.

Example:

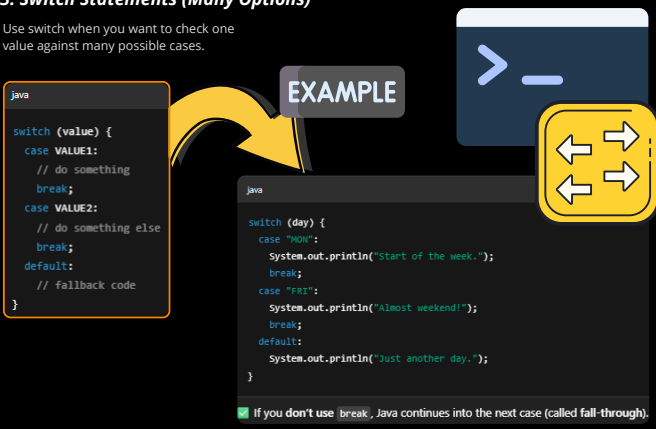

```java
if (door.isClosed()) {
  if (door.isLocked()) {
    door.unlock();
  }
  door.open();
}
advance(); // runs no matter what
```

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

🔀 3. Switch Statements (Many Options)

Use switch when you want to check one value against many possible cases.



```java
switch (value) {
  case VALUE1:
    // do something
    break;
  case VALUE2:
    // do something else
    break;
  default:
    // fallback code
}
```

EXAMPLE

```java
switch (day) {
  case "MON":
    System.out.println("Start of the week.");
    break;
  case "FRI":
    System.out.println("Almost weekend!");
    break;
  default:
    System.out.println("Just another day.");
}
```

✅ If you **don't use** `break`, Java continues into the next case (called **fall-through**).

🌀 4. While Loop

✅ Use when: You don't know how many times to repeat — depends on a condition.

```java
while (condition) {
  // repeat this block while condition is true
}
```

EXAMPLE

```java
int j = 0;
while (j < data.length && data[j] != target) {
  j++;
}
```

✅ Condition is checked BEFORE the first run.
⚠️ May run zero times if the condition is false from the beginning.

🔁 5. Do-While Loop

✅ Use when: You want the loop to run at least once, no matter the condition.

```java
do {
  // run this block
} while (condition);
```

EXAMPLE

```java
String input;
do {
  input = getInputString();
  handleInput(input);
} while (input.length() > 0);
```

✅ Condition is checked AFTER the first run.
💡 Useful for user input validation.

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

6. For Loop (Traditional)

✅ Use when: You know how many times to repeat something (counting loop).

```java
for (initialization; condition; update) {
  // loop body
}
```

EXAMPLE

```java
for (int j = 0; j < n; j++) {
  System.out.println(j);
}
```

➡️ Same as:

```java
int j = 0;
while (j < n) {
  System.out.println(j);
  j++;
}
```

🔹 Real Example – Sum an Array:

```java
public static double sum(double[] data) {
  double total = 0;
  for (int j = 0; j < data.length; j++) {
    total += data[j];
  }
  return total;
}
```

7. For-Each Loop (Enhanced For Loop)

✅ Use when: You want to go through all elements of an array or collection.

```java
for (ElementType variable : arrayOrList) {
  // use variable directly
}
```

EXAMPLE

```java
for (double val : data) {
  total += val;
}
```

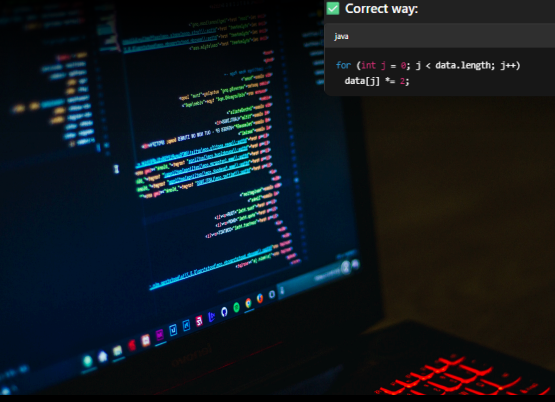
⚠️ You can read values, but not change the original array.

