1. You could create the new subclass object but use super class to represent it however, you can not do it reversely.
2The idea of an ADT is to separate the **notions of specification** (what kind of thing we're working with and what operations can be performed on it) and **implementation** (how the thing and its operations are actually implemented). (1)Code is easier to understand (e.g., it is easier to see "high-level" steps being performed, not obscured by low-level code).(2)Implementations of ADTs can be changed (e.g., for efficiency) without requiring changes to the program that uses the ADTs.(3)ADTs can be reused in future programs.each ADT corresponds to a class (or Java interface - more on this later) and the operations on the ADT are the class/interface's public methods. The user, or client, of the ADT only needs to know about the method interfaces(the names of the methods, the types of the parameters, what the methods do, and what, if any, values they return), not the actual implementation (how the methods are implemented, the private data members, private methods, etc.).
3. an Iterator, which is an interface defined in java.util. Every Java class that implements the Iterable interface (which includes classes that implement the subinterface Collection) provides an iterator method that returns an Iterator for that collection **import java.util.*; // or import java.util.Iterator**;
4. Every exception is either a **checked exception** or an **unchecked exception**. If a method includes code that could cause a checked exception to be thrown, then: ● the exception must be declared in the method header using a throws clause, or● the code that might cause the exception to be thrown must be inside a try block with a catchclause for that exception.in general, you must always include some code that acknowledges the possibility of a checked exception being thrown. If you don't, you will get an error when you try to compile your code. (The compiler does not check to see if you have included code that acknowledges the possibility of an unchecked exception being thrown.)The answer lies in the exception hierarchy: ● **If an exception is a subclass of RuntimeException, then it is unchecked.● If an exception is a subclass of Exception but not a subclass of RuntimeException, then it ischecked.**
5. A function T(N) is O(F(N)) if for some constant c and for all values of N greater than some value n0:T(N) <= c * F(N)The idea is that T(N) is the exact complexity of a method or algorithm as a function of the problem size N and that F(N) is an upper-bound on that complexity (i.e., the actual time/space or whatever for a problem of size N will be no worse than F(N)). In practice, we want the smallest F(N). **-- the least upper bound on the actual complexity**.
6The only item that can be taken out (or even seen) is the most recently added (or top) item; a Stack is a Last-In-First-Out (LIFO) abstract data type.
7. A Queueisa First-In-First-Out(FIFO)abstractdatatype.Itemscanonlybeaddedatthere arofthe queue and the only item that can be removed is the one at the front of the queue.
8 **E[] tmp = (E[])(new Object[items.length*2]);**
9 warp around
private int incrementIndex(int index) {
if (index == items.length-1) return 0;
Else return index + 1; }
10. *** **RECURSION RULE #1** ***Every recursive method must have a base case -- a condition under which no recursive call is made -- to prevent infinite recursion.*** **RECURSION RULE #2** ***Every recursive method must make progress toward the base case to prevent infinite recursion.
11 non-linear structure: (1)More than one item can follow another. (2) The number of items that follow can vary from one item to another.● each letter represents one node● the arrows from one node to another are called edges● the topmost node (with no incoming edges) is the root (node A)● the bottom nodes (with no outgoing edges) are the leaves (nodes D, I, G & J)A path in a tree is a sequence of (zero or more) connected nodes;The length of a path is the number of nodes in the pathThe height of a tree is the length of the longest path from the root to a leafThe depth of a node is the length of the path from the root to that nodeNode A is called the parent, and node B is called the child.A subtree of a given node includes one of its children and all of that child's descendants. The descendants of a node N are all nodes reachable from N (N's children, its children's children, etc.).In a binary tree: (1) Each node has 0, 1, or 2 children. (2)Each child is either a left child or a right child.

	Array List	Linked List
Constructor	O(1)	O(1)
Add(E)	O(N)	O(1)
Add(int,E)	O(N)	O(N)
Contains(int)	O(N)	O(N)
Size	O(1)	O(1)
isEmpty	O(1)	O(1)
Get(int)	O(1)	O(N)
Remove(int)	O(N)	O(N)

13 Array Implemented Stack

Operation	Worst-case Time	Average-case Time
constructor	O(1)	O(1)
isEmpty	O(1)	O(1)
push	O(N)	O(1)
pop	O(1)	O(1)
peek	O(1)	O(1)

