CLASS LIBRARIES

References:

- Paul Deitel and Harvey Deitel, ***"Java How to Program, Late Objects Version"***, Pearson Education Inc.**, 2010 (8th edition)**
- Y. Daniel Liang, ***"Introduction to Java Programming, Comprehensive Version"***, Pearson Education Inc., **2012 (9th edition)**
- David J. Barnes & Michael Kölling. "***Objects First with Java A Practical Introduction using BlueJ"***. Sixth Edition, Pearson, 2016
- https://www.javatpoint.com
- https://www.tutorialspoint.com
- https://www.geeksforgeeks.org/
- https://docs.oracle.com/

Class Libraries

- A *class library* is a collection of classes that we can use when developing programs
- The *Java standard class library* is part of any Java development environment
- Its classes are not part of the Java language per se, but we rely on them heavily
- Various classes we've already used (`System`, `Scanner`, `String`) are part of the Java standard class library
- Other class libraries can be obtained through third party vendors, or you can create them yourself

Packages

- The classes of the Java standard class library are organized into *packages*
- Some of the packages in the standard class library are:

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities
java.net	Network communication
java.util	Utilities
javax.xml.parsers	XML document processing

The import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Scanner
```

- Or you can *import* the class, and then use just the class name

```
import java.util.Scanner;
```

- To import all classes in a particular package, you can use the * wildcard character

```
import java.util.*;
```

The import Declaration

- All classes of the `java.lang` package are imported automatically into all programs
- It's as if all programs contain the following line:

```
import java.lang.*;
```

- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs
- The `Scanner` class, on the other hand, is part of the `java.util` package, and therefore must be imported

MATH

The Math Class

- The `Math` class is part of the `java.lang` package
- The `Math` class contains methods that perform various mathematical functions
- These include:
  - absolute value
  - square root
  - exponentiation
  - trigonometric functions

The Math Class

- The methods of the `Math` class are *static methods* (also called *class methods*)
- Static methods can be invoked through the class name – no object of the `Math` class is needed

```
value = Math.cos(90) + Math.sqrt(delta);
```

Math Methods

- `public static double pow(double a, double b)`
- `public static double random()`
- `public static double sqrt(double a)`
- `public static double max(double a, double b)`
- `public static float max(float a, float b)`
- `public static long max(long a, long b)`
- `public static int max(int a, int b)`
- `public static double min(double a, double b)`
- `public static float min(float a, float b)`
- `public static long min(long a, long b)`
- `public static int min(int a, int b)`

Math.random()

- public static double **random**()

  // Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
- Random number in the range (n_1-n_2)

```
(int)(Math.random() * (n2 – n1 + 1)) + n1
```

- For example,
- Random the number in the range 1-6

```
int i=(int)(Math.random()*6)+1;
```

Math Constant

- `public static final double PI`

- For example,

```
System.out.println(Math.PI);

//3.141592653589793
```

RANDOM

The Random Class

- The `Random` class is part of the `java.util` package
- It provides methods that generate pseudorandom numbers
- A `Random` object performs complicated calculations based on a *seed value* to produce a stream of seemingly random values
  - `public int nextInt(int n) //0(incl.)- n(excl.)`
  - `public int nextInt()`
  - `public double nextDouble() //same as Math.random()`
  - `public float nextFloat()`

The Random Class

- Random number in the range (n_1-n_2)

```
randObj.nextInt(n2 – n1 + 1) + n1
```

- For example,
- Random the number in the range 20-34

```
Random generator = new Random();
 int num1 = generator.nextInt(15) + 20;
```

The Random Class

- Random with seed
  - Creates a new random number generator using a single long seed.
  - The seed is the initial value of the internal state of the pseudorandom number generator which is maintained by method `next(int)`
  - The result is not truly random***
- Two ways to generate the random with seed number
  - Using constructor

```
Random r = new Random(seed);
```

  - Using `setSeed(seed)` method

```
Random rnd = new Random();
rnd.setSeed(seed);
```

OUTPUT FORMAT

Formatting Output

- It is often necessary to format values in certain ways so that they can be presented properly
- The Java standard class library contains classes that provide formatting capabilities
- The `NumberFormat` class allows you to format values as currency or percentages
- The `DecimalFormat` class allows you to format values based on a pattern
- Both are part of the `java.text` package

NumberFormat

- The `NumberFormat` class has static methods that return a formatter object

