Republic of the Philippines
Department of Education
REGION III
SCHOOLS DIVISION OFFICE OF NUEVA ECIJA

LEARNING ACTIVITY SHEET
SPECIAL PROGRAM IN ICT 9
BASIC PROGRAMMING
Second Quarter, Week 6

Java Identifiers, Java Keywords & Java Literals

BACKGROUND INFORMATION FOR LEARNERS

In programming languages, identifiers are used for identification purposes. In Java, an identifier can be a class name, method name, variable name, or label. For example :

```
public class Test
{
    public static void main(String[] args)
    {
        int a = 20;
    }
}
```

In the above java code, we have 5 identifiers namely :

- **Test** : class name.
- **main** : method name.
- **String** : predefined class name.
- **args** : variable name.
- **a** : variable name.

Rules for defining Java Identifiers

There are certain rules for defining a valid java identifier. These rules must be followed, otherwise we get compile-time error. These rules are also valid for other languages like C,C++.

- The only allowed characters for identifiers are all alphanumeric characters([**A-Z**],[**a-z**],[**0-9**]), '**$**'(dollar sign) and '**_**' (underscore).For example "geek@" is not a valid java identifier as it contain '@' special character.
- Identifiers should **not** start with digits(**[0-9]**). For example "123geeks" is a not a valid java identifier.

- Java identifiers are **case-sensitive**.
- There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.
- **Reserved Words** can't be used as an identifier. For example "int while = 20;" is an invalid statement as while is a reserved word. There are **53** reserved words in Java.

Examples of valid identifiers :

```
MyVariable
MYVARIABLE
myvariable
x
i
x1
i1
_myvariable
$myvariable
sum_of_array
geeks123
```

Examples of invalid identifiers :

```
My Variable  // contains a space
123geeks   // Begins with a digit
a+c // plus sign is not an alphanumeric character
variable-2 // hyphen is not an alphanumeric character
sum_&_difference // ampersand is not an alphanumeric character
```

Reserved Words

Any programming language reserves some words to represent functionalities defined by that language. These words are called reserved words. They can be briefly categorised into two parts : **keywords**(50) and **literals**(3). Keywords define functionalities and literals define a value. Identifiers are used by symbol tables in various analyzing phases(like lexical, syntax, semantic) of a compiler architecture.

Note: The keywords const and goto are reserved, even though they are not currently used. In place of const, the final keyword is used. Some keywords like strictfp are included in later versions of Java.

Java Keywords

Keywords or Reserved words are the words in a language that are used for some internal process or represent some predefined actions. These words are therefore not allowed to use as a variable names or objects. Doing this will result into a **compile time error**.

Java also contains a list of reserved words or keywords. These are:

1. **abstract** -Specifies that a class or method will be implemented later, in a subclass
2. **assert** -Assert describes a predicate (a true–false statement) placed in a Java program to indicate that the developer thinks that the predicate is always true at that place. If an assertion evaluates to false at run-time, an assertion failure results, which typically causes execution to abort.
3. **boolean** – A data type that can hold True and False values only
4. **break** – A control statement for breaking out of loops
5. **byte** – A data type that can hold 8-bit data values
6. **case** – Used in switch statements to mark blocks of text
7. **catch** – Catches exceptions generated by try statements
8. **char** – A data type that can hold unsigned 16-bit Unicode characters
9. **class** -Declares a new class
10. **continue** -Sends control back outside a loop
11. **default** -Specifies the default block of code in a switch statement
12. **do** -Starts a do-while loop
13. **double** – A data type that can hold 64-bit floating-point numbers
14. **else** – Indicates alternative branches in an if statement
15. **enum** – A Java keyword used to declare an enumerated type. Enumerations extend the base class.
16. **extends** -Indicates that a class is derived from another class or interface
17. **final** -Indicates that a variable holds a constant value or that a method will not be overridden
18. **finally** -Indicates a block of code in a try-catch structure that will always be executed
19. **float** -A data type that holds a 32-bit floating-point number
20. **for** -Used to start a for loop
21. **if** -Tests a true/false expression and branches accordingly
22. **implements** -Specifies that a class implements an interface
23. **import** -References other classes
24. **instanceof** -Indicates whether an object is an instance of a specific class or implements an interface

