Programming Paradigms
Assignment 2

Jacek Wasilewski

1 Description

Solve problems defined below. You should submit one Scala Worksheet file (`.sc` file extension) containing all your work. Separate and comment each problem appropriately. Include only code that compiles - any compilation error will result in 20 points penalty. Totally you can get 100 points, problems are not equally weighted. Submit before the deadline stated on Moodle.

2 Problems

2.1 Problem 1 (40 points)

We live in a world where there are three shapes: square, triangle and circle. These shapes are used to build well-structured but simple pictures. Actually there are only three types of pictures: picture with one row of elements, picture with two rows of elements and picture with three row of elements. Each row can be built using one, two of three shapes.

Each shape should store its properties: square has side, triangle has base and height, circle has radius.

Your task is to transform the above scenario into an object-oriented solution. Your solution should present the use of classes, abstract classes and/or case classes. You should be able to create and store pictures using your models.

Consider the following examples of pictures.

- Picture with 1 row, row of 1 element:

```
| X |
```

- Picture with 2 rows, each of 1 element:

```
| X |
| X |
```

- Picture with 2 rows. First row of 1 element, second row of 2 elements:

```
| X |
| X | X |
```

- Picture with 3 rows. First row of 2 elements, second row of 1 element, third row of 3 elements:

```
| X | X |
| X |
| X | X | X |
```

It should be possible to create one object representing whole picture. Element (`X`) can be any of the shapes.

2.2 Problem 2 (10 points)

Shapes can be of different colours, actually either red, green or blue. Modify your code that each shape stores its colour. Colour needs to be specified if a specific shape is being created. Model the colours appropriately using classes.

2.3 Problem 3 (10 points)

Your classes should have nice default string representation to make it easier to visualise the pictures. Here are the requirements:

- shape is of the following format: <`shape_type`>(<`colour`>)
- square - `S`
- triangle - `T`
- circle - `C`
- red - `r`
- green - `g`
- blue - `b`

So a green triangle would be `T(g)` and a red circle `C(r)`.

Rows should concatenate all elements in one line. Pictures should have one row per line in their representation.

Representation of a sample picture of 2 rows and few elements could look like:

```
| T(r) | C(g) |
| T(g) | T(e) | S(b) |
```

Make your implementation as simple and as clean as possible.

2.4 Problem 4 (15 points)

Pictures, rows, shapes have different dimensions and we need space to store them. Let's say we need to know the area of a picture to know if we have enough space to store it. Typically area is member of a class but in our world we have compressors that know the areas of everything.

Implement a class called `GreatAreaCompressor` with a method to store all logic to calculate the area of any objected passed it (make sure that you handle unknown objects or never accept them). There should be only one instance of that class ever created. Areas of shapes are defined by the dimensions. Area of a row is just the sum of areas of all elements in the row. Area of a picture is the sum of its rows' areas.

For a following picture:

```
| S(r) |
| T(r) |
```

where side of square is 5 and triangle's base and height is 5 as well, the total area is 37.5 (25 for square, 12.5 for triangle, area of all elements is added).

2.5 Problem 5 (25 points)

There are some rules that all the elements have to obey (in the order):

1. if there are only two elements in the row, and these elements are of the same shape, these should be merged together; for squares you add the sides together, for triangles you add bases together and heights together, for circles you add radiuses together,

2. if there are three shapes in a row, and all are of a different colour, the row should be removed and picture shrinks,

3. if there are three shapes in a row, and all are of a different type, shapes should get twice times bigger,

4. if there are three shapes in a row, and first and third are of the same type, swap them.

Implement a class called `WorldRuler` that allows to transform objects according to the above rules.

Below are some examples:

```
| S(r) | C(g) | T(g) |
| S(r) | S(r) |
```

Areas are:

```
| 1.00 | 3.14 | 1.00 |
| 1.00 | 1.00 |
Total area: 7.14
```

After transforming (rules 1 and 3) this becomes:

```
| S(r) | C(g) | T(g) |
| S(r) |
```

with areas:

```
| 4.00 | 12.57 | 4.00 |
| 4.00 |
Total area: 24.56
```

Other example:

```
| S(r) | C(g) | T(b) |
| S(r) | S(r) |
| S(g) | C(g) | S(b) |
```

becomes

```
| S(r) |
| S(b) | C(g) | S(g) |
```

First row removed (2), second row merged (3), places in third swapped (4).