Programming Task: "Property Management"

A program for managing apartments in a property management system.

1. Abstract Class *Apartment*

The abstract class **`Apartment`** is used to store information about apartments in a property management system. The following data should be stored as private instance variables, and the necessary public access methods (set... and get...) should be created:

- *Id* (Property number - a unique number, though not necessarily consecutive)
- *Area*
- *Number of rooms*
- *Floor* (0 = Ground floor)
- *Year of construction*
- *Address* (Postal code, Street, House number, Apartment number)

Choose appropriate data types.

A constructor should be implemented to allow setting the corresponding instance variables directly. Ensure that the values are plausible (e.g., the year of construction must not be in the future, etc.).

If conditions are not met, throw an **`IllegalArgumentException`** with a predefined error message (see point 7).

The method **`getAge()`** should calculate the age of the apartment.

Additionally, there should be a method **`getTotalCost()`** that calculates the monthly total cost of an apartment (see point 2).

2. Classes *OwnedApartment* and *RentedApartment*

Two concrete subclasses, **`OwnedApartment`** and **`RentedApartment`**, should be derived from the abstract class **`Apartment`**. The class **`OwnedApartment`** has additional private instance variables for the **operating costs** (per square meter) and the contribution to the **maintenance reserve** (per square meter). The class **`RentedApartment`** has additional instance variables for the **monthly rent** (per square meter) as well as the **number of tenants** in an apartment.

The monthly total cost for owned apartments is calculated from the monthly operating costs and the maintenance reserve, plus a surcharge. For rented apartments, the monthly total cost is derived from the monthly rent in addition to a surcharge.

For an owned apartment, the monthly surcharge is calculated based on the floor level (2% surcharge per floor; no surcharge for the ground floor). For a rented apartment, the surcharge depends on the number of tenants—no surcharge for one tenant, and 2.5% for each additional tenant, with the maximum surcharge capped at 10%.

The **`toString()`** method (inherited from **`Object`**) should be overridden so that all apartment data is returned as a string according to the specified format (see point 8).

3. Interface *PropertyManagementDAO*

This interface defines methods for accessing apartment data, independent of the specific implementation of persistent data storage (cf. Data Access Object).

The **`PropertyManagementDAO`** interface contains abstract methods for reading and saving apartment objects:

- The method **`getApartments()`** returns all persistently stored apartment objects as a **`java.util.List`**.
- The method **`getApartmentById(int id)`** returns an apartment object based on the apartment number (ID). If the apartment is not found, it should return null.
- The method **`saveApartment(Apartment apartment)`** should persistently store an apartment object. Ensure that the ID of a newly saved apartment is not already used by a previously stored apartment. If this occurs, throw an **`IllegalArgumentException`** with an appropriate error message (see point 7).
- The method **`deleteApartment(int id)`** should delete an apartment from the persistent data source. If the apartment does not exist, throw an **`IllegalArgumentException`** with the corresponding error message (see point 7).

4. Class *PropertyManagementSerializationDAO*

The **`PropertyManagementSerializationDAO`** class implements the **`PropertyManagementDAO`** interface.

Implement persistent storage of apartment data using ***Java Object Serialization***.

A string with the filename should be passed to the class through its constructor.

In case of file operation errors, a single-line error message should be displayed that starts with:

- "Serialization error:" or
- "Deserialization error:"

The program should terminate using **`System.exit(1)`** after displaying the error message.

5. Class *PropertyManagement*

The **`PropertyManagement`** class should implement the business logic. The class should have a private instance variable propertyManagementDAO (of type **`PropertyManagementDAO`**) to access apartment data.

Implement methods to provide the following functionality:

- Provide data for all apartments
- Provide data for a single apartment
- Add a new apartment
- Delete an existing apartment
- Determine the total number of all apartments / owned apartments / rented apartments
- Calculate the average monthly total cost of all apartments
- Determine the ID(s) of the oldest apartment(s)

6. Command-Line Interface

Write a Java program **`PropertyManagementClient`** that, using the **`PropertyManagement`** class, provides the command-line interface described below. Ensure that the program output matches the examples provided **exactly**!

The program should support calls in the following format:

```
java PropertyManagementClient <File> <Parameter>
```

<File>: The name of the file used for serialization. If the file does not exist, it should be created.

<Parameter>: One of ***list***, ***add***, ***delete***, ***count***, ***meancosts***, or ***oldest***. Only one parameter can be provided per call.

