Super Keyword Practice Problems

Problem 1: Library Management System (Single Inheritance + Method Overriding)

Scenario: Create a library management system with books and e-books.

Requirements:

1. Create a `Book` class with:
    - Fields: `title`, `author`, `price`, `pages`
    - Constructors: default and parameterized
    - Methods: `displayInfo()`, `calculateLateFee()`, `getReadingTime()`
2. Create an `EBook` class extending `Book` with:
    - Additional fields: `fileSize`, `downloadLink`
    - Constructor overloading (3 different constructors)
    - Override `displayInfo()` using `super.displayInfo()` first, then add e-book specific info
    - Override `calculateLateFee()` (e-books have different fee structure)
    - Override `getReadingTime()` (e-books are read 20% faster)
3. **Super keyword usage:**
    - Use `super()` in all EBook constructors
    - Use `super.method()` in overridden methods
    - Access parent variables using `super.variableName`

Test cases:

- Create objects using different constructors
- Call overridden methods to see parent method called first
- Show late fee calculation differences

Problem 2: Restaurant Management (Multilevel Inheritance + Constructor Chaining)

Scenario: Create a restaurant system with Food → MainCourse → PremiumDish hierarchy.

Requirements:

1. Create `Food` class:
    - Fields: `name`, `basePrice`, `preparationTime`
    - Multiple constructors (overloading)
    - Methods: `prepare()`, `serve()`, `calculateTotalPrice()`
2. Create `MainCourse` class extending `Food`:

- Additional fields: `portion`, `spiceLevel`
- Constructor overloading with `super()` calls
- Override `prepare()` method using `super.prepare()` first
- Method overloading: `season()` with different parameters

3. Create `PremiumDish` class extending `MainCourse`:

- Additional fields: `chefSpecial`, `presentationStyle`
- All constructors must use `super()` appropriately
- Override `serve()` calling parent's version first
- Override `calculateTotalPrice()` adding premium charges

Super keyword focus:

- Demonstrate 3-level constructor chaining
- Show method calls flowing through inheritance hierarchy
- Access grandparent methods through inheritance

Test cases:

- Create PremiumDish objects and trace constructor calls
- Call methods to see multilevel inheritance in action

Problem 3: Vehicle Manufacturing (Hierarchical Inheritance + Method Overloading)

Scenario: Create a vehicle manufacturing system with one parent and multiple child classes.

Requirements:

1. Create `Vehicle` base class:

- Fields: `make`, `model`, `year`, `price`
- Constructor overloading (3 constructors)
- Methods: `start()`, `displaySpecs()`, `calculateInsurance()`
- Method overloading: `calculateTax()` with different parameters

2. Create `Car` class extending `Vehicle`:

- Additional fields: `doors`, `fuelType`
- Constructor using `super()` appropriately
- Override `start()` using `super.start()` first
- Method overloading: `park()` with different parameters
- Override `calculateInsurance()` with car-specific logic

3. Create `Motorcycle` class extending `Vehicle`:

- Additional fields: `engineCC`, `hasStorage`
- Constructor using `super()` appropriately
- Override `start()` using `super.start()` first
- Override `displaySpecs()` calling parent version first

- Method overloading: `wheelie()` with different parameters

4. Create `Truck` class extending `Vehicle`:
   - Additional fields: `loadCapacity`, `axles`
   - Constructor using `super()` appropriately
   - Override all inherited methods using `super` appropriately
   - Method overloading: `loadCargo()` with different parameters

Super keyword requirements:

- Each child constructor must call parent constructor
- All overridden methods must call parent version using `super`
- Demonstrate accessing parent variables when child has same-named variables

Test cases:

- Create objects of all child classes
- Test overloaded methods in each class
- Show parent method execution before child-specific code

Problem 4: Banking System (Single + Method Overriding + Variable Access)

Scenario: Create a banking system with accounts and premium accounts.

Requirements:

1. Create `BankAccount` class:
   - Fields: `accountNumber`, `holderName`, `balance`, `interestRate`
   - Constructor overloading (default, with name, with all details)
   - Methods: `deposit()`, `withdraw()`, `calculateInterest()`, `displayAccount()`
   - Method overloading: `transfer()` with different parameters

2. Create `PremiumAccount` class extending `BankAccount`:
   - Fields: `accountNumber` (same name as parent), `creditLimit`, `rewardPoints`
   - Constructor overloading ensuring `super()` usage
   - Override `deposit()` - call `super.deposit()` then add reward points
   - Override `withdraw()` - call parent method, add premium features
   - Override `displayAccount()` - show parent info using `super.displayAccount()`, then premium details
   - Method overloading: `redeemRewards()` with different parameters

Super keyword challenges:

- Handle variable name conflicts using `super.variableName`
- Chain constructor calls properly
- Extend parent functionality without losing original behavior
- Access parent's version of overridden methods

Test cases:

- Create accounts with different constructors
- Test deposit/withdraw showing parent method calls
- Display account info showing both parent and child data
- Demonstrate variable access conflicts resolution

Problem 5: Educational Institution (Multilevel + Complex Super Usage)

Scenario: Create an educational system with Person → Student → GraduateStudent hierarchy.

Requirements:

1. Create `Person` class:
    - Fields: `name`, `age`, `id`, `address`
    - Constructor overloading (3 different constructors)
    - Methods: `introduce()`, `updateInfo()`, `getDetails()`
2. Create `Student` class extending `Person`:
    - Fields: `id` (same as parent), `course`, `semester`, `gpa`
    - Constructor overloading with proper `super()` calls
    - Override `introduce()` calling `super.introduce()` first
    - Override `getDetails()` using parent version then adding student info
    - Method overloading: `study()` with different parameters
    - Method overloading: `takeExam()` with different parameters
3. Create `GraduateStudent` class extending `Student`:
    - Fields: `researchArea`, `advisor`, `thesis`
    - Constructor overloading with proper inheritance chain
    - Override `introduce()` - call parent's version, add graduate info
    - Override `study()` - use `super.study()` then add research activities
    - Method overloading: `conductResearch()` with different parameters
    - Special method: `showInheritanceChain()` - demonstrate accessing methods from all levels

Complex Super keyword usage:

- 3-level constructor chaining
- Method calls flowing through multiple inheritance levels
- Variable name conflict resolution across multiple levels
- Accessing grandparent methods through inheritance chain

Test cases:

- Create GraduateStudent objects using different constructors
- Call overridden methods to trace execution through inheritance hierarchy
- Test variable access at different inheritance levels
- Demonstrate complete inheritance chain functionality

General Implementation Guidelines:

1. **Constructor Rules:**
   - Every child class constructor must call `super()` as first statement
   - Demonstrate both explicit and implicit `super()` calls
   - Show constructor parameter passing through inheritance chain
2. **Method Overriding Rules:**
   - Use `@Override` annotation
   - Call parent method using `super.methodName()` where appropriate
   - Extend functionality, don't completely replace it
3. **Variable Access:**
   - Create scenarios with same-named variables in parent and child
   - Use `super.variableName` to access parent variables
   - Demonstrate the difference between `this.variable` and `super.variable`
4. **Testing Requirements:**
   - Create comprehensive test cases for each problem
   - Show inheritance hierarchy in action
   - Demonstrate proper `super` keyword usage
   - Include edge cases and error scenarios