Worksheet: Creation of Objects 1

Goals

In object-oriented programming, if an object of a class `A` wants to call methods in an object of class `B`, class `A` only needs to know the most general type in which these methods are declared. This can be an interface implemented by class `B` or a base class of `B`.

However, to create an *instance* of class `B`, the `new` statement must specify the concrete class name of class `B`. Class `A` is thus dependent on class `B`, i.e. class `A` is strongly coupled to class `B`.

The following tasks will help you to think about how this strong coupling can be reduced.

Context

The *JDraw*-project as it was distributed at the beginning of this course contains test classes (in the directory `src/test/java`). One of them, `RectangleTest`, tests the behavior of class `Rect` in its role as an observable.

In the meantime, you have (hopefully…) implemented additional figures, probably a group figure and some decorators. For all these figures, it would be interesting to apply the same tests as those in class `RectangleTest`.

Tasks

a) Study class `RectangleTest`. Today, we are less interested in the behavior of the individual tests than in the dependency of class `RectangleTest` on class `Rect` under test.

   To apply the same tests to a second class (e.g. to class `Line`), you could simply copy `RectangleTest.java` to `LineTest.java`. Where would you have to change something in this copied class so that it actually tests class `Line` instead of class `Rect`?

b) Copying source code as proposed in a) is not a good approach. If for example an additional test method were added later, you would have to add this method to each copy. A better approach would be to factor out shared code in a common (abstract) base class.

   Implement this approach in your *JDraw* to test your self-programmed figures (or - if you do not have one - for a rectangle decorated with the `GreenDecorator`.) You can find class `GreenDecorator` in the materials provided with lecture *09_Decorator1*).

   Try to find a solution that would still work if a figure had to be instantiated not just at one location but at several places in class `RectangleTest`.

c) Draw your solution of task b) (together with the tested figure classes) schematically in a UML class diagram. Do you recognize a certain structure in this diagram?

d) If you have completed the tasks a) to c), then in addition to your figure implementations test a decorated figure and your group figure implementation. Are the tests successful? If not, you can now use the time to debug and fix these problems.

   Alternatively, you could also fill out the evaluation of this course if you haven't done it already.