Download code from github/gigjacker/rlg-dotnet-fun and extract the Kata folder to do these exercises

1 Publishing House revisited.

----------------Part 1 (easy)----------------

Follow the factory pattern and produce an object hierarchy for a publishing house.

There are a number of printing machines that can produce can produce publications in the formats shown at the foot of this description.

Your objectives for Part 1 are:

-- to produce a structure that reflects the division of labour across the available machines.
-- to note which extra detail can be set aside in order to achieve the first objective.

----------------Part 2 (not too bad)----------------

-- abstract the extra data into groups and encapsulate the groups in the appropriate data types
-- encapsulate the new types in a single class that can be used across the entire system

----------------Part 3 (not so easy)----------------

-- embed a second factory pattern in the same way as the MenuItem -> Variant factory was produced and incorporate your encapsulated data to represent every type of publication detailed below.

-- display the entire printing systems capabilities in a loop exactly like the other factory pattern loops, overriding ToString() on the appropriate objects and make any other additions you need to standardise your formatting so that main factory pattern loop does not require modification.

----------------Printing formats----------------
Paperbacks
- A6 pages
- soft cover
- bound with gum

Firmcover
- B5 pages
- firm cover
- bound with gum

Hardbacks
- hardcover
- some have dust jackets
- novels, technical books and reference books are printed on A5 pages and bound with cord
- spiral bound books
--- cookbooks, B5 glossy paper
--- diaries, A6 and A4 matt paper
--- sketch pads, A3, A4 and B4 paper

Magazines
- glossy cover
- A4 glossy pages
- bound with gum

Comics
- matt cover
- A4 recycled pages
- bound with staples

One printing press produces hardcover publications. A second machines handles all the others.

2 Part I (easy)

Use the factory pattern to design a class hierarchy for recycling. There is a separate bin for each material. Glass, Paper, Tin, Plastic and Organic.

Your design should show specific items of each material being contained in the appropriate receptacle.

Part 2 (not too bad)

Write a loop to get input from the console to select a specific item and report back the bin it should go in. Use polymorphism to identify the correct bin.

Part 3 (not so easy)

Create a list of items to be recycled and populate it with a random selection of items. Loop thru the resultant list, using polymorphism to identify the correct bin.

3 Part 1 (easy)

use the factory pattern to design a class hierarchy for the booker of a music festival. the booker has a number of venues to schedule. each is suited to a particular style of music: indie, folk, jazz & prog-rock

your design should show how artists or bands are distributed among the venues.

Part 2 (not too bad)

each venue has different opening times and the changeover between artists varies according to the style of music. set length is 45 minutes for all artists. extend your design to reflect these requirements.

Part 3 (not so easy)

Either by getting input from the console or by generating different artists at random, fill the schedule fo the festival across all venues. each artist plays one style of music only and has a finite length of time they're able to play for.

Accept or reject the artists applications based on:

* reject the artists that cannot play for long enough to fill the time slot.
* reject the artists that the schedule for that style cannot accommodate because it's full

-----

Part 4 (pushing it)

let each artist have a reputation and order them in the schedule so that the act with the biggest reputatio plays last on the schedule.

as new applications are encountered, move the already scheduled artists to earlier in the schedule as required.

if the schedule is full, allow higher reputation acts to push the lower profile acts off the schedule.

show the process taking place with messages to the console.

4 Part 1 (Easy)

Use the factory pattern to create an object hierarchy for a game of draughts or backgammon.

The creator is the game board and the sets of pieces are the product.

Draughts has 2 sets of 12 identical pieces - one white and one black.
Backgammon has 2 set of 15 identical pieces - one white, one black and a pair of dice.

Part 2 (Not too bad)

Extend your design to include a second factory pattern, allowing the set of pieces to be the creator and th
individual piece to be the product.

Part 3 (Not so easy)

Extend your design to include chess.

For chess you'll need a King, Queen, 2 Bishops , 2 Knights, 2 Rooks and 8 Pawns each for black and white.

Part 4 (Pushing it)

Extend your design to include the playing area. By default this will be an 8 x 8 grid but in the case of backgammon, this grid is 24 x 5. Your design should provide for using the same playing area Property to implement either board so that the code referring to the playing area can be reused transparently.

Display the boards in a rough representation of their real life arrangement, with the pieces at their starting positions.

Use your google-fu to get those starting positions if you're not familiar with the game(s) :-)

Part 5 (You must be joking)

(Seriously, people, I really am joking with this one. I'm only putting this here so I don't forget. A bit further down the road when we're focusing on Interface Segregation, this is gonna be a good exercise :-))

* Write an interface to manage playing each game.

* Establish the commonality between your interfaces and segregate them into their abstract functions.

5 use the factory pattern to set up a class structure for different types of album. you should take into consideration the number of concrete creators you identify so that you choose the best element as your abstract product.

the elements to consider:

- Vinyl has 2 sides, a CD has 1
- An album can be single, double, triple or a boxset with any number of discs.
- on a compilation album, each track is by a different artist.

hint: the backing store doesn't have to be a "List"...

6 use the factory pattern to set up the finer detail of the bookshop system.

each book has:
- title,
- author(s)
- isbn
- genre(s)
- age range
- language
- media - print, audio or digital

the bookshop system is able cross-reference or group the books in multiple ways

7 use 1 or more factory patterns to represent the composition of a wide variety of board games.

the list below provides the contents of a few board games, compare these lists to derive your abstract view of their components and establish their commonality. you can include any board game you like - you're not restricted to the ones listed here.

Monopoly
- Board
- 3 decks of cards
- 7 types of cash
- 2 dice
- 6 mascots/counters
- houses
- hotels

cluedo
- pads
- pencils
- a die
- character pieces
- 3 decks of cards
- murder envelope
- board

let's buy hollywood
- board
- 5 decks of cards
- 7 types of cash
- 2 sets of mascots

snakes and ladders
- board
- die
- counters

backgammon
- board/box
- 2 sets of counters
- 2 dice

trivial pursuit
- board
- question cards
- pie counters
- pie slices
- dice

cranium
- board
- 4 decks of activity cards
- plasticine
- pencils
- paper
- hourglass
- die

"

8

```
Use 1 or more factory patterns to setup an object hierarchy for a tidying up app.  the initial
implementation is to tidy a bedroom.

Storage space:

The following containers are available:

Wardrobe
- Dimensions
  -- Shelf  (x 5)
     W x H x D = 10 x 12 x 16
  -- RailWell (x 1)
     W x H x D = 40 x 24 x 16
  -- Rail (x 1)
     W x H x D = 36 x 36 x 16
  -- Hanger
     W x H x D = 2 x 36 x 16
 A hanger can hold 1 x Trousers + (1 x Shirt or 1 x Jacket)
  -- Roof (x 1)
     W x H x D = 40 x 24 x 30

- Quantity: 1

Drawer
- Dimensions: W x H x D = 24 x 12 x 26
- Quantiy: 5

Box
- Dimensions: 10 x 10 x 10
- Quantity: 4


Items to tidy:

Shirt
- Primary storage location = Hanger (Dimensions hung = 2 x 36 x 16)
- Secondary storage location = Shelf (Dimensions folded = 16 x 6 x 12)
- States = clean, dirty, torn

T-Shirt
- Primary storage = Shelf (Dimensions folded = 16 x 4 x 12)
- Secondary storage = Drawer (Dimensions folded = 16 x 4 x 24)
- States = clean, dirty, torn

Trousers
- States = clean, dirty, torn, casual, smart
- Primary = Wardrobe (Dimensions hung 2 x 30 x 16)
- Secondary = Shelf (Dimensions folded 16 x 6 x 10)
- Tertiary = Drawer (Dimensions folded 16 x 6 x 10)
- Restrictions
-- smart can only go to primary
-- casual can only go to secondary or tertiary

Pants
- Dimensions 1 x 4 x 3
- Primary = Drawer
- States = clean, dirty, torn

Socks
- Dimensions = 1 x 1 x 1
- Primary = drawer
- states = clean, dirty, torn, complete, incomplete
- set size = 2

Shoes
- Dimensions = 2 x 4 x 4
- Primary = railwell
- states = complete, incomplete
- set size = 2

CD
- Dimensions: 2 x 0.5 x 2
- Primary = Box
- states = viable, missing case, missing disc, scratched

LP
- Dimensions: .25 x 6 x 6
- Primary = Box
- states = viable, missing case, missing disc, scratched

Games/DVDs/Bluray
- states = viable, missing case, missing disc, scratched
- Primary = Box
- Dimensions = .4 x 5 x 3

PlayingCards
```

