Types of Fields,Types of Methods,this Keyword, and this() Method

-K.L.MADHAVI

Java Fields and their Types

Fields are variables declared within a class. They store data associated with objects of that class. Fields can be of various data types, including primitive types like `int` and `double`, or reference types like `String` and `ArrayList`. The scope of a field defines where it can be accessed within the class.

The syntax for declaring a Java field involves various components, each of which serves a specific purpose:

```
[access_modifier] [static] [final] type name [= initial value] ;
```

Continue…

Access Modifiers
Control the visibility of a field.
- public
- private
- protected

Data Types
Determine the type of data a field can store.
- int
- double
- String
- ArrayList

Field Names
Identify a specific field within a class.
- age
- name

Final Fields and Constants

Final fields
Final fields can only be assigned a value once, making them immutable. Cannot be modified after initialization. Used for representing constant values.We can assign a value to a final variable at runtime, but once assigned, it cannot be changed.

Constants
Typically declared as `public static final`. Used to represent values that never change.These must be assigned a value at compile time and cannot be changed at runtime.

Static vs. Non-static Fields

Java fields can either be static or non-static..

1.Static Fields: A static field belongs to the class itself rather than instances (objects) of the class. This means that all objects of the class share the same static field and its value is consistent across all instances. Static fields are declared using the static keyword.
Shared across all objects of the class.

2.Non-static Fields: Non-static fields are specific to individual instances (objects) of the class. Each object of the class has its copy of non-static fields, and their values can vary from one instance to another. Non-static fields are declared without the static keyword.
Unique to each object of the class.

Example

Consider a class `Employee` with a `company` field. `company` is static, shared by all employees, while `salary` is instance or non-static, unique per employee.

Types of Methods

In java the methods can be classified into different types based on their functionality and purpose:

1. Instance Methods: These methods are associated with an instance of a class and can access instance variables and other instance methods directly. They are typically used to perform actions specific to an individual object.

2. Static Methods: Static methods are associated with the class itself rather than with any specific instance of the class. They are invoked using the class name and are often used for utility functions or tasks that do not require access to instance-specific data.

3. Abstract Methods: These methods are declared in an abstract class or interface but do not contain implementations.The body of abstract method can be implemented in the derived class.

4. Overloaded Methods: When a class has more than one method having the same name but different parameters, it is known as method overloading. This allows a class to have more than one method having the same name if their parameter lists are different.

Continue…

5. Overridden Methods: When a method in a subclass has the same name, type, and parameters as a method in its superclass, then the method in the subclass is said to override the method in the superclass. This is known as method overriding.

Example:

```
    // Instance method
    public class Animal {
           public void makeSound() {
               System.out.println("Some sound");
           }
}
     // static method
    public static int addNumbers(int a, int b) {
        return a + b;
    }
```

```
// Abstract method example in an interface
    interface Activity {
        void perform();
    }
```

Continue…

```
    // **Overloaded methods**
    public void move() {
        System.out.println("Moving forward");
    }
    public void move(int speed) {
        System.out.println("Moving forward at speed: " + speed);
    }
}
```

```
// **Overridden Methods**
     class P
     {
         void add()
        {
          System.out.print(“addition1”);
        }
    }
    class C extends P
    {
          void add()
           {
              System.out.println(“addition2”);
            }
}
```

this Keyword

The this keyword in Java refers to the current object. It can be used within a method to access the object's instance variables and methods.
For example, this.name would refer to the name of the current object.

Example:

```
class Example {
    int x;


    Example(int x) {
        this.x = x;  // 'this.x' refers to the instance variable, 'x' is the parameter
    }
}
```

this() method

The this() method in Java is used to call one constructor from another within the same class, facilitating constructor chaining.It must be the first statement in the constructor and helps in reducing code duplication.

Example:

```
class Example {
    int x, y;

    Example(int x) {
        this(x, 0);  // Calls the constructor with two parameters
    }

    Example(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```