Fundamental Programming Structures

Fundamental Programming Structures

- What is java?
  - Introduction:
    - Created in 1995, Java is a prominent programming language.
  - Ownership:
    - Oracle currently owns Java.
  - Ubiquity:
    - Over 3 billion devices globally run Java.
  - Applications:
  - Used for diverse purposes, including:
    - Mobile applications (particularly Android apps)
    - Desktop applications
    - Web applications
    - Web servers and application servers
    - Games
    - Database connection
  - Versatility:
    - Widely employed across different domains and industries.

- Why use java?
  - Versatile Cross-Platform Language:
    - Java seamlessly operates on diverse platforms.
  - Global Demand and Popularity:
    - Highly sought after in the job market; globally popular.
  - Simplicity and Accessibility:
    - Easy to learn, use, and favored for its simplicity.
  - Open-Source Security and Speed:
    - Free, open-source, and known for robust security and speed.
  - Strong Community and Object-Oriented Structure:
    - Massive developer community support; object-oriented for clear, reusable code.

Fundamental Programming Structures

- **Java syntax: #Main.java**

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

Output: Hello World!

- All Java code must be contained within a **class**, and in our illustration, we designated the **class** as "**Main**."
- It's a convention for **class names** to begin with an **uppercase** letter.
- Keep in mind that Java is **case-sensitive**; "**MyClass**" and "**myclass**" are distinct.
- The Java file's name should correspond to the class name.
- When saving, ensure the **filename** aligns with the **class name** and appends "**.java**" at the end.

Outline

- **Whitespace, Identifiers, Literals, Comments, Separators, Keywords:**
  - Code formatting and structure essentials.
- **Data Types and Conversion:**
  - Defining data and changing types if needed.
- **Variables**:
  - Memory spaces for data storage.
- **Constants**:
  - Fixed, unchanging values in the program.
- **Operators**:
  - Tools for arithmetic, comparison, and logical operations.
- **Strings**:
  - Sequences of characters.
- **Control Structures:**
  - Decision-making and flow control.
- **Loops**:
  - Repeating code execution.
- **Methods**:
  - Reusable blocks of code with parameters and return values.
- **Arrays**:
  - Ordered collections of elements with indexing.

Whitespace, Identifiers, Literals, Comments, Separators, and Keywords

- **Whitespace**
  - Spaces and tabs used for formatting and visual clarity.
- Identifiers
  - Names given to variables, methods, etc., adhering to naming conventions.
- Literals
  - Represent fixed values like numbers or strings directly in the code.

```java
public class SimpleProgram {
    public static void main(String[ ] args) {
        // Whitespace for clarity
        int number1 = 5;
        int number2 = 10;
        // Identifiers
        String greeting = "Hello, ";
        String name = "John";
        // Literals
        int age = 25;
        char firstLetter = 'A';
        // Output using identified variables
        System.out.println(greeting + name);
        System.out.println("Age: " + age);
        System.out.println("First Letter: " + firstLetter);
        // Mathematical operation with identified variables
        int sum = number1 + number2;
        System.out.println("Sum: " + sum);
    }
}
```

Whitespace, Identifiers, Literals, Comments, Separators, and Keywords

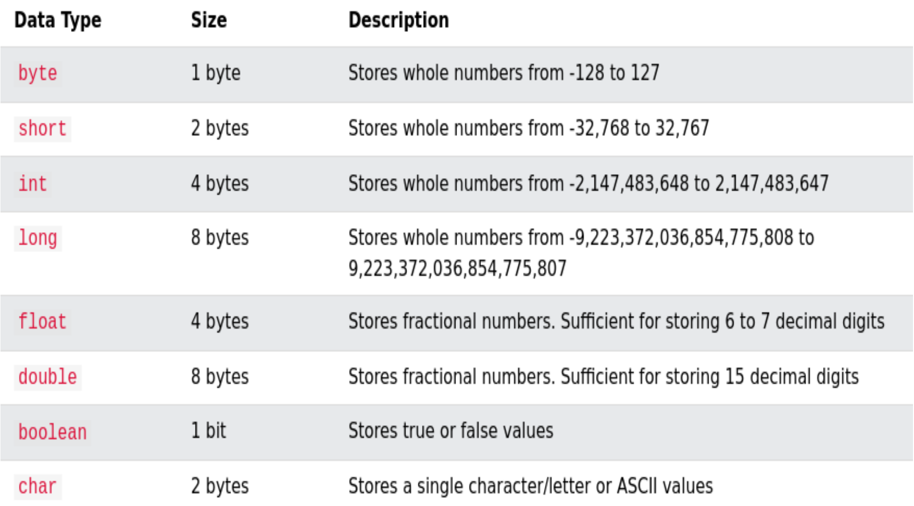
- Comments
  - Annotations for documentation or clarification, not affecting code execution.
- Separators
  - A separator is a symbol or character used to differentiate and structure elements in the code.
- Keywords
  - Reserved words that organize code and convey specific meanings to the compiler.

```
public class CommentSeparatorKeywordExample {
    public static void main(String[ ] args) {
        // This is a single-line comment
        /*This is a
         multi-line comment */
        int number1 = 5, number2 = 10; // 'int' is a keyword
        String greeting = "Hello, "; // 'String' is a keyword
        // Concatenate strings using '+'
        String message = greeting + "World";
        System.out.println(message);
        // Mathematical operation with identified variables
        int sum = number1 + number2;
        System.out.println("Sum: " + sum);
    }  // ;, :, {}, (), and , etc are separator
}
```

