1 Introduction to Java Programming

This document summarizes a tutorial on starting to code with Java. The tutorial covers installing necessary software and writing a basic "Hello, World!"-style program.

1.1 Setting up your Java Development Environment

To begin coding in Java, you need two key components:

- **Java Development Kit (JDK)**: This contains the compiler, which translates your code into machine-readable bytecode. Download the JDK from Oracle (link to be provided in the original video's description). The tutorial uses JDK 23 as an example. The specific version may vary.
- **Integrated Development Environment (IDE)**: This provides a workspace for writing and running code. The tutorial recommends IntelliJ IDEA Community Edition (free version).

The tutorial details the installation process for both the JDK (including selecting the appropriate installer for your operating system) and IntelliJ IDEA. It also notes the option to update path variables, but it's not necessary for this specific tutorial.

1.2 Creating Your First Java Program

After installing the JDK and IDE, the tutorial guides you through creating a new project in IntelliJ IDEA. Key steps include:

- Creating a new project (e.g., "My First Project").
- Selecting the appropriate JDK version.
- Creating a main Java file (e.g., "Main.java").
- Adjusting font size in IntelliJ IDEA settings (using `Control` + mouse wheel).

The core of the program is the `main` method:

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

This method is essential for running Java code. The tutorial explains that while the meaning of the keywords will be covered later, it's a necessary component for program execution.

To print output to the console, the tutorial uses:

```
System.out.println("I like pizza");
```

This line prints the text "I like pizza" to the console. Running the program (via the green arrow or `Shift` + `F10`) will show this output, confirming successful program execution. Error handling is briefly touched on; a successful run shows "processed finished with exit code zero".

"'latex

2 First Java Program

2.1 Basic Output

- Using `system.out.print` displays output on the same line. For example: `system.out.print("I like pizza");`
- Using `system.out.println` (or `print Ln` which is an escape sequence) moves the output to the next line.
- Comments are added using `//` for single-line comments and `/* ... */` for multi-line comments. These are ignored by the compiler.

2.2 Console Customization

- In IntelliJ, you can customize the console font and colors under `File` → `Settings` → `Color Scheme` → `Console Font`.

2.3 Shortcuts

- Typing `sout` and pressing Tab in IntelliJ auto-generates a `system.out.println` statement.

2.4 Homework

Post three `println` statements (e.g., a poem or song lyrics) in the comments section.

3 Variables in Java

3.1 Introduction to Variables

A variable is a reusable container for a value. Variables can hold numbers, characters, and words. There are different data types, including:

- **Integers (`int`):** Whole numbers.
- **Doubles (`double`):** Numbers with decimals.

3.2 Variable Types

There are two categories of variables:

- **Primitive variables:** Simple values stored directly in memory (stack).
- **Reference variables:** Hold memory addresses (stack) that point to locations in the heap. Think of it like an IOU.

3.3 Creating Variables

Creating a variable involves two steps:

1. **Declaration:** Specify the data type and variable name (e.g., `int age;`).
2. **Assignment:** Assign a value to the variable (e.g., `age = 21;`).

3.4 Integer Example

The code `int age = 21;` declares an integer variable named `age` and assigns it the value 21. `system.out.println(age);` would then print 21 to the console.

Summary: The text explains the basics of creating a simple Java program, including printing to the console, using comments, and customizing the console appearance. It then introduces the concept of variables in Java, differentiating between primitive and reference types, and illustrating how to declare and assign values to integer variables. "'

4 Data Types in Programming

This text summarizes basic data types in a programming language (likely Java, given the 'system.out.println' reference), focusing on integers, doubles, characters, and booleans. It also briefly introduces reference data types like strings.

4.1 Integers (`int`)

Integers are whole numbers. Examples include:

- Years (e.g., 2025)
- Quantities (e.g., 1)

4.2 Doubles (`double`)

Doubles are numbers that can contain decimal portions. Examples include:

- Prices (e.g., $19.99)
- GPAs (e.g., 3.5)
- Temperatures (e.g., 12.5)

Assigning an integer to a double variable will append a `.0` to the integer value, making it a double.

