Method Overriding in Java - All Rules with Examples

What is Method Overriding in Java?

Method overriding in Java occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. The method in the child class must have the same name, return type, and parameters as in the parent class.

It enables runtime polymorphism and dynamic method dispatch.

Real-World Example:

```
class Bank {
    public double getInterestRate() {
        return 5.0;
    }
}

class SBI extends Bank {
    @Override
    public double getInterestRate() {
        return 6.0;
    }
}

class HDFC extends Bank {
    @Override
    public double getInterestRate() {
        return 6.5;
    }
}

// Runtime Polymorphism
public class Test {
    public static void main(String[] args) {
        Bank bank = new HDFC();
        System.out.println(bank.getInterestRate()); // Output: 6.5
    }
}
```

1. Same Method Signature

```
class Parent {
    public void show() {
        System.out.println("Parent");
    }
}
class Child extends Parent {
    @Override
    public void show() {
        System.out.println("Child");
    }
}
```

2. Access Modifier Rule

```
class Parent {
    public void display() {
        System.out.println("Public method in Parent");
    }
}
class Child extends Parent {
    @Override
    public void display() {
        System.out.println("Public method in Child");
    }
}
```

3. Final Methods Cannot Be Overridden

```
class Parent {
    public final void finalMethod() {
        System.out.println("Final method");
    }
}
class Child extends Parent {
    // Cannot override final method
}
```

4. Static Methods Are Not Overridden (Method Hiding)

```
class Parent {
    public static void staticMethod() {
        System.out.println("Static in Parent");
    }
}
class Child extends Parent {
    public static void staticMethod() {
        System.out.println("Static in Child");
    }
}
```

5. Constructors Cannot Be Overridden

```
class Parent {
    Parent() {
        System.out.println("Parent constructor");
    }
}
class Child extends Parent {
    Child() {
        System.out.println("Child constructor");
    }
}
```

6. Covariant Return Type

```
class Animal {}
class Dog extends Animal {}
class Parent {
    Animal getAnimal() {
        return new Animal();
    }
}
class Child extends Parent {
    @Override
    Dog getAnimal() {
        return new Dog();
    }
}
```

7. Exception Handling Rule

```
class Parent {
    public void test() throws IOException {
        System.out.println("Parent method");
    }
}
class Child extends Parent {
    @Override
    public void test() throws FileNotFoundException {
        System.out.println("Child method");
    }
}
```

8. Use of @Override Annotation

```
class Parent {
    public void greet() {}
}
class Child extends Parent {
    @Override
    public void greet() {
        System.out.println("Hello from Child");
    }
}
```

9. Private Methods Cannot Be Overridden

```
class Parent {
    private void secret() {
        System.out.println("Parent secret");
    }
}
class Child extends Parent {
    private void secret() {
        System.out.println("Child secret");
    }
}
```

10. Abstract Methods Must Be Overridden

```
abstract class Animal {
    abstract void makeSound();
}
class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Bark");
    }
}
```