```
getCurrencyInstance()

getPercentInstance()
```

- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format

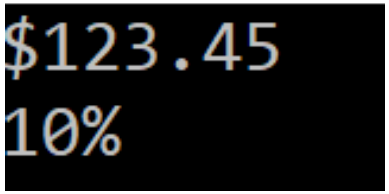
NumberFormat

```
double tax = 123.45;
double TAX_RATE = 0.1;

NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
System.out.println(fmt1.format(tax));

NumberFormat fmt2 = NumberFormat.getPercentInstance();
System.out.println(fmt2.format(TAX_RATE));
```

- Output



DecimalFormat

- The `DecimalFormat` class can be used to format a floating point value in various ways
- For example, you can specify that the number should be truncated to three decimal places
- The constructor of the `DecimalFormat` class takes a string that represents a pattern for the formatted number

DecimalFormat

```
double area = 54.352952;
DecimalFormat fmt = new DecimalFormat ("0.###");
System.out.println("The circle's area: " +
  fmt.format(area));
```

- Output

```
The circle's area: 54.353
```

printf() or format()

- The printf() is a method of java PrintStream
- System.out.printf(“format-string” [, arg1, arg2, ...]);
- Format-string can consist of:
  - % [flags] [width] [.precision] conversion-character

printf()

- Flags:
  - - :left-justify (default is to right-justify)
  - + : output a plus (+) or minus (-) sign for a numerical value
  - 0 : forces numerical values to be zero-padded (default is blank padding)
  - , : comma grouping separator (for numbers > 1000)
  - : space will display a minus sign if the number is negative or a space if it is positive

printf()

- Width:
  - Specifies the field width for outputting the argument and represents the minimum number of characters to be written to the output.

- Precision:
  - Used to restrict the output depending on the conversion. It specifies the number of digits of precision when outputting floating-point values or the length of a substring to extract from a String.

- Conversion-Characters:
  - d : decimal integer [byte, short, int, long]
  - f : floating-point number [float, double]
  - c : character Capital C will uppercase the letter
  - s : String Capital S will uppercase all the letters in the string
  - h : hashcode A hashcode is like an address. This is useful for printing a reference
  - n : newline Platform specific newline character- use %n instead of \n for greater compatibility

  For more character conversion, visit
  https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

printf()

```
long n = 461012;
System.out.printf("Here is format number %d%n", n);   //  -->  "461012"
System.out.printf("%08d%n", n);    //  -->  "00461012"
System.out.printf("%+8d%n", n);    //  -->  " +461012"
System.out.printf("%,8d%n", n);    // -->  " 461,012"
System.out.printf("%+,8d%n%n", n); //  -->  "+461,012"

double pi = Math.PI;

System.out.printf("%f%n", pi);       // -->  "3.141593"
System.out.printf("%.3f%n", pi);     // -->  "3.142"
System.out.printf("%10.3f%n", pi);   // -->  "     3.142"
System.out.printf("%-10.3f%n", pi);  // -->  "3.142"
System.out.printf(Locale.FRANCE,
                  "%-10.4f%n%n", pi); // -->  "3,1416"

Calendar c = Calendar.getInstance();
System.out.printf("%tB %te, %tY%n", c, c, c); // -->  "May 29, 2006"
System.out.printf("%tl:%tM %tp%n", c, c, c);  // -->  "2:34 am"
System.out.printf("%tD%n", c);    // -->  "05/29/06"
```

printf()

```
String[] product = {"pencil", "pen", "ruler", "rubber"};
double[] price = {5.127, 10.3235, 12.1235, 4.12567};
double[] stock = {9, 2003, 150, 12};
for (int index = 0; index < product.length; index++)
{
   System.out.printf("    %-10s%-10.2f%10.2f%n", product[index], price[index],
  stock[index]);
}
```

- Output