4.3 Characters (`char`)

Characters are single characters enclosed in single quotes. Examples include:

- Letter grades (e.g., `'A'`)
- Symbols (e.g., `'!'`)
- Currency symbols (e.g., `'$'`)

4.4 Booleans (`boolean`)

Booleans represent true or false values. Examples include:

- `isStudent` (true/false)
- `forSale` (true/false)
- `isOnline` (true/false)

Booleans are often used in conditional statements (like `if` statements) to control program flow. The text shows a basic example using an `if-else` statement to check the value of a boolean variable.

4.5 Strings

Strings are a sequence of characters. (Further detail is omitted from the provided text).

The text emphasizes that these are basic data types; more advanced types (floats, long doubles) exist but aren't necessary for beginners.

5 Introduction to Variables in Java

5.1 Strings

Objects are complex; we'll revisit them later. We'll create strings (series of characters). Examples include names, emails, and car models. Strings are enclosed in double quotes, while characters are in single quotes.

- `String name = "Bro Code";`
- `String food = "pizza";`
- `String email = "fake123@gmail.com";`
- `String car = "Mustang";`
- `String color = "red";`

Strings can contain numbers, but are treated as characters, not numerical values. String concatenation combines strings using the + operator. For example: `System.out.println("Hello " + name);`

The font color of strings often differs from primitive data types (booleans, chars, doubles, ints) in IDEs.

5.2 Variable Usage and Examples

- Printing variables: `System.out.println("Your age is " + age + " years old");`
- Combining multiple variables in a single `println` statement: `System.out.println("Your choice is a " + color + " " + year + " " + car);`
- Using a boolean variable: `if (forSale) { System.out.println("There is a " + car + " for sale"); } else { System.out.println(car + " is not for sale"); }`

5.3 Variable Types

There are two categories of variables:

- **Primitive data types:** Simple values (integers, floats, characters, booleans) stored directly in memory (stack).
- **Reference variables:** More complex (strings); store a memory address pointing to a location in the heap.

5.4 Homework Assignment

Create five variables: a string, an integer, a double, a char, and a boolean.

6 User Input in Java

To accept user input, we use the `Scanner` class. We need to import it: `import java.util.Scanner;`.

The `Scanner` object allows us to accept input, process it, and produce output. A typical programming flow involves accepting input, processing it, and then producing output. This is why accepting user input is important. We want to do something with the input.

This document summarizes a Java tutorial covering variables and user input. It explains string manipulation, variable types (primitive and reference), and the use of the `Scanner` class for user input. A homework assignment is included.

7 User Input in Java using Scanner

This document summarizes how to accept various data types (strings, integers, doubles, booleans) as user input in Java using the 'Scanner' class. It emphasizes proper resource management (closing the 'Scanner') and demonstrates different input methods.

7.1 Scanner Basics and Resource Management

- Create a 'Scanner' object: `Scanner scanner = new Scanner(System.in);`
- Close the 'Scanner' at the end of your program: `scanner.close();` (Essential to prevent resource leaks).

7.2 Accepting Different Data Types

7.2.1 Strings

- To read a line of text (including spaces): `String name = scanner.nextLine();`
- To read a single word (stops at the first space): `String firstName = scanner.next();`

`System.out.print("Enter your name: ");` followed by `String name = scanner.nextLine();` will prompt the user for their name and store it in the `name` variable. Outputting the name: `System.out.println("Hello " + name);`

7.2.2 Integers

- To read an integer: `int age = scanner.nextInt();`
- Error handling is required if the user inputs a non-integer value.

Example: `System.out.print("Enter your age: ");` followed by `int age = scanner.nextInt();` and then `System.out.println("You are " + age + " years old");`

7.2.3 Doubles

- To read a double (floating-point number): `double gpa = scanner.nextDouble();`

Example: `System.out.print("What is your GPA? ");` followed by `double gpa = scanner.nextDouble();` and then `System.out.println("Your GPA is " + gpa);`

7.2.4 Booleans

- To read a boolean (true/false): `boolean isStudent = scanner.nextBoolean();`

Example: `System.out.print("Are you a student (true/false)? ");` followed by `boolean isStudent = scanner.nextBoolean();` and then using an `if` statement to handle the boolean value (e.g., to print different messages based on whether the user is a student). This example demonstrates the usage of an `if` statement, which is explained further in the text but not detailed here.

