- JAVA INTERFACES -

AN INTERFACE PROVIDES A WAY TO SPECIFY METHOD SIGNATURES (ALSO CALLED PROTOTYPES) WITHOUT SUPPLYING ANY IMPLEMENTATION DETAILS.

INTERFACES CAN BE USED TO SPECIFY SOME COMMON BEHAVIOR AMONGST SEVERAL DIFFERENT CLASSES. YOU CAN THEN DESIGN A METHOD TO WORK WITH A VARIETY OF OBJECT TYPES THAT EXHIBIT THIS COMMON BEHAVIOR BY SPECIFYING THE INTERFACE AS A PARAMETER TYPE FOR THE METHOD, INSTEAD OF A CLASS. THIS ALLOWS THE METHOD TO UTILIZE THE COMMON BEHAVIOR AS LONG AS ITS ARGUMENTS HAVE IMPLEMENTED THE INTERFACE

EX. Java.util.Collection IS AN INTERFACE WHICH PROVIDES METHOD PROTOTYPES (BUT NO DEFINITIONS) FOR MANAGING COLLECTIONS OF OBJECTS (SUCH AS SETS, LISTS, ETC...)

IT CONTAINS THE FOLLOWING PROTOTYPE

```
Public boolean Contains (Object element );
```

USE THE implements KEYWORD TO CREATE A CLASS THAT USES THE METHODS PROTOTYPES IN Java.util.Collection.

```
import java.util.Collection;
Public class myCollection implements Collection {
    :
    public boolean contains(Object element) {
        :
        // IMPLEMENTATION OF Contains
        :
    }
}
```

NOW ANY METHOD WHICH IS DEFINED TO TAKE A PARAMETER OF TYPE Collection CAN BE PASSED AN ARGUMENT OF TYPