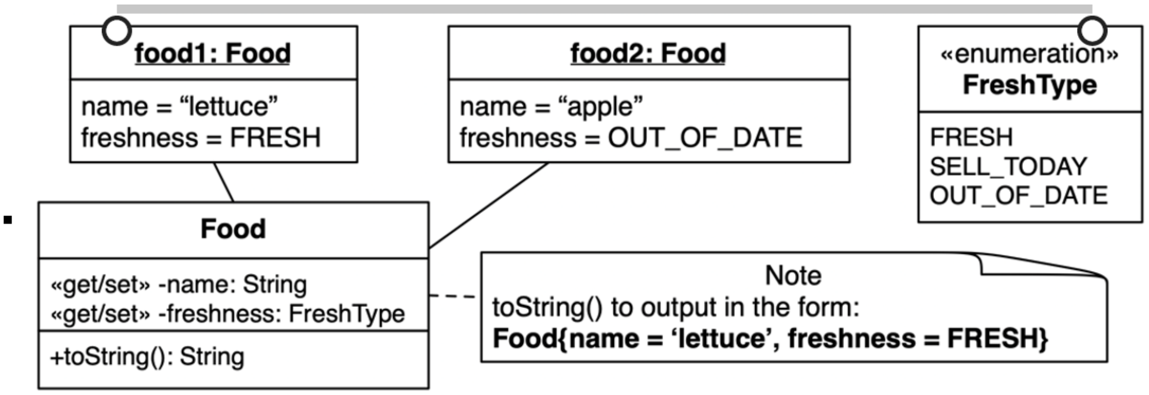
OO Programming

Lab 05

Week 5 Java exercises

Exercise 1 (Book 2 - 1.10)

Exercise - create a solution using an enumeration class



Class-Object diagram for Food-FreshType enum.

AIM:

- explore use of enums for a specified set of values for a variable

ACTION:

- enumeration class `FreshType` (file: `FreshType.java`)
    - declare an enumeration class named `FreshType` with 3 values `{FRESH, SELL_TODAY, OUT_OF_DATE}`
- class `Food` (file: `Food.java`)
    - declare a variable `freshness` which stores a `FreshType` value
    - public get and set methods for both variables
    - with public `toString()` method to return a String summary of the object’s state (as shown in the diagram)

- class Main (file: Main.java)
    - Create an instance of Food named food1, which is still fresh lettuce
    - Create an instance of Food named food2, which is an out of date apple
    - print out each object's state via its toString() method, i.e.

```
System.out.println(food1);
System.out.println(food2);
```

OUTPUT:

```
$ java Main
Food{name='lettuce', freshness=FRESH}
Food{name='apple', freshness=OUT_OF_DATE}
```

Exercise 2 Use public accessor methods in subclass code

(Lecture Protected Book 2 – ex 2.8)

