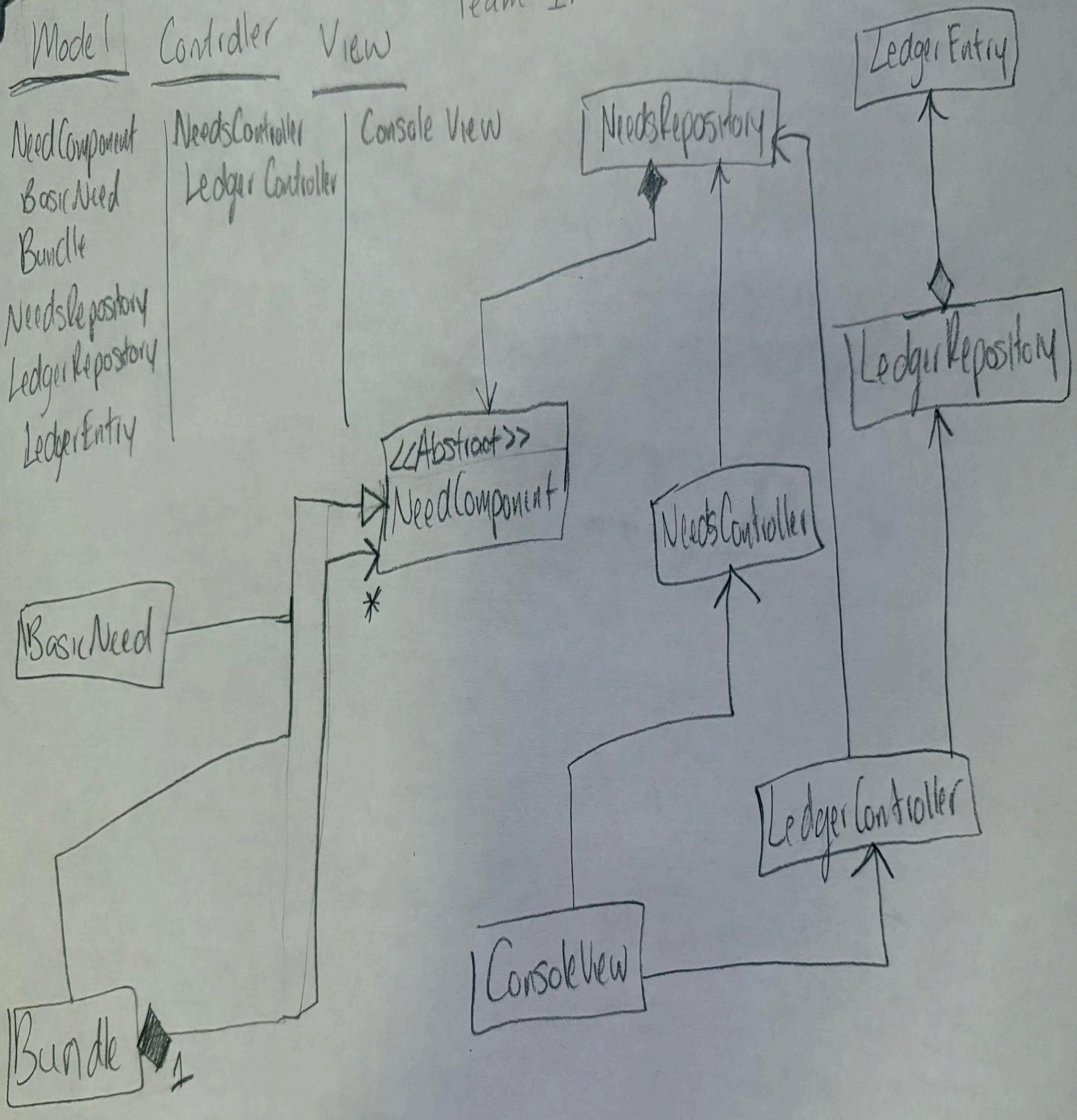
