in this chapter you will learn how to determine whether two objects are equal

API stands for application programming interface. In Java, an interface is something special. In the context of an API, it can be a group of class or interface definition that gives you access to a service or functionality.

* Creating and manipulating Strings:
  + These two snippets both create a string: String name = “fluffy” String name = new String (“Fluffy”);
    - Both give you a reference variable of type name pointing to the string object.
* Concatenation
  + I understand it
* Immutability:
  + Once a string object is created, it is not allowed to change. It cannot be made larger or small, and you canot change of the characters inside it.
* The String Pool
  + Since strings are everywhere in Java, they use up a lot of memory. In some production applications, they can use up 25-40 percent of the memory in the entire program. Java realizes that many strings repeat in the program and solves this issue by reusing common ones.
  + The string pool, also known as the intern pool, is a location in the JVM that collects all these strings.
  + The string pool contain literal values that appear in your program.
    - String name = “fluffy”; //efficient because it goes into the string pool
    - String name = new String (“fluffy”); //not efficient because it does not go into the string pool and it creates a new object
* Important String methods:
  + indexOf(): looks at the character in the string and finds the first index that matches the desired value.
    - indexOf can work with an individual character or a whole string as input.
    - It can also start from a requested position.
    - Returns -1 when there is no match found
  + Substring():
    - Looks for a character in a string. It returns parts of the string.
    - The first parameter is the index to start with for the returned string. As usual, this is a zero-based index
    - If starting and ending index are equal it returns an empty string
    - If starting index is less than ending index it throws an exception
    - If ending index is out of range it throws an exception
* toLowerCase() and toUpperCase()
  + String string = “animals” System.out.println(string.toLowerCase()) does not change the string object just prints it out
    - Do not forget that strings are immutable
* Equals() and equalsIgnoreCase()
  + This methods checks whether two string objects contain exactly the same characters in the same order
* startsWith() and endsWith()
  + they look at whether the provided value matches part of the String.
* Contains(String value)
  + I got it
* Replace()
  + It does a simple search and replace on the string. There is a version that takes char parameters as well as a version that takes CharSequence parameters
* Trim()
  + There is a technique called method chaining
    - String result = “animal “.trim().toLowerCase().replace(“a”,”A”);
* Using the StringBuilder class
  + This class creates a String without storing all those iterim String value.
  + StringBuilder is mutable
* Mutability and chaining
  + StringBuilder a = new StringBuilder(“abc”) StringBuilder b = a.append(“de”)
    - Both a and b are pointing to a so all changes will occur in both
* Creating String Builder
  + You can create by giving it a string or empty or a number to show how big the value will eventually be
* Important StringBuilder Methodss:
  + charAt(),indexOf(), and substring(): work same as string class
  + append(): fully understand it
  + insert(): adds characters to the StringBuilder at the requested index and returns a reference to the current StringBuilder, however it does not replace the character
  + delete() and deleteCharAt(): removes characters from the sequence and returns a reference to the current StringBuilder.
  + Reverse(): reverses the characters in the sequences and returns a reference to the current StringBuilder
  + toString(): converts a StringBuilder to a String
* StringBuilder vs StringBuffer
  + When writing new code that concatenates a lot of String Objects together, you should use StringBuilder
  + StringBuffer is older than StringBuilder, it does the same thing but it is slower since it is thread safe.
* Understanding Equality:
  + In chapter 2, you learned how to use == to compare numbers and that object references refer to the saem object.
  + StringBuilder one = newStringBuilder(); StringBuilder two = new StringBuilder(); StringBuilder three = one.append (“a”);
    - System.out.println(one == two); this is false because they do not refer to the same object
    - System.out.println(one == three); this is true because they refer to the same object
  + Never use == to compare string objects. The only time you should have to deal with == for String is on the exam
  + Authors of StringBuilder() did not implement equals(). If you call it on two StringBuilder instances, it will check reference equality
  + An object has to implement equals in order for it to evaluate, otherwise it will return false
* Understanding Java Arrays:
  + Int [] numbers1 = new int[3];
    - When using this form to instantiate an array, all elements are set to the default value for that type, which is 0.
  + Int [] numbers2 = new int[] {42,55,99};
    - Another way of initializing above array is int [] numbers2 = {42,55,99};
* Creating an array with Reference Variables:
  + You can call equals() on an array because it is an object. A1.equals(A2) returns true because of reference equality. The Equals() methods on arrays does not look at the elements of the array.
  + The array does not allocate space for the string objects. Instead, it allocated space for a reference to where the objects are really sotred.
  + Object is a broader type than String.
  + You can’t add a stringBuilder object into a string array.
* Sorting
  + Java makes it easy to sort an array by providing a sort method.
  + You can pass almost anything to Arrays.sort()
  + Arays is the first class provided by Java we have used that requires an import.
  + If you pass a string array that has integers as the string, the sort will do it from 1 – 9
* Searching
  + Java provides a way to search, but only if the array is already sorted.
