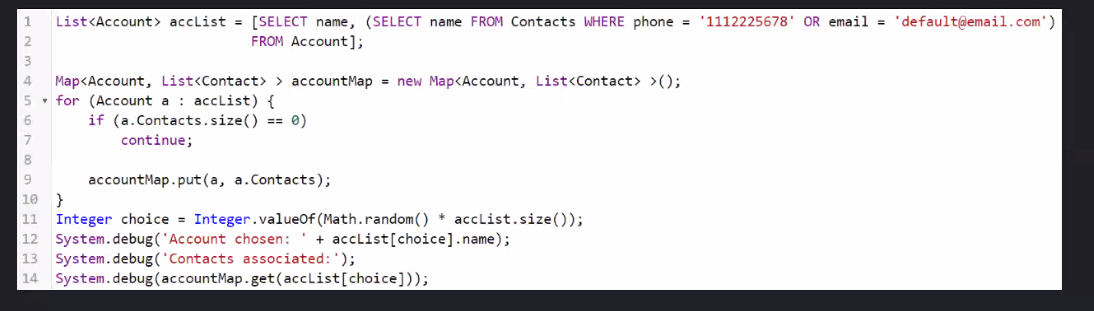
* Solution for coding exercise 4
  + 
  + You can only access what you query for
* SOQL
  + We have governor limits
    - Salesforce puts the limits on us
    - Main limitations are on SOQL and DML
    - Limits to remember for SOQL
      * 100 synchronous
        + In sequential order
      * 200 asynchronous
        + Only runs when there are resources to do so
        + More flexibility when it is run
      * These are per transaction
    - If you exceed these limits, everything gets rolled back
      * Don’t do that
* DML
  + 150 allowed operations per transaction
* Best practices
  + Avoid SOQL and DML in loops
    - You can do this usually by getting everything you need at the start and then operating on that
  + There is a limits class that can help track how many we have left
    - Methods
      * Get number you have used
        + Limits.getQueries()
        + Limits.getDMLStatements()
      * Get number you have access to at peak
        + Limits.getLimitQueries()
        + Limits.getLimitDMLStatements()
    - These can help track where we are so that nothing is rolled back
      * Kill the loop and move on with the program
* Transaction control language
  + Pretty niche
  + Set save points
  + Roll back to save points
  + Example
    - Savepoint sp = Database.setSavepoint();
    - Database.rollback(sp);
  + Any DML after a save point that is rolled back is discarded
  + Still under the same governor limits
  + Making a save point counts as a DML call
* Object Oriented Programming
  + 4 pillars
    - Encapsulation
      * In apex, is controlled by access modifiers, static and instance keywords
      * Practice of taking your data and ensuring outside access to that data is controlled
      * 4 access modifiers
        + Global

Data can be accessed in any namespace

* + - * + Public

Data can only be accessed within the same namespace

* + - * + Protected

Only the parent and child classes can access the data

* + - * + Private

Only the owning class can access the data

* + - Polymorphism
      * Objects can take on many different forms
      * This is the parent/child relationship
        + The child is both a child and parent object, but the parent is only a parent object
  + Definitions
    - Namespaces are grouping
    - Most classes are public
    - Get and set methodology
      * You can validate data someone puts into the list
      * Also called public through private
    - Cloning
      * A deep copy is a complete duplicate of the value in memory
      * Do this to operate by value instead of reference
    - Property
      * A variable with get and set methodology
    - Static belongs to all instances of the class
      * Is the same across every object of the class
      * Also allows you to call it without an instance of the class
      * You can also do a static method with no name that is called the first time the class is used
      * Static variables can only be accessed in a static method
        + Likewise, static methods can only access other static methods
    - Abstract keyword
      * Taking what doesn’t need to be know now, and putting it off till the future
      * Can make abstract classes and methods
      * Abstract methods have no body
      * Another tool for abstraction is an interface
        + Not a class but think of it as a class with only abstract methods
        + All methods must be public or global
        + Methods in an interface can only be abstract
        + Think of it as a contract

If you want to use this class, you must implement these methods

* + - * + Abstract classes cannot be instantiated
    - Final cannot be changed
      * Any class to be extended can either abstract or virtual but not final
    - Can use implements keyword to simply implement
      * We can implement as many as we want
      * This gets around the rule of only extending one class
    - Super() calls parent constructor
      * This helps keep your code clean and modular so that you can have children easily implement your code
    - Override is the only way you can override a function
      * This means it gets priority
      * It is similar to overloading, but the signature is the same
    - Virtual
      * Can be overridden
      * Has a body unlike abstract
    - This
      * References the current object
      * Differentiate between parent and child
      * Perform constructor chaining
    - Super
      * References the parent object
      * Can call parent constructor and parent methods
    - Text

      Description automatically generated
* Interesting things in Apex
  + Can define a class within a class
  + Access modifier cannot be less struct than the outer class
  + Exceptions
    - Try catch finally
      * Try – code we think will throw an error
      * Catch – what happens if we get an error
      * Finally – executes regardless of if caught or not
    - Exceptions are also polymorphic
    - Can specify which type in a catch parameter
    - Put in order of most specific to least specific
    - Exception methods
      * getMessage() – returns the message that is thrown without displaying directly
      * getLineNumber() – returns the line number thrown on it
      * getStackTraceString() – gets the stack trace
      * setMessage() – can set the message to be whatever we wnat
    - Only the system can throw exceptions
    - We can create our own custom exceptions
      * Often, we use an inner class of the exception class to do this
      * Throw new innerException(‘Bad new’)
    - Know where to find exceptions, not all the different ones you might get
* Triggers
  + A block of code that is executed on certain DML events
  + Before and after triggers
  + Before trigger happens right before DML statement executes
  + After trigger happens right after DML statement executes
  + Example
    - Trigger AccountTrigger on Account(before insert, before update, before delete, after insert, after update, after delete, after undelete){
    - After undelete has no before, only after since it doesn’t make sense to do it before
  + Best practice
    - Have a single trigger for each object
      * Otherwise we will not know which fires first, you want to have full control
    - Use switch-when statements to specify what to do based on which trigger
    - Graphical user interface, text, application, email

      Description automatically generated
  + Use handler classes to do all the work in triggers
    - Don’t actually write code in the trigger itself
    - Handler class should be static so you don’t have to instantiate it
    - Trigger context variables tell use about the trigger that was fired
      * Common one is trigger.operationType
      * trigger.new holds a list of objects that caused the trigger to fire
      * trigger.old is only available after update and delete
        + Holds the old, unmodified version
        + This helps validate that the change actually happened
    - Bulkify your handler
      * Make it available for any number of items
    - Often it is helpful for validation
      * Make it complicated
      * We have another tool for simple validation
    - Use maps to have little to no SOQL and DML in the handler
  + Avoid recursion in triggers
    - We are limited as to how many layers deep we can go
  + Handlers can do object.addError(‘This is an error’) and it will cause the insert to fail, which can cause all of them to fail
  + Triggers do not actually need to insert or delete… because that is handled afterward
* After triggers
  + Cannot modify fields
  + Typically, you want to perform logic on other objects