Lab 2: Defining a scene of multiple objects in your domain

Task 0: *For any changes, always update your UML class diagram!*

Task 1: *Improve your code*

- There should at least be 10 classes in the hierarchy to define the animal. The depth of the hierarchy should at least be 3 (the deeper the better).
- Each class should implement one constructor which initialises all properties.
- Classes should be similar in size, i. e. regarding their number of lines.
- Avoid magic values everywhere: introduce constants and variables with meaningful identifier names.
  - Verify whether all your constants and variables are correctly defined either locally in methods or as properties of classes.
- drawAt(int left, int bottom)-methods:
  - make sure that all drawAt-methods work properly, so that the entire animal can be drawn at different places (determined by the parameters *left* and *bottom*).

Task 2: *Associations: aggregates and composites*

- Determine aggregates and composites of your domain. Write a comment after each object-property:

```
private Facade facade; // composite
private Door aDoor; // aggregate
```

- Define a class called *Scene* aggregating many instances of your domain:
  - It is instantiated as the sole object in the *paintComponent*-method of class *DrawingArea*.
  - In *Scene* define an *ArrayList* (multi-piece association) called cats, if your animal is cat.

– Each such instance is to be drawn at a different place on the scene. (You might have to scale down your object if you draw it very large.)

TASK 3: *Interfaces*

- Let your main class of your domain (e.g. cat) implement the interface LocatedRectangle (provided in moodle: define a new Interface in your package with that name and copy & paste it to this new class).
- don't change LocatedRectangle! Only the package name.
- LocatedRectangle provides a method called intersects. This is to be used to verify for each domain object to be drawn, whether it intersects with another object, more precisely, its bounding rectangle.
- Hint: all objects are to be stored in an *ArrayList* according to Task 2.

TASK 4: *Inheritance*

- Identify a superclass-subclass relationship in your domain.
- E.g., Door could be superclass of FrontDoor and of FrenchWindow.
- In domain Car: WheelRims as super class of SportRims and WinterRims.

TASK 5: *Variations of the objects of your domain*

- Overloading:
  Provide different draw-methods which take into account different details.
  E.g., a window might have curtains, a blend, bars, or nothing.
  Put common code in new private methods.
- Use different draw-methods for the different instances on your scene.
- Use a random number generator to variate objects of your domain automatically. Such a generator can be restricted to a static method as follows:

```
public class RandomNumber {

   public static int between(int min, int max) {
      return (int) (Math.random() * ((max - min) + 1) + min);
   }
}
```

- Use it like: int someVaule = RandomNumber.between(10, 20);

Software Quality: Code Conventions

a) Identifiers are in English.

b) Identifiers are meaningful, but not too long.

c) Variable identifiers begin with a small letter. Multiple words composed as CamelCase.

d) Identifiers for classes and interfaces begin with a capital letter. Multiple words composed as CamelCase.

e) Identifiers for constants consist only of uppercase letters. Multiple words composed by underline.

f) Left curly braces not in a new line. New line after left curly braces.

g) New line after right curly braces.
Exception: keyword else is in the same line.

h) Logical sections within a method have a comment as a heading.

i) Each block level is horizontally tap-indented by one level.

j) There is a blank line between methods.

k) There is a blank line between classes.

l) Classes and interfaces are separated by a blank line of import and package statements.

m) No more than one blank line in a row.

n) Order within a class or an interface:

1. properties (constants and variables)
2. constructors
3. getter and setter for properties, but only if required
4. other methods