❌ Incorrect way to modify array:

```java
for (double val : data)
  val *= 2; // Only changes the local copy, not the array
```

✅ Correct way:

```java
for (int j = 0; j < data.length; j++)
  data[j] *= 2;
```



Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

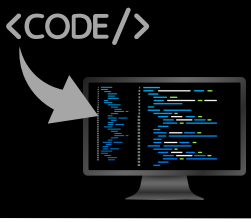
Simple Input and Output

System.out – Console Output

- System.out is used to display output to the console.
- It's an instance of the java.io.PrintStream class.

Common Methods:

Method	Description
`print()`	Prints text without a new line
`println()`	Prints text followed by a new line



Example:

```java
System.out.print("Java values: ");
System.out.print(3.1416);
System.out.print(',');
System.out.print(15);
System.out.println(" (double,char,int).");
```

System.in + Scanner – Keyboard Input

- System.in is used for user input.
- We commonly wrap it in a Scanner object to make input easier.

Import:

```java
import java.util.Scanner;
```

Basic Example:

```java
Scanner input = new Scanner(System.in);
System.out.print("Enter your age: ");
double age = input.nextDouble();
```

Scanner Methods

Reading different data types:

Method	Reads...
`next()`	String (one word)
`nextLine()`	Full line of text
`nextInt()`	Integer
`nextDouble()`	Decimal (double)
`nextBoolean()`	true or false

Input validation example:

```java
System.out.print("Please enter an integer: ");
while (!input.hasNextInt()) {
    input.nextLine(); // clear invalid line
    System.out.print("Invalid input. Try again: ");
}
int number = input.nextInt();
```

Full Program – Heart Rate Calculator

```java
import java.util.Scanner;

public class InputExample {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter your age in years: ");
        double age = input.nextDouble();

        System.out.print("Enter your maximum heart rate: ");
        double rate = input.nextDouble();

        double fb = (rate - age) * 0.65;
        System.out.println("Your ideal fat-burning heart rate is " + fb);
    }
}
```

EXAMPLE

Sample Output:

```yaml
Enter your age in years: 21
Enter your maximum heart rate: 220
Your ideal fat-burning heart rate is 129.35
```

More Scanner Methods

Method	Description
`hasNext()`	Checks if another token is available
`hasNextInt()`	Checks if the next token can be read as an integer
`next()`	Returns the next token
`nextLine()`	Returns the entire line of input
`hasNextLine()`	Checks if there is another line to read
`findInLine(String a)`	Looks for a pattern match using a regular expression

Tokens and Delimiters

- A token is a piece of text separated by whitespace.
- Example: "123 hello 9.8" has 3 tokens → 123, hello, 9.8.

Gabriel Miño V.

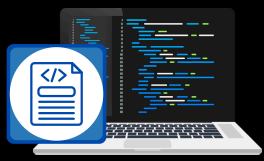
DATA STRUCTURES & ALGORITHMS

An Example Program

🎯 *Purpose:*

To model a simplified credit card system, showing key OOP principles in Java:

- Class design
- Constructors
- Instance & static methods
- Encapsulation
- Arrays of objects
- Method invocation
- Control structures (loops)