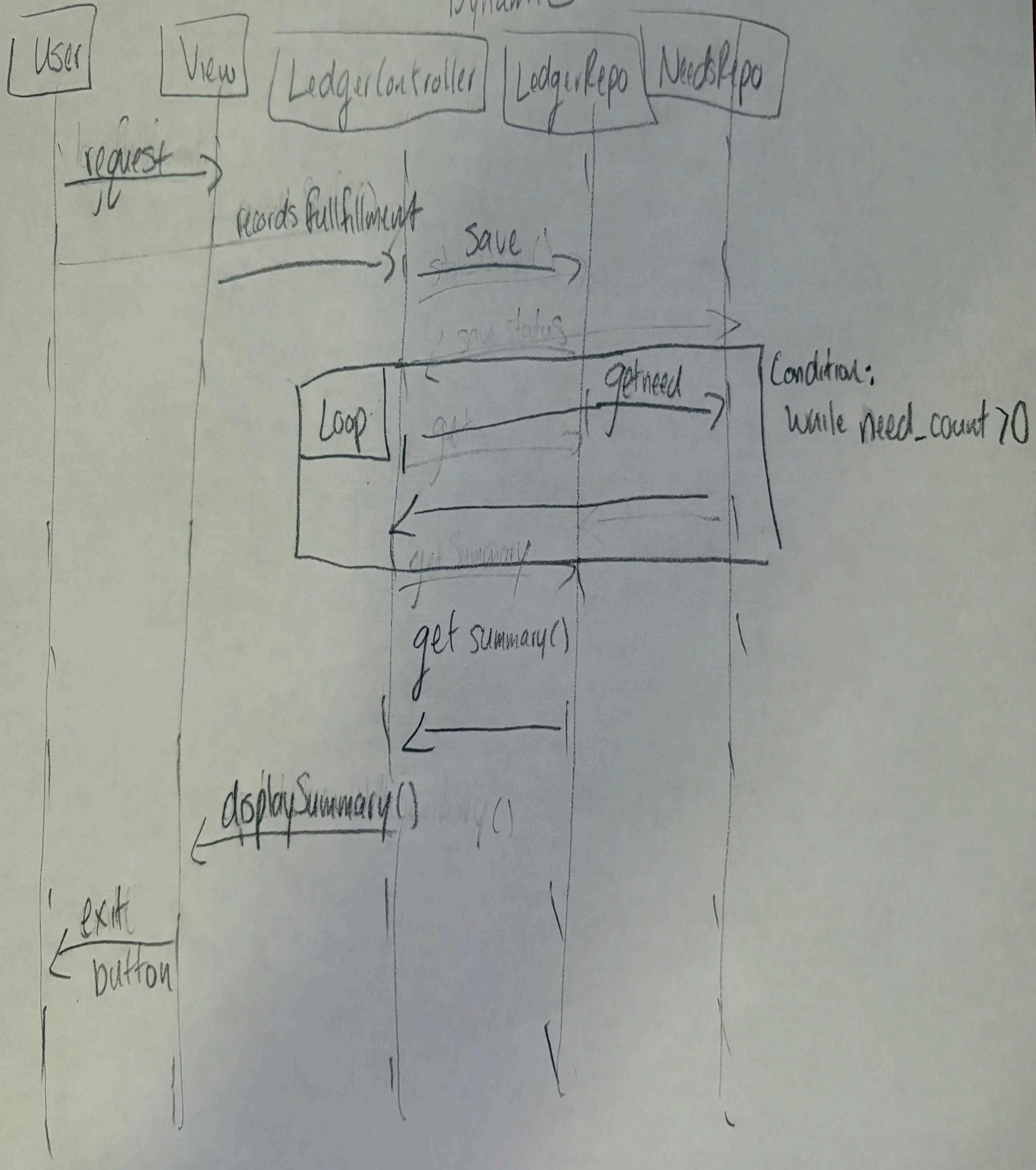
Design Sketch