7.3 Common Issues and Best Practices

- Mixing `next()` and `nextLine()`: Be mindful of how `next()` leaves a newline character in the input buffer, potentially causing unexpected behavior when following a `next()` call with a `nextLine()` call.
- Error Handling: Include error handling (e.g., `try-catch` blocks) when reading numbers to gracefully handle cases where the user enters invalid input.

This summary provides a concise overview of the Java 'Scanner' class functionalities for user input and highlights essential aspects for efficient and robust code. More advanced concepts like `if` statements and detailed error handling are mentioned but not explicitly coded here.

8 Java Input and Output

8.1 Handling Newline Characters

This section discusses a common issue in Java when reading user input: the newline character left in the input buffer after reading an integer or double before reading a string. This leads to unexpected behavior.

- The problem arises when using `next()` for integers/doubles and `nextLine()` for strings. The `nextLine()` method consumes the newline character left behind by the previous input.
- Solution: After reading an integer or double, call `scanner.nextLine()` without assigning it to a variable to clear the newline character from the buffer.

8.2 Calculating the Area of a Rectangle

This section describes a program that calculates the area of a rectangle based on user input.

- Declare variables: `double width = 0;`, `double height = 0;`, `double area = 0;`.

- Use a `Scanner` object to get user input for width and height using `nextDouble()`.
- Calculate the area: `area = width * height;`.
- Output the result: `System.out.println("The area is " + area + " centimeters$^2$");`. Note the use of `\textsuperscript2` for the superscript.

8.3 Mad Libs Game

This section outlines a Mad Libs game implemented in Java. Mad Libs is a game where a user provides words (adjectives, nouns, verbs) to fill in blanks in a story.

- Import `java.util.Scanner`.
- Declare string variables: `adjective1`, `noun1`, `adjective2`, `verb1`, `adjective3`.
- Use a `Scanner` to prompt the user for input for each variable using `nextLine()`.
- Construct the Mad Libs story using string concatenation, incorporating the user-provided words.

9 Summary

The text provides examples of basic Java input/output operations. It highlights the issue of newline characters in the input buffer and demonstrates how to solve it. It then gives examples of programs to calculate a rectangle's area and to create a simple Mad Libs game, showcasing the use of `Scanner` for user input and string manipulation for output.

10 Mad Libs Game in Java

This section describes a simple Mad Libs game implemented in Java. The program prompts the user for various words (adjectives, nouns, verbs) and then generates a short story using these inputs.

10.1 Program Logic

The core logic involves:

- Prompting the user to enter an adjective, noun, another adjective, a verb (present tense, ending in -ing), and a final adjective.
- Using a `Scanner` to read user input.

- Storing the inputs in variables (`adjective1`, `noun1`, `adjective2`, `verb1`, `adjective3`).
- Concatenating the inputs to create the Mad Libs story.
- Closing the `Scanner` to avoid resource leaks.
- Displaying the generated story.

10.2 Example Interaction

An example interaction might look like this:

- **Prompt:** Enter an adjective
- **User Input:** suspicious
- **Prompt:** Enter a noun
- **User Input:** Mark Zuckerberg
- **Prompt:** Enter an adjective
- **User Input:** smelly
- **Prompt:** Enter a verb ending with -ing
- **User Input:** screaming
- **Prompt:** Enter an adjective
- **User Input:** happy

This would result in a story like: "I went to a suspicious zoo. In an exhibit I saw a Mark Zuckerberg. Mark Zuckerberg was smelly and screaming. I was happy."

11 Arithmetic Operators in Java

This section covers basic arithmetic operators in Java, including augmented assignment operators and increment/decrement operators.