Data Types and Conversion

- **Data Types:**
  - In Java, data types define the type of data a variable can hold.
- Data types are divided into two groups:
- **Primitive** data types -
  - byte, short, int, long, float, double, boolean and char
- **Non-primitive** data types
  - String, Arrays and Classes

- A primitive data type specifies the size and type of variable values, and it has no additional methods.
- There are eight primitive data types in Java:



Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Data Types and Conversion

- **Data Types:**
  - In Java, data types define the type of data a variable can hold.
- Data types are divided into two groups:
- **Primitive** data types -
  - byte, short, int, long, float, double, boolean and char
- **Non-primitive** data types
  - String, Arrays and Classes

```
public class Main {
 public static void main(String[ ] args) {
  int myNum = 5;              // integer (whole number)
  float myFloatNum = 5.99f;   // floating point number
  char myLetter = 'D';        // character
  boolean myBool = true;      // boolean
  String myText = "Hello";    // String
  System.out.println(myNum);
  System.out.println(myFloatNum);
  System.out.println(myLetter);
  System.out.println(myBool);
  System.out.println(myText);
 }
}
```

Data Types and Conversion

- Conversion (Type Casting):
- Type casting is when you assign a value of one primitive data type to another type.
- Types of casting:
  - Widening Casting (automatically) - converting a smaller type to a larger type size
    - **byte -> short -> char -> int -> long -> float -> double**
  - Narrowing Casting (manually) - converting a larger type to a smaller size type
    - **double -> float -> long -> int -> char -> short -> byte**

```
public class Main {
  public static void main(String[ ] args) {
    int myInt = 9;
 double myDouble = myInt;
// Automatic casting: int to double
    System.out.println(myInt);      // Outputs 9
   System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Data Types and Conversion

- What are the two groups into which primitive number types are divided, and what types are included in each group?
- What are the main difference between primitive and non-primitive data types?
- Implement the concept of narrowing type casting as discussed in class.

Variables

- Variables are containers for storing data values.
- To create a variable, you must specify the type and assign it a value:
  - ***type variableName = value;***
- *All Java variables must be identified with unique names.*
- *These unique names are called identifiers.*

