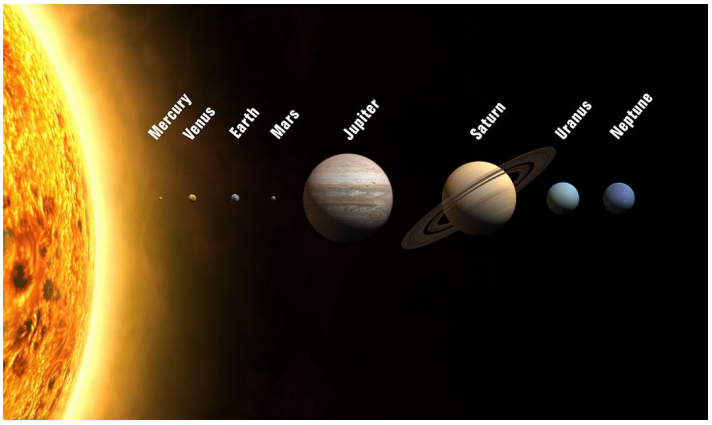

Figure 1: Solar system, source: wikipedia

Java2 Planets and Filters

Pieter van den Hombergh

February 17, 2017

Planets and Filters

This week you have studied enums and generics. They give you more experience with the subject matter, this weeks task is an extension to the shapes exercise.

Start with the shapes solution code given in the repository. Note that we changed a few details. In particular we changed the name of the `Shape.draw` method to `Shape.paint` and added two methods in the `Shape` interface, called `void draw(Graphics2D)` and `void fill(Graphics2D)`. This allows the implementation of `paint(...)` in the abstract class and the drawing and filling can be deferred to the sub-classes.

Planets

Start with the `Planet`-enum example from the Oracle enum tutorial site and add the following functionality.:

1. Stick to the named planets in the example.
2. Add the distance to the sun in astronomical units and make a getter for this value.
3. Add a method to calculate the gravitational pull that the Sun exerts on the Planet. Call the method `double sunGravitationalPull()`. The gravitational force that two bodies exert on each other can be calculated with

$$f_g = \frac{G \times m_1 \times m_2}{d^2}$$

in which m_1 and m_2 are the masses of the bodies concerned and d is the distance between the center of gravity between those bodies. Make sure you get your dimensions right, as in always use meters and kilograms, then the result will be in Newton.

4. Add a method to calculate the speed with which the planet is moving around the sun in km/h. This is the speed required to keep the planet at its distance to the Sun, given its mass and gravitational pull of the Sun. The speed can be derived from the fact that the pulling force and the centripetal force that keeps the planet in its orbit are at equilibrium. From that you can derive with some math that the speed is

$$v = \sqrt{\frac{f_g \times d}{m}}$$

with f_g the force computed earlier, d the distance between sun and planet and m the mass of the planet concerned.

5. Add a method to compute the average density of the planet. You may assume the planet is a sphere with given radius. The volume of a sphere is defined as

$$V = \frac{4}{3}\pi r^3$$

.

6. Add output in the main method of the Planet enum to show the calculated values.

You can use wikipedia `http://en.wikipedia.org/wiki/Solar_System` as the source of many parts of the required information and continue from there.

Add the Planet enum to the shapes project from week 1.

New Shapes

Again add to the shapes project the following:

In the package `shapes.basic` add two shapes:

Disk which extends circle and paints it selves without border. As an extra (in addition to center and radius) constructor. It paints its interior as a flat disk. Disk has also a setter for an `image` which, when set, is drawn on the the disk. The paint method must make sure that the when drawing the image nothing of the image is spilled over the edge of the disk. parameter it takes a fillPaint which is of type Paint.

Sphere which extends Disk. Sphere creates a 3D like image of a sphere or Ball. The fillpaint that is given as constructor parameter is assumed to be a color and is used as the starting (as in darkest) colour when drawing the sphere like interior of this shape.

The essential detail here is that there are sub classes of Circle. There is no need to really improve or overwrite the painting methods etc.

Shape filter

This part is about generics. Remember the PECS acronym.

In the `Canvas` class add two methods:

1. `void addShapes(Collection ... shapesToAdd)` Make sure to use the proper Collection definition with generics.
   You would use this method to add a bulk of shapes to the canvas.

2. `boolean shapeFilter( receiver, type)` This method takes two methods, a collection (like a list) in which you can put shapes and the type of shape you want filter.
   You would use this method to collect the shapes from the canvas and can specify the kind of shapes you want to receive.

Remember that the reason for using the wild card in `extends` or `super` is to let the compiler do the type checking as accurate as possible. At the same time you want it to be sufficiently flexible to allow assignments that *should* be possible[1] *are* possible.

The best way to go about this exercise is to experiment a bit with what the NetBeans and compiler combination accept.

Add a functionality to the main class of the Shapes exercise to call both methods. In the main class, add a lot more shapes to your canvas. Hint: use a random generator to determine size, position and color of the shapes. A nice touch is to use semi-transparant colours.

For the `addShapes` you could chose to use this method instead of the code in the given solution.

Add a menu and a menu item to the gui, in which the menu item lets you generate a new drawing. After this generation, use the filter method to find how many shapes of kind `circle`.

[1] so called compatible assignments