Team 1A

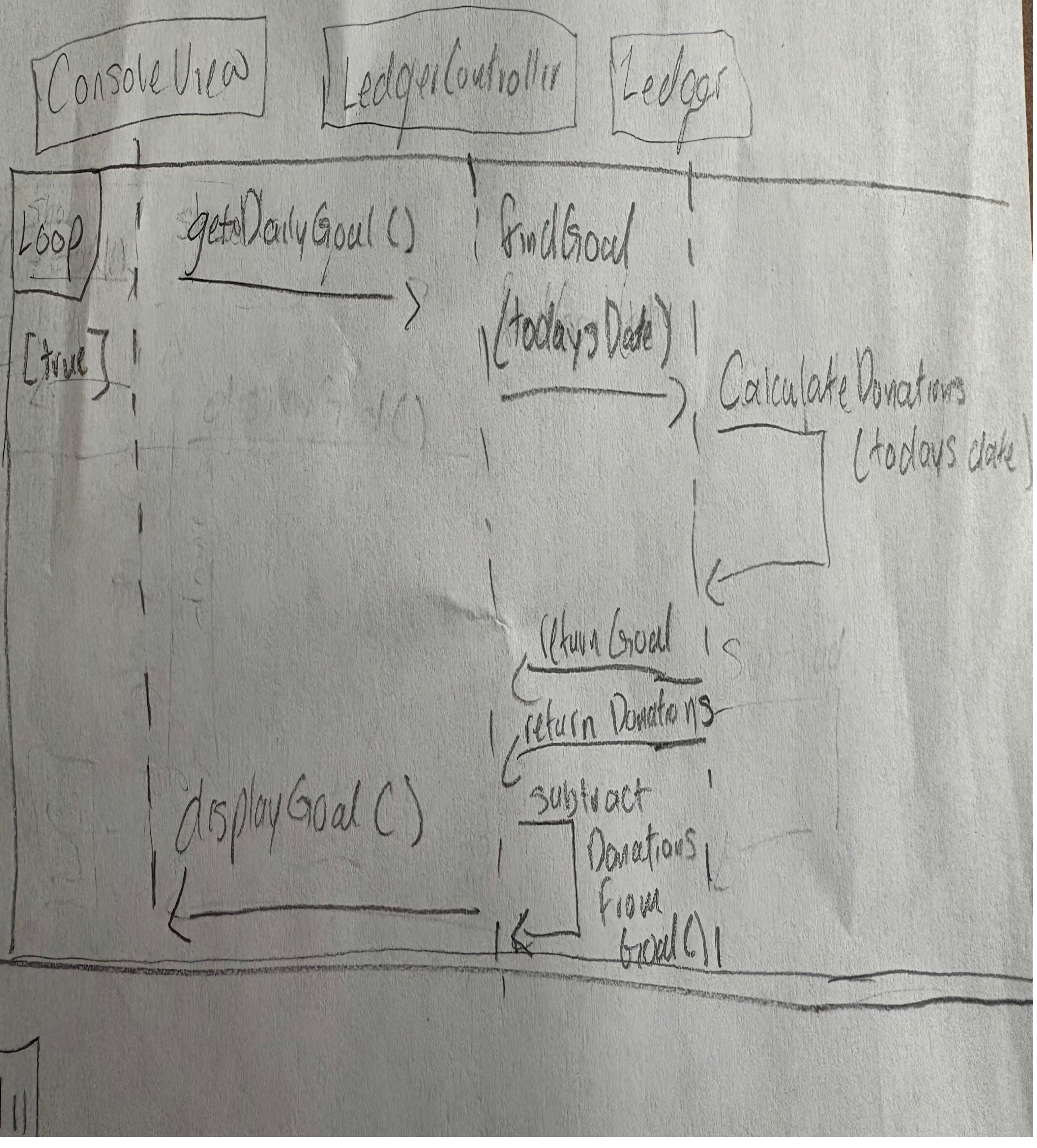
Model	Controller	View
NeedComponent	NeedsController	Console View
BasicNeed	Ledger Controller	
Bundle		
NeedsRepository		
LedgerRepository		
LedgerEntry		



Design Sketch
Team 1A
Dynamic



#3



ConsoleView
LedgerController
Ledger
Loop
[true]
getDailyGoal()
findGoal (todaysDate)
CalculateDonations (todays date)
return Goal
return Donations
subtract Donations From Goal()
displayGoal()

Note

Instances of the word ***Ledger or LedgerRepo*** in the diagrams refer to the ***Ledger Repository*** in our class diagram.