9 define an interface to manage a recycling plant. you'll need methods to:

- direct the trucks to the correct silo for their material load (glass, plastic, etc)

- separate the different types of glass into stained glass, shatter glass, glassware, ceramics

- separate different types paper as for glass. those types are arbitrary for illustration.

- clean stickers etc off the glass items

- crush cardboard boxes

- separate plastics

- melt plastics

- unbag organic waste and put in composter

- dispose of organic waste bags in landfill silo.

the method signatures are all you have to write. the types named are abitrary and you can use types that don't exist etc to get your concept across.

10 part 1

define an interface that accepts a generic type. the interface is IPlayMedia and the method is Play.

The Play method accepts a type describing a media item that has not been defined as part of a hierarchy.

Possible items are Vinyl12, Vinyl7, Vinyl10, C60, C90, C120, CD

part 2

define a class that implements the interface for the Vinyl12 type.

implement the Play method and write the FullName of the type to the console.

part 3

modify the class so that it accepts a generic type, which it passes on to the interface.

create stub classes for the media types listed above. (e.g. public class Vinyl12 {} is all you need)

write a program to instantiate the PlayMedia<T> class specifying each of your stub types in turn.

part 4

create an interface IMediaMaintenance with a generic type and method Clean that takes an argument of the same generic type.

modify your stub classes to implement this interface and write out a message saying "cleaning " + the name of the type passed in.

modify your program to clean each media item before playing.

part 5 (getting trickier)

create a parent class for your media types and have them inherit from it as well as implementing the interface.

add a boolean property IsClean to the parent class.

combine your knowledge of access modifiers, properties and anonymous methods to allow:

-- IPlayMedia.Play to access the IsClean property but not modify it.
-- IMediaMaintenance.Clean to set the property to true before returning.

modify your implementation of Play to write a message out if the media is not clean and then clean it.

11 Define as many interfaces as ISP (Interface Segregation Principle) dictates to represent the operational components of goods-in department at a large company. Over the course of a week the following take place

- deliveries arrive by lorry, van and courier.
- van and lorry deliveries are ID checked at the rear security gate.
- courier deliveries are made to reception, which are then passed on to goods-in for processing. courier i
is not checked but a full airport-style scan is required.
- lorry and van deliveries are made to goods-in and the items security scanned before entering the premises
- food and beverage items are taken to the kitchen, a tracking receipt is obtained with a contact name and employee ID.
- stationery is taken to the office manager to be signed for.
- out of hours (mon-fri 8-4.30) kitchen and office supplies are placed in a holding cage to be delivered on the next working day.
- letters and parcels with department or individual addresses are sorted by department and floor.
- department deliveries are carried out at 11am and 4pm.
- shift changes occur at 7am and 3.30pm.
- a handover of ongoing tasks is done between shifts.
- the night shift begins at 23:30. a security guard is also on this shift.
- weekend shift is 7-19 and 1900-0700 with security present from night-shift Friday to morning shift Monday
- Tuesday for Bank Holidays.
- out of hours deliveries are permitted with prior arrangement and authentication. ight time deliveries an
weekend
- shifts must be full staffed. on-call team members are available to fill shifts at short notice.