◆ *Class Declaration*

```java
public class CreditCard {
```

➡ Declares a public class named CreditCard.

```java
public class CreditCard {
    // Instance variables:
    private String customer;
    private String bank;
    private String account;
    private int limit;
    protected double balance;

    // Constructors:
    public CreditCard(String cust, String bk, String acnt, int lim, double initialBal) {
        customer = cust;
        bank = bk;
        account = acnt;
        limit = lim;
        balance = initialBal;
    }

    public CreditCard(String cust, String bk, String acnt, int lim) { // Partial Constructor
        this(cust, bk, acnt, lim, 0.0); // use a balance of zero as default
    }

    // Accessor methods:
    public String getCustomer() { return customer; }
    public String getBank() { return bank; }
    public String getAccount() { return account; }
    public int getLimit() { return limit; }
    public double getBalance() { return balance; }

    // Update methods:
    public boolean charge(double price) { // make a charge
        if (price + balance > limit) // if charge would surpass limit
            return false;            // refuse the charge
        balance += price;            // update the balance
        return true;                 // announce the good news
    }

    public void makePayment(double amount) { // make a payment
        balance -= amount;
    }

    // Utility method to print a card's information
    public static void printSummary(CreditCard card) {
        System.out.println("Customer = " + card.customer);
        System.out.println("Bank = " + card.bank);
        System.out.println("Account = " + card.account);
        System.out.println("Balance = " + card.balance);
        System.out.println("Limit = " + card.limit);
    }

    // main method
    public static void main(String[] args) {
        CreditCard[] wallet = new CreditCard[3];
        wallet[0] = new CreditCard("John Bowman", "California Savings", "5391 0375 9387 5309", 5000);
        wallet[1] = new CreditCard("John Bowman", "California Federal", "3485 0399 3395 1954", 3500);
        wallet[2] = new CreditCard("John Bowman", "California Finance", "6011 0996 2142 7986", 9500, 300);

        for (int val = 1; val <= 16; val++) {
            wallet[0].charge(3 * val);
            wallet[1].charge(2 * val);
            wallet[2].charge(val);
        }

        for (CreditCard card : wallet) {
            CreditCard.printSummary(card); // calling static method
            while (card.getBalance() > 200.0) {
                card.makePayment(200);
                System.out.println("New balance = " + card.getBalance());
            }
        }
    }
}
```

◆ *Instance Variables*

Modifier	Variable	Description
private	customer	Name of the card owner
private	bank	Issuing bank
private	account	Account number
private	limit	Credit limit in dollars
protected	balance	Current balance (used later in inheritance)

➡ These variables store information about each credit card:

- customer: the name of the cardholder
- bank: the name of the issuing bank
- account: the card number (as a String)
- limit: maximum allowed balance
- balance: current amount owed (protected so child classes can access it)

◆ *Constructors*

```java
public CreditCard(String cust, String bk, String acnt, int lim, double initialBal)
    ...
}
```

➡ This constructor is used when all five values are provided, including the initial balance.

◆ *Partial Constructor*

```java
public CreditCard(String cust, String bk, String acnt, int lim) {
    this(cust, bk, acnt, lim, 0.0);
}
```

➡ This constructor only takes 4 values and calls the full constructor with a default balance of 0.0 using this().

◆ *Accessor Methods*

```java
public String getCustomer()  // returns customer
public String getBANK()      // returns bank
public String getAccount()   // returns account
public int getLimit()        // returns limit
public double getBalance()   // returns balance
```

These methods return values from the object:

◆ *Mutator Methods (Update Methods)*

```java
public boolean charge(double price)
```

➡ Tries to add a charge to the card:

- If the new balance would exceed the limit, it returns false (rejected).
- Otherwise, it updates the balance and returns true.

```java
public void makePayment(double amount)
```

➡ Subtracts a payment from the balance.

📄 *Utility Method*

```java
public static void printSummary(CreditCard card) {
    System.out.println("Customer = " + card.customer);
    System.out.println("Bank = " + card.bank);
    System.out.println("Account = " + card.account);
    System.out.println("Balance = " + card.balance);
    System.out.println("Limit = " + card.limit);
}
```

➡ Prints all the important details of the card. It is static, so you call it with the class name like CreditCard.printSummary(...).

◆ *Create an Array of CreditCards*

```java
CreditCard[] wallet = new CreditCard[3];
wallet[0] = new CreditCard(...);
wallet[1] = new CreditCard(...);
wallet[2] = new CreditCard(..., 300); // with initial balance
```

➡ Creates an array called wallet to store 3 CreditCard objects, each initialized with different data.

