Photo by Marvin Meyer on Unsplash

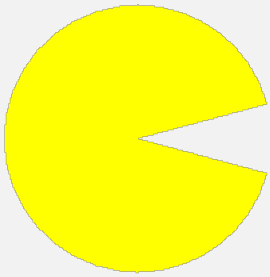
Lab 1

Pacman · Toolbelt · House · Colors · Swiss Flag · Rhombus · Ski Slope · Promote to Toolbelt · Eyes · Flower · Yin and Yang · Me

A. Pacman

Let's construct the main character of the Pac-Man game!

Task A1

Class:	`Pacman`
Task:	Implement the `pacman` method.
Run in JShell:	`show(Pacman.pacman(100, 30))`
Output:	

B. Toolbelt

Let's add some methods to our toolbelt, so we can later use them.

Task B1

Class:	`Toolbelt`
Task:	Implement the `square` method.
Run in JShell:	`show(Toolbelt.square(50, GREEN))`
Output:	

Task B2

Class:	`Toolbelt`
Task:	Implement the `circle` method.
Run in JShell:	`show(Toolbelt.circle(50, YELLOW))`
Output:	

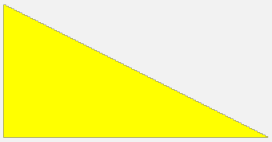
Task B3

Class:	`Toolbelt`
Task:	Implement the `equilateralTriangle` method.
Run in JShell:	`show(Toolbelt.equilateralTriangle(100, CYAN))`
Output:	

Task B4

Class:	`Toolbelt`
Task:	Implement the `isoscelesTriangle` method.
Run in JShell:	`show(Toolbelt.isoscelesTriangle(150, 30, MAGENTA))`
Output:	

Task B5

Class:	`Toolbelt`
Task:	Implement the `rightTriangle` method.
Run in JShell:	`show(Toolbelt.rightTriangle(200, 100, YELLOW))`
Output:	

C. House

Task C1

Class:	`House`
Task:	Implement the `house` method. **Use** the methods from your toolbelt!
Run in JShell:	`show(House.house(100, RED, WHITE))`
Output:	

Task C2

In the old town of the Swiss capital Bern, most houses are green-ish.

Class:	`House`
Task:	Create a method named `berneseHouse`, with one parameter named `width` of type `double`. Your method should create a house with the given width, a green roof, and a green wall. **Use** the `house` method you implemented in the previous task.
Run in JShell:	`show(House.berneseHouse(80))`
Output:	

D. Colors

Task D1

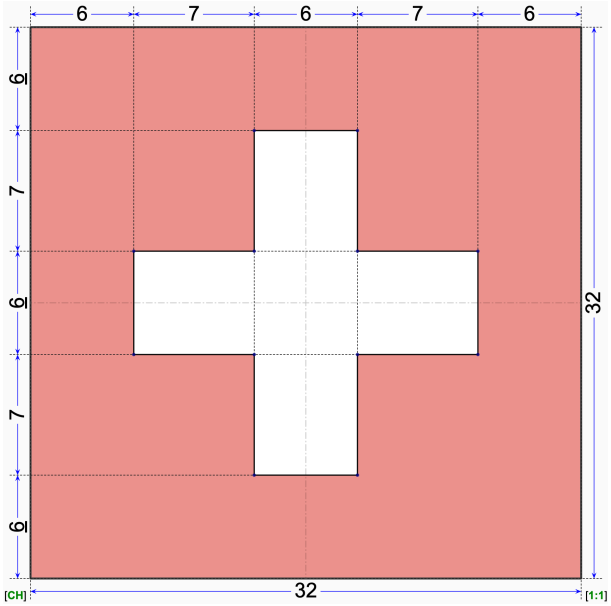
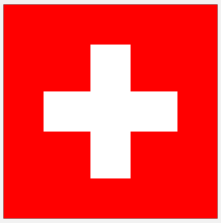
Class:	`BaseColors`
Task:	Create a method named `rgbCircles`, with one parameter named `diameter` of type `double`. Your method should create three circles next to each other, in red, green, and blue color. **Assert** that the parameter has an acceptable value. **Use** your toolbelt.
Run in JShell:	`show(BaseColors.rgbCircles(80))`
Output:	

