The following table shows the access to members permitted by each modifier.

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Example 2: Use a linked list parameter

```
printList( Listnode<E> l ) {
   if (l == null) {
      return;
   }
   System.out.println(l.getData());
   printList(l.getNext());
}
```

Example 1: Use a String parameter

```
printStr( String S ) {
   if (S.equals("")) {
      return;
   }
   System.out.print( S.substring(0, 1) + " " );
   printStr( S.substring(1) );
}
```

```
   public int length() {
         if (getNext() == null) return 1;
         else return  1 + getNext().length();
           } ... }

public String toString(){
         if (next == null) return data.toString();
         else return data.toString() + ",” + next.toString();
}
```

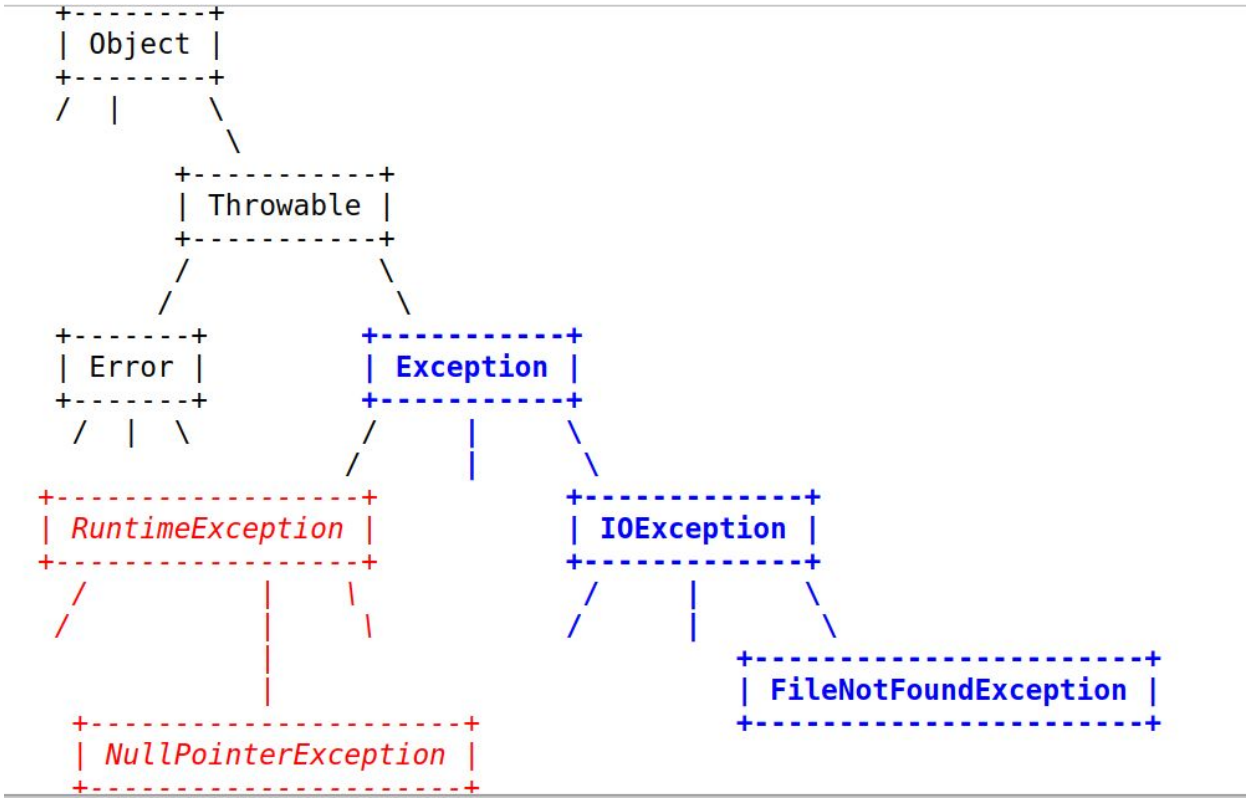
```
public static void printInt(int k) {
  if (k == 0) {
     return;
```

```
  }
  printInt(k/2);
  for (int j = 0; j < k; j++) {
     System.out.println(j);
  }
  printInt(k/2);
}
```

```
private int incrementIndex(int index) {
   if (index == items.length-1)
      return 0;
   else
      return index + 1;
}
```

Note that instead of using incrementIndex we could use the mod operator (%), and write: rearIndex = (rearIndex + 1) % items.length. However, the mod operator is quite slow and it is easy to get that expression wrong, so we will use the auxiliary method (with a check for the "wrap-around" case) instead.

E[] tmp = (E[])(new Object[items.length*2]);



	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)

Worst case complexity of ArrayStack

- Push: O(N)
- Pop: O(1)
- Peek: O(1)
- isEmpty: O(1)

Complexity of LLStack

Because we add and remove stack elements at the head of the list, all stack operations are O(1)!

```
public class Listnode<E> {
        //*** fields ***
   private E data;
   private Listnode<E> next;

   public int length() {
        if (getNext() == null) return 1;
        else return  1 + getNext().length();
         } ... }
```

```
public String toString(){
        if (next == null) return data.toString();
        else return data.toString() + ",” + next.toString();
}
```

```
boolean isPalindrome(String s){
    if (s.length() == 0 ||
        s.length() == 1 )
      return true;
    char first =  s.charAt(0);
    char last =
        s.charAt(s.length()-1);
    return (first == last) &&
      isPalindrome(
        s.substring(1, s.length()-1));
  }
```

```
public class Listnode<E> {
  //*** fields ***
    private E data;
    private Listnode<E> next;

    public int length() {
      if (getNext() == null)
         return 1;
       else return
         1 + getNext().length();
     } ... }
```