```
    pencil    5.13          9.00
    pen       10.32      2003.00
    ruler     12.12       150.00
    rubber    4.13          12.00
```

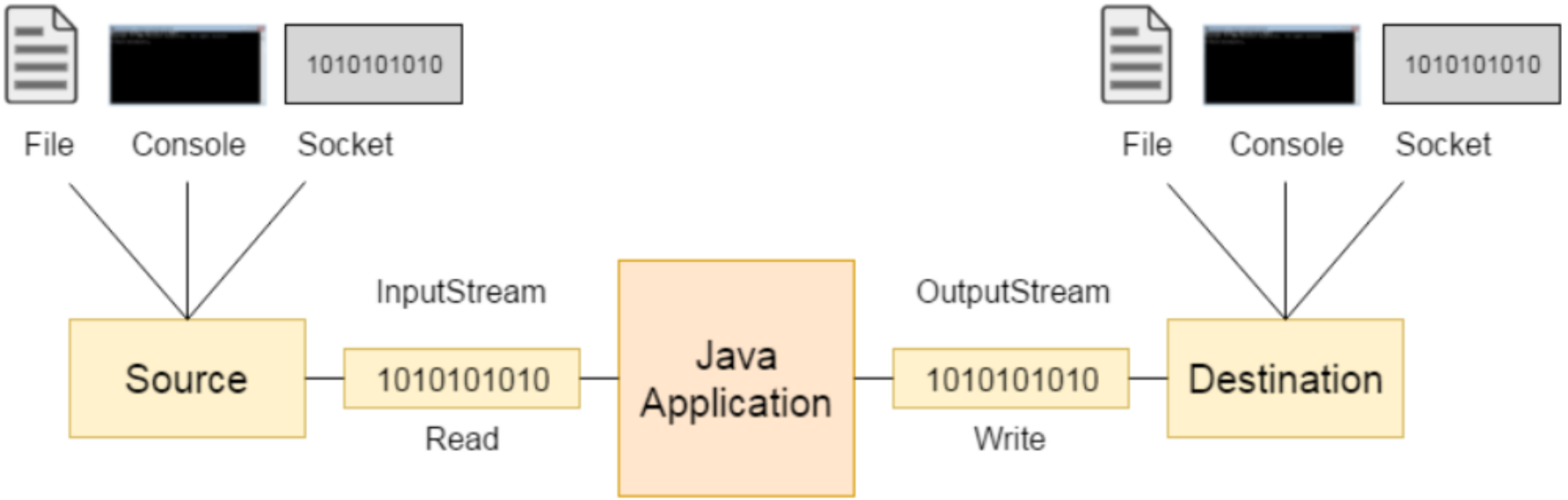
FILE I/O

Input and Output

- The java.io package contains classes you might ever need to perform input and output.
- Java uses concept of a stream to make I/O operation.
- Stream is a sequence of data.
- In Java, 3 streams are created automatically.
  - `System.out:` standard output stream
  - `System.in:` standard input stream
  - `System.err:` standard error stream

Input and Output

- Java uses an output stream to write data to a destination
- Java uses an input stream to read data from a source



File I/O

- The File class is intended to provide an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion.

- The filename is a string. The File class is a wrapper class for the file name and its directory path.

- A File object encapsulates the properties of a file or a path, but does not contain the methods for reading/writing data from/to a file. In order to perform I/O, you need to create objects using appropriate Java I/O classes. The objects contain the methods for reading/writing data from/to a file. This section introduces how to read/write strings and numeric values from/to a text file using the Scanner and PrintWriter classes.

File I/O

- Example of how to read a file.

```
import java.io.File;
import java.util.Scanner;

public class ReadFile {
    public static void main(String[] args) throws Exception {

        // Create a File instance
        File file = new File("scores.txt");
        // Create a Scanner for the file
        Scanner input = new Scanner(file);

        // Read data from a file
        while (input.hasNext()) {
            String firstName = input.next();
            String lastName = input.next();
            int score = input.nextInt();
            System.out.println(firstName + " " + lastName + " " + score);
        }
        input.close(); // Close the file
    }
}
```

File I/O

- Example of how to write a file.