11.1 Basic Arithmetic

The basic arithmetic operators are demonstrated using variables `x`, `y`, and `z`:

- **Addition:** `z = x + y` (e.g., 10 + 2 = 12)
- **Subtraction:** `z = x - y` (e.g., 10 - 2 = 8)
- **Multiplication:** `z = x  y` (e.g., 10 2 = 20)

- **Division:** `z = x / y` (e.g., 10 / 2 = 5. Note integer division truncates decimals.)
- **Modulus:** `z = x % y` (e.g., 10 % 2 = 0; 10 % 3 = 1. Returns the remainder of the division.)

11.2 Augmented Assignment Operators

These operators provide a shorthand for combining an arithmetic operation with an assignment:

- `x += y` (equivalent to `x = x + y`)
- `x -= y` (equivalent to `x = x - y`)
- `x = y` (equivalent to `x = x  y`)
- `x /= y` (equivalent to `x = x / y`)
- `x %= y` (equivalent to `x = x % y`)

11.3 Increment and Decrement Operators

These operators increment or decrement a variable by 1:

- `x++` (increments `x` by 1)
- `x--` (decrements `x` by 1)

Summary: The text describes a simple Mad Libs game implemented in Java and explains basic arithmetic operations, augmented assignment operators, and increment/decrement operators in Java. It highlights the importance of closing resources (like the `Scanner`) and points out the effects of integer division.

12 Order of Operations

The order of operations follows the acronym PEMDAS (Parentheses, Exponents, Multiplication, Division, Addition, Subtraction). You might also hear "Please Excuse My Dear Aunt Sally" or, as the author prefers, "Please Excuse My Dope Ass Swag".

Let's consider the equation:

```
double result = 3 + 4 * (7 - 5) / 2.0;
```

Following PEMDAS, we solve the equation from left to right:

- **Parentheses**: (7 - 5) = 2
- **Multiplication**: 4 * 2 = 8
- **Division**: 8 / 2.0 = 4

- **Addition**: 3 + 4 = 7
- **Subtraction**: (There is no subtraction in this equation)

Therefore, the result is 7.0.

13 Shopping Cart Program

This section details the creation of a simple shopping cart program in Java.

13.1 Setup and User Input

The program utilizes the `Scanner` class to accept user input. The necessary import statement is:

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

A `Scanner` object is created:

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

Remember to close the scanner when finished:

```
scanner.close();
```

Variables are declared to store item name (`String item`), price (`double price`), quantity (`int quantity`), currency (`char currency`), and total cost (`double total`).

13.2 Program Logic

The program prompts the user for the item name, price, and quantity using `System.out.print` and retrieves the input using `scanner.nextLine()`, `scanner.nextDouble()`, and `scanner.nextInt()`, respectively. The total cost is calculated as `price * quantity`.

13.3 Output

The program then outputs a summary of the purchase, including the quantity, item name, price, and total cost. Example output might look like:

"You have bought 3 pizzas. Your total is $14.97."

The program demonstrates basic arithmetic operations and user input handling in Java. The use of \System.out.print and \System.out.println for output is shown. Error handling and more sophisticated input validation are not included in this simplified example.

"'latex

14 Summary

This text describes a Java program that uses 'if', 'else if', and 'else' statements to demonstrate conditional logic. The program first shows a simple example

checking if a user's age is greater than or equal to 18, then expands to include multiple conditions (age ranges and a check for negative age) and user input using the 'Scanner' class. Finally, it briefly mentions extending the logic to handle string input (user's name).

15 Shopping Cart Program (Example)

A brief example of a simple shopping cart program in Java is mentioned as an exercise. The example involves calculating the total cost of multiple pizzas.

16 If Statements in Java

16.1 Basic If Statement

- An `if` statement executes a block of code if its condition is true.
- Boolean variables are used to control the execution of the `if` statement.
- Example: Checking if age is greater than or equal to 18. If true, print "You are an adult."

16.2 Adding `else` and `else if` Clauses

- An `else` clause provides an alternative block of code to execute if the `if` condition is false.
- `else if` clauses allow for multiple conditional checks.
- Example: Adding checks for age ranges (child, baby, senior citizen) using `else if`. The `else` clause handles cases where none of the previous conditions are met.