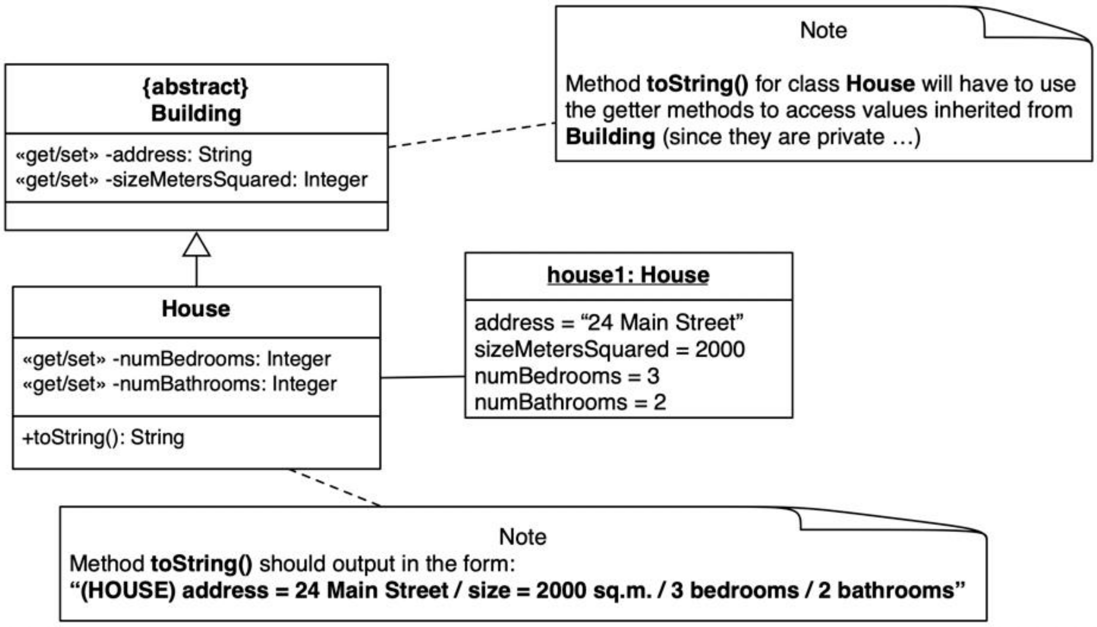

Note
Method **toString()** for class **House** will have to use the getter methods to access values inherited from **Building** (since they are private …)
{abstract}
Building
«get/set» -address: String
«get/set» -sizeMetersSquared: Integer
House
«get/set» -numBedrooms: Integer
«get/set» -numBathrooms: Integer
+toString(): String
house1: House
address = "24 Main Street"
sizeMetersSquared = 2000
numBedrooms = 3
numBathrooms = 2
Note
Method **toString()** should output in the form:
"(HOUSE) address = 24 Main Street / size = 2000 sq.m. / 3 bedrooms / 2 bathrooms"

Figure 2.2: Class-Object diagram for toString() method of class Building

AIM:

- reflect on when to use the public getter/setter methods to overcome `private` variables inherited from a superclass

ACTION:

- class Building (File: Building.java) abstract, properties are private variables.
- class `House` (File: `House.java`)
  - add a `toString()` method to your `House` class, that outputs in the following form:

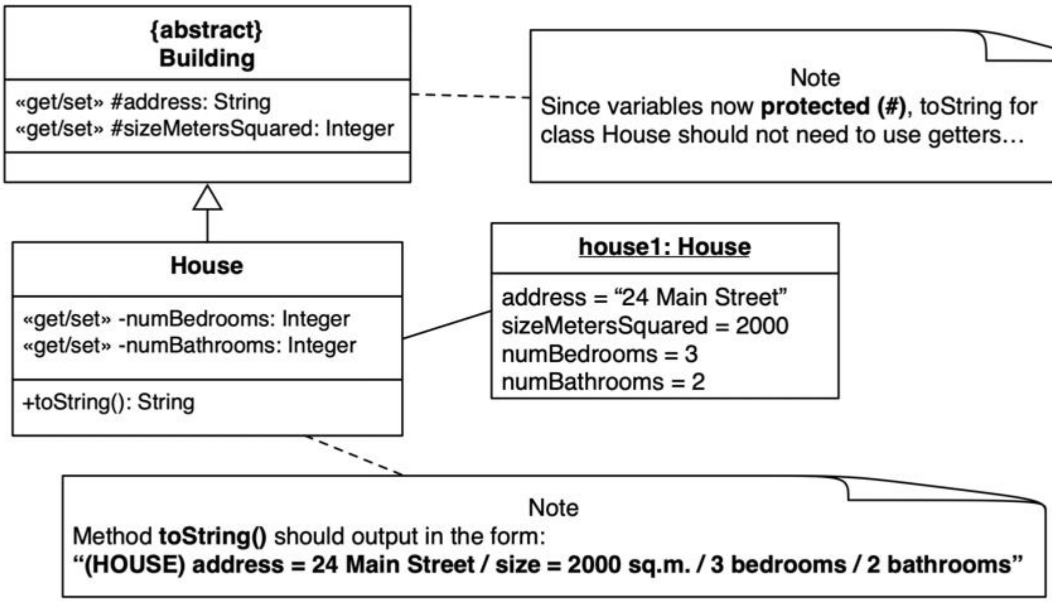
    ```
    (HOUSE) address = 24 Main Street / size = 2000 sq.m. / 3
    bedrooms / 2 bathrooms
    ```
- class `Main` (File: `Main.java`)
  - method `main()`:
    - use the setter methods to create `house1`, and instance-object of class `House` with values:
      - address = `24 Main Street` / size = 2000 sq.m. / 3 bedrooms / 2 bathrooms
    - use `System.out.println()` to output the details of `house1`

OUTPUT:

```
$ java Main
(HOUSE) address = 24 Main Street / size = 2000 sq.m. / 3 bedrooms / 2 bath
rooms
```

HINT: Since the properties in `Building` are `private`, you'll have to use the public `get<>()` methods in the `toString()` method of class `House`

Exercise 3 (Lecture Protected Ex 2.9 from Book)
Use protected visibility to allow subclass methods to directly access inherited properties


{abstract}
Building
«get/set» #address: String
«get/set» #sizeMetersSquared: Integer
Note
Since variables now **protected (#)**, toString for class House should not need to use getters…
House
«get/set» -numBedrooms: Integer
«get/set» -numBathrooms: Integer
+toString(): String
house1: House
address = "24 Main Street"
sizeMetersSquared = 2000
numBedrooms = 3
numBathrooms = 2
Note
Method **toString()** should output in the form:
"(HOUSE) address = 24 Main Street / size = 2000 sq.m. / 3 bedrooms / 2 bathrooms"

Class-Object diagram for `toString()` method of class `Building`.

