**Printing, Keywords & Identifiers**

**Printing Statements**

In Java, printing statements are commonly used to display information to the console or output window. The most commonly used methods for printing in Java are System.out.print(), System.out.println(), and System.out.printf(). Each of these methods has specific functionality that can be useful depending on the situation.

**1. System.out.print()**

The System.out.print() method prints text to the console **without moving the cursor to a new line** after the text is printed. It is useful when you want to print multiple pieces of information on the same line.

**Example:**

public class PrintExample {

public static void main(String[] args) {

System.out.print("Hello, ");

System.out.print("World!");

}

}

**Output:**

Hello, World!

**2. System.out.println()**

The System.out.println() method prints text to the console **and moves the cursor to a new line** after printing. This is useful when you want each printed statement to appear on a new line.

**Example:**

public class PrintExample {

public static void main(String[] args) {

System.out.println("Hello, ");

System.out.println("World!");

}

}

**Output:**

Hello,

World!

**3. System.out.printf()**

The System.out.printf() method allows formatted output. It is useful when you need to print values in a specific format, such as for floating-point numbers, alignment, padding, or precision. The format specifiers (like %d, %f, %s) help you control how data is displayed.

**Example:**

public class PrintExample {

public static void main(String[] args) {

int age = 25;

double price = 45.99;

System.out.printf("Age: %d\n", age);

// Shows price with two decimal places

System.out.printf("Price: %.2f\n", price);

}

}

**Output:**

Age: 25

Price: 45.99

**Format Specifiers in printf():**

* %d: Prints an integer.
* %f: Prints a floating-point number.
* %s: Prints a string.
* %c: Prints a character.
* %.2f: Prints a floating-point number with 2 decimal places (useful for currency, for example).

**4. System.out.format()**

System.out.format() is similar to printf(), as both use the same format specifiers and provide formatted output. It's another way to achieve formatted output in Java, with the same syntax and functionality as printf().

**Example:**

public class PrintExample {

public static void main(String[] args) {

String name = "Alice";

int score = 95;

System.out.format("Name: %s\n", name);

System.out.format("Score: %d\n", score);

}

}

**Output:**

Name: Alice

Score: 95

**5. Using Escape Sequences**

In Java, escape sequences allow you to include special characters in strings. Here are some common escape sequences:

* \n: New line
* \t: Tab
* \\: Backslash
* \': Single quote
* \": Double quote

**Example:**

public class PrintExample {

public static void main(String[] args) {

System.out.println("Hello\tWorld!");

System.out.println("This is a new line.\nNext line starts here.");

System.out.println("A backslash: \\");

System.out.println("Quote: \"Java\"");

}

}

**Output:**

Hello World!

This is a new line.

Next line starts here.

A backslash: \

Quote: "Java"

**6. Concatenating Strings**

In Java, you can concatenate (combine) strings using the + operator. This is useful when you want to combine different variables or expressions into a single output statement.

**Example:**

public class PrintExample {

public static void main(String[] args) {

String name = "Bob";

int age = 30;

System.out.println("Name: " + name + ", Age: " + age);

}

}

**Output:**

Name: Bob, Age: 30

### ****Java Keywords****

#### ****Description:****

In Java, **keywords** are predefined, reserved words that have a special meaning in the language syntax. They cannot be used as identifiers (such as variable names, function names, or class names) because they are integral to the functionality of the language.

Here is a table with **Java keywords**:

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | continue | for | new |
| assert | default | goto | package |
| boolean | do | if | private |
| break | double | implements | protected |
| byte | else | import | public |
| case | enum | instanceof | return |
| catch | extends | int | short |
| char | final | interface | static |
| class | finally | long | strictfp |
| const | float | native | super |
| default | for | null | synchronized |
| do | goto | package | this |
| double | if | private | throw |
| else | implements | protected | transient |
| enum | import | public | try |
| extends | instanceof | short | void |
| final | int | static | volatile |
| finally | interface | super | while |

### ****Identifiers in Java****

#### ****Description:****

In Java, an **identifier** is the name given to a variable, method, class, or other user-defined items. Identifiers are essential for distinguishing between various entities in a program. They are used to identify variables, functions, classes, packages, and other elements in Java programs.

#### ****Rules for Naming Identifiers:****

1. **Starting Character:** An identifier must start with a letter (a-z, A-Z), a dollar sign ($), or an underscore (\_). It cannot start with a digit (0-9).
   * **Valid:** age, \_counter, total$
   * **Invalid:** 1stNumber, @value
2. **Subsequent Characters:** After the first character, the identifier can contain letters, digits (0-9), underscores (\_), or dollar signs ($).
   * **Valid:** totalValue, total\_Amount\_2, value$
   * **Invalid:** total@Amount, #name
3. **Case Sensitivity:** Java identifiers are **case-sensitive**, meaning Total and total would be considered different identifiers.
   * **Valid:** total, Total, TOTAL
4. **Reserved Keywords:** Identifiers cannot be named using Java’s reserved keywords (e.g., class, public, void, int, etc.), as these are predefined in the language.
   * **Invalid:** class, public, if
5. **Length of Identifier:** There is no explicit limit to the length of an identifier, but it's good practice to choose meaningful and concise names to improve code readability.
6. **Meaningful Names:** While Java allows flexibility in naming, it is considered best practice to use descriptive and meaningful names for identifiers, which makes the code more readable and maintainable. For example:
   * **Good identifiers:** studentName, totalAmount, isEligible
   * **Bad identifiers:** a, x, temp

#### ****Best Practices for Naming Identifiers:****

1. **Use Descriptive Names:** Ensure that the name reflects the purpose of the variable, method, or class.
   * Example: int totalAmount; instead of just int x;
2. **Follow Naming Conventions:**
   * **Classes and Interfaces:** Use **PascalCase** (capitalize the first letter of each word). Example: StudentInfo, AccountDetails
   * **Variables and Methods:** Use **camelCase** (start with a lowercase letter, and capitalize subsequent words). Example: totalAmount, calculateTotal, lastIndexOf.
   * **Constants:** Use **UPPERCASE** letters with underscores separating words. Example: MAX\_SIZE, PI\_VALUE
3. **Avoid Single-Character Names:** Unless used for loop counters, avoid single-character names like a, x, etc. They don’t convey any meaningful information about the data.
4. **Avoid Using Underscores in Method and Variable Names:** Use underscores in constants or variables only if it is necessary, such as in **UPPERCASE** constants.
5. **Be Consistent:** Stick to one naming convention throughout your program or project to maintain uniformity and readability.

### ****Examples of Valid and Invalid Identifiers:****

| **Valid Identifiers** | **Invalid Identifiers** |
| --- | --- |
| age | 1stNumber |
| totalAmount | total#Amount |
| isValid | @name |
| \_counter | public |
| studentName | int |
| StudentDetails | 123abc |
| calculateTotal() | class |
| total\_Amount\_2 | void |