◆ *Charge Each Card in a Loop*

```java
for (int val = 1; val <= 16; val++) {
    wallet[0].charge(3 * val);
    wallet[1].charge(2 * val);
    wallet[2].charge(val);
}
```

➡ Simulates charges:

- Card 0: adds 3×1, 3×2, ..., 3×16
- Card 1: adds 2×1, 2×2, ..., 2×16
- Card 2: adds 1, 2, ..., 16

◆ *Charge Each Card in a Loop*

```java
for (CreditCard card : wallet) {
    CreditCard.printSummary(card);
    while (card.getBalance() > 200.0) {
        card.makePayment(200);
        System.out.println("New balance = " + card.getBalance());
    }
}
```

➡ For each card in the wallet:

- Prints the summary.
- If the balance is over $200, it makes payments of $200 until it is below $200.
- After each payment, it prints the new balance.

💻 *Sample Output*

```
Customer = John Bowman
Bank = California Savings
Account = 5391 0375 9387 5309
Balance = 408.0
Limit = 5000
New balance = 208.0
New balance = 8.0
...
```

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

Packages and Imports

What is a Package?

A package is a way to group related Java classes together, just like putting similar files into the same folder.

✅ **Why use packages?**

- To organize code better.
- To avoid name conflicts. (You can have multiple Window classes in different packages.)
- To share code easily with other developers.

Declaring a Package

At the top of your .java file, you declare the package like this:

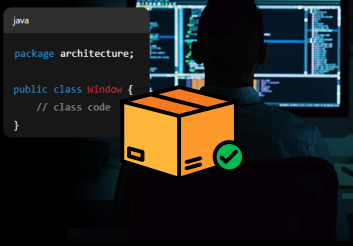
```
package architecture;
```

➡ This tells Java that your class belongs to the architecture package.

- The file must be inside a folder named architecture.

Package Example

- If we create a file Window.java and write:
- That file must be saved in a folder called architecture.



```java
package architecture;

public class Window {
    // class code
}
```

Subpackages

Packages can have subfolders, called subpackages.
Example:

- java.util is a package
- java.util.zip is a subpackage

So to use a class like Deflater, you'd write:

```java
java.util.zip.Deflater def = new java.util.zip.Deflater();
```

Fully Qualified Names

- You can refer to any class using its full name (called the fully qualified name).
- For example:
- But this is long... that's where import helps!

```java
java.util.Scanner input = new java.util.Scanner(System.in);
```

Using import in Java

Instead of writing the full name every time, you can import a class:

```
import java.util.Scanner;
```

Now you can write:

```
Scanner input = new Scanner(System.in);
```

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

🔸 *Importing Whole Packages*

If you're using many classes from the same package:
➡ This lets you use any class from java.util without writing the full name.

```
import java.util.*;
```

❗ *Import Conflicts*

- Java doesn't allow two classes with the same name from different packages to be imported without qualification.

```java
import architecture.*;
import gui.*; // both have a Window class
```

Now you **must write**:

```java
architecture.Window myWin1 = new architecture.Window();
gui.Window myWin2 = new gui.Window();
```

You **cannot** just write:

```java
Window myWin; // ❌ Java won't know which one you mean
```

🟣 *Default Package*

- If you don't write package something; at the top, your class goes into the default package (no name).
- But this is not recommended for large projects.

✅ *Summary Table*

Concept	Example	Purpose
Package declaration	`package myapp;`	Groups related classes
Import one class	`import java.util.Scanner;`	Lets you write `Scanner` instead of `java.util.Scanner`
Import all in package	`import java.util.*;`	Lets you use all classes in `java.util`
Fully qualified name	`java.util.Scanner`	Avoids import but is longer
Conflict resolution	Use full names (e.g., `gui.Window`)	Avoids ambiguity between same-named classes

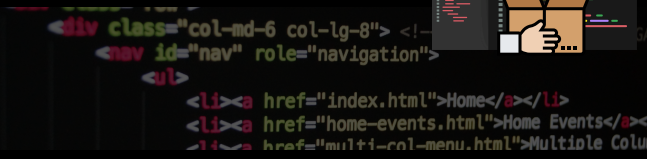
Software Development

📌 *3 Main Steps in Software Development*

1- Design – Plan how your program will work.

2- Coding – Write the actual Java code.

3- Testing & Debugging – Run the code and fix errors.



Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

🧱 *Step 1: Design (The Most Important Step)*

Goal: Decide what your program needs to do and how to organize it using classes.

Key Concepts:

- Class: A blueprint for objects. It holds data (variables) and actions (methods).
- Object-Oriented Programming (OOP): Break your program into small parts (classes).

💡 Design Tips:

- Responsibilities: What will each class do? Use action verbs like CalculateTotal, StoreData, etc.
- Independence: Each class should work mostly on its own.
- Behaviors: Define what each class can do (methods).

🛠️ Tools:

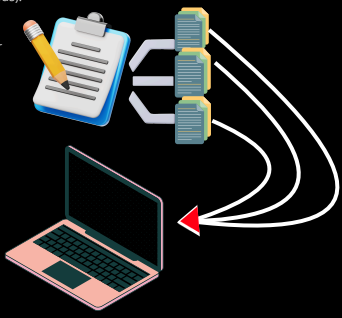
- CRC Cards = Class + Responsibility + Collaborator
- Example:

- **CRC Cards** = Class + Responsibility + Collaborator
  Example:

```
Class: CreditCard
Responsibilities: store info, make charges, return balance
Collaborators: none or other classes it talks to
```

- **UML Diagrams:** Visual maps of your classes.
  - Example:

```
+ getBalance() : double  // public method
- customer : String      // private variable
```



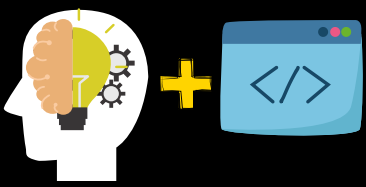
✍️ *Step 2: Pseudocode (Plan Before You Code)*

- **Pseudocode** = A mix of English + code-like instructions.
- Helps you think like a programmer before writing real Java code.

📄 Pseudocode Examples:

```
if age >= 18 then
   print "You can vote"
else
   print "Too young"
```

```
while not atEndOfFile do
   read line
   print line
```



💻 *Step 3: Coding in Java*

🧱 Build Your Program:

1. Use an editor (like Eclipse or VS Code).
2. Write your .java files with classes, variables, and methods.
3. Compile using javac MyFile.java.
4. Run using java MyFile.

⚠️ Common Errors:

- Missing semicolons ;
- Wrong variable types
- Not compiling before running

🛠️ Example:

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

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

📝 *Documentation and Coding Style*

Use Javadoc to explain your code and generate documentation.

📘 Javadoc Example:

```java
/**
 * Calculates the total price after tax.
 * @param price The original price.
 * @return The final price including tax.
 */
public double calculateTotal(double price) {
    return price * 1.12;
}
```

📕 Common Tags:

- @author
- @param – describe input
- @return – describe output
- @throws – describe possible errors

✅ *Java Coding Best Practices*

🗂️ Naming Rules

- Use clear names: customerName, isFull()
- Classes: Capitalize every word → CreditCard, StudentAccount
- Methods & variables: First word lowercase → chargeCard(), creditLimit
- Constants: All UPPERCASE → MAX_CREDITS

📏 Indentation

- Use 2–4 spaces to indent code for readability.

📦 Code Structure Order

1. Constants
2. Variables
3. Constructors
4. Methods

💬 Comments

- Use // for short notes
- Use /** */ for method descriptions

🧪 *Testing & Debugging*

🧪 Testing = Check if program works

- Test every method at least once.
- Include special cases, like:
  - Empty list
  - Only one element
  - All items the same
  - Sorted / reverse sorted lists

✅ Testing Strategies

- Top-down: Start testing high-level methods first.
- Bottom-up: Test small, low-level methods first (best for finding bugs).

💡 Example:
Test payment with 0 balance or over-limit charge.

🛠️ Automated Testing

- Use a class's main() method to test it.
- For more advanced tests, use JUnit (external tool).

