Java Classes and Objects

Here's a table of contents for Java Classes and Objects:

Table of Contents

1. **Introduction to Classes**
    - 1.1 What is a Class?
    - 1.2 Why Use Classes?
    - 1.3 Example: Defining a Class
2. **Introduction to Objects**
    - 2.1 What is an Object?
    - 2.2 Creating and Using Objects
    - 2.3 Example: Creating and Using an Object
3. **Constructors**
    - 3.1 What is a Constructor?
    - 3.2 Types of Constructors
    - 3.3 Example: Using Constructors
    - 3.4 Example: Creating Objects with Constructors
4. **Access Modifiers**
    - 4.1 What are Access Modifiers?
5. **Static Members**
    - 5.1 What are Static Members?
    - Example: Using Static Members
6. **Object References and Passing Objects**
    - 6.1 Object References
7. **Object Comparison**
    - 7.1 Comparing Objects
8. **Additional Common Questions**
    - 8.1 Differences Between Classes and Objects
    - 8.2 Common Questions

---

1. Introduction to Classes

1.1 What is a Class?

A class in Java is a blueprint from which individual objects are created. It defines the data and methods that will be used to create objects.

1.2 Why Use Classes?

- **Encapsulation**: Classes bundle data (attributes) and methods (functions) into a single unit.
- **Modularity**: Classes help break down a complex system into manageable pieces.
- **Code Reusability**: Once a class is defined, it can be used to create multiple objects with similar properties.

1.3 Example: Defining a Class

```
//* Car.java
public class Car {
    // Attributes
    String color;
    String model;
    int year;

    // Method to display car details
    void displayInfo() {
        System.out.println("Model: " + model);
        System.out.println("Color: " + color);
        System.out.println("Year: " + year);
    }
}
```

Explanation:

- **Attributes**: Variables that represent the state of an object.
- **Methods**: Functions that define the behavior of the class.

2. Introduction to Objects

2.1 What is an Object?

An object is an instance of a class. It represents a concrete realization of the class with specific values for its attributes.

2.2 Creating and Using Objects

You create an object using the `new` keyword followed by the class constructor. You can then access the object's methods and attributes.

2.3 Example: Creating and Using an Object

```
public class Main {
    public static void main(String[] args) {
        // Create an object of the Car class
```

```
        Car myCar = new Car();

        // Set attributes
        myCar.color = "Red";
        myCar.model = "Toyota Corolla";
        myCar.year = 2020;

        // Call method to display car details
        myCar.displayInfo();
    }
}
```

Output:

```
Model: Toyota Corolla
Color: Red
Year: 2020
```

3. Constructors

3.1 What is a Constructor?

A constructor is a special method that is called when an object is instantiated. It initializes the object's attributes.

3.2 Types of Constructors

- **Default Constructor**: A no-argument constructor provided by Java if no constructors are defined.
- **Parameterized Constructor**: A constructor that accepts arguments to initialize the object's attributes.

3.3 Example: Using Constructors

```
//* Car.java
public class Car {
    // Attributes
    String color;
    String model;
    int year;

    // Default constructor
    public Car() {
        color = "Unknown";
        model = "Unknown";
        year = 0;
    }

    // Parameterized constructor
    public Car(String color, String model, int year) {
        this.color = color;
```

```
        this.model = model;
        this.year = year;
    }

    // Method to display car details
    void displayInfo() {
        System.out.println("Model: " + model);
        System.out.println("Color: " + color);
        System.out.println("Year: " + year);
    }
}
```

3.4 Example: Creating Objects with Constructors

```
public class Main {
    public static void main(String[] args) {
        // Create objects using different constructors
        Car defaultCar = new Car();
        Car myCar = new Car("Blue", "Honda Civic", 2022);

        // Display car details
        System.out.println("Default Car:");
        defaultCar.displayInfo();
        System.out.println("\nMy Car:");
        myCar.displayInfo();
    }
}
```

Output:

```
Default Car:
Model: Unknown
Color: Unknown
Year: 0

My Car:
Model: Honda Civic
Color: Blue
Year: 2022
```

4. Access Modifiers

4.1 What are Access Modifiers?

Access modifiers determine the visibility of classes, methods, and attributes. They include:

- `public`: Accessible from anywhere.
- `private`: Accessible only within the class.

- `protected`: Accessible within the same package and subclasses.
- **Default (no modifier)**: Accessible only within the same package.