1. Parameter 'list'
    - Output all data for all apartments.

Example Call:

```
java PropertyManagementClient <File> list
```

Output:

```
Type:              RA
Id:                3
Area:              125.80
Rooms:             5
Floor:             2
Year Built:        1890
Postal Code:       1090
Street:            Berggasse
House Number:      19
Apartment Number:  5
Rent/m2:           10.35
Number of Tenants: 1

Type:              OA
Id:                7
```

```
Area:               100.00
Rooms:              3
Floor:              1
Year Built:         1890
Postal Code:        1090
Street:             Berggasse
House Number:       19
Apartment Number:   4
Operating Costs:    2.55
Reserve Fund:       0.95
```

2. Parameter 'list <id>'
    - Output all data of a single apartment

Example Call:
```
java PropertyManagementClient <File> list 3
```

Output:
```
Type:               RA
Id:                 3
Area:               125.80
Rooms:              5
Floor:              2
Year Built:         1890
Postal Code:        1090
Street:             Berggasse
House Number:       19
Apartment:          5
Rent/m2:            10.35
Number of Tenants: 1
```

3. Parameter 'add OA <id> <area> <rooms> <floor> <year_of_construction> <postal_code> <street> <house_number> <apartment_number> <operating_costs/m2> <reserve_fund/m2>' o Add an owned apartment persistently.
    - Add owned apartment persistently.

Example Call:
```
java PropertyManagementClient <File> add OA 1 95 3 4 1898 1080
    Florianigasse 42 20 1.55 0.45
```
Output:
```
Info: Apartment 1 added.
```

4. Parameter 'add RA <id> <area> <rooms> <floor> <year_of_construction> <postal_code> <street> <house_number> <apartment_number> <rent/m2> <tenants>'
    - Add a rented apartment persistently.

Example Call:
```
java PropertyManagementClient <File> add RA 2 95.0 3 3 1894
    1080 "Lange Gasse" 15 16 8.75 2
```

Output:

```
Info: Apartment 2 added.
```

5. Parameter 'delete <id>'
    - Delete apartment.

Example Call:

```
java PropertyManagementClient <File> delete 2
```

Output:

```
Info: Apartment 2 deleted.
```

6. Parameter 'count'
    - Calculate the total number of recorded apartments.

Example Call:

```
java PropertyManagementClient <File> count
```

Output:

```
3
```

7. Parameter 'count <type>'
    - Calculate the total number of owned apartments/rented apartments.

Example Call:

```
java PropertyManagementClient <File> count RA
```

Output:

```
1
```

Example Call:

```
java PropertyManagementClient <File> count OA
```

Output:

```
2
```

8. Parameter 'meancosts'
    - Calculate the average monthly total costs of all apartments.

Example Call:

```
java PropertyManagementClient <File> meancosts
```

Output:

```
621.41
```

9. Parameter 'oldest'
    - Find the oldest apartment(s).

Example Call:

```
java PropertyManagementClient <File> oldest
```

Output:

```
Id: 3
Id: 7
```

7. Error Messages

All exceptions thrown due to invalid or incorrect input must be caught, and the program must terminate with an error message. Only one of the following error messages should be displayed at a time:

- "Error: Invalid parameter."
- "Error: Invalid year of construction."
- "Error: Apartment already exists. (id=<id>)"
- "Error: Apartment not found. (id=<id>)"

Example Call:
```
java PropertyManagementClient <File> add OA 7 100.0 3 1 1890
    1090 Berggasse 19 4 2.55
```
Output:
```
Error: Invalid parameter.
```

Example Call:
```
java PropertyManagementClient <File> add OA x 100.0 3 1 1890
    1090 Berggasse 19 4 2.55 0.95
```
Output:
```
Error: Invalid parameter.
```

Example Call:
```
java PropertyManagementClient <File> add OA 7 100.0 3 1 18900
    1090 Berggasse 19 4 2.55 0.95
```
Output:
```
Error: Invalid year of construction.
```

Example Call:
```
java PropertyManagementClient <File> add RA 2 95.0 3 3 1894
    1080 Lange Gasse 15 16 8.75 2
```
Output:
```
Error: Apartment already exists. (id=2)
```

Example Call:
```
java PropertyManagementClient <File> delete 4711
```
Output:
```
Error: Apartment not found. (id=4711)
```

8. Additional Remarks:

- **Inputs**

If parameters contain spaces, you can use quotation marks (" ") (see the example call for 'add').

- **Outputs**

Always output floating-point numbers with a '.' (dot) as the decimal separator and exactly 2 digits after the decimal point, e.g., 12.35.
You can use the method **`Apartment.getDecimalFormat()`** for this.

Example:

```
Double area = 124.345;
DecimalFormat df = Apartment.getDecimalFormat();
System.out.println("Area:                " + df.format(area));
```

Output:

```
Area:                124.34
```

9. Submission Guidelines

Submission Deadline: Wednesday, 06.11.2021, 12:00

Use the Java classes/interfaces provided in the archive Aufgabe1PLC24WS.zip as the basis for developing your program. Under no circumstances should you change the class names or the **`PropertyManagementDAO`** interface. All classes and interfaces should be left in the default package.

The **program has to be submitted before the deadline on the online platform** after it has passed all checks. Using **TRY OUT** will save the last state of your program. Further information is provided in the lectures and on Moodle.