Worksheet 2: Polymorphism

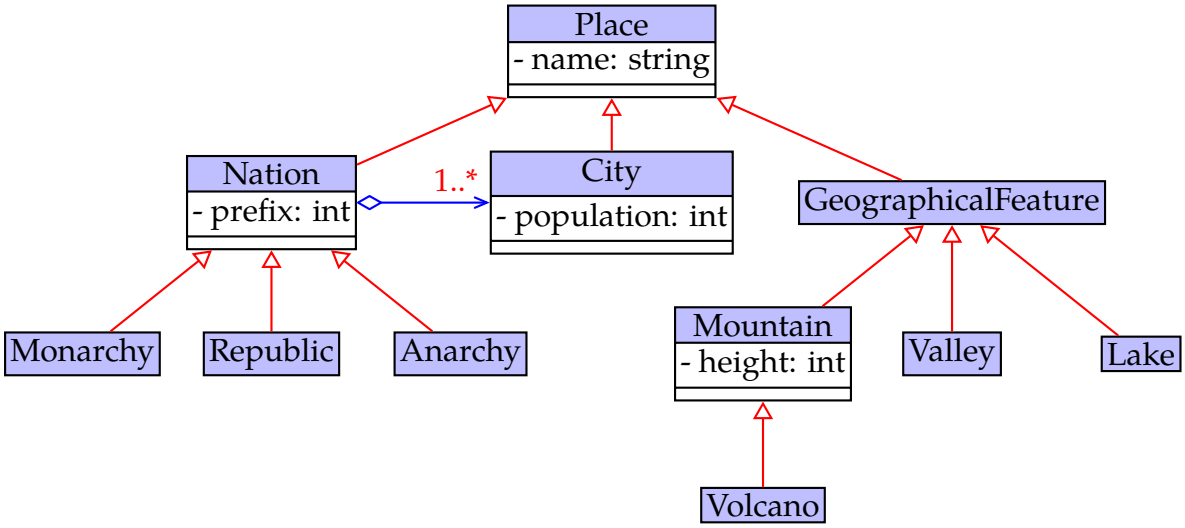
Updated: 5th March, 2019

Note: Despite all the code in this worksheet (and those following), remember that this unit is all about *design*. Typically, once you have a workable design, it ought to be clear that it will work.

While testing is important in general, it can't tell you whether you have a good design or not. That's what your reasoning skills are for!

1. Discussion: Reasoning about Types

Consider the following inheritance hierarchy:



For each of the following, what class name(s) could you use in place of "???"? (Where there are multiple ???s, each one can be different.)

(a)
```
??? var = new Mountain();
```

(b)
```
Mountain var = new ???();
```

(c)
```
???1 var = new Mountain();
???2 var2 = var;
```

(d)
```
public Nation createNation(???1 x)
{
    return new ???2(x.getName());
}
```

(e)
```
public ??? getBetterPlace(Place p)
{
    if(p instanceof Anarchy)
    {
```

```
        return new Republic();
    }
    else if(p instanceof Valley)
    {
        return new Volcano();
    }
    return null;
}
```

(f)
```
private boolean isInside(GeographicalFeature f, ???1 y) {...}

public ???3 findInside(???2 thing, List<Nation> nationList)
{
    for(Nation nat : nationList)
    {
        if(isInside(thing, nat))
        {
            return thing;
        }
    }
    return null;
}
```

(g)
```
public void find(List<???1> list, int code)
{
    for(???2 element : list)
    {
        if(test(element, code))
        {
            System.out.println(element.getName());
        }
    }
}

public boolean test(???3 element, int code)
{
    return element.getPrefix() == code;
}
```

2. Discussion: Template Methods in the Java API

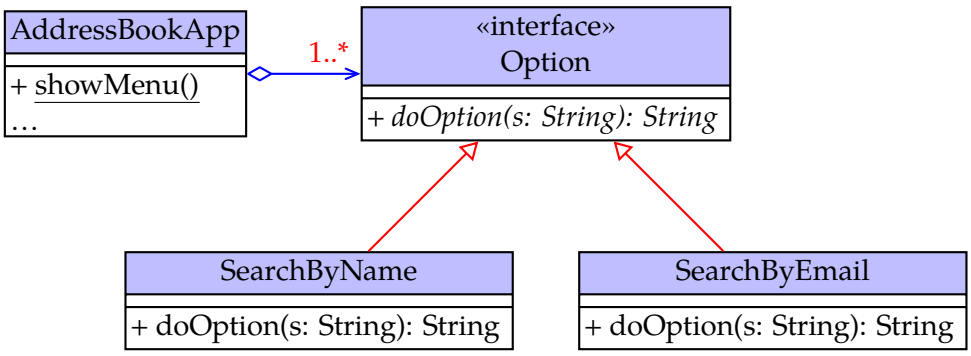
The standard Java classes `InputStream` and `OutputStream` are two cases in which the Template Method Pattern (slightly modified) has been used. Can you explain how?