Task D2

Class:	`BaseColors`
Task:	Create a method named `cmySquares`, with one parameter named `side` of type `double`. Your method should create three squares next to each other, in cyan, magenta, and yellow color. **Assert** that the parameter has an acceptable value. **Use** your toolbelt.
Run in JShell:	`show(BaseColors.cmySquares(80))`
Output:	

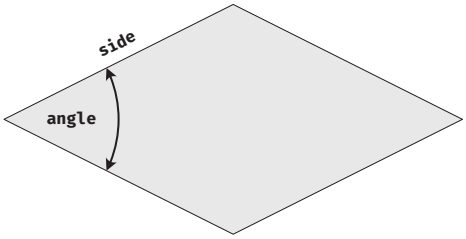
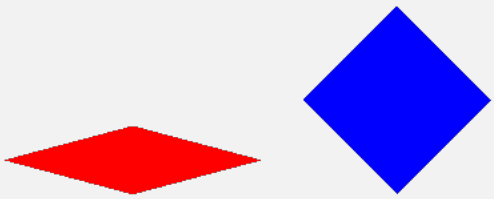
E. Swiss Flag

Task E1

Class:	`SwissFlag`
Task:	Implement a method named `flag` that has a parameter named `unit` of type `double` that represents the size of the unit from which the flag is created (the flag is 32 units wide and 32 units high).  To reduce the size of your expression, and to avoid code duplication, create additional methods to produce the parts of the flag. Also **use your toolbelt** methods where appropriate.
Run in JShell:	`show(SwissFlag.flag(5))`
Output:	

F. Rhombus

Task F1

Class:	`Rhombus`
Task:	Implement a method named `rhombus` that has a parameter named `side` of type `double` and a parameter named `angle` of type `double`, that produces a rhombus. A rhombus is a quadrilateral (a 4-sided polygon) where all sides have the same length.  **Assert** that the parameters have acceptable values. What are the minimum and maximum possible values for the angle? **Use** the most appropriate toolbelt methods.
Run in JShell:	`show(Rhombus.rhombus(100, 30, RED))` `show(Rhombus.rhombus(100, 90, BLUE))`
Output:	

G. Ski Slope

Difficult ski slopes are usually marked in black. In North America, slopes that are more difficult than normal black slopes are marked with a “black diamond”. The most difficult slopes are marked with a “double black diamond”.

Task G1

Class:	`DoubleBlackDiamond`
Task:	Implement a method named `blackDiamond` that has a parameter named `side` of type `double` that represents the side length, which produces a black rhombus with a 110-degree angle. **Assert** that the side is positive! **Use** your toolbelt methods where appropriate.
Run in JShell:	`show(DoubleBlackDiamond.blackDiamond(40))`
Output:	◆

Task G2

Class:	`DoubleBlackDiamond`
Task:	Implement a method named `doubleBlackDiamond` that has a parameter named `side` of type `double` that represents the side length of a rhombus. The method should produce the “double black diamond” sign. **Assert** that the side is positive! **Use** your method from the previous task.
Run in JShell:	`show(DoubleBlackDiamond.doubleBlackDiamond(40))`
Output:	◆◆

Task G3

Class:	`DoubleBlackDiamond`
Task:	Implement a method named `labeledDoubleBlackDiamond` that has a parameter named `side` of type `double` that represents the side length of a rhombus. The method should produce the “double black diamond sign” with the “EXPERTS ONLY” label. Use `SANS_SERIF` as the font (or a font you have) and make the font size half of the side. **Assert** that the side is positive! **Use** your method from the previous task.
Run in JShell:	`show(DoubleBlackDiamond.labeledDoubleBlackDiamond(40))`
Output:	◆◆ EXPERTS ONLY

H. Promote to Toolbelt

When you realize that a method is **useful in multiple contexts**, do consider adding it to your toolbelt. We had to create a rhombus to create the double-black diamond graphics. We don't want to duplicate our code.

Thus, let's "promote" the `rhombus` method from class `Rhombus` to our `Toolbelt` class. We will keep our `Toolbelt` class around for future homework assignments, and so we will always have the `rhombus` method readily available.