4. Narrative

4.1 - Brief description of each of the application classes

NeedComponent Class

Knows:

- Its `name` (identifier for the financial need)
- The interface for total, fixed, variable, and fee costs

Can do:

- Provide abstract methods for computing cost details
- Allow derived classes to calculate total costs in their own way.

BasicNeed Class

Knows:

- `name` (e.g., “Monthly Rent”).
- The dollar values for `total`, `fixed`, `variable`, and `fees`.

Can do:

- Return total or individual cost components.
- Provide its cost breakdown directly (no subcomponents).
- Serialize/deserialize itself to/from a CSV line.

Bundle Class

Knows:

- `name` of the bundle (e.g., “Monthly Essentials”).

- A list of `(NeedComponent, count)` pairs that make up the bundle.

Can do:

- Add or remove sub-needs (BasicNeeds or other Bundles).
- Compute total, fixed, variable, and fee costs by summing across all components Composite pattern).
- Serialize/deserialize itself to/from a CSV line (respecting “no forward reference” rule).

NeedsRepository Class

Knows:

- All `NeedComponent` objects (both basic needs and bundles), stored in a dictionary keyed by name.
- The location and format of the `needs.csv` file.

Can do:

- Load all needs and bundles from `needs.csv` while enforcing ordering and uniqueness.
- Save current needs back to the CSV.
- Add, retrieve, or delete needs by name.
- Resolve bundle components when loading.

LedgerEntry Class

Knows:

- The `date` of the entry.
- The `type` of record (`'f'` for funds, `'g'` for goal, `'n'` for fulfilled need).
- The relevant `name` (if applicable).
- The `value` (amount, goal, or quantity fulfilled).

Can do:

- Identify what type of entry it represents.
- Provide its stored data for processing or saving.
- Convert to and from a CSV line.

LedgerRepository Class

Knows:

- A list of all `LedgerEntry` objects currently recorded.
- The location and format of the `log.csv` file.

Can do:

- Load all ledger entries from `log.csv`.
- Save the entries to the file when requested.
- Add new ledger entries (funds, goals, or fulfillments).
- Retrieve all entries for a given date.
- Find the most recent funds or goal entry prior to a specific date.
- Compute totals for a date (via the controller).

NeedsController Class

Knows:

- The `NeedsRepository` that manages all needs and bundles.

Can do:

- Add new basic needs or bundles.
- Retrieve cost data for a specific need or bundle.
- Trigger saving or reloading of the repository file.
- Validate user input before creating needs or bundles.

LedgerController Class

Knows:

- The `LedgerRepository` for daily logs.
- The `NeedsRepository` for cost reference when computing totals.

Can do:

- Record funds, goals, and fulfilled needs for a given date.
- Retrieve and compute daily summaries (total spent, distribution, remaining funds).
- Determine if the user met their daily funding goal.
- Trigger saving or reloading of ledger data.

4.2 - Rationale for organizing the system

The system is organized using the **Model–View–Controller (MVC)** architecture and the **Composite pattern**, with clear separation of responsibilities among classes.

Benefits, a huge separation of concerns. The Model layer includes classes such as `NeedComponent`, `BasicNeed`, `Bundle`, `NeedsRepository`, `LedgerEntry`, and `LedgerRepository`, which manage all data and business logic. The Composite relationship between `Bundle` and `NeedComponent` allows bundles to contain both basic needs and other bundles, providing a flexible way to represent complex financial structures without duplicating logic. Repositories isolate file storage and retrieval, allowing other components to focus on logic and presentation. The Controllers (`NeedsController` and `LedgerController`) coordinate data flow and enforce rules between the Model and the user-facing View. Importantly, the `NeedsController` manages creation and validation of `BasicNeed` and `Bundle` objects.

Drawbacks, the MVC architecture requires a controller to be made for each view. In this case, if we add another logging element to the system we would need to build a controller for it. The repository classes must contain lots of data validation from the csv files as the logs are the main source of truth. If they contain errors then the system fails. The main drawback is that this design introduces more classes and relationships, which increases complexity and coordination effort.

5. Skeleton Classes

```java
public abstract class NeedComponent {
	protected String name;

	public NeedComponent(String name) {
		this.name = name;
	}

	public abstract double calculateTotalCost();
	public abstract double calculateFixedCost();
	public abstract double calculateVariableCost();
	public abstract double calculateFeeCost();

	public String getName() {
		return name:
	}
}
```

Java
```java
// Insert skeleton code here in this code block
public class BasicNeed extends NeedComponent {
      private final double fixedCost;
      private final double variableCost;
      private final double fees;

      public BasicNeed(String name, double, fixed, double variable, double
fees) {
            super(name);
            this.fixedCost = fixed;
            this.variableCost = variable;
            this.fees = fees;
      }

      @Override
      public abstract double calculateTotalCost() {
            return (fixedCost + variable + fees);
      }

      @Override
      public abstract double calculateFixedCost() {
            return fixedCost;
      }

      @Override
      public abstract double calculateVariableCost() {
            reutrn variableCost;
      }

      @Override
      public abstract double calculateFeeCost() {
            return fees;
      }
}
```

Java
```java
public class Bundle extends NeedComponent {
      private final List<String> componentNames = new ArrayList<>();
      private final List<Integer> componentCounts = new ArrayList<>();

      public Bundle(String name) {
            super(name);
```

```
        }

        publid void addComponent(String needName, int count) {
                this.componentNames.add(needName);
                this.componentCounts.add(count);
        }

        @Override
        public abstract double calculateTotalCost() {
                System.out.println("Calculating recursive cost for Bunlde: " +
name);

                return 0.0:
        }

        @Override
        public abstract double calculateFixedCost() {
                return 0.0:
        }

        @Override
        public abstract double calculateVariableCost() {
                return 0.0:
        }

        @Override
        public abstract double calculateFeeCost() {
                return 0.0:
        }
}
```

Java
```
public class LedgerEntity {
        public enum EntryType {FUND, GOAL, NEED}

        private final LocalDate date;
        private final EntryType type;
        private final String needName;
        private final double amount;
```

```
	private final int count;

	//Constructor for FUND/DOAL entries
	public LedgerEntity(LocalDate date, EntryType type, double amount) {
		this.date = date;
		this.type = type;
		this.amount = amount;
		this.needName = null;
		this.count = 0;
	}

	//Contructor for NEED fulfillment entries
	public LedgerEntity(LocalDate date, EntryType type, String needName, int
	count) {
		this.date = date;
		this.type = type;
		this.needName = needName;
		this.count = count;
		this.amount = 0.0;
	}

	public String getNeedName() {
		return needName;
	}

	public double getAmount() {
		return amount;
	}

	public LocalDate getDate() {
		return date;
	}
}
```

Java
```
public class NeedsRepository {
	private final List<NeedComponent> needsCatalog = new ArrayList<>();

	public void getNeeds(String csvSource) {
		//Reads BasicNeeds from CSV
	}
```

```java
        public void convertNeedsToObject(List<String[]> rawNeeds) {
                //Parses data into BasicNeed objects
        }

        public void getBundles(String csvSource) {
                //Reads Bundles from CSV
        }

        public void convertBundlesToBundlesObject(List<String[]> rawBundles) {
                //Parses data into Bundle objects
        }

        public void addNeedsToNeedsArray(List<NeedComponent> basicNeeds) {
                this.needsCalalog.addAll(basicNeeds);
        }

        public void addBundlesToBundlesArray(List<NeedComponent> bundles) {
                this.needsCalalog.addAll(bundles);
        }

        public void finishedAlert(ConsoleView view) {
                view.finishedAlert();
        }

        public NeedComponent getNeedByName(String name) {
                return (needsCatalog.stream()
                                .filter(n -> n.getName().equalsIgnoreCase(name))
                                .findFirst()
                                .orElse(null));
        }
}
```