Hint: look at the various `read(...)` and `write(...)` methods in the Java API documentation.

3. The Strategy Pattern

(a) Consider the AddressBook application again. It asks the user to enter a number, then searches by name or email (or quit) depending on the number entered.

However, consider the following changed design:



Refactor your code (based on the above diagram) so that `AddressBookApp` does the following instead:

- Stores `Option` objects in a container (in a static field), to be initialised in `main()`. Each option must be identified by an integer "label", so that the right one can be found when required.
- When `showMenu()` is called, it asks the user to enter a number and then a search term (unless the number is the quit option).
- Finds the corresponding `Option` object (if any).
- Calls the `doOption()` method on that object, and displays whatever it returns.

You can still hard-code a number for the "quit" option, since that's a more natural part of `showMenu()`.

You'll also need to create the `Option` interface and its two implementing classes. Have their `doOption()` methods perform the search and *return* (not display) the entry string to be displayed.

> **Note:** Although they are side issues, the following are also both good ideas:
>
> - Converting `AddressBookApp` to use *non-static* methods and fields. Keep `main()` static (because you have to), and in `main()` create an `AddressBookApp` *object*.
> - Have an `addOption()` mutator in `AddressBookApp`. This will help us split up `AddressBookApp` into two parts later on.

(b) Determine how you could add another option to display all entries.

This *does not* require a search string. So, to fit it in with the two search options, add a "`boolean requiresText()`" method to the `Option` interface and its subclasses. This should be called by `AddressBookApp` to determine whether to ask the user for a search term.

4. The Template Method Pattern

Create a design, based on the Template Method Pattern, for performing a set of three image operations: scaling, rotating, and colour-inverting. Algorithms for these operations are shown below.

Your design should involve a set of new classes and relationships. You should aim for maximum code re-use! Draw your design in UML, and code it in the language of your choice.

Note: We're certainly not claiming that the following code is the best way to perform these operations. It isn't! It's just written this way for simplicity.

- Scaling by a factor of 0.5:

```
ImageData scale(ImageData oldImage)
{
    int newWidth = oldImage.getWidth() / 2;
    int newHeight = oldImage.getHeight() / 2;
    ImageData newImage = new ImageData(newWidth, newHeight);
    for(int y = 0; y < newHeight; y++)
    {
        for(int x = 0; x < newWidth; x++)
        {
            newImage.setPixel(x, y, oldImage.getPixel(x * 2, y * 2));
        }
    }
    return newImage;
}
```

- Rotating 90 degrees:

```
ImageData rotate(ImageData oldImage)
{
    int newWidth = oldImage.getHeight();
    int newHeight = oldImage.getWidth();
    ImageData newImage = new ImageData(newWidth, newHeight);
    for(int y = 0; y < newHeight; y++)
    {
        for(int x = 0; x < newWidth; x++)
        {
            newImage.setPixel(x, y, oldImage.getPixel(newHeight - y, x));
        }
    }
    return newImage;
}
```

- Colour-inverting (nb. "~" is the bitwise "not" operator):

```
ImageData invert(ImageData oldImage)
{
    int newWidth = oldImage.getWidth();
```

```
    int newHeight = oldImage.getHeight();
    ImageData newImage = new ImageData(newWidth, newHeight);
    for(int y = 0; y < newHeight; y++)
    {
        for(int x = 0; x < newWidth; x++)
        {
            newImage.setPixel(x, y, ~oldImage.getPixel(x, y));
        }
    }
    return newImage;
}
```

Note: these algorithmic operations use a class called `ImageData`, which is just a wrapper around a 2D array of `int`s, as follows:

```
public class ImageData      // Don't modify this particular code.
{                           // (It's not the interesting bit!)
    private int[][] image;

    public ImageData(int width, int height)
    {
        image = new int[height][width];
    }

    public void setPixel(int x, int y, int value)
    {
        image[y][x] = value;
    }

    public int getPixel(int x, int y)
    {
        return image[y][x];
    }
}
```

End of Worksheet