Task H1

Classes:	`Toolbelt, Rhombus, DoubleBlackDiamond`
Task:	Copy the `rhombus` method from class `Rhombus` into class `Toolbelt`. You now have multiple copies of the same code! Bad!
Task:	Replace the implementation (the body of the method) of `Rhombus.rhombus` with a call to `Toolbelt.rhombus`. Make sure that your `Rhombus.rhombus` method still produces a rhombus! We don't want to have code duplication, but we still want to keep a working Rhombus class.
Task:	Make sure that `DoubleBlackDiamond.blackDiamond` calls your `Toolbelt.rhombus` method. Make sure that calling your `DoubleBlackDiamond` class methods still produces the expected graphics!

I. Eyes

Task I1

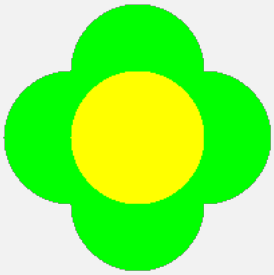
Class:	`Eyes`
Task:	Create a method named `eye`, with one parameter named `diameter` of type `double`, which specifies the outer diameter of the eye. The diameter of the pupil should be 55% of the diameter of the whole eye. Your method should create an eye, like the one below. **Assert** that the parameter has an acceptable value. **Use** your toolbelt.
Run in JShell:	`show(Eyes.eye(50))`
Output:	

Task I2

Class:	`Eyes`
Task:	Create a method named `eyes`, with one parameter named `diameter` of type `double`. Your method should create a pair of eyes, like the one below. **Assert** that the parameter has an acceptable value. **Use** your toolbelt.
Run in JShell:	`show(Eyes.eyes(50))`
Output:	

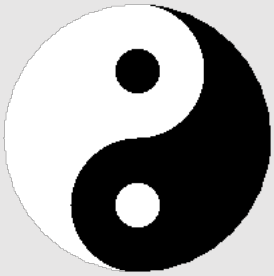
J. Flower

Task J1

Class:	`FourPetalFlower`
Task:	Create a method named `flower`, with one parameter named `diameter` of type `double`. Your method should create a flower like the one below. Create additional methods for various parts to keep methods small. **Assert** that the parameter has an acceptable value. **Use** your toolbelt.
Run in JShell:	`show(FourPetalFlower.flower(100))`
Output:	

K. Yin and Yang

Task K1

Class:	`YinYang`
Task:	Create a method named `yinYang`, with one parameter named `diameter` of type `double`. Your method should create a yin and yang symbol like the one below. Add additional methods, with well-chosen names, for the various components that make up the yin and yang symbol. **Use** your toolbelt.
Run in JShell:	`show(YinYang.yinYang(200))`
Output:	

L. Me

Task L1

Class:	`Toolbelt`
Task:	In your `Toolbelt` class, create the following seven methods, each producing a value of type `String`: 1. `firstName` – produces your first name 2. `lastName` – produces your last name / surname 3. `fullName` – produces your full name (concatenation of first name, a space, last name) 4. `usiUserName` – produces your USI username 5. `usiLongEmail` – produces your long USI email address (based on your full name) 6. `usiShortEmail` – produces your short USI email address (based on your USI username) 7. `gitHubUserName` – produces your GitHub username To avoid code duplication, where possible, use the string concatenation operator `+` to produce a string from other strings. For example, to implement the `fullName` method, use the `firstName` and `lastName` methods you implemented earlier.
Run in JShell:	`Toolbelt.firstName()`
Value:	`Matthias` (yours will differ)
Run in JShell:	`Toolbelt.lastName()`
Value:	`Hauswirth` (yours will differ)
Run in JShell:	`Toolbelt.fullName()`
Value:	`Matthias Hauswirth` (yours will differ)
Run in JShell:	`Toolbelt.usiUserName()`
Value:	`hauswirm` (yours will differ)
Run in JShell:	`Toolbelt.usiLongEmail()`
Value:	`Matthias.Hauswirth@usi.ch` (yours will differ)
Run in JShell:	`Toolbelt.usiShortEmail()`
Value:	`hauswirm@usi.ch` (yours will differ)
Run in JShell:	`Toolbelt.gitHubUserName()`
Value:	`hauswirth` (yours will differ)

Task L2

Class:	`Me`
Task:	Create a method named `twistedName`, with one parameter named `fontSize` of type `double`. Your method needs to produce a graphic showing your blue first name above your rotated red last name, in the given font size (use `SANS_SERIF` as the font). **Use** the toolbelt methods you just created!
Run in Code Pad:	`show(Me.twistedName(40))`
Output: