Rubrics:

Q3:

Seven classes: 7 marks each

Stock::displayInfo() const : 2 Marks

Market::getNumStocks() const: 1 Mark

Q4A:

Stock::isEligibleToBuy() const: 4 Marks

TechStock::isEligibleToBuy(): 3 Marks

PharmaStock::isEligibleToBuy(): 3 Marks

Q4B:

Investor::canBuyStock() : 2 Marks

DayTrader::canBuyStock(): 4 Marks

LongTermInvestor::canBuyStock(): 4 Marks

Q4C:

Operator : 4 Marks

==========

Solution

```
#include <iostream>
#include <string>
using namespace std;
#define MAX_STOCKS 100
class Stock {
protected:
    string symbol;
    string companyName;
    double price;
    int availableQuantity;
    int maxQuantityPerInvestor;
    int stockCategoryQuantity; // Number of stocks available for each category

public:
```

```
Stock(const string& stockSymbol, const string& company, double stockPrice, int quantityLimit, int categoryQuantity)
    : symbol(stockSymbol), companyName(company), price(stockPrice), availableQuantity(quantityLimit),
      maxQuantityPerInvestor(quantityLimit / 10), stockCategoryQuantity(categoryQuantity) {}

string getSymbol() const {
    return symbol;
}

string getCompanyName() const {
    return companyName;
}

double getPrice() const {
    return price;
}

bool isEligibleToBuy(int purchaseQuantity) const {

if (purchaseQuantity <= 0) {
    cout << "Cannot buy stock. Invalid purchase quantity." << endl;
    return false;

}
if (purchaseQuantity > maxQuantityPerInvestor) {
    cout << "Cannot buy stock. Maximum quantity limit per investor exceeded." << endl;
    return false;
}

if (purchaseQuantity > availableQuantity) {
    cout << "Cannot buy stock. Insufficient stock quantity." << endl;
    return false;
}
```

```
    return true;
}



    void displayInfo() const {
        cout << "Symbol: " << symbol << endl;
        cout << "Company Name: " << companyName << endl;
        cout << "Price: " << price << endl;
        cout << "Available Quantity: " << availableQuantity << endl;
        cout << "Max Quantity Per Investor: " << maxQuantityPerInvestor << endl;
        cout << "Category Quantity: " << stockCategoryQuantity << endl;
    }

    // Overload != operator
    bool operator!=(const Stock& other) const {
        return symbol != other.symbol;
    }
};

class TechStock : public Stock {
public:
    TechStock(const string& stockSymbol, const string& company, double stockPrice, int quantityLimit)
        : Stock(stockSymbol, company, stockPrice, quantityLimit, 0) {}

    bool isEligibleToBuy(int purchaseQuantity) {
        if (!Stock::isEligibleToBuy(purchaseQuantity)) {
            return false;
        }

        // Additional condition specific to TechStock
        if (purchaseQuantity % 10 != 0) {
```

```
        cout << "Cannot buy stock. Quantity must be a multiple of 10 for TechStock." << endl;
        return false;
    }

     if (purchaseQuantity > 100) {
    cout << "Cannot buy stock. Maximum purchase quantity for TechStock is 100." << endl;
    return false;
  }

    return true;
  }
};

class PharmaStock : public Stock {
public:
  PharmaStock(const string& stockSymbol, const string& company, double stockPrice, int quantityLimit)
    : Stock(stockSymbol, company, stockPrice, quantityLimit, 0) {}

  bool isEligibleToBuy(int purchaseQuantity) {
    if (!Stock::isEligibleToBuy(purchaseQuantity)) {
      return false;
    }

    // Additional condition specific to PharmaStock
    if (purchaseQuantity < 50) {
      cout << "Cannot buy stock. Minimum purchase quantity for PharmaStock is 50." << endl;
      return false;
    }
    // Additional condition specific to PharmaStock
  if (purchaseQuantity % 5 != 0) {
    cout << "Cannot buy stock. Quantity must be a multiple of 5 for PharmaStock." << endl;
    return false;
```

```
    }

        return true;
    }
};

class Investor {
protected:
    string name;
