ICSE Class 12 Computer Science

Chapter 9: Methods

Introduction

In object-oriented programming (OOP), methods play a crucial role in encapsulating behavior within classes. A method is a set of code that performs a specific task. In Java, which is the language prescribed for ICSE Class 12 Computer Science, methods are used to define the actions that an object can perform. This chapter covers the definition, declaration, invocation, overloading, and scope of methods in Java.

Understanding methods is key to writing modular, readable, and reusable code. By the end of this chapter, students should be able to define and use different types of methods effectively in their programs.

Detailed Chapter Content

---

1. Definition of a Method

A **method** is a block of code that executes when it is called or invoked. It helps in code reusability and modular programming.

Syntax:

```
javaCopyEditreturnType methodName(parameterList) {
    // body of method
    // return statement if required
}
```

---

2. Method Declaration

The basic components of a method declaration:

- **Access Specifier**: Determines the visibility (e.g., `public`, `private`, `protected`)
- **Return Type**: Type of data the method returns (e.g., `int`, `double`, `void`)
- **Method Name**: Identifier used to call the method
- **Parameters**: Optional input to the method

- **Method Body**: Block of code executed when the method is called

Example:

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

---

3. Method Calling

Methods are called using the object of the class (for non-static methods) or directly (for static methods).

Example:

```
javaCopyEdit// Calling a static method
int sum = add(5, 10);

// Calling a non-static method
MyClass obj = new MyClass();
obj.display();
```

---

4. Types of Methods

1. **Predefined Methods**: Methods already defined in Java (e.g., `System.out.println()`, `Math.sqrt()`)
2. **User-defined Methods**: Methods created by the programmer to perform custom tasks

---

5. Return Type of a Method

A method can return a value using the `return` keyword. If no value is returned, `void` is used as the return type.

Example:

```
javaCopyEditdouble area(double radius) {
    return 3.14 * radius * radius;
}
```

---

6. Parameter Passing

Java supports **pass-by-value**. This means that the actual parameter's value is copied into the method's parameter.

Types of Parameters:

- **Formal Parameters**: Defined in the method declaration
- **Actual Parameters**: Values passed when the method is called

Example:

```
javaCopyEditvoid greet(String name) {
    System.out.println("Hello " + name);
}

greet("Alice"); // actual parameter
```

---

7. Method Overloading

Method overloading allows the same method name to be used with different parameter lists.

Rules of Overloading:

- Must differ in the number or type of parameters
- Cannot overload just by return type

Example:

```
javaCopyEditint multiply(int a, int b) {
    return a * b;
}

double multiply(double a, double b) {
    return a * b;
}
```

---

8. Static and Non-static Methods

- **Static Methods**: Belong to the class rather than an instance; can be called without creating an object
- **Non-static Methods**: Belong to objects and require an object to be called

Example:

```
javaCopyEdit// Static method
static void showMessage() {
    System.out.println("Hello!");
}

// Non-static method
void display() {
```

```
    System.out.println("This is a non-static method");
}
```

---

9. Scope and Lifetime of Variables in Methods

- **Local Variables**: Declared inside a method and accessible only within it
- **Instance Variables**: Declared inside a class but outside any method; accessible to all methods of the class

---

10. Recursion

A method that calls itself is known as a **recursive method**.

Example: Factorial Using Recursion

```
javaCopyEditint factorial(int n) {
    if (n == 0) return 1;
    else return n * factorial(n - 1);
}
```

---

11. Void Methods

Void methods do not return any value. They are used for actions, not calculations.

Example:

```
javaCopyEditvoid printWelcome() {
    System.out.println("Welcome to Computer Science!");
}
```

---

12. Best Practices While Using Methods

- Keep methods short and focused on one task
- Use meaningful names
- Avoid side-effects in methods
- Limit the number of parameters (ideally less than 5)
- Document with comments if necessary

Summary

- Methods encapsulate behavior and promote modular programming.
- A method has a return type, name, and (optional) parameters.
- Methods can be **static** or **non-static**, **void** or **returning**.
- Java supports **method overloading** but not based solely on return types.
- Methods can be **recursive** and can help break down problems into smaller tasks.
- Using methods efficiently enhances code readability, maintainability, and reusability.
