Overview

This C++ program implements a set of interactive tasks that users can choose to perform. Each task is encapsulated within its own class, which inherits from a base abstract class called `LabTask`. The available tasks include creating and displaying points, dynamically allocating a 2D array, and managing building and resident information. Users can select a task from a menu, and the corresponding functionality is executed.

Key Classes and Components

LabTask

`LabTask` is an abstract base class that defines a pure virtual function `execute()`. All derived classes must implement this function to provide specific functionality for each task.

PointTask

This class manages the creation and display of points.

- **Point Structure**: Represents a point with dynamically allocated `x` and `y` coordinates.
- **Constructor**: Dynamically allocates a vector to store pointers to points.
- **Destructor**: Frees the memory allocated for points and the vector.
- **inputPoints()**: Prompts users to enter coordinates for a specified number of points.
- **displayPoints()**: Outputs the coordinates of all points stored in the vector.
- **execute()**: Asks the user for the number of points, calls `inputPoints()` and `displayPoints()`, and frees the memory for the count.

DynamicArrayTask

This class creates and displays a dynamically allocated 2D array.

- **Constructor**: Initializes the array pointer to `nullptr` and dynamically allocates `rows` and `cols` pointers.
- **Destructor**: Frees the memory allocated for the 2D array and the `rows` and `cols` pointers.
- **createArray()**: Dynamically allocates the 2D array based on the specified number of rows and columns, and fills it with odd numbers.
- **displayArray()**: Outputs the contents of the 2D array.
- **execute()**: Prompts the user for the number of rows and columns, calls `createArray()` and `displayArray()`.

BuildingTask

This class manages building and resident information.

- **Resident Structure**: Represents a resident with dynamically allocated room number, number of residents, and rent.
- **Building Structure**: Represents a building with a dynamically allocated address and a vector of pointers to residents.
- **Constructor**: Dynamically allocates a building with an address and a vector for residents.

- **Destructor**: Frees the memory allocated for the building, address, and residents.
- **addTestData()**: Adds sample data for a building and its residents.
- **displayBuildingInfo()**: Outputs the building address and resident information.
- **execute()**: Calls `addTestData()` and `displayBuildingInfo()` to demonstrate building and resident management.

Execution Flow

Main Menu:

The `main()` function presents a menu to the user, allowing them to select one of three tasks:

1. Create and display points
2. Dynamically allocate 2D array
3. Building and residents info

Task Execution:

Based on user input, the corresponding task's `execute()` method is invoked. Each task handles its own input/output, allowing for independent execution of functionalities.

Program Termination:

After executing a selected task, the program terminates gracefully without prompting for further actions, as there is no loop implemented for repeated task selection.

Example User Interaction

Points Task

```
Select a task:
1. Create and display points
2. Dynamically allocate 2D array
3. Building and residents info
>> 1
How many points do you want to create? 3
Enter coordinates for point 1 (X Y): 10 20
Enter coordinates for point 2 (X Y): 30 40
Enter coordinates for point 3 (X Y): 50 60
Displaying Points:
Point (X: 10, Y: 20)
Point (X: 30, Y: 40)
Point (X: 50, Y: 60)
```

Dynamic Array Task

```
Select a task:
1. Create and display points
2. Dynamically allocate 2D array
3. Building and residents info
>> 2
```

```
Enter the number of rows: 3
Enter the number of columns: 4
Dynamically allocated 2D array:
1    3    5    7
9    11    13    15
17    19    21    23
```

Building Task

```
Select a task:
1. Create and display points
2. Dynamically allocate 2D array
3. Building and residents info
>> 3
Building Address: 123 Example Street
Residents Info:
Room Number: 101, Number of Residents: 2, Rent: $1200.5
Room Number: 102, Number of Residents: 3, Rent: $1300
Room Number: 103, Number of Residents: 1, Rent: $1100.75
```

Enhancements & Improvements

Error Handling:

- Implement error handling for invalid inputs (e.g., non-integer values when entering coordinates or dimensions).

Dynamic Input:

- Allow users to dynamically input building and resident information instead of using hardcoded test data.

Additional Features:

- Add functionality to modify or remove points, residents, or building information.
- Implement sorting capabilities for points, residents, or buildings based on specific criteria.

Conclusion

This program effectively demonstrates object-oriented programming principles such as inheritance and polymorphism in C++. Each task is encapsulated within its respective class, making it modular and easy to extend or modify in future iterations. With enhancements, this program can become even more user-friendly and robust in handling various data structures and operations.