```
import java.io.File;
import java.io.PrintWriter;

public class WriteFile {
    public static void main(String[] args) throws Exception {

            File file = new File("scores_write.txt");
            if (file.exists()) {
                System.out.println("File already exists");
                System.exit(0);
            }
            PrintWriter output = new PrintWriter(file); // Create a file

            // Write formatted output to the file
            output.println("Hello, My name is John.");
            output.print("I'm 30 years old."+" ");
            output.print("I'm Computer Scientist.");
            output.close(); // Close the file
    }
}
```

ARRAYS

Arrays

- Arrays class various methods for manipulating arrays (such as sorting and searching).
- It is in the java.util package.
- Methods that used for manipulating arrays are defined as static.

Arrays

- Some common methods
  - public static void sort(int[] a) // sort an array,
  - public static String toString(int[] a) // print the array
  - public static vod fill(int[] a, int val) // assign specific int value to each element of the array
  - public static fill(int[] a, int from Index, int toIndex, int val) // assign specific int value to each element of the specific range of the array
  - public static boolean equal(int[] a, int a2) // return true if the two arrays are equal to one another

- These methods can also be used for an array of char short byte long float

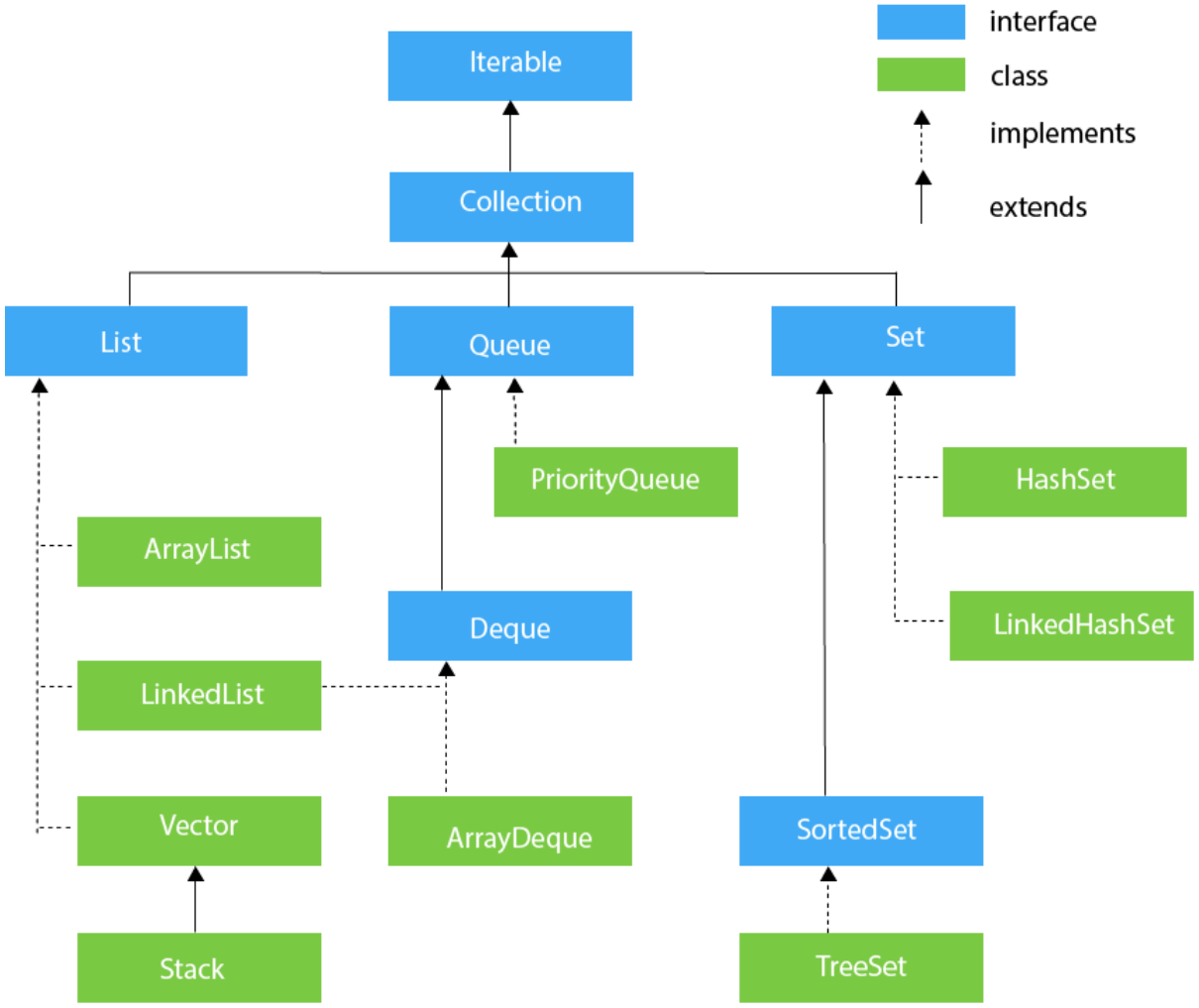
COLLECTION

Collection

- Java provide collection classes for storing data with several purposes.

- Features of collection
  - It increases its capacity as necessary
  - It keeps a private count (size() accessor).
  - It keeps the object in order.
  - Details of how all this is done are hidden.

Java Collection Framework



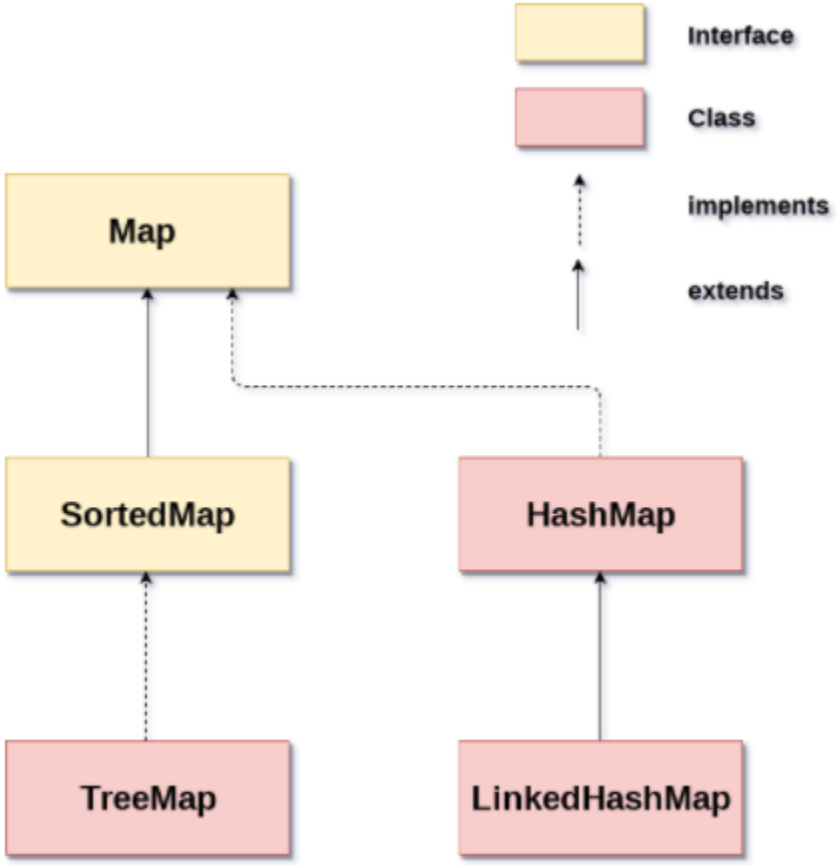
Interface Collection

- add(o) Add a new element
- clear() Remove all elements
- contains(o) Membership checking.
- isEmpty() Whether it is empty
- iterator() Return an iterator
- remove(o) Remove an element