Linked-list implemented stack, constructor O(1) isEmpty O(1) push O(1) pop O(1) peek O(1)
14 For the **array implementation**, the worst-case times for the **push and enqueue** methods O(N) for the naive implementation, for a stack/queue with N items (to allocate a new array and copy the values); using the "shadow array" trick, those two operations are O(1). **For the linked-list implementation, push and enqueue are always O(1)**.
15 The compareTo method returns a negative value to mean "less than", it returns zero to mean "equal to", and it returns a positive value to mean "greater than". To make your class implement the Comparable interface you must: (1) include implements Comparable<C> after the class name in the file that defines the class, and (2) define a compareTo method with the following signature: int compareTo(C other)
16. Level Order Trees

```
Q.enqueue(root) while (!Q.empty()) { Treenode<T> n = Q.dequeue(); System.out.print(n.getData()); ListADT<Treenode<T>> kids = n.getChildren(); Iterator<Treenode<T>> it = kids.iterator(); while (it.hasNext()) { Q.enqueue(it.next()); } }
```

17 BST : In a BST, each node stores some information including a unique key value and perhaps some associated data. A binary tree is a BST iff, for every node n, in the tree: (a) All keys in n's left subtree are less than the key in n, and (b) all keys in n's right subtree are greater than key in n.
18 the worst-case time required to do a lookup in a BST is **O(height of tree).** In the worst case (a "linear" tree) this is O(N), where N is the number of nodes in the tree. In the best case (a "full" tree) ~ O(log N).
19 delete BST : There are two possibilities that work: the largest value in n's left subtree or the smallest value in n's right subtree. We'll arbitrarily decide to use the smallest value in the right subtree.

```
private BSTnode<K> delete(BSTnode<K> n, K key) {    if (n == null) {        return null;    }        if (key.equals(n.getKey())) {        // n is the node to be removed        if (n.getLeft() == null && n.getRight() == null) {            return null;        }        if (n.getLeft() == null) {            return n.getRight();        }        if (n.getRight() == null) {            return n.getLeft();        }        // if we get here, then n has 2 children        K smallVal = smallest(n.getRight());        n.setKey(smallVal); n.setRight( delete(n.getRight(), smallVal) ); return n;     }     else if (key.compareTo(n.getKey()) < 0) {        n.setLeft( delete(n.getLeft(), key) ); return n;     }     else {        n.setRight( delete(n.getRight(), key) ); return n;     }}
```

19. Set<String> dictionary = new TreeSet<String>();Set<String> misspelled = new TreeSet<String>();
private Map<Integer, Employee> staff = new TreeMap<Integer, Employee>();
20 if the tree is reasonably balanced (shaped more like a "full" tree than like a "linear" tree), the insert, lookup, and delete operations can all be implemented to run in O(log N) time, where N is the number of stored items. For a linked list, although insert can be implemented to run in O(1) time, lookup and delete take O(N) time in the worst case. It is important to remember that for a "linear" tree (one in which every node has one child), the worst-case times for insert, lookup, and delete will be O(N).
21 A red-black tree is a binary search tree in which each node has a color (red or black) associated with it ()the following 3 properties hold:
(root property) The root of the RB tree is black
(red property) The children of a red node are black.
(black property) For each node with at least one null child, the number of black nodes on the path from the root to the null child is the same.
22 Red-Black Tree Insert (a) looking : O(lg n); (b) color the new node red. This step is O(1) ;(C) Restructuring is O(1) since it involves changing at most five pointers to tree nodes; (d) Changing the colors of nodes during recoloring is O(1). However, we might then need to handle a double-red situation further up the path from the added node to the root. In the worst-case, we end up fixing a double-red situation along the entire path from the added node to the root. So, in the worst-case, the recoloring that is done during insert is O(log N) (= time for one recoloring * max number of recolorings done = O(log N)).=> Overall O(log N)
Balanced search trees have a height that is always O(log N). One consequence of this is that lookup, insert, and delete on a balanced search tree can be done in O(log N) worst-case time.
23. A heap is a binary tree (in which each node contains a Comparable key value), with two special properties:The **ORDER property**:For every node N, the value in N is greater than or equal to the values in its children (and thus is also greater than or equal to all of the values in its subtrees).**The SHAPE property:**All leaves are either at depth d or d-1 (for some value d).All of the leaves at depth d-1 are to the right of the leaves at depth d.(a) There is at most 1 node with just 1 child. (b) That child is the left child of its parent, and (c) it is the rightmost leaf at depth d