  + If the array is not sorted, then the output will be unpredictable
  + If binary search does not find the element you are looking for then it will return to you a value of where it should be inserted by negating the index and subtracting 1
* Varargs
  + Public static void main (String… args)
    - This examples uses a syntax called varargs(variable arguments)
    - You will learn how to call a method using varargs
* Multidimensional Arrays
* Creating a multi dimensnional array
  + You can create 2d and 3D arrays
  + Be careful on the exam because the creators will try to confuse you with how they create array
* Using a multidimensional array
  + Example by code
* Understanding an ArrayList
  + In Arrays you have to know how many elements are going into the array, but that is not the case in arrayList
  + ArrayList requires an import to use it (impot java.util.\* or import java.util.ArrayList)
* Creating an arrayList
  + ArrayList lsit1 = new ArrayList();
  + ArrayList lsit1 = new ArrayList(10);
  + ArrayList lsit1 = new ArrayList(list2); //copies size and elements in list2
  + ArrayList <String> lsit4 = new ArrayList<String>();
  + ArrayList <String> lsit5= new ArrayList< >();
  + ArrayList implements an interface called List. In other words, an ArrayList is a List.
  + You know you can store an ArrayList in a List reference variable but not vice versa
    - The reason is that List is an interface and interfaces can’t be instantiated
* Using an arrayList
  + Before reading any further, you will see a “class” named E. E is used by convention in generics to mean “any class that this array can hold”. If you did not specify a type when creating the ArrayList, E means objects.
  + ArrayList implements toString(), so you can easily see the contents by printing it.
  + If you do not specify a type for the array list then the type is an object.
* Remove() in arrayList()
  + The remove method removes the first matching value in ArrayList or removes the element at a specified index
* Set() in arrayList()
  + The set method changes one of the elements of the ArrayList without changing the size
  + You can’t perform a set on an index that does not exist and does not have a value since the method does not insert, it just updates.
* isEmpty() and size()
  + understood
* clear()
  + provides an easy way to discard all elemnts of the ArrayList
* contains()
  + checks whether a certain value is in the arrayList.
  + Returns true if the array has what you are looking for
* Equals() in arrayList
  + ArrayList has a custom implementation of equals() so you can compare two list to see if they contain the same elements in the same order
* Wrapper classes
  + Here is a list of what you can also insert into an ArrayList: Boolean,Byte,Short,Integer,Long,Float,Dobule,Character
  + The wrapper classes also have a method that converts back to a primitive. You don’t need to know much about the constructors or intValue() tyoe methods for the example.
  + Int primitive = Integer.parseInt(“23”); convert String to int primitive
  + Integer wrapper = Integer.valueOf(“234”); convert string to Integer wrapper class.
  + Parse and value are not applicable for characters
* Autboxing:
  + Since Java 5, you can just type the primitive value and Java will convert it to the relevant wrapper class for you, this is called autoboxing
  + List<Double> weights = new ArrayList<>();
  + Weights.add(50.5); //this autoboxes the double primitive into a Double objects
  + Be careful when you call a remove method on int array list, remove method takes both int for index, but if you want to remove an actual number then you have to autobox it
* Converting between array and list:
  + Examples using code
  + Converting from an array to a List is more interesting. The original array and created array backed List are linked. When a change is made to one, it is available in the other. It is a fixed-size list and is also knows as a backed List because the array changes with it.
* Sorting()
  + You sort an arrayList using Collections.sort(ArrayList)
* Working with Dates and Times
  + LocalDate: contains just a date – no time and no time zone.
  + LocalTime: contains just time. No date and no time zone,
  + LocalDateTime: contains both date and time but no time zone.
  + Oracle recommends avoiding time ones unless you really need them. Try to act as if everyone is in the same time zone.
  + Java tends to use a 24 hours clock
  + When creating dates (Year,Month,day) there are two formats: (int,int,int) or (int,Month,int)
  + When creating time you can do hour, mint seconds, nanonseconds
  + You can also combine date and time: year,month,day, hour, minute, second, nanosecond
  + We do not use constructors in local date and time because they are private and you are being used to use static
* Manipulating Dates and Times
  + Date.plusDays(number of days)
  + Date.plusWeeks(number of weeks)
  + Date.plusMonth(number of months)
  + Dates.plusYears(number of years)
  + You can also go backwards just replaces plus with minus
* Working with Periods
  + To check if a one date is before the other use the following method date1.isBefore(date2)
  + Java jas a Period class that you can use to increment by months.
  + Period period = Period.ofMonths(1); //create a period
  + Date.plus(period)
  + You can create different types of peridos
    - Period.ofYears(1)
    - Period.ofMonths(3)
    - Period.ofWeeks(3);
    - Period.ofDays(2)
    - Period.of(1,0,7) every year and 7 days -- y, month,days
  + You can not chain methods when creating a period. When you do chain while creating a period only the last method is used.
* Formatting Dates and Times:
  + LcaoDAte date = LocalDate.of(2020,Month.January,20)
  + getDayofWeek() Monday,Tuesday date.getMonth, date.getYear() date.getDayOfYear()
  + there are two types of date format short and medium
* Parsing Dates and Times:
  + DateTimeFormatter f = DateTimeFormatter.ofPattern(“MM dd yyyy”)
  + LocalDate date = LocalDate.parse(“01 02 2015, f);
  + LocalTime time = LocalTime.parse(“11:22”);