Noah Parker

Chapter 16 Study Sheet

Implementing a Linked List Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at the contiguous location, the elements are linked using pointers.

|  |
| --- |
| class LinkedList {  Node head; // head of list  /\* Linked list Node\*/  class Node {  int data;  Node next;  // Constructor to create a new node  // Next is by default initialized  // as null  Node(int d) { data = d; }  }} |

The node class:, is a class which is used to create the individual data holding blocks for various data structure, which organise data in a non-sequential fashion.

|  |
| --- |
| public void addFirst(Object element)  {  Node newNode = new Node();  newNode.data = element;  newNode.next = first;  first = newNode;  } |

Java ArrayList Implementation The Java Collection framework has a class called ArrayList. The objects are stored in a dynamic array. It's similar to Array, however it doesn't have a size restriction. We have complete control over the elements and can add or remove them at any time. The ArrayList can be used to store the duplicate element; it maintains the insertion order internally.

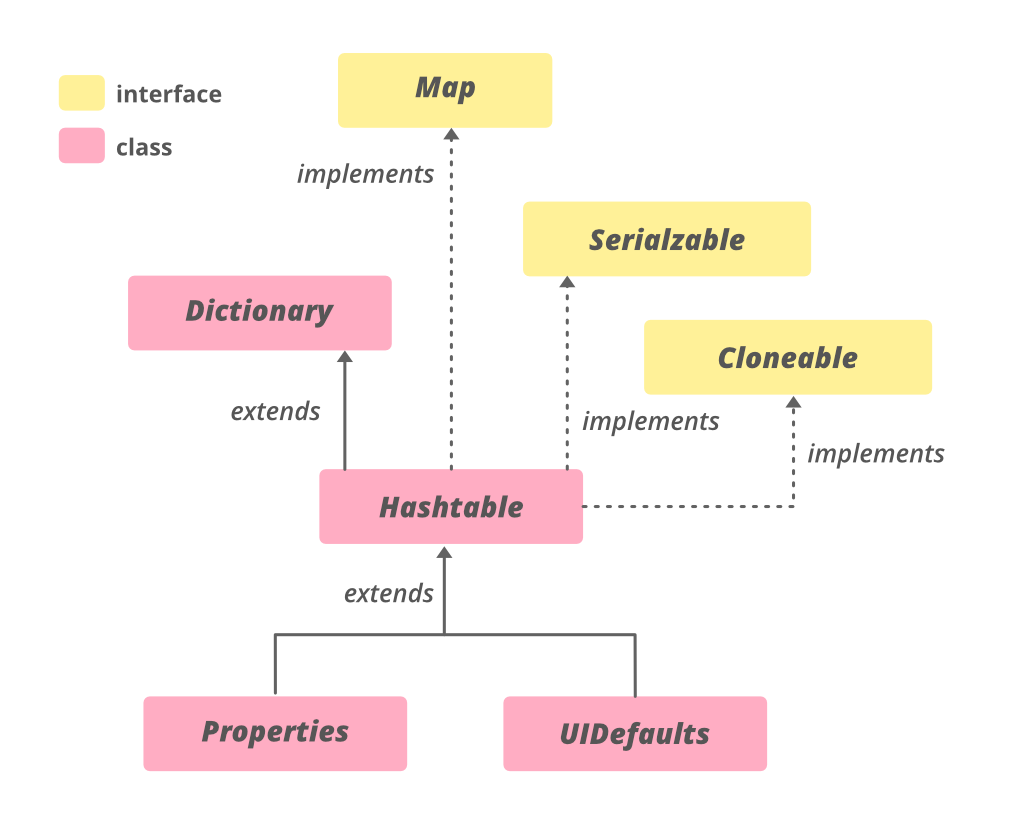
|  |
| --- |
| public class ALExample {  public static void main(String[] args) {  List<String> l = new ArrayList<>(); //List Implementation  l.add("Sam"); //adding objects to list  l.add("Sandy");  l.add("Joe");  l.add("Arya");  l.add("Nik");  System.out.println("List objects are: " +l); // printing the list  l.remove("Nik"); //removing objects from list  System.out.println("After Removing Nik, List Objects are" +l);  } } |

This table above creates a new array list that we store names in

A queue can be implemented using two stacks. Let queue to be implemented be q and stacks used to implement q be stack1 and stack2. q can be implemented in two ways:

|  |
| --- |
| static class Queue{  static Stack<Integer> s1 = new Stack<Integer>();  static Stack<Integer> s2 = new Stack<Integer>();  static void enQueue(int x) {  // Move all elements from s1 to s2  while (!s1.isEmpty()){  s2.push(s1.pop());  //s1.pop();}  // Push item into s1  s1.push(x);  // Push everything back to s1  while (!s2.isEmpty()){  s1.push(s2.pop());  //s2.pop(); } }  // Dequeue an item from the queue  static int deQueue() {  // if first stack is empty  if (s1.isEmpty()) {  System.out.println("Q is Empty");  System.exit(0);}  // Return top of s1  int x = s1.peek();  s1.pop();  return x;}}; |

Hash table is implemention using the Hashtable class, which maps keys to values. As a key or a value, any non-null object can be used. The objects used as keys must implement the hashCode and equals methods in order to successfully store and retrieve objects from a hashtable.

public class Hashtable<K,V> extends Dictionary<K,V> implements Map<K,V>, Cloneable, Serializable The table below is how a hash table is laid out