Static vs. Instance Methods

Static Methods:

- **Definition**: Static methods belong to the class rather than any specific instance of the class. This means you can call a static method without creating an object of the class.
- **Usage**: Useful when the behavior of the method is not dependent on object states (e.g., utility functions like `Math.sqrt()`).
- **Characteristics**:
  - Cannot access instance variables directly.
  - Accessed directly using the class name (e.g., `ClassName.methodName()`).

Instance Methods:

- **Definition**: Instance methods are tied to a specific instance of the class, allowing them to operate on instance variables specific to that object.
- **Usage**: Ideal for behaviors that require information stored in object properties.
- **Characteristics**:
  - Can access both instance and static variables.
  - Called on an object instance (e.g., `objectName.methodName()`).

Example Program:

```
class Example {
    static void staticMethod() {
        System.out.println("Static method called.");
    }

    void instanceMethod() {
        System.out.println("Instance method called.");
    }

    public static void main(String[] args) {
        Example.staticMethod(); // Calls static method
        Example obj = new Example();
        obj.instanceMethod(); // Calls instance method
    }
}
```

2. Static vs. Instance Variables

Static Variables:

- **Definition**: Static variables are shared across all instances of a class; they are created once and have a single memory allocation.
- **Usage**: Useful for constants or data that should be consistent across all instances (e.g., `count` of created objects).
- **Example**: `public static int count = 0;`

Instance Variables:

- **Definition**: Instance variables are unique to each instance of a class and represent the object's state.
- **Usage**: Used to store data specific to each object.

Example Program:

```
class Counter {
    static int staticCount = 0;
    int instanceCount = 0;

    Counter() {
        staticCount++;
        instanceCount++;
    }

    void display() {
        System.out.println("Static Count: " + staticCount);
        System.out.println("Instance Count: " + instanceCount);
    }

    public static void main(String[] args) {
        Counter c1 = new Counter();
        Counter c2 = new Counter();
        c1.display();
        c2.display();
    }
}
```

3. The `Object` Class as a Parent Class

- **Definition**: In Java, all classes inherit from the `Object` class by default, making it the root of the class hierarchy.
- **Purpose**: Provides basic methods that all Java objects can use, such as `toString()`, `hashCode()`, and `equals()`.
- **Example Usage of `toString()`**:
    - By default, `toString()` returns the class name and hashcode.
    - You can override `toString()` to provide a meaningful string representation of an object.

Example Program:

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
```

```
    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }

    public static void main(String[] args) {
        Person person = new Person("Alice", 30);
        System.out.println(person); // Implicitly calls person.toString()
    }
}
```

4. Inheritance Revisited

Concepts Revisited:

- **Method Overriding**: Allows a subclass to provide a specific implementation of a method already defined in its superclass.
- **Static Methods in Inheritance**: Static methods are not inherited in the usual sense and cannot be overridden but can be hidden.
- **Static Variables in Inheritance**: Static variables are shared across all instances of the class and subclass.

Example Program Using Inheritance:

```
class Animal {
    static String kingdom = "Animalia";
    String name;

    Animal(String name) {
        this.name = name;
    }

    static void displayKingdom() {
        System.out.println("Kingdom: " + kingdom);
    }

    void sound() {
        System.out.println("Animal makes sound");
    }
}

class Dog extends Animal {
    Dog(String name) {
        super(name);
    }

    @Override
    void sound() {
        System.out.println("Dog barks");
    }

    static void displayKingdom() {
```

```
        System.out.println("Dog is also part of the kingdom: " + kingdom);
    }

    public static void main(String[] args) {
        Animal animal = new Animal("Generic Animal");
        Dog dog = new Dog("Buddy");

        animal.sound(); // Calls Animal's sound
        dog.sound();    // Calls Dog's overridden sound

        Animal.displayKingdom(); // Calls static method from Animal
        Dog.displayKingdom();    // Calls static method from Dog
    }
}
```