Al-Mustaqbal University
College of Sciences
Intelligent Medical System Department

جامـــــعة المـــــستـقبـل
AL MUSTAQBAL UNIVERSITY

كلية العلوم
قـســـــــم الانظمة الطبية الذكية

Lecture: (3)

The fundamental concepts of computer programming, Structured Programming, Program Errors .

Subject: Computer Programming (I)

Level: First

Lecturer: Dr. Maytham N. Meqdad

- **The fundamental concepts of computer programming" java"**

The fundamental concepts of computer programming are principles and techniques that form the basis for writing efficient, maintainable, and scalable software. These concepts are applicable across various programming languages, and understanding them is crucial for anyone learning to program. Here are some key fundamental concepts:

1. **Variables and Data Types:**
   - Variables are used to store and manipulate data. Data types define the nature of the data that a variable can hold, such as integers, floating-point numbers, characters, and booleans.

2. **Operators:**
   - Operators are symbols that perform operations on variables and values. Common types include arithmetic operators (+, -, *, /), comparison operators (==, !=, <, >), and logical operators (&&, ||, !).

3. **Control Flow:**
   - Control flow structures determine the order in which statements are executed. Common control flow statements include:
     - **Conditional Statements:** if, else if, else
     - **Looping Statements:** for, while, do-while
     - **Switch Statements:** Selects one of many code blocks to be executed

4. **Functions/Methods:**
   - Functions (or methods, depending on the language) allow you to group code into reusable blocks. They take inputs, perform a specific task, and can return a value.

5. **Data Structures:**
   - Data structures organize and store data. Common data structures include arrays, linked lists, stacks, queues, and trees. The choice of a data structure depends on the specific needs of the program.

6. **Algorithms:**
   - Algorithms are step-by-step procedures or formulas for solving problems. They involve a sequence of well-defined, unambiguous instructions for performing a task or solving a particular problem.

7. **Object-Oriented Programming (OOP):**
   - OOP is a paradigm that represents concepts as "objects" that have attributes (data) and methods (functions). Key OOP principles include encapsulation, inheritance, and polymorphism.

8. **Error Handling:**

- Error handling involves dealing with unexpected situations or errors in a program. This may include using try-catch blocks to catch and handle exceptions.

9. **Debugging:**
- Debugging is the process of identifying and fixing errors in the code. This often involves using debugging tools, print statements, and logical reasoning to trace and correct issues.

10. **Comments and Documentation:**
- Adding comments to code helps explain its functionality. Documentation, whether in-line or external, provides a comprehensive understanding of the program's structure and purpose.

11. **Version Control:**
- Version control systems, such as Git, help track changes to code over time, enabling collaboration and providing a way to revert to previous versions if needed.

12. **Testing:**
- Testing is crucial for ensuring the correctness and reliability of a program. Unit testing, integration testing, and other testing methodologies help identify and fix issues.

13. **Modularity:**
- Breaking down a program into smaller, manageable modules promotes code reusability, maintainability, and collaboration among team members.

14. **Concurrency and Parallelism:**
- Understanding how to handle multiple tasks or processes simultaneously (concurrency) and how to perform tasks concurrently to improve performance (parallelism) is important for certain types of applications.

==========

Structured Programming:

- ## structured Programming in Java

Structured programming is a program written with only the three constructions sequence, repetition, and decision.

1. **Sequence.** Lines or blocks of code are written and executed in sequential order.

   Example:

```
x = 5;
y = 11;
z = x + y;
System.out.println(z);
```

2. **Repetition.** Repeat a block of code (Action) while a condition is true. There is no limit to the number of times that the block can be executed.

```
while(condition)
{
    action
}
```

   When the body of a while statement, or of a for or switch statement consists of a single line, no braces are necessary.

   Example:

```
while(x < 100)
{
    System.out.println(x);
    x = x * x
}
```

3. **Decision.** Execute a block of code (Action) if a condition is true. The block of code is executed at most once.

```
if (condition)
{
    action
}
```

   Example:

```
if (x % 2 == 0)
{
    System.out.println("The number is even.");
}
```

Program Errors:

In Java, program errors can be broadly categorized into three types: syntax errors, runtime errors, and logic errors. Let's explore each type in detail:

1. **Syntax Errors:**

   Description: Syntax errors occur during the compilation phase when the Java compiler detects a violation of the language syntax rules. These errors prevent the program from being successfully compiled.

   Example:

```
public class SyntaxErrorExample {
    public static void main(String[] args) {
        // Missing semicolon
        System.out.println("Hello, world!")
    }
}
```

- **Resolution:** Carefully review the code for typos, missing semicolons, parentheses, or other syntax-related issues and correct them.

2. **Runtime Errors:**
   Description: Runtime errors occur during the execution of a program. These errors are not detected by the compiler but rather by the Java Virtual Machine (JVM) at runtime. Common runtime errors include division by zero, array index out of bounds, and null pointer exceptions.

Example:

```
public class RuntimeErrorExample {
    public static void main(String[] args) {
        // Division by zero
        int result = 5 / 0;
    }
}
```

Resolution: Add appropriate error-checking mechanisms, such as conditional statements or exception handling, to prevent or handle runtime errors.

3. **Logic Errors:**

Description: Logic errors occur when the program runs successfully, but it does not produce the expected output due to incorrect program logic. These errors are often subtle and may not result in immediate failure.

Example:

```
public class LogicErrorExample {
 public static void main(String[] args) {
     // Incorrect logic: sum should be subtraction
     int result = add(5, 3);
     System.out.println("Result: " + result);
 }

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

Resolution: Review the program's logic and correct any mistakes in the algorithm or calculations.

4. **Exception Handling:**

Description: Exception handling is a mechanism to deal with runtime errors by explicitly specifying how the program should respond to exceptional conditions.

Example:

```
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        try {
            // Division by zero
            int result = divide(5, 0);
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static int divide(int a, int b) {
        return a / b;
    }
}
```

- **Resolution:** Implement appropriate try-catch blocks to catch and handle specific exceptions, providing a graceful way to respond to errors.