Java
```java
public class LedgerRepository {
        private final List<LedgerEntity> logEntries = new ArrayList<>();

        public void save(LetgetEntity entity) {
                this.name = name;
        }

        public String getSummary() {
                //Logic to calculate sumary
```

```
		return "Daily Summary: $150.00 funding available.";
	}

	public double findGoal(LocalDate todaysDate) {
		//Logic to find the current goal for the date
		return 200.00;
	}

	public double calculateDonations(LocalDate todaysDate) {
		//Logic to aggregate donations for the date
		return 50.00;
	}

	public void addDonations(double donation1, double donation2) {
		LocalDate today = LocalDate.now();

		this.logEntries.add(new LedgerEntity(today,
LedgerEntity.EntryType.FUND, donation1));
		this.logEntries.add(new LedgerEntity(today,
LedgerEntity.EntryType.FUND, donation2));
	}
}
```

```java
public class NeedsController {
	private final NeedsRepository needsRepository;

	public NeedsController(NeedsRepository needsRepository) {
```

```
                this.needsRepository = needsRepository;
        }

        //Placeholder for need management methods (future implementation
possibly)
}
```

```
Java
public class LedgerController {
        private final Needsrepository needsRepository;
        private final Ledgersrepository ledgerRepository;

        public LedgerController(NeedsRepository needsRepository, ledgerRepository
ledgerRepository) {
                this.needsRepository = needsRepository;
                this.ledgerRepository = ledgerRepository;
        }

        public void recordsFullfillment(String needName, int count) {
                LedgerEntity fulfillment = new LedgerEntity(LocalDate.now(),
LedgerEntity.EntryType.NEED, needName, count);

                ledgerRepository.save(fulfillment);

                needsRepository.getNeedByName(needName);
        }

        public void displaySummary(ConsoleView view) {
                String summary = ledgerRepository.getSummary();

                view.displaySummary(summary);
        }

        public void processDailyGoal(ConsoleView view) {
                LocalDate today = LocalDate.now();
                int need_count = 5; // Placeholder for loop condition

                while (need_count > 0) {
                        double goal = ledgerRepository.findGoal(today);
```

```
                    double donations =
        ledgerRepository.calculateDonations(today);
                    double remainingGoal = goal - donations;

                    view.displayGoal(remainingGoal);

                    need_count = (need_count - 1);
                }
        }

        public void registerDonations(double donation1, double donation2,
ConsoleView view) {
                ledgerRepository.addDonations(donation1, donation2);

                //Logic to recalculate the goal status

                view.donationCompleted();
        }
}
```

Java

```java
public class ConsoleView {
        private final NeedsController needsController;
        private final LedgerController ledgerController;

        public ConsoleView(NeedsController needsController, LedgerController
ledgerController) {
                this.needsController = needsController;
                this.ledgerController = ledgerController;
        }

        public void startup(NeedsRepository needsRepository) {
                System.out.println("ConsoleView: Initiating Model startup...");

                needsRepository.getNeeds("needs.csv");
                needsRepository.getBundles("bundles.csv");

                //Alert will be triggered by NeedsRepository (if we allow the
import)
                //If not, the application main method must call finishedAlert()
after the repository operations.
        }
```

```
      public void finishedAlert() {
            System.out.println("ConsoleView: Data loading complete.");
      }

      public void recordsFullfillment(String needName, int count) {
            System.out.println("ConsoleView: Request to fulfill need: " +
needName);

            ledgerController.recordsFullfillment(needName, count);
      }

      public void displaySummary(String summary) {
            System.out.println("\n--- Displaying Summary ---");
            System.out.println(summary);
      }

      public void exitButton() {
            System.out.println("ConsoleView: Exiting program.");
      }

      public void getDailyGoal() {
            System.out.println("ConsoleView: Requesting daily goal
      calculation...");

            ledgerController.processDailyGoal(this);
      }

      public void displayGoal(double remainingGoal) {
            System.out.printf("ConsoleView: Current Remaining Goal: $%.2f\n",
      remainingGoal);
      }

      public void enterDonations(double donation1, double donation2) {
            registerDonations(donation1, donation2);
      }

      public void registerDonations(double donation1, double donation2) {
            System.out.printf("ConsoleView: Registering donations $%.2f and
            $%.2f\n", donation1, donation2);

            ledgerController.registerDonations(donation1, donation2, this);
      }
```

```java
        public void donationCompleted() {
                thankYouAlert();
        }

        public void thankYouAlert() {
                System.out.println("ConsoleView: Thank you message displayed.");
        }
}
```