Abstract Factory Pattern Exercise
IN710 Object Oriented System Development

Introduction

The *Abstract Factory Pattern* can be used to create factories that produce families of related objects.

In this exercise you will implement an abstract factory pattern to represent postal addresses and phone numbers in the USA and the UK.

sectionThe problem We want to model personal contact information for people in various counries, for now we'll focus on the US and UK. We will store and represent postal addresses and phone numbers. This means that our classes will need to adapt to the data and display formats of the two countries. When we create a US contact, we want a US address and a US phone number. For UK contacts, we will use UK addresses and phone numbers. This mean that right now we have four classes in two groups on our hands.

- US addresses and US phone numbers
- UK addresses and UK phone numbers

We need to create the right set of objects at runtime based on the locations of particular contacts. This is the kind of situation where the *Abstract Factory Pattern* works well.

1 The task

You need to define `AbstractPhoneNumber` and `AbstractPostalAddress` classes. classes. From these, derive `UKPhoneNumber`, `USPhoneNumber`, `UKAddress`, and `USAddress` Classes. These derived classes will validate their data and need to supply `print` methods that print their data in the correct formats.

Then, define an `AbstractContactFactory` class that provides two public methods, `create_phone_number` and `create_address`. From this, derive the classes `UKContactFactory` and `USContactFactory` that implement create methods that return the right kinds of contact objects.

Don't forget to write appropriate unit tests.

1.1 Abstract classes in Python

Abstract bases classes are not often used in Python. In the absence of compile-time type checking they are less important than they are in language like Java or C++. But they still have some value, in part in defining the interface derived classesa are expected to support. There are a couple of ways you could define an abstract class.

Example 1:

```
class FooClass:

  def bar():
      pass

  def quux(a):
      pass
```

Or, if you want to enforce the abstract nature a bit more,
Example 2:

```
class FooClass:

  def bar():
      raise NotImplementedError()
```

1.2 Phone numbers

US phone numbers are 10 digits long. The first three digits are the area code. The next three are the prefix and the remaining four are the number. Display them with a hyphen between each part, e.g., 206-555-1212.

UK phone numbers are 10 or 11 digits long. They begin with a three to six digit area code that begins with a zero. Next is an optional prefix that is three or four digits long. The remaining digits are the individual number. Display them with brackets around the area code, e.g., (020) 3221 8735.

1.3 Addresses

US addresses have the following format:

```
recipient
organisation (optional)
address line 1
address line 2 (optional)
city, state post code
```

The state/territory field is a two letter abbreviation. Postcodes are nine digits long.
UK addresses have the following format:

```
recipient
organisation (optional}
building name (optional)
address line
locality (optional)
city
post code
```

UK post codes are seven alphanumeric characters, e.g., SO31 4NG.
Note that these address specifications are somewhat simplified. Real world addresses are horribly messy.