- size() The number of elements

Interface List

- add(i,o) Insert o at position i
- add(o) Append oto the end
- get(i) Return the i-th element
- remove(i) Remove the i-th element
- remove(o) Remove the element o
- set(i,o) Replace the i-th element with o

Java Map Framework



Interface Map

clear()	Remove all mappings
containsKey(k)	Whether contains a mapping for k
containsValue(v)	Whether contains a mapping to v
EntrySet()	Set of key-value pairs
get(k)	The value associated with k
isEmpty()	Whether it is empty
keySet()	Set of keys
put(k,v)	Associate vwith k
remove(k)	Remove the mapping for k
size()	The number of pairs
values()	The collection of values

Concrete Collections

Concrete collection	implements	description
HashSet	Set	hash table
TreeSet	SortedSet	balanced binary tree
ArrayList	List	resizable-array
LinkedList	List	linked list
Vector	List	resizable-array
HashMap	Map	hash table
TreeMap	SortedMap	balanced binary tree
Hashtable	Map	hash table

Concrete Collections

- The collections are realized by several collection classes.

- Some well-known classes are:
  - ArrayList
  - LinkedList
  - HashSet
  - HashMap
  - Etc.

ArrayList

- It is like an array, but there is *no size limit* (dynamic arrays). We can add or remove elements anytime.
- It implements the List interface so we can use all the methods of List interface
- It maintains insertion order
- Whenever we remove an element, internally, the array is traversed and the memory bits are shifted.

ArrayList – Some Useful Methods

`public int size()`	Return number of elements
`public boolean isEmpty()`	Return whether it is empty
`public boolean contains(Object o)`	Check whether o is in the array list
`public void clear()`	Remove all elements
`public boolean add(E o)`	Add a new element to the end
`public void add(int i, E o)`	Insert o at position i
`public E get(int i)`	Return the i-th element
`public E remove(int i)`	Remove the i-th element
`public boolean remove(Object o)`	Remove the element o
`public E set(int i, E o)`	Replace the i-th element with o

*noted that `E` is Generics (depends on input type)

ArrayList

```
String mango = new String("Mango");
String apple = new String("Apple");
String banana = new String("Banana");
String grapes = new String("Grapes");

//Creating arraylist
ArrayList<String> list=new ArrayList<String>();

list.add(mango);//Adding object in arraylist
list.add(apple);
list.add(banana);
list.add(grapes);

//Printing the arraylist info
System.out.println(list);
```

ArrayList Cont.

```
System.out.println(list); //[Mango, Apple, Banana, Grapes]
list.remove(2);
System.out.println(list); //[Mango, Apple, Grapes]
list.remove(apple);
System.out.println(list); //[Mango, Grapes]
list.set(0, apple);
System.out.println(list); //[Apple, Grapes]
list.add(1, banana);
System.out.println(list); //[Apple, Banana, Grapes]
System.out.println(list.contains(grapes)); //true
System.out.println(list.contains(mango)); //false
System.out.println(list.size()); //3
```

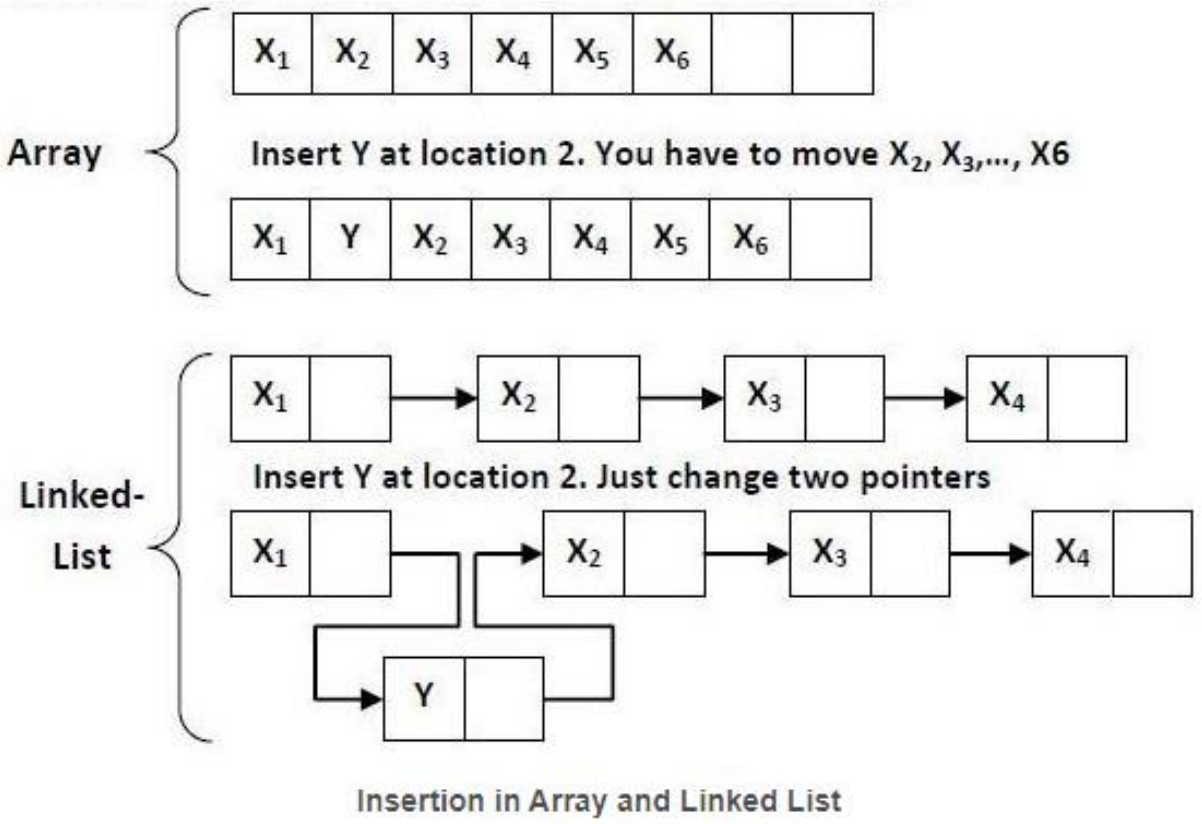
LinkedList

- It uses a doubly linked list to store the elements. It provides a linked-list data structure.

fig- doubly linked list

- It implements the List and Deque interface
- It maintains insertion order
- There is no concept of shifting the memory bits. The list is traversed and the reference link is changed.
- Java LinkedList class can be used as a list, stack or queue.

LinkedList vs ArrayList



Insertion in Array and Linked List

Credit: https://www.interviewbit.com/tutorial/arrays-vs-linked-lists/

LinkedList