Example: Using Access Modifiers

```
//* Person.java
public class Person {
    // Attributes
    public String name;
    private int age;
    protected String address;

    // Constructor
    public Person(String name, int age, String address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }

    // Public method
    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Address: " + address);
    }

    // Private method
    private void incrementAge() {
        age++;
    }
}
```

5. Static Members

5.1 What are Static Members?

Static members belong to the class rather than any object instance. They are shared among all instances of the class.

Example: Using Static Members

```
//* MathUtils.java
public class MathUtils {
    // Static attribute
    public static final double PI = 3.14159;

    // Static method
    public static int square(int number) {
        return number * number;
```

```
    }
}
```

Usage:

```
public class Main {
    public static void main(String[] args) {
        // Access static attribute
        System.out.println("Value of PI: " + MathUtils.PI);

        // Call static method
        int result = MathUtils.square(5);
        System.out.println("Square of 5: " + result);
    }
}
```

Output:

```
Value of PI: 3.14159
Square of 5: 25
```

6. Object References and Passing Objects

6.1 Object References

Objects are passed by reference, meaning that methods receive a reference to the original object, not a copy.

Example: Passing Objects to Methods

```
//* Person.java
public class Person {
    // Attributes
    String name;

    // Constructor
    public Person(String name) {
        this.name = name;
    }

    // Method to update name
    public void updateName(String newName) {
        name = newName;
    }

    // Method to display name
    public void displayInfo() {
        System.out.println("Name: " + name);
```

```
    }
}

//* Main.java
public class Main {
    public static void main(String[] args) {
        // Create an object
        Person person = new Person("John");

        // Pass the object to a method
        updatePersonName(person);

        // Display updated name
        person.displayInfo();
    }

    public static void updatePersonName(Person p) {
        p.updateName("Jane");
    }
}
```

Output:

```
Name: Jane
```

7. Object Comparison

7.1 Comparing Objects

Objects are compared using the `==` operator for reference equality (whether they point to the same object) and the `.equals()` method for logical equality (whether their states are equal).

Example: Comparing Objects

```
//* Person.java
public class Person {
    String name;

    // Constructor
    public Person(String name) {
        this.name = name;
    }

    // Override equals() method
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
```

```
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Person person = (Person) obj;
        return name.equals(person.name);
    }
}

//* Main.java
public class Main {
    public static void main(String[] args) {
        // Create objects
        Person person1 = new Person("Alice");
        Person person2 = new Person("Alice");
        Person person3 = new Person("Bob");

        // Compare objects
        System.out.println(person1.equals(person2)); // true
        System.out.println(person1.equals(person3)); // false
    }
}
```

Output:

```
true
false
```

8. Additional Common Questions

8.1 Differences Between Classes and Objects

Aspect	Class	Object
Definition	Blueprint/template for creating objects	Instance of a class
Purpose	Defines structure and behavior	Represents a specific entity with defined attributes
Creation	Defined using the `class` keyword	Created using the `new` keyword followed by the class constructor
State	Does not hold data; defines data fields	Holds actual data values in its attributes
Usage	Used to create objects and define methods	Used to perform actions and store data

Aspect	Class	Object
Access Modifiers	Determines visibility of class members	Accessed via class methods and attributes
Static Members	Shared across all instances	Not shared; specific to each instance

8.2 Common Questions

Questions	Answers
Why do we need classes?	Classes provide a way to bundle data and methods into a single unit, promoting modularity and reusability in programming. They help in organizing code and managing complex systems.
What is the difference between a class and an object?	A class is a blueprint or template for creating objects, whereas an object is an instance of a class. A class defines attributes and methods, but an object holds actual values and interacts with those methods.
What are constructors?	Constructors are special methods used to initialize objects. They set up the initial state of an object and are called when an object is created. There are default constructors (provided by Java) and parameterized constructors (defined by the programmer).
What are static members?	Static members belong to the class rather than any object instance. They are shared across all instances and can be accessed without creating an object of the class. Static members are used for constants and utility methods that do not depend on instance-specific data.
Why use access modifiers?	Access modifiers control the visibility of class members, allowing you to protect data and encapsulate functionality. They help enforce encapsulation and prevent unauthorized access or modification of class attributes and methods.
How do we compare objects?	Objects can be compared using the `==` operator for reference equality and the `.equals()` method for logical equality. The `==` operator checks if two references point to the same object, while `.equals()` checks if two objects have the same logical state.