**Chapter 4 Goals:**

* I understand the concepts of collections and generic classes
  + Collection objects can store an arbitrary number of other objects, therefore allowing us to refer to them and manage them all together.
  + Generic classes are not limited in what it can store but will require a second type to be specified as a parameter when the class is used to declare fields or other variables
    - ArrayList <String>
* I can use a for-loop to iterate over an Array.
  + ArrayList <String> list1 = new ArrayList<String> ();

for (int i = 0 ; i < list1.size(); i++)

{

String p = list1.get(i);

}

* I am able to create an ArrayList of specific objects
  + ArrayList <*ClassType*> *nameOfArrayList* = new ArrayList<*ClassType*> ();
  + ArrayList <String> list1 = new ArrayList<String>();
* I am able to use the methods of the ArrayList class
  + add
    - list1.add(int index, java.lang.Object o)
    - list1.add(java.lang.Object o)
  + size
    - list1.size()
  + get
    - list1.get(int index)
  + remove
    - list1.remove(int index);
  + iterator
    - list1.iterartor()
      * must create an iterator object
* I understand the difference between flexible and fixed size collections
  + Flexible collection: you can store an arbitrary number of objects in a single collection object, never have to specify how many items a particular collection will hold, number of objects can change throughout the lifetime of the object
    - ArrayList
  + Fixed size collection: the capacity of the collection is fixed, meaning that at the point when the collection is created, we have to specify the max number of items it can store and this cannot be changed
    - Array
* I can use a for each loop to iterate over a collection
  + ArrayList <String> list1 = new ArrayList <String> ();

for (String p: list1)

{

*statements*

}

* I can use an iterator to iterate over a collection
  + ArrayList <String> list1 = new ArrayList <String> ();

Iterator <String> it = list1.iterator();

while (it.hasNext())

{

String p = it.next();

}

* I am able to read and write files using Java library classes.
  + import java.util.Scanner;

import java.io.File;

import java.io.FileNotFoundException;

public static void main(String [] args) throws FileNotFoundException

{

File text = new File(“*NAME*.txt”);

Scanner scan = new Scanner(text);

String line = “”;

while(scan.hasNextLine())

{

line = scan.nextLine();

System.out.println(line);

}

}

* + import java.io.\*;

public class WriteFile{

private FileOutputStream out;

private PrintStream ps;

public WriteFile(){

try {

out = new FileOutputStream(“*NAME*.txt”);

ps = new PrintStream(out);

}

catch (Exception e){

System.out.println(“Error opening the output file”);

}

}

public void write(String s){

try {

ps.println(s);

}  
 catch (Exception e) {

System.out.println(“Error when writing file.”);

}

public void close() {

try {

ps.close();

}

catch (Exception e) {

System.out.println(“Error when closing file.”);

}

}

}

* I am able to use the Random class
  + import java.util.Random;
  + Random randomGenerator;

randomGenerator = new Random();

int index = randomGenerator.nextInt(*top limit + whatever number you want to add to it*);

System.out.println(index);

* I am able to create HashMaps and HashSets
  + HashMap: A collection that stores key/value pairs as entries, values can be looked up by providing the key
    - Uses hashing technique, which a technique to convert a large String to a small String that represents the same string
    - Two types can be different
    - Allows one null key and multiple null values
    - HashMap <String, String> contacts = new HashMap<>();

contacts.put(“Charles Nguyen”, “(531) 9392 4587”);

String number = contacts.get(“Charles Nguyen”);

* + HashSet: a collection that stores each individual element at most once
    - Allows only 1 null element
    - Special case of set
    - import java.util.HashSet;

…

HashSet <String> mySet = new HashSet < > ( );

mySet.add(“one”);

mySet.add(“two”);

mySet.add(“three”);

* + - Can still use the same type of for each loop as ArrayList
* I am able to differentiate between an ArrayList and a HashSet
  + ArrayList
    - Implements List interface
    - Allows duplicates
    - Ordered collection, maintains insertion order of elements
    - Backed by an Array
    - Index based, meaning you can call get(index) or remove(index)
  + HashSet
    - Implements Set interface
    - Doesn’t allow duplicates
    - Unordered collection, doesn’t maintain any order
    - Backed by HashMap instance
    - Completely object based, doesn’t provide get() method
* I am able to write to a file using FileWriter and/or PrintWriter class .
  + import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

…

public class SimpleFileWriterExample {

public static void main(String[] args) {

try {

FileWriter writer = new FileWriter(“…*name of file…*”);

while (*there is more text to write*) {

…

writer.write(*next piece of text*);

}

writer.close();

}

catch (IOException ex) {

ex.printStackTrace();

}

}

}

* + import Java.io.PrintWriter;

import Java.io.FileWriter;

…

FileWriter writer = new FileWriter(“*…name of file…”*);

PrintWriter printWriter = new PrintWriter(writer);

printWriter.print(true);

printWriter.print((int) 123);

printWriter.print((float) 123.456);

printWriter.close();

* I am able to create an anonymous object
  + enterLot method
    - lots.add(new Lot(nextLotNumber, description)); 🡪 anonymous object
      * newLot variable is not used after it is added 🡪 avoids defining a variable with such limited ise
      * Create an anonymous object (an object without a name) by passing it straight to the method that uses it
    - Lot newLot = new Lot(newLotNumber, description);

lots.add(newLot);

* I am able to understand how to create chaining method calls
  + new Hotel().name("ARSALAN").show().order(0).order(1).eat().pay();
    - read left to right, method on the previous method

**Vocabulary:**

Collections: Objects that store an arbitrary number of other objects

For loop: An iterative control structure that is often used when an index variable is required to select consecutive elements from a collection, such as an ArrayList or an array.

While loop: repeatedly executes a target statement as long as a given condition is true

ArrayList: part of a collection framework, dynamic arrays, flexible-sized list, unsorted list

for each loop: one way to perform a set of actions repeatedly on the items in a collection but without having to write out those actions more than once

* starts out with keyword for like a normal for-loop
* instead of declaring and initializing a loop counter variable, declare variable that is the same type as the base type of the array, followed by a colon, followed by array name
* used to iterate over an array or any Collections class (ArrayList)
* for each *element* in *collection* do: {

*loop body*

}

Iterator: An object that provides functionality to iterate over all elements of a collection

* not a class because it cannot be created on its own
* interface 🡪 can be implemented by other objects (ArrayList)

Null: The Java reserved word null is used to mean “no object” when an object variable is not currently referring to a particular object. A field that has not explicitly been initialized will contain the value null by default

HashMap: A particular implementation of Map. It stores the data in (Key, Value) pairs

* Uses a technique called Hashing = converting a large String to a small String that represents the same String
* Most important methods = put and get

HashSet: Implements the Set interface, backed by a has table which is actually a HashMap instance

* Doesn’t maintain any specific order 🡪 Doesn’t guarantee the constant order of elements over time
* Permits a null element
* Can store each individual element only once

Diamond notation: < > specify the type of the elements that can be stored in the collection

* Needed for ArrayList constructor

Anonymous objects: An object without a name, created by passing it straight to the method that uses it rather than defining a variable with such limited use

Chaining method calls: When you call a chain of methods on an object

* Read left to right
* Methods to the right are performed on the object which has already already had the methods on the left performed on it