    string CNIC;
    string email;
    int availableFunds;
    bool hasLoan;

public:
    Investor(const string& investorName, const string& cnic, const string& investorEmail, int funds)
        : name(investorName), CNIC(cnic), email(investorEmail), availableFunds(funds), hasLoan(false) {}

    virtual bool canBuyStock(const Stock& stock, int purchaseQuantity) const = 0;

    // Other member functions
};

class DayTrader : public Investor {
public:
    DayTrader(const string& traderName, const string& cnic, const string& traderEmail, int funds)
        : Investor(traderName, cnic, traderEmail, funds) {}

    bool canBuyStock(const Stock& stock, int purchaseQuantity) const override {
        if (hasLoan) {
            cout << "Cannot buy stock. Loan availed." << endl;
            return false;
```

```
        }

        double totalPrice = stock.getPrice() * purchaseQuantity;
        if (totalPrice > availableFunds) {
            cout << "Cannot buy stock. Insufficient funds." << endl;
            return false;
        }

        return stock.isEligibleToBuy(purchaseQuantity);
    }
};

class LongTermInvestor : public Investor {
public:
    LongTermInvestor(const string& investorName, const string& cnic, const string& investorEmail, int funds)
        : Investor(investorName, cnic, investorEmail, funds) {}

    bool canBuyStock(const Stock& stock, int purchaseQuantity) const override {
        if (hasLoan) {
            cout << "Cannot buy stock. Loan availed." << endl;
            return false;
        }

        double totalPrice = stock.getPrice() * purchaseQuantity;
        if (totalPrice > availableFunds) {
            cout << "Cannot buy stock. Insufficient funds." << endl;
            return false;
        }

        return stock.isEligibleToBuy(purchaseQuantity);
    }
};
```

```
template <class T>
class Market {
public:
    T* stocks[MAX_STOCKS];
    int numStocks;

public:
    Market() : numStocks(0) {}

    void addStock(T* stock) {
        if (numStocks < MAX_STOCKS) {
            if (stock == NULL) {
                cout << "Cannot add stock. Invalid stock object." << endl;
                return;
            }

            stocks[numStocks] = stock;
            numStocks++;
        } else {
            cout << "Cannot add stock. Maximum number of stocks reached." << endl;
        }
    }

    void tradeStocks() {
        // Simulate trading of stocks
        for (int i = 0; i < numStocks; i++) {
            T* stock = stocks[i];
            // Perform trading operations
        }
    }
```

```
    int getNumStocks() const {
        return numStocks;
    }
};

int main() {
    // Create stocks and investors
    TechStock* techStock = new TechStock("AAPL", "Apple Inc.", 150.50, 1000);
    PharmaStock* pharmaStock = new PharmaStock("PFE", "Pfizer Inc.", 35.75, 500);
    techStock->displayInfo();
    pharmaStock->displayInfo();
    DayTrader dayTrader("John Doe", "1234567890", "john.doe@example.com", 10000);
    LongTermInvestor longTermInvestor("Jane Smith", "0987654321", "jane.smith@example.com", 50000);

    // Create market
    Market<Stock> market;

    // Add stocks to the market
    market.addStock(techStock);
    market.addStock(pharmaStock);

    // Trade stocks in the market
    cout << "Trading stocks..." << endl;
    for (int i = 0; i < market.getNumStocks(); i++) {
        Stock* stock = market.stocks[i];

        // Check if investors can buy the stock
        if (dayTrader.canBuyStock(*stock, 20)) {
            cout << "DayTrader bought stock: " << stock->getSymbol() << endl;
        }

        if (longTermInvestor.canBuyStock(*stock, 200)) {
```

```
            cout << "LongTermInvestor bought stock: " << stock->getSymbol() << endl;
        }
    }

    // Cleanup
    delete techStock;
    delete pharmaStock;

    return 0;
}
```