25. Heap Complexity For the insert operation, we start by adding a value to the end of the array (constant time, assuming the array doesn't have to be expanded); then we swap values up the tree until the order property has been restored. In the worst case, we follow a path all the way from a leaf to the root (i.e., the work we do is proportional to the height of the tree). Because a heap is a balanced binary tree, the height of the tree is O(log N), where N is the number of values stored in the tree.The removeMax operation is similar: in the worst case, we follow a path down the tree from the root to a leaf. Again, the worst-case time is O(log N).
26. The array is called the **hashtable**.We will refer to the size of the array as **TABLE_SIZE**.The function that maps a key value to the array index in which that key (and its associated data) will be stored is called the **hash function**.
27. Hash Complexity **: lookup** : In the worst case, all of the keys will hash to the same place. In that case, if linked lists are used, the worst-case time for lookup will be O(N), where N is the number of values stored in the hashtable. If a balanced search tree is used, the time will be O(log N). **Insert** The time for insert is similar to the time for lookup: the sum of the time for the hash function and the time to insert k into array[v]. However, if linked lists are used, the time to insert k into the array will always be O(1) rather than O(N) in the worst case. **Delete** he worst-case time is the same as for lookup, since the value has to be found before it can be deleted.
28. Summary Given a fixed table size and a hash function that distributes keys evenly in the range 0 to TABLE_SIZE-1, the expected times for insert, lookup, and delete are all O(1), as long as the number of keys in the table is less than TABLE_SIZE.**Using balanced trees** as array elements, the worst-case times for insert, lookup, and delete are all O(log N), where N is the number of keys stored in the table.**Using linked lists as array** elements, the worst-case times are O(1) for insert and O(N) for lookup and delete.A disadvantage of hashtables (compared to binary-search trees and red-black trees) is that it is not easy to implement a print method that prints all values in sorted order.
29. Sorting Summary
Selection Sort: N passes on pass k: find the k-th smallest item, put it in its final place **always O(N^2)**
Insertion Sort:N passes on pass k: insert the k-th item into its proper position relative to the items to its left **worst-case O(N^2)given an already-sorted array: O(N)**
Merge Sort: recursively sort the first N/2 items recursively sort the last N/2 items merge (using an auxiliary array) **always O(N log N)**
Quick Sort: choose a pivot value partition the array:left part has items <= pivot right part has items >= pivot recursively sort the left part recursively sort the right part **worst-case O(N^2) expected O(N log N)**
Heap Sort:use heapify to convert the unsorted array into a heap, then do N removeMax operations. Each operation frees one more space at the end of the array; put the returned max value into that space.**always O(N log N)**
Radix Sort:make len passes through the N sequences to be sorted, right-to-left on each pass, put the values into the queue in position p of the auxiliary array, where p is the value of the current "digit"then put the values back from the auxiliary array into the original arrayno comparisons of values are done (i.e., radix sort is not a comparison sort).**always O(N + range) * len)**
30. Stable Sorting Algorithms:Insertion Sort; Merge Sort Bubble Sort Tim Sort; Counting Sort
Unstable Sorting Algorithms:Heap Sort; Selection sort; Shell sort; Quick Sort
31. Graphs are a generalization of trees. graphs have nodes and edges. (The nodes are sometimes called vertices and the edges are sometimes called arcs.) However, graphs are more general than trees: in a graph, a node can have any number of incoming edges (in a tree, the root node cannot have any incoming edges and the other nodes can only have one incoming edge). Every tree is a graph, but not every graph is a tree. [2] -> [1], [1] -- [3]
In the directed graph, there is an edge from node 2 to node 1; therefore:The two nodes are adjacent (they are neighbors).Node 2 is a predecessor of node 1.Node 1 is a successor of node 2.The source of the edge is node 2 and the target of the edge is node 1. **In the undirected graph**, there is an edge between node 1 and node 3; therefore: Nodes 1 and 3 are adjacent (they are neighbors). 32. The first two paths are **acyclic paths: no node is repeated**; the last path is **a cyclic path** because node 1 occurs twice. an edge can connect a node to itself.
32. **An undirected graph is connected** if there is a path from every node to every other node**. A directed graph is strongly connected** if there is a path from every node to every other node. A **directed graph is weakly connected** if, treating all edges as being undirected, there is a path from every node to every other node.