```
String mango = new String("Mango");
String apple = new String("Apple");
String banana = new String("Banana");
String grapes = new String("Grapes");
LinkedList<String> linkedlist =new LinkedList<String>();
linkedlist.add(mango);
linkedlist.add(apple);
linkedlist.add(banana);
linkedlist.add(grapes);
System.out.println(linkedlist); //[Mango, Apple, Banana, Grapes]
linkedlist.remove(2);
System.out.println(linkedlist); //[Mango, Apple, Grapes]
linkedlist.remove(apple);
System.out.println(linkedlist); //[Mango, Grapes]
linkedlist.set(0, apple);
System.out.println(linkedlist); //[Apple, Grapes]
linkedlist.add(1, banana);
System.out.println(linkedlist); //[Apple, Banana, Grapes]
System.out.println(linkedlist.contains(grapes)); //true
System.out.println(linkedlist.contains(mango)); //false
System.out.println(linkedlist.size()); //3
```

LinkedList – Some useful methods

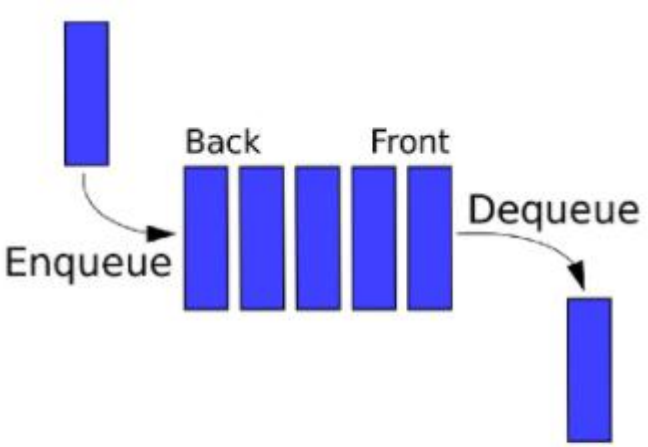
- Most methods are similar to ArrayList since they both implement List interface

- However, LinkedList also implements Deque interface (which is Queue + double ended features) therefore it can be used for many types of data structures.

LinkedList – Some useful methods

- Queue

Queue Method	Equivalent Deque Method
add(e)	addLast(e)
offer(e)	offerLast(e)
remove()	removeFirst()
poll()	pollFirst()
element()	getFirst()
peek()	peekFirst()



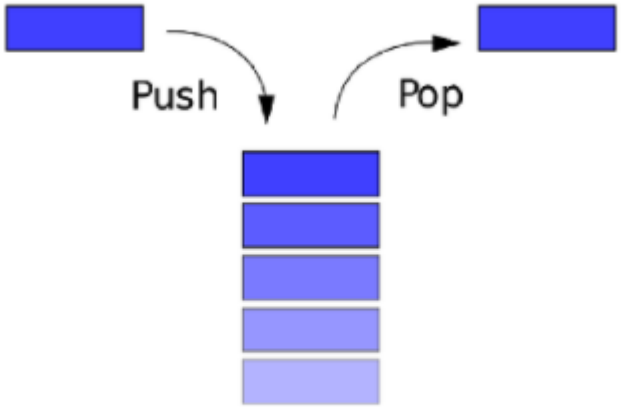
Queue (FIFO) first in first out

Credit: https://dunglai.github.io/2018/07/26/linked-list/

LinkedList – Some useful methods

- Stack

Stack Method	Equivalent Deque Method
`push(e)`	`addFirst(e)`
`pop()`	`removeFirst()`
`peek()`	`peekFirst()`



Stack (LIFO) last in first out

Credit: https://dunglai.github.io/2018/07/26/linked-list/

HashSet

- It implements Set which is used to create the mathematical set. It is an unordered collection which duplicates are not allowed.
- HashSet create a collection that uses a hash table for storage. It stores the elements by using a mechanism called hashing.
- It contains unique element only
- It doesn’t maintain the insertion order, therefore we cannot manipulate elements by using specific index

HashSet