16.3 Order of Conditions

- The order of `if` and `else if` statements matters. Conditions are checked sequentially from top to bottom.
- If multiple conditions are true, only the code associated with the first true condition is executed.

16.4 User Input with Scanner

- The program is modified to take user input for age using the `java.util.Scanner` class.
- `scanner.nextInt()` is used to read integer input.
- It's important to close the scanner after use (`scanner.close()`).

16.5 String Input

- The program is briefly mentioned to be expandable to handle string input (user's name) using `scanner.nextLine()`.

16.6 Comparison Operator

- The double equals sign (`==`) is used for comparison, while a single equals sign (`=`) is for assignment.

"'

17 If-Else Statements in Java

17.1 Group 1: Name Input

- Checks if the name variable is empty using `name.isEmpty()`.
- If empty, prints "You didn't enter your name". Adds a pouting face emoji.
- If not empty, prints "Hello *name*!". Adds a smiley face emoji.

17.2 Group 2: Age Input

- Checks the user's age. (Details of age checking logic not explicitly provided in text).
- Outputs age-related messages (e.g., "you are an adult", "you haven't been born yet"). Adds relevant emojis.

17.3 Group 3: Student Status

- Uses a Boolean variable `isStudent` to store user input.
- If `isStudent` is true, prints "You are a student". Adds a school emoji.
- If `isStudent` is false, prints "You are not a student". Adds an office emoji.

17.4 Summary

The code uses three groups of if-else statements to handle user input: name, age, and student status. Each group uses conditional logic and outputs different messages and emojis based on the user's responses.

18 Generating Random Numbers in Java

18.1 Importing and Creating a Random Object

- Imports the `java.util.Random` class.
- Creates a `Random` object using `Random random = new Random();`.

18.2 Generating Random Integers

- Uses the `nextInt(bound)` method to generate random integers.
- `nextInt(bound)` generates a random integer between 0 (inclusive) and `bound` (exclusive). For example, `nextInt(7)` generates numbers from 0 to 6.
- To get a random number in a range [a, b], use `a + random.nextInt(b - a + 1)`.

18.3 Generating Random Doubles

- The text mentions the ability to generate random doubles but does not provide code examples.

18.4 Summary

The code demonstrates how to use the `java.util.Random` class to generate random integers within specified bounds. The example shows how to generate single and multiple random numbers. The capability of generating random doubles is also noted.

19 Generating Random Numbers and Using the Random Module in Java

19.1 Generating Random Numbers

- Change the data type to `double`.
- Use the `nextDouble()` method to generate a random number between 0 and 1.
- Generate a random boolean value using `nextBoolean()`. This can be used to simulate a coin flip.
- Example: Simulate a coin flip using an `if` statement to display "Heads" or "Tails" based on a random boolean value.

20 Useful Math-Related Constants and Methods in Java

20.1 Constants

- `Math.PI`: Provides the value of π.
- `Math.E`: Provides the value of Euler's number (e).

20.2 Methods

- `Math.pow(base, exponent)`: Raises a `base` to a given `exponent`.
- `Math.abs(number)`: Returns the absolute value of a number.
- `Math.sqrt(number)`: Returns the square root of a number.
- `Math.round(number)`: Rounds a number to the nearest whole integer.
- `Math.ceil(number)`: Rounds a number up to the nearest integer.
- `Math.floor(number)`: Rounds a number down to the nearest integer.
- `Math.max(number1, number2)`: Returns the maximum of two numbers.
- `Math.min(number1, number2)`: Returns the minimum of two numbers.

21 Math-Related Exercises

21.1 Hypotenuse of a Right Triangle

This exercise involves calculating the hypotenuse of a right triangle using user input for the lengths of the other two sides. The formula used is $c = \sqrt{a^2 + b^2}$, where c is the hypotenuse, and a and b are the lengths of the other two sides. The solution utilizes the `Scanner` class for user input, the `Math.pow()` method for squaring the sides, and the `Math.sqrt()` method for calculating the square root. Error handling and resource management (closing the `Scanner`) are important considerations. Units of measurement (e.g., centimeters) can be optionally added to the output.