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

Exercises

✅*Reinforcement*

R-1.1 *Write a short Java method, inputAllBaseTypes, that inputs a different value of each base type from the standard input device and prints it back to the standard output device*

```java

public class BaseTypeInput {

    public static void inputAllBaseTypes() {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a byte: ");
        byte b = sc.nextByte();

        System.out.print("Enter a short: ");
        short s = sc.nextShort();

        System.out.print("Enter an int: ");
        int i = sc.nextInt();

        System.out.print("Enter a long: ");
        long l = sc.nextLong();

        System.out.print("Enter a float: ");
        float f = sc.nextFloat();

        System.out.print("Enter a double: ");
        double d = sc.nextDouble();

        System.out.print("Enter a boolean (true/false): ");
        boolean bool = sc.nextBoolean();

        System.out.print("Enter a char: ");
        char c = sc.next().charAt(0);

        System.out.println("\nYou entered:");
        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("int: " + i);
        System.out.println("long: " + l);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
        System.out.println("boolean: " + bool);
        System.out.println("char: " + c);
    }

    public static void main(String[] args) {
        inputAllBaseTypes();
    }
}
```

✅ ***R-1.1 – Java Method: inputAllBaseTypes***

Write a Java method that reads one value of each primitive base type from the user and then prints them.

R-1.2 *Suppose that we create an array A of GameEntry objects, which has an integer scores field, and we clone A and store the result in an array B. If we then immediately set A[4].score equal to 550, what is the score value of the GameEntry object referenced by B[4]?*

```java
class GameEntry {
    int score;

    public GameEntry(int s) {
        score = s;
    }
}

public class Main {
    public static void main(String[] args) {
        GameEntry[] A = new GameEntry[5];
        for (int i = 0; i < A.length; i++) {
            A[i] = new GameEntry(i * 100);
        }

        GameEntry[] B = A.clone();  // Shallow copy

        A[4].score = 550;

        System.out.println("B[4].score: " + B[4].score); // Output: 550
    }
}
```

✅ ***R-1.2 – Array Cloning with Objects***

Answer:
The value of B[4].score will be 550.

Why?

- Cloning an array of objects in Java only copies the references, not the actual objects.
- So A[4] and B[4] point to the same object in memory.
- Changing A[4].score changes the same object B[4] refers to.

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

R-1.3 *Write a short Java method, isMultiple, that takes two long values, n and m, and returns true if and only if n is a multiple of m, that is, n = mi for some integer i.*

java

```java
public class MathUtils {

    public static boolean isMultiple(long n, long m) {
        if (m == 0) return false; // avoid division by zero
        return n % m == 0;
    }

    public static void main(String[] args) {
        System.out.println(isMultiple(10, 2));  // true
        System.out.println(isMultiple(10, 3));  // false
    }
}
```

✅ ***R-1.3 – Method isMultiple***

Goal:
Check if n is a multiple of m, i.e., n = m * i for some integer i.

R-1.4 *Write a short Java method, isEven, that takes an int i and returns true if and only if i is even. Your method cannot use the multiplication, modulus, or division operators, however.*

java

```
i & 1 == 0 → even
i & 1 == 1 → odd
```

java

```java
public class MathUtils {

    public static boolean isEven(int i) {
        return (i & 1) == 0;
    }

    public static void main(String[] args) {
        System.out.println(isEven(4));  // true
        System.out.println(isEven(7));  // false
    }
}
```

✅ ****R-1.4 – Method isEven (No , /, or % allowed)***

Goal:
Check if a number is even without using multiplication (*), division (/), or modulus (%) operators.

Hint:
Use bitwise operator:
Even numbers have their least significant bit as 0.

R-1.5 *Write a short Java method that takes an integer n and returns the sum of all positive integers less than or equal to n.*

java

```java
public static int sumUpTo(int n) {
    if (n <= 0) return 0;
    return n * (n + 1) / 2;
}
```

✅ ***R-1.5 – Sum of all positive integers less than or equal to n***

Goal:
Return the sum 1 + 2 + 3 + ... + n.