```
String mango = new String("Mango");
String apple = new String("Apple");
String banana = new String("Banana");
String grapes = new String("Grapes");
HashSet<String> set =new HashSet<String>();
set.add(mango);
set.add(apple);
set.add(banana);
set.add(grapes);
System.out.println(set); //[Apple, Grapes, Mango, Banana]
set.remove(apple);
System.out.println(set); //[Grapes, Mango, Banana]
System.out.println(set.contains(grapes)); //true
System.out.println(set.contains(apple)); //false
System.out.println(set.size()); //3
```

HashSet Cont.

```
String kiwi = new String("Kiwi");
LinkedList<String> list = new LinkedList<String>();
list.add(grapes);
list.add(apple);
list.add(kiwi);

set.addAll(list);
System.out.println(set); //[Apple, Kiwi, Grapes, Mango, Banana]
```

HashMap

- HashMap implements the Map interface which allows us *to store key and value pair*, where keys should be unique.

- It contains values based on key and contains only unique keys

- It doesn’t maintain order

HashMap

```
String mango = new String("Mango");
String apple = new String("Apple");
String banana = new String("Banana");
String grapes = new String("Grapes");

//Creating HashMap
HashMap<Integer,String> map = new HashMap<Integer,String>();
map.put(1, mango);
map.put(2, apple);
map.put(3, banana);
map.put(4, grapes);
System.out.println(map); //{1=Mango, 2=Apple, 3=Banana, 4=Grapes}
```

HashMap

```
map.put(1, grapes);
System.out.println(map); //{1=Grapes, 2=Apple, 3=Banana, 4=Grapes}
System.out.println(map.get(2)); //Apple
System.out.println(map.containsValue(banana)); //true
System.out.println(map.containsValue(mango)); //false
System.out.println(map.containsKey(1)); //true
System.out.println(map.containsKey(7)); //false
System.out.println(map.size()); //4
map.replace(1, grapes, mango);
System.out.println(map); //{1=Mango, 2=Apple, 3=Banana, 4=Grapes}
map.remove(1);
System.out.println(map); //{2=Apple, 3=Banana, 4=Grapes}
map.remove(2, grapes);
System.out.println(map); //{2=Apple, 3=Banana, 4=Grapes}
map.replace(3, mango);
System.out.println(map); //{2=Apple, 3=Mango, 4=Grapes}
```

Iterator

- **Java Iterator** is practiced in order to iterate over a collection of Java object components entirety one by one.

- We can apply these iterators to any of the classes of the Collection framework

- In Java Iterator, we can use both of the read and remove operations. Just like for-each loop, but it can remove elements

Iterator

```
ArrayList<String> list = new ArrayList<String>();
String mango = new String("Mango");
String apple = new String("Apple");
String banana = new String("Banana");
String grapes = new String("Grapes");
list.add(mango);//Adding object in arraylist
list.add(apple);
list.add(banana);
list.add(grapes);
System.out.println(list); //[Mango, Apple, Banana, Grapes]

Iterator<String> iterator = list.iterator();
while (iterator.hasNext()){
        //print Mango Apple Banana Grapes line by line
        System.out.println(iterator.next());
        iterator.remove();
}
System.out.println(list); //[]
```