Vending Machine Program Flowchart/Skeleton

```
1. Initialization:
   a. Load items from file into memory.
   b. Display welcome message.
   c. Display items and their prices.
   d. Display option to deposit money or exit.

2. User Actions:
   a. Deposit Money:
      i. Prompt for money deposit.
     ii. Store the deposited amount.

   b. Select Item:
      i. Prompt user to select an item.
     ii. Check if the item is available (inventory > 0).
         1. If no, display "Item out of stock" and return to main display.
    iii. Check if the user has deposited enough money.
         1. If insufficient, throw `InsufficientFundsException`.
            - Display "Insufficient funds" error message.
            - Display the amount the user deposited.
            - Return to main display.
         2. If sufficient:
            - Deduct item cost from deposited amount.
            - Update the item's inventory.
            - Display the change using the `Change` class.
            - Display "Thank you for your purchase" message.
            - Return to main display.
   c. Exit:
      i. Update items' file with the current inventory.
     ii. Display goodbye message.
    iii. End program.

3. Change Calculation:
   - Using the `Change` class:
      a. Convert the remaining deposited amount (in pennies) into quarters, dimes, nickels, and pennies.
      b. Display the calculated change.

4. Exception Handling:
   a. `InsufficientFundsException`:
      - Handled in the "Select Item" process.
   b. `NoItemInventoryException`:
      - Handled in the "Select Item" process.
   c. `FilePersistenceException`:
      - When there's an issue reading or writing to the file.
      - Display "Technical difficulty" error message.
      - Exit program gracefully.

5. End Program.
```

Development Guidelines:

1. Design First: Sketch out the program's structure and flow.
2. MVC Pattern:
   a. `VendingMachineApp` class as entry point.
   b. `Controller` for main business logic and flow.
   c. `View` for user interaction.
   d. `Service Layer` for high-level operations.
   e. `DAO` for data persistence.
3. Dependency Injection: Constructor based.
4. Use BigDecimal: For all monetary calculations.
5. Application Specific Exceptions: Ensure program fails gracefully.

```
Start
 |
 |
 v
[Initialization]
 |-> Load items from file into memory
 |-> Display welcome message
 |-> Display items and their prices
 |-> Display option to deposit money or exit
 |
 v
[User Actions]
 |-> (Option A) Deposit Money:
 |    |-> Prompt for money deposit
 |    |-> Store the deposited amount
 |
 |-> (Option B) Select Item:
 |    |-> Prompt user to select an item
 |    |    |-> Check if the item is available
 |    |    |    |-> If No: "Item out of stock" -> Main Display
 |    |    |    |-> If Yes: Check if enough money is deposited
 |    |    |        |-> If No: "Insufficient funds" -> Main Display
 |    |    |        |-> If Yes:
 |    |    |            - Deduct item cost
 |    |    |            - Update inventory
 |    |    |            - Display change using "Change" class
 |    |    |            - "Thank you for your purchase"
 |    |    |            -> Main Display
 |
 |-> (Option C) Exit:
 |    |-> Update item file with current inventory
 |    |-> Display goodbye message
 |
 v
[Change Calculation]
 |-> Convert remaining amount to quarters, dimes, nickels, pennies
 |-> Display calculated change
 |
 v
[Exception Handling]
 |-> InsufficientFundsException
 |   - Handle in "Select Item"
 |-> NoItemInventoryException
 |   - Handle in "Select Item"
 |-> FilePersistenceException
 |   - Display "Technical difficulty"
 |   - Exit program
 |
 v
End Program
```