Lecture 3: Keywords, Expressions, Methods, and Method Overloading

Lecture overview

This is an outline of the topics covered in **Lecture 3** of CS 104T. The goal of this lecture is to introduce the foundational concepts of Java and provide a hands-on demonstration of writing basic Java programs.

---

1. Keywords in Java

Definition:
Keywords are reserved words in Java that have a predefined meaning and are part of the syntax of the language. They cannot be used as identifiers (variable names, method names, class names, etc.).

Importance:
Keywords define the structure and control flow of a Java program. They guide the compiler on how to interpret the code.

Examples of Common Keywords:

- `class`: Used to declare a class.
- `public`: An access modifier indicating that the method or class can be accessed from any other class.
- `static`: Indicates that a method or variable belongs to the class rather than instances of the class.
- `void`: Specifies that a method does not return a value.

Example Program:

```
public class KeywordExample {
    public static void main(String[] args) {
        // 'public', 'static', 'void' are keywords
        System.out.println("Keywords in Java are reserved words.");
    }
}
```

Best Practices:

- Avoid using keywords as identifiers to prevent confusion and compilation errors.
- Always capitalize class names and use camel case for method names to enhance readability.

---

2. Expressions

Definition:
Expressions are combinations of variables, constants, and operators that produce a value. They are the building blocks of Java programs.

Importance:
Expressions perform calculations and evaluations, allowing the program to manipulate data.

Example of an Expression:

- `int aSmallNumber = 2;`
  Here, `2` is a literal value, and `aSmallNumber` is a variable.

Example Program:

```
public class ExpressionExample {
    public static void main(String[] args) {
        int aSmallNumber = 2; // Variable assignment
        if (aSmallNumber <= 10) {
            System.out.println("It really is a small number");
        }
    }
}
```

Best Practices:

- Keep expressions simple to enhance code readability.
- Use parentheses to clarify complex expressions.

---

3. Statements, Whitespaces, and Indentation

Definition:
A statement is a complete line of code that performs an action and ends with a semicolon. Statements can include declarations, assignments, and control flow instructions.

Importance:
Properly formatted statements improve the readability and maintainability of code.

Example Statement:

- `int aSmallNumber = 2;`

Combining Statements: Statements can be combined, but it's not recommended for clarity.

```
int a = 5; int b = 10; // Not recommended
```

Best Practices:

- Use one statement per line.
- Indent code blocks for better readability.

---

4. Indentation

Definition:
Indentation refers to the practice of adding spaces or tabs at the beginning of a line of code to structure it visually.

Importance:
Proper indentation enhances readability, making it easier to understand the code structure, especially in nested blocks.

Example of Indentation:

```
if (true) {
    System.out.println("Indented for clarity");
}
```

Tools:

- Use IDE features (e.g., IntelliJ's auto-formatting) to maintain consistent indentation.

Best Practices:

- Use consistent indentation (e.g., 4 spaces) throughout your code.
- Indent code blocks (e.g., loops, conditionals) to visually distinguish them.

---

5. Code Blocks and If-Then-Else Statements

Definition:
A code block is a group of statements enclosed in curly braces `{}`. It defines the scope of variables and the execution flow.

Importance:
Code blocks help organize code and control the execution based on conditions.

If-Then-Else Example:

```
if (aSmallNumber <= 10) {
    System.out.println("Small number");
} else {
    System.out.println("Not a small number");
}
```

Multiple Conditions Example:

```
int grade = 85;
if (grade >= 90) {
    System.out.println("A");
} else if (grade >= 80) {
    System.out.println("B");
} else {
    System.out.println("C");
}
```

Accessing Variables: Variables defined within a code block cannot be accessed outside that block.

```
{
    int x = 10; // x is accessible within this block
}
// System.out.println(x); // This will cause a compile error
```

Best Practices:

- Use clear and descriptive variable names.
- Avoid nesting too many if-else statements for clarity.

---

6. Functions

Definition:
A function in Java is a block of code designed to perform a specific task. Functions can take parameters and return values.

Importance:
Functions promote code reuse, modularity, and abstraction.

Declaring a Function:

```
public int add(int a, int b) {
    return a + b;
}
```

Example Program:

```
public class FunctionExample {
    public static void main(String[] args) {
        System.out.println("Sum: " + add(5, 10));
    }

    public static int add(int a, int b) {
        return a + b; // Function returning a value
    }
}
```

Best Practices:

- Keep functions focused on a single task.
- Use meaningful names for functions that indicate their purpose.

---

7. Method Overloading

Definition:
Method overloading allows multiple methods to have the same name with different parameter lists (i.e., different types or numbers of parameters).

Importance:
It provides flexibility in method usage, allowing different behaviors based on input parameters.

Example Program:

```
public class OverloadingExample {
    public static void main(String[] args) {
        System.out.println(add(5, 10));          // Calls add(int, int)
        System.out.println(add(5.5, 10.5));      // Calls add(double, double)
    }

    public static int add(int a, int b) {
        return a + b;
    }

    public static double add(double a, double b) {
        return a + b;
    }
}
```

Best Practices:

- Ensure that overloaded methods are distinct in functionality.
- Avoid confusion by maintaining clear parameter types.