1.Depth first search

```
public static void dfs (Graphnode<T> n)
{     n.setVisited(true);
for (Graphnode<T> m : n.getSuccessors()) {
 if (! m.getVisited())
 {            dfs(m);         }     }}
```

Usage : Cycle Detection For node

```
public boolean hasCycle(Graphnode<T> n) {
n.setMark(IN_PROGRESS);
 for (Graphnode<T> m : n.getSuccessors()) {
 if (m.getMark() == IN_PROGRESS) { return true; }
 if (m.getMark() != DONE) { if (hasCycle(m)) { return true; } }
}
n.setMark(DONE); return false; }
```

For Graph.

```
public boolean graphHasCycle() {
mark all nodes unvisited for each node k in the graph
{ if (node k is marked unvisited) { if (hasCycle(k)) { return
true; } } } return false; }
```

Topological Numbering

```
public int topNum(Graphnode<T> n, int num) throws
CycleException {
n.setMark(IN_PROGRESS);
 for (Graphnode<T> m : n.getSuccessors()) {
 if (m.getMark() == IN_PROGRESS) {
// no topological ordering for a cyclic graph!
 throw new CycleException(); }
 if (m.getMark() != DONE) { num = topNum(m, num); } } //
here when n has no more successors n.setMark(DONE);
n.setNumber(num); return num - 1; }
```

2. Breadth first search

Breadth-first search uses a queue rather than recursion (which actually uses a stack); the queue holds "nodes to be visited". If the graph is a tree, breadth-first search gives you a level-order traversal. Here's the pseudo code:

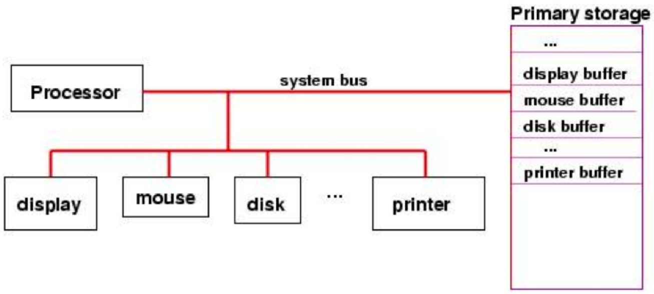
```
public void bfs(Graphnode<T> n) { Queue<Graphnode>
queue = new Queue<Graphnode>(); n.setVisited(true);
queue.enqueue( n );
while (!queue.isEmpty()) {
Graphnode<T> current = queue.dequeue();
for (Graphnode<T> k : current.getSuccessors())
{ if (! k.getVisited()){ k.setVisited(true); queue.enqueue(k);
} // end if k not visited } // end for every successor k } //
end while queue not empty }
```

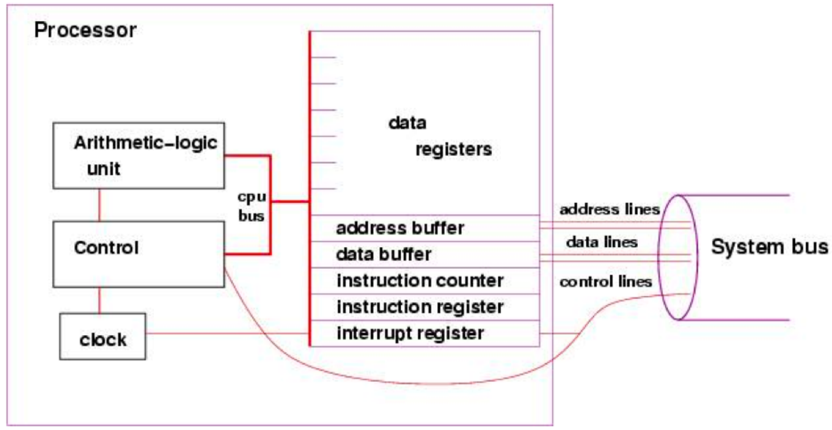
3. **Dijkstra's algorithm** : A clever algorithm that can be used to solve this problem (to find shortest paths in **a weighted graph with non-negative edge weights**) has been defined by Edsgar Dijkstra (and so is called "Dijkstra's algorithm"). **The worst-case running time of the algorithm is O(E log N), where E is the number of edges and N is the number of nodes.**

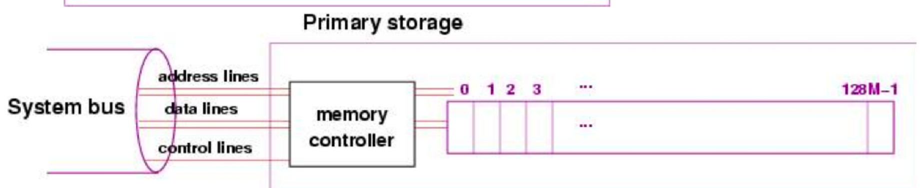
4. The format of each instruction is:

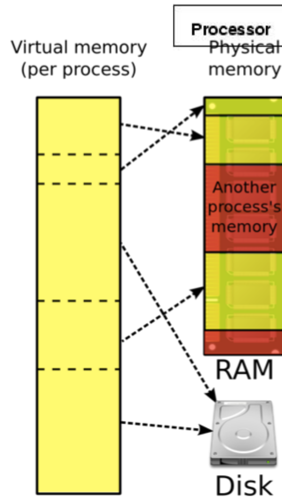
IIII RRRR DDDD DDDD, where IIII is the coding that states the operation required, RRRR is the coding of which data register to use, and DDDD DDDD is the data, which is either a storage address or another register number.