```
public class Main {
  public static void main(String[ ] args) {
    String name = "John";
     int x = 5, y = 6, z = 50;
    System.out.println(x + y + z);
    System.out.println(name);
  }
}
```

Variables

- *The general rules for naming variables are:*
  - *Names can contain letters, digits, underscores, and dollar signs*
  - *Names must begin with a letter*
  - *Names should start with a lowercase letter and it cannot contain whitespace*
  - *Names can also begin with $ and _ (but we will not use it in this tutorial)*
  - *Names are case sensitive ("myVar" and "myvar" are different variables)*
  - *Reserved words (like Java keywords, such as int or boolean) cannot be used as names*

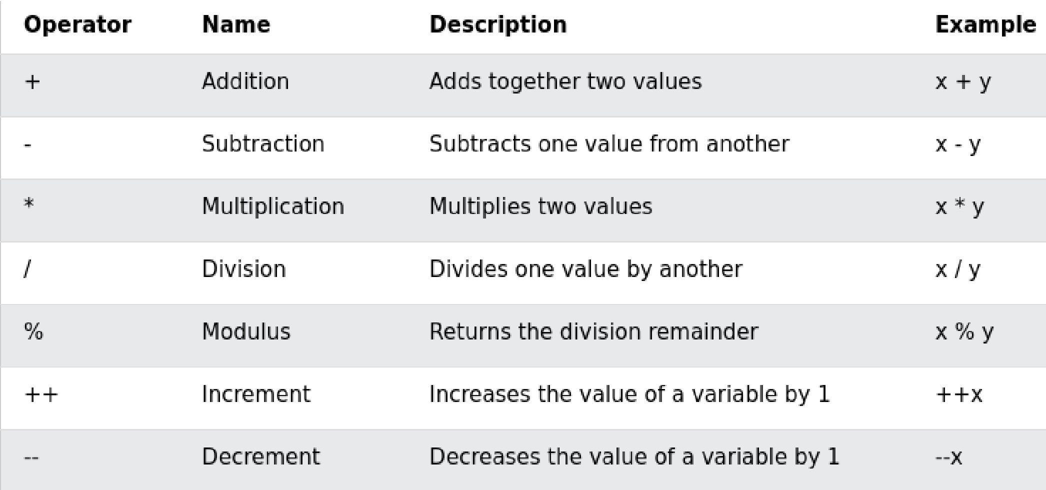
```
public class Main {
  public static void main(String[ ] args) {
    String name = "John";
     int x = 5, y = 6, z = 50;
    System.out.println(x + y + z);
    System.out.println(name);
  }
}
```

Operators

- *Operators are used to perform operations on variables and values.*
- *Java divides the operators into the following groups:*
  - *Arithmetic operators*
  - *Assignment operators*
  - *Comparison operators*
  - *Logical operators*
  - *Bitwise operators*

Operators

- *Arithmetic Operators:*
  - *Arithmetic operators are used to perform common mathematical operations.*

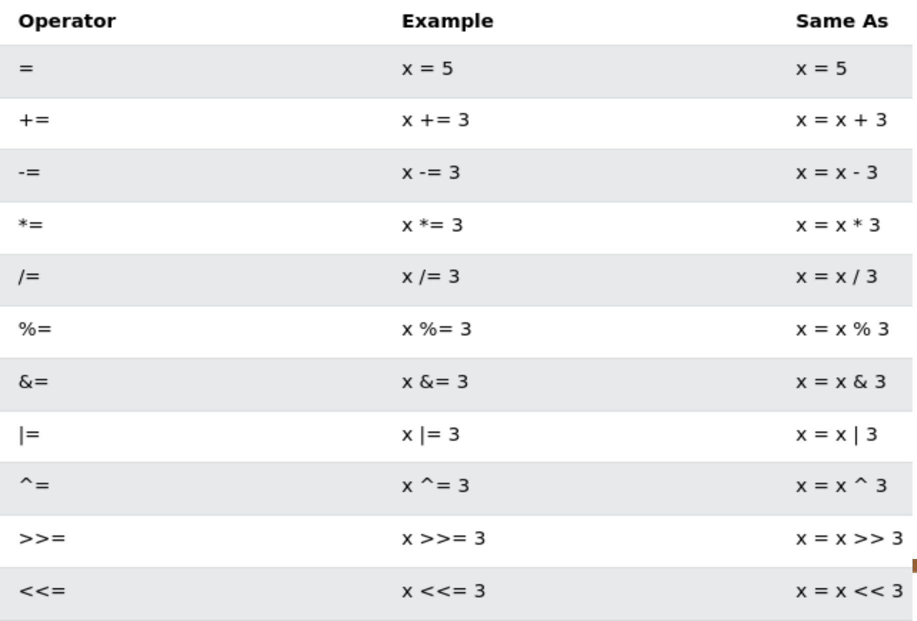


Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

```
public class Main {
  public static void main(String[ ] args) {
    int sum = 100 + 50;
    int diff= sum - 250;
    float div = sum / diff;
    System.out.println(sum);
    System.out.println(diff);
    System.out.println(div);
  }
}
```

Operators

- *Assignment Operators:*
  - *Assignment operators are used to assign values to variables.*

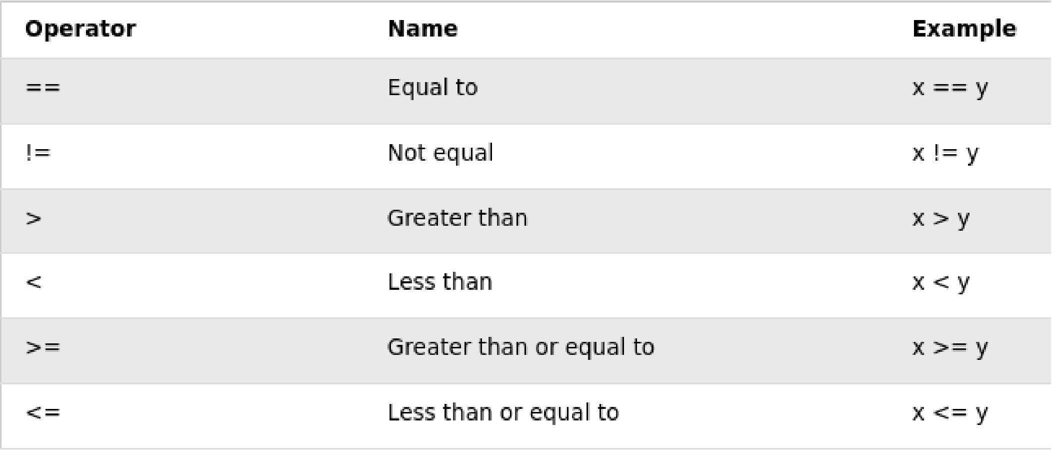


Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

```
public class Main {
  public static void main(String[ ] args) {
    int x = 10;
    x += 5;
    System.out.println(x);
  }
}
```

Operators

- *Comparison Operators:*
  - *Comparison operators are used to compare two values (or variables).*



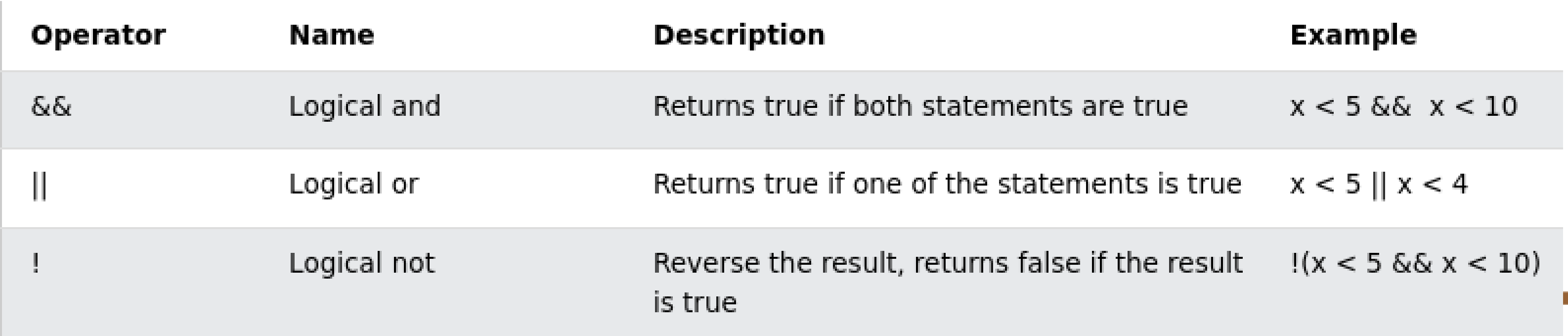
Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

```
public class Main {
  public static void main(String[ ] args) {
    int x = 5;
    int y = 3;
    System.out.println(x > y);
// returns true, because 5 is higher
than 3
  }
}
```

Operators

- *Logical Operators:*
  - *You can also test for true or false values with logical operators.*

```
public class Main {
  public static void main(String[ ] args) {
    int x = 5;
    System.out.println(x > 3 && x < 10); // returns true because 5 is greater than 3 AND 5 is less than 10
  }
}
```



Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Strings

- *In Java, a* ***String*** *is a* ***class*** *that represents a* ***sequence of characters.***
- *It is one of the most commonly used classes in Java and is part of the java.lang package, which is automatically imported into all Java programs*
- *Strings in Java are* ***immutable****, meaning their values cannot be changed once they are created.*

```
public class StringExample {
    public static void main(String[] args) {
        // Creating strings
        String str1 = "Hello";
        String str2 = new String("World");

        // Concatenation
        String greeting = str1 + " " + str2;
        System.out.println("Concatenated String: " + greeting);
        // String length
        int length = greeting.length();
        System.out.println("Length of the String: " + length);
        // IndexOf
        int indexOfWorld = greeting.indexOf("World");
        System.out.println("Index of 'World': " + indexOfWorld);
        // Uppercase and lowercase
        String upperCaseString = greeting.toUpperCase();
        String lowerCaseString = greeting.toLowerCase();
        System.out.println("Uppercase: " + upperCaseString);
        System.out.println("Lowercase: " + lowerCaseString);
    }
}
```

Strings

- Write short notes on escape sequences in java with example.
- Implement the following concepts of strings in java
  - Finding a Character in a String
  - String Concatenation
  - Adding Numbers and Strings
  - Strings - Special Characters