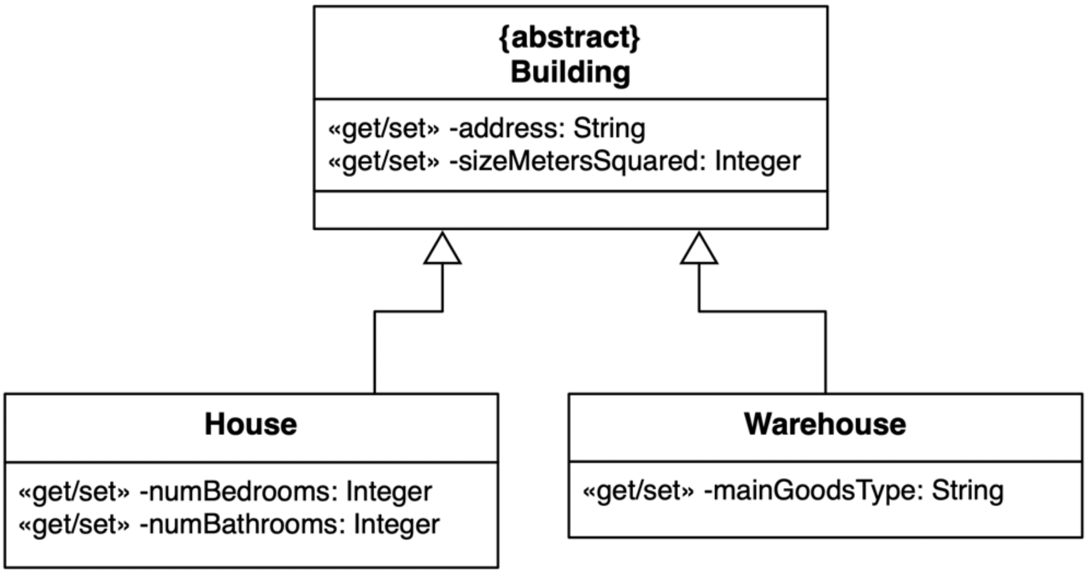
AIM:

- reflect on when to use the `protected` accessibility modifier to allow methods in subclasses to directly access inherited variables

Do the following:

- duplicate your folder for the previous question (abstract `Building` using get/set methods)
- class `Building` (File: `Building.java`)
  - change the visibility of the variables for class `Building` to `protected`
- class `House` (File: `House.java`)
  - improve the `toString()` method of class `House`, to directly access the inherited `protected` variables for `address` and `sizeMetersSquared`
    - so now your code should **not** need to use any getter methods, since it has direct access to the inherited variables
- the output should be the same…

Exercise 4 (4.19)
Explore Abstract Classes


{abstract}
Building
«get/set» -address: String
«get/set» -sizeMetersSquared: Integer
House
«get/set» -numBedrooms: Integer
«get/set» -numBathrooms: Integer
Warehouse
«get/set» -mainGoodsType: String

Class-Object diagram for `abstract` class `Building`.

AIM:

- practice working with abstract classes

ACTION:

- class Building (File: Building.java)
    - class Building as an abstract class, with private properties and public getters/setters:
        - address: String
        - sizeMetersSquared: Integer
- class House (File: House.java)
    - class House, a subclass of Building, with private properties and public getters/setters:
        - numBedrooms: Integer
        - numBathrooms: Integer
- class Warehouse (File: Warehouse.java)
    - class Warehouse, a subclass of Building, with private properties and public getters/setters:
        - mainGoodsType: String
- class Main (File: Main.java)
    - main() method to:
        - attempt to create an object of class Building
            - you should get COMPILER ERROR since Building is abstract
        - create instance-objects of classes House and Warehouse

Exercise 5 – Final Class (4.20 from Book)

AIM:

- understand the result of having a `final` class

ACTION:

- create a `Person` superclass and a `Student` subclass
  - they do not need any methods or properties for this exercise
- declare the `Person` class as final
- compile your files

OUTPUT:

- you should get an error, stating that `Student` cannot extend Person because `Person` is final

Exercise 6 – Final Method (4.21 from Book)

AIM:

- practice working with `final` methods

ACTION:

- create a `Employee` superclass and a `Caretaker` subclass
  - in class `Employee` declare a final method `calculateSalary()` that returns double 20.0
  - in class `Caretaker` declare a method `calculateSalary()` that returns double 55.0
- compile your files

OUTPUT:

- you should get an error, stating that `Caretaker` cannot override inherited final method `calculateSalary()`

Exercise 7 – Abstract Method (4.22 from Book)

AIM:

- practice working with `abstract` methods

ACTION:

- create a `Vehicle` superclass and a `Boat` subclass
    - in class `Vehicle` declare an abstract method `getTopSpeed()` that returns a double
        - there should be no method body, just a semi-colon after the parentheses
    - declare class `Boat`
        - but do **not** write any method `getTopSpeed()` for this class
- compile your files

OUTPUT:

- you should get an error, stating that `Boat` cannot extend class `Vehicle` because it does not implement abstract method `getTopSpeed()`

REFINEMENT STEP:

- now declare class `Boat` as abstract
    - you should now be able to compile your classes - since an abstract subclass does **not** need to implement all inherited abstract methods