pp. 211-17

Encapsulation

* Keeping some data in class private.
* Keep methods and fields private from other classes.
* Can only be accessed from inside the class.
* Helps prevent bugs.
* Easy to maintain- use method to access private data rather than access field directly.
* Keep fields up to date.
* Make changes to how things operate whilst not affecting rest of software.
* E.g. agent name – only access by using methods that have access to private parts of class.
* E.g. if you had access to random numbers, you could input values that aren’t random.
* Chef –1) can’t assign int on string, temperature 2) ingredient supplier 4) secret ingredient. 7) Customer amount was set to 60 - so 54 won’t work – wont print secret ingredient.

Decomposition – Process of break down complex solutions into strict bullet point statements (e.g. breaking down brief to make requirements).

Abstraction – Only using required data. Limiting how much we need (e.g. booking reference + seat numbers for plane, don’t need fuel, engines).

Conditional (If/Selection) Statements + switch cases

* Used to change outcomes of data being handles
* If(condition is true){ //do something}

Else if (different condition){ do something else}

Else {final do something else}

* Avoid nested conditionals – use switch case
* Can’t use switches to test values – use for constants

Pp.324-331

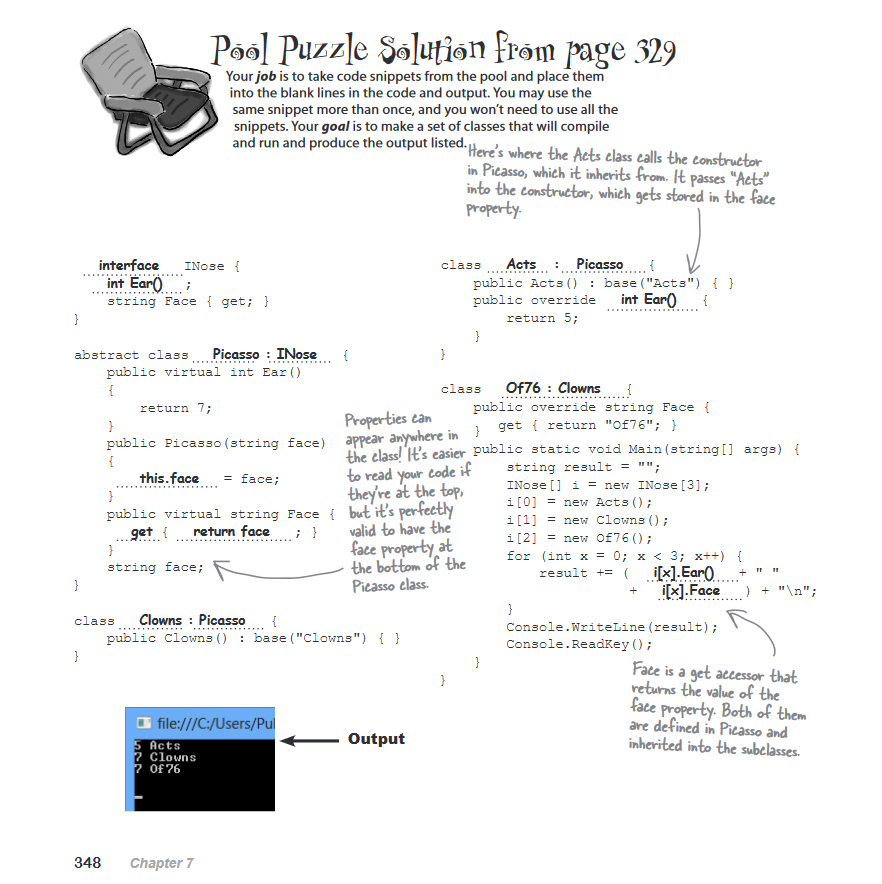
Abstract Class

* Type of class that can’t be instantiated
* Inherited by subclasses that implement or override its methods
* Either partially implemented or not implemented at all
* Have virtual methods, abstract methods or concrete
* Can’t inherit from 2 classes
* Declared without implementation

Abstract method

* Declared with no implementation
* Declared with purpose of having child class provide implementation
* Interfaces implement inheritance, don’t use them.

Polymorphism – one object can take different forms



Arrays

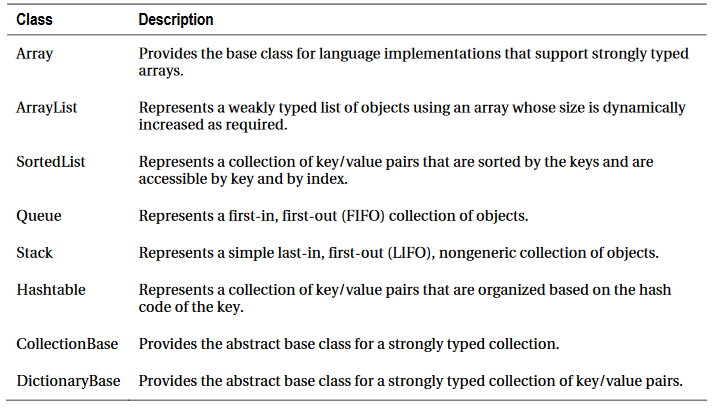
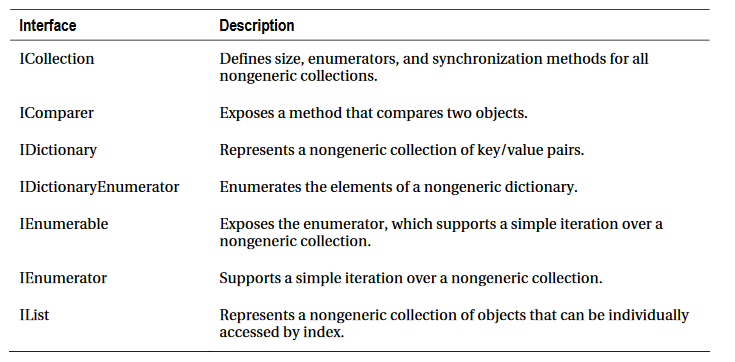
* Another method to store data into values
* Pieces of info that share same name
* Fixed number of variables in same type
* 2 types
* Single Dimension (4x1 grid)
* Instead of int num1, int num2 etc. – use var num = new int [5]; = 5 int with 5 values.
* To reference – use index. E.g. to access int num3 – reference num2. (0-5)
* Arrays start at 0.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 22 | 23 | 24 | 25 | 26 |

* Multidimensional arrays 2D or 3D – grids of data built into a structure (4x4 grid or cube). –More sophisticated management of data that 1D array could not handle as well.
* Rectangular or Jagged array.
* Jagged – unequal distribution, different lengths of grid (e.g. 4 rows top 5 rows middle, 3 rows bottom). Single arrays stacked on one another.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |

Collections

* Specialized classes for data storage and retrieval
* The System.Collections namespace contains interfaces and classes that define various types of collections, such as lists, stacks, queues, hash tables, and dictionaries.
* 
* 
* Generic Collection - The limitation of these collections is that while retrieving items, you need to cast into the appropriate data type, otherwise the program will throw a runtime exception. It also affects on performance, because of boxing and unboxing.
* To overcome this problem, C# includes generic collection classes in the System.Collections.Generic namespace.
* 