25. **int** – A data type that can hold a 32-bit signed integer
26. **interface** – Declares an interface
27. **long** – A data type that holds a 64-bit integer
28. **native** -Specifies that a method is implemented with native (platform-specific) code
29. **new** – Creates new objects
30. **null** -Indicates that a reference does not refer to anything
31. **package** – Declares a Java package
32. **private** -An access specifier indicating that a method or variable may be accessed only in the class it's declared in
33. **protected** – An access specifier indicating that a method or variable may only be accessed in the class it's declared in (or a subclass of the class it's declared in or other classes in the same package)
34. **public** – An access specifier used for classes, interfaces, methods, and variables indicating that an item is accessible throughout the application (or where the class that defines it is accessible)
35. **return** -Sends control and possibly a return value back from a called method
36. **short** – A data type that can hold a 16-bit integer
37. **static** -Indicates that a variable or method is a class method (rather than being limited to one particular object)
38. **strictfp** – A Java keyword used to restrict the precision and rounding of floating point calculations to ensure portability.
39. **super** – Refers to a class's base class (used in a method or class constructor)
40. **switch** -A statement that executes code based on a test value
41. **synchronized** -Specifies critical sections or methods in multithreaded code
42. **this** -Refers to the current object in a method or constructor
43. **throw** – Creates an exception
44. **throws** -Indicates what exceptions may be thrown by a method
45. **transient** -Specifies that a variable is not part of an object's persistent state
46. **try** -Starts a block of code that will be tested for exceptions
47. **void** -Specifies that a method does not have a return value
48. **volatile** -Indicates that a variable may change asynchronously
49. **while** -Starts a while loop

** The keywords **const** and **goto** are reserved, even they are not currently in use.

- **const** -Reserved for future use
- **goto** – Reserved for future use

** **true, false** and **null** look like keywords, but in actual they are **literals**. However they still can't be used as identifiers in a program.

Java Literals

Literals are data used for representing fixed values. They can be used directly in the code.

For example,

```
int a = 1;
float b = 2.5;
char c = 'F';
```

Here, `1`, `2.5`, and `'F'` are literals.

Here are different types of literals in Java.

1. Boolean Literals

In Java, boolean literals are used to initialize boolean data types. They can store two values: true and false. For example,

```
boolean flag1 = false;
boolean flag2 = true;
```

Here, `false` and `true` are two boolean literals.

2. Integer Literals

An integer literal is a numeric value(associated with numbers) without any fractional or exponential part. There are 4 types of integer literals in Java:

1. binary (base 2)
2. decimal (base 10)
3. octal (base 8)
4. hexadecimal (base 16)

For example:

```
// binary
int binaryNumber = 0b10010;
// octal
int octalNumber = 027;
```

```
// decimal
int decNumber = 34;

// hexadecimal
int hexNumber = 0x2F; // 0x represents hexadecimal
// binary
int binNumber = 0b10010; // 0b represents binary
```

In Java, binary starts with **0b**, octal starts with **0**, and hexadecimal starts with **0x**.

Note: Integer literals are used to initialize variables of integer types like `byte`, `short`, `int`, and `long`.

3. Floating-point Literals

A floating-point literal is a numeric literal that has either a fractional form or an exponential form. For example,

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

    double myDouble = 3.4;
    float myFloat = 3.4F;

    // 3.445*10^2
    double myDoubleScientific = 3.445e2;

    System.out.println(myDouble);  // prints 3.4
    System.out.println(myFloat);    // prints 3.4
    System.out.println(myDoubleScientific);   // prints 344.5
  }
}
```

Note: The floating-point literals are used to initialize `float` and `double` type variables.

4. Character Literals

Character literals are unicode character enclosed inside single quotes. For example,

```
char letter = 'a';
```

Here, `a` is the character literal.

We can also use escape sequences as character literals. For example, **\b** (backspace), **\t** (tab), **\n** (new line), etc.

5. String literals

A string literal is a sequence of characters enclosed inside double-quotes. For example,

```
String str1 = "Java Programming";
String str2 = "Programiz";
```

Here, `Java Programming` and `Programiz` are two string literals.

LEARNING COMPETENCY

1. Define what Java identifiers are and identify its guidelines
2. List the different Java keywords
3. Name the literals used in Java

ACTIVITIES: **Type your answer in the private comment section of our google classroom**

ACTIVITY 1: DEFINITION: Define what is being asked.

1. In your own words, define what **Java Identifier** is.

ACTIVITY 2: List 5 different **JAVA KEYWORDS** and give its definition.

ACTIVITY 3: What are the **Five(5) Literals** in **JAVA**?

REFERENCES

https://www.geeksforgeeks.org/java-identifiers/
https://www.geeksforgeeks.org/list-of-all-java-keywords/
https://www.programiz.com/java-programming/variables-literals

Prepared by:

Maximo A. Luna Jr

Name of Writer

Noted by:

Laberne A. Ladignon Jr

Division ICT Coordinator/ OIC EPS