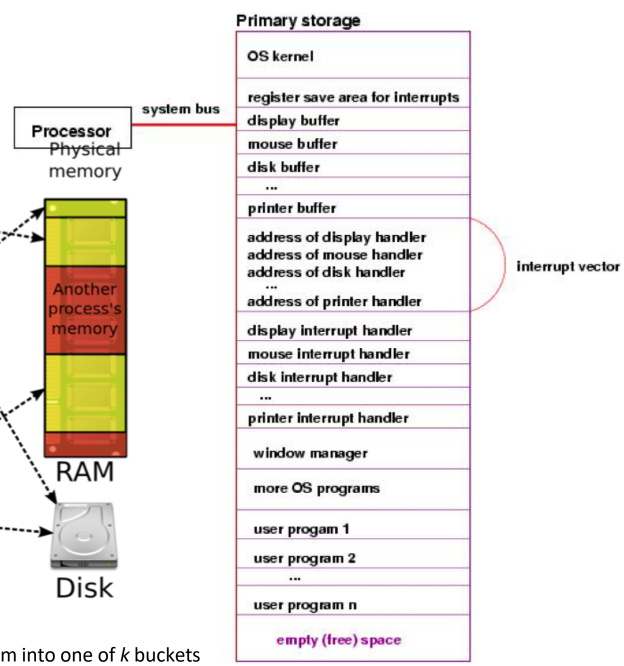
5. The instructor cycle are the actions taken by the processor to execute one instruction. Each time the processor's clock pulses (ticks) the control unit does these steps: (1)uses the number in the instruction counter to fetch an instruction from primary storage and copy it into the instruction register (2)reads the pattern of bits in the instruction register and decodes the instruction (3) based on the decoding, tells the ALU to execute the instruction, which means that the ALU manipulates the registers accordingly.(4) There is a fourth step in the instruction cycle, an interrupt check, After the execution step, the control unit examines the contents of the interrupt register, checking to see if any bit in the register is set to 1. If all bits are 0, then no device has completed an action, so the processor can start a new instruction. But if a bit is set to 1, then there is an interrupt --- the processor must pause its execution and do whatever instructions are needed:For example, perhaps the user has pressed the mouse button. The device controller for the mouse sends a signal on the system bus to set to 1 the bit for a ``mouse interrupt'' in the interrupt register. When the control unit examines the interrupt register at the end of its current execution cycle, it sees that the bit for the mouse is set to 1. it resets the bit to 0 and resets the instruction counter to the address of the program that must be executed whenever the mouse button is pressed. Once the mouse-button program finishes the processor can resume the work it was doing. The mouse-button program is called an interrupt handler.











Bucket Sort:

- Map each item into one of *k* buckets
- no comparisons of values are done (not a comparison sort).
- always O(N + *k*)

Virtual Memory guarantees non-interference

The addresses used by a program are virtual. This means an address used by a program isn't the address actually accessed. When executing, a program is given two special registers, called **base and bound**. The base register stores an the first memory address assigned to a program. When a program reads or writes address v it actually accesses address base + v. Similarly, the bound register represents the largest address a program may access. This guarantees memory addresses beyond a program's allocation are untouched. With virtual memory no program can touch another program's memory allocation, guaranteeing peaceful co-existence.

Paging : a very clever extension of virtual memory, Memory is divided into a large number of fixed size units called *pages*. Page size is commonly in the range 4k to 32k bytes. If a program needs 100k bytes, it is allocated 25 pages. But, each page is mapped into processor memory *independently*. And some are *not mapped* at all! Thus the first page allocated may be at location 100,000. The second may be at location 40,000. The third may not have any memory allocation (yet). If a program grows too large, parts of it (in terms of pages) may be removed from memory and copied to the hard drive. This means 2 programs, each of size 1 megabyte, can co-exist in a single 1 megabyte block of processor memory!