# Coffee Machine Simulator

The Coffee Machine Simulator offers a user-friendly experience where users can simulate the operation of a coffee machine. It handles resource management for ingredients such as water, milk, coffee beans, and disposable cups, ensuring that the correct resources are available before making a coffee. The simulator distinguishes between different coffee types using inheritance and polymorphism, allowing for a modular and extendable design.

## 1. CoffeeMachine Class

**Description**:
The CoffeeMachine class represents the main machine and handles the machine's resources, including water, milk, coffee beans, disposable cups, and money. It provides methods for managing these resources and performing actions based on user input.

**Variables**:

- int water: Amount of water available in the machine.
- int milk: Amount of milk available in the machine.
- int coffeeBeans: Amount of coffee beans available in the machine.
- int disposableCups: Number of disposable cups available.
- int money: Total money collected by the machine.
- static Scanner scanner: Scanner object to handle user input.

**Methods**:

- CoffeeMachine(): Default constructor.
- CoffeeMachine(int water, int milk, int coffeeBeans, int disposableCups, int money): Parameterized constructor for initializing resources.
- int getWater(): Returns the current amount of water.
- void setWater(int water): Sets the amount of water.
- int getMilk(): Returns the current amount of milk.
- void setMilk(int milk): Sets the amount of milk.
- int getCoffeeBeans(): Returns the current amount of coffee beans.
- void setCoffeeBeans(int coffeeBeans): Sets the amount of coffee beans.
- int getDisposableCups(): Returns the number of disposable cups.
- void setDisposableCups(int disposableCups): Sets the number of disposable cups.
- int getMoney(): Returns the current money.
- void setMoney(int money): Sets the amount of money.
- static void showSupply(CoffeeMachine coffeeMachine): Displays the current status of the machine's resources.
- static void buy(CoffeeMachine coffeeMachine): Allows the user to buy a coffee, adjusting resources accordingly.
- static String checkWhatIsNotEnough(CoffeeMachine coffeeMachine, Coffee coffee): Checks if sufficient resources are available to make the selected coffee and provides feedback.

- static void fill(CoffeeMachine coffeeMachine): Prompts the user to refill the machine's resources.
- static void take(CoffeeMachine coffeeMachine): Collects the money from the machine.

**2. Coffee Class**

**Description**:
Coffee is an abstract class that represents a generic coffee. It serves as the base class for all specific coffee types and extends the CoffeeMachine class to inherit its resource management functionality.

**Variables**:

- String name: Name of the coffee.

**Methods**:

- Coffee(int water, int milk, int coffeeBeans, int money, String name): Constructor to initialize the coffee with specific resource requirements.

**3. Espresso Class**

**Description**:
Represents the Espresso coffee type. It extends the Coffee class and sets predefined values for resources required to make an Espresso.

**Methods**:

- Espresso(): Constructor that initializes an Espresso with specific resource needs.

**4. Latte Class**

**Description**:
Represents the Latte coffee type. It extends the Coffee class and sets predefined values for resources required to make a Latte.

**Methods**:

- Latte(): Constructor that initializes a Latte with specific resource needs.

**5. Cappuccino Class**

**Description**:
Represents the Cappuccino coffee type. It extends the Coffee class and sets predefined values for resources required to make a Cappuccino.

**Methods**:

- Cappuccino(): Constructor that initializes a Cappuccino with specific resource needs.

**6. Main Class**

**Description**:
Contains the main entry point of the application. It initializes a CoffeeMachine object with default resources and runs a loop to handle user actions like buying coffee, filling resources, taking money, and checking the remaining supplies.

**Methods**:

- public static void main(String[] args): The main method that controls the flow of the application, prompting the user for actions and calling the appropriate methods.