🧠 **Logic:**
You can use either a loop or a simple math formula:
Formula:
sum = n * (n + 1) / 2

R-1.6 *Write a short Java method that takes an integer n and returns the sum of all the odd positive integers less than or equal to n.*

java

```java
public static int sumOddUpTo(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i += 2) {
        sum += i;
    }
    return sum;
}
```

✅ ***R-1.6 – Sum of all odd positive integers less than or equal to n***

Goal:
Return the sum 1 + 3 + 5 + ... up to n (if n is odd).

🧠 **Logic:**
Loop from 1 to n, incrementing by 2 to only get odd numbers.

Gabriel Miño V.

DATA STRUCTURES & ALGORITHMS

R-1.7 *Write a short Java method that takes an integer n and returns the sum of the squares of all positive integers less than or equal to n.*

```java
public static int sumSquaresUpTo(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i * i;
    }
    return sum;
}
```

✅ *R-1.7 – Sum of the squares of all positive integers less than or equal to n*

Goal:
Return the sum $1^2 + 2^2 + 3^2 + \ldots + n^2$

🧠 **Logic:**
Use a loop to add up squares of numbers.
Optional formula (if you want it faster):
sum = n(n + 1)(2n + 1) / 6

R-1.8 *Write a short Java method that counts the number of vowels in a given character string.*

```java
public static int countVowels(String str) {
    int count = 0;
    String vowels = "aeiouAEIOU";

    for (int i = 0; i < str.length(); i++) {
        if (vowels.indexOf(str.charAt(i)) != -1) {
            count++;
        }
    }
    return count;
}
```

✅ *R-1.8 – Count the number of vowels in a string*

Goal:
Create a method that receives a String and returns the number of vowels (a, e, i, o, u, case-insensitive).

R-1.9 *Write a short Java method that uses a StringBuilder instance to remove all the punctuation from a string s storing a sentence, for example, transforming the string "Let's try, Mike!" to "Lets try Mike".*

java Copiar

```java
public static String removePunctuation(String s) {
    StringBuilder result = new StringBuilder();

    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (Character.isLetterOrDigit(c) || Character.isWhitespace(c)) {
            result.append(c);
        }
    }

    return result.toString();
}
```

✅ *R-1.9 – Remove all punctuation using StringBuilder*

Goal:
Remove all punctuation from a sentence, keeping only letters, digits, and spaces.

R-1.10 *Write a Java class, Flower, that has three instance variables of type String, int, and float, which respectively represent the name of the flower, its number of petals, and price. Your class must include a constructor method that initializes each variable to an appropriate value, and your class should include methods for setting the value of each type, and getting the value of each type.*

```java
import java.util.Scanner;

public class BaseTypesInput {

    public static void inputAllBaseTypes() {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a byte: ");
        byte b = sc.nextByte();

        System.out.print("Enter a short: ");
        short s = sc.nextShort();

        System.out.print("Enter an int: ");
        int i = sc.nextInt();

        System.out.print("Enter a long: ");
        long l = sc.nextLong();

        System.out.print("Enter a float: ");
        float f = sc.nextFloat();

        System.out.print("Enter a double: ");
        double d = sc.nextDouble();

        System.out.print("Enter a boolean (true/false): ");
        boolean bool = sc.nextBoolean();

        System.out.print("Enter a char: ");
        char c = sc.next().charAt(0);

        System.out.println("\nYou entered:");
        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("int: " + i);
        System.out.println("long: " + l);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
        System.out.println("boolean: " + bool);
        System.out.println("char: " + c);
    }

    public static void main(String[] args) {
        inputAllBaseTypes();
    }
}
```

```java
public static void main(String[] args) {
    Flower rose = new Flower("Rose", 32, 5.99f);

    System.out.println("Name: " + rose.getName());
    System.out.println("Petals: " + rose.getPetalCount());
    System.out.println("Price: $" + rose.getPrice());

    // Update values
    rose.setPrice(6.49f);
    System.out.println("Updated Price: $" + rose.getPrice());
}
```

✅ *R-1.10 – Flower class with instance variables and getters/setters*

Goal:
Create a Flower class with:

- name (String)
- petalCount (int)
- price (float)

Include:

- A constructor to initialize all fields
- Getter and setter methods for each field

Gabriel Miño V.