Summary: The text describes how to generate random numbers and use the `random` module in Java, along with a detailed explanation of several useful mathematical constants and methods within the `Math` class. A practical exercise demonstrating the calculation of the hypotenuse of a right triangle using user input and several `Math` class methods is also provided.

22 Summary

This text covers two main topics: basic geometric calculations in Java and the use of the 'printf' statement for formatted output. The geometric calculations section demonstrates calculating the circumference, area, and volume of a circle/sphere given its radius, incorporating user input and mathematical formulas. The second section details the 'printf' statement, explaining its use as an alternative to 'println' and 'print', showcasing its ability to format output by using placeholders and specifiers for various data types.

23 Geometric Calculations in Java

23.1 Calculating Circle/Sphere Properties

- The program prompts the user to enter the radius of a circle or sphere.
- It calculates the circumference ($2 \times \pi \times radius$), area ($\pi \times radius^2$), and volume ($\frac{4}{3} \times \pi \times radius^3$) using Java's 'Math' class.
- It displays the results, including appropriate units (cm, cm^2, cm^3).
- The use of 'printf' for formatted output (limiting decimal places) is mentioned as a future topic.

23.2 Code Snippet (Partial)

The code uses a 'Scanner' object to get user input, declares variables ('radius', 'circumference', 'area', 'volume'), and performs calculations using 'Math.PI' and 'Math.pow()'. Output is initially done using 'println', with later mention of using 'printf' for improved formatting.

24 The `printf` Statement in Java

24.1 Formatted Output

- `printf` is an alternative to `println` and `print` for formatted output.
- It uses placeholders (`%`) followed by specifiers (e.g., `s` for String, `c` for char, `d` for int, `f` for double, `b` for boolean) to insert variables into the output string.
- Example: `printf("Hello %s", name);` inserts the value of the `name` variable into the string.

24.2 Example with Different Data Types

The example shows how to use `printf` with variables of different data types (String, char, int, double, boolean) to create a formatted output string. This demonstrates the flexibility of `printf` in handling various data types within a single output statement.

25 Formatting Specifiers in `printf`

This text describes how to use format specifiers in Java's `printf` method for various data types. It covers inserting variables, handling newlines, precision, and flags for formatting output.

25.1 Basic Data Types

- **Strings (`s`):** Use `%s` as a placeholder, followed by a comma and the string variable. Example: `printf("hello %s", variableName)`.
- **Characters (`c`):** Use `%c` as a placeholder, followed by a comma and the character variable. Example: `printf("Your name starts with a %c", charVariable)`.
- **Integers (`d`):** Use `%d` as a placeholder, followed by a comma and the integer variable. Example: `printf("You are %d years old", ageVariable)`.
- **Doubles/Floats (`f`):** Use `%f` as a placeholder, followed by a comma and the double/float variable. Example: `printf("You are %f inches tall", heightVariable)`.
- **Booleans (`b`):** Use `%b` as a placeholder, followed by a comma and the boolean variable. Example: `printf("Employed:  %b", employedVariable)`.

25.2 Newlines

Remember to add `\n` after the format specifier to insert a new line character in your output.

25.3 Precision

To limit the number of digits after the decimal point in floating-point numbers, add a dot followed by the desired number of digits between the percent sign and the `f`. For example, `%.1f` displays one digit after the decimal.

25.4 Flags

Several flags can modify the output format:

- `+`: Adds a plus sign before positive numbers.

- `,`: Adds a comma as a thousands separator.
- `(`: Encloses negative numbers in parentheses.
- ` `: Adds a leading space before positive numbers.

25.5 Width

By specifying a width before the format specifier (e.g., `%8d`), you can align integers by padding with spaces.

25.6 Example Summary

The text provides a comprehensive guide to using format specifiers in `printf` statements in Java, covering basic data types, newlines, precision control for floating-point numbers, various flags for enhanced formatting, and width specification for aligning numerical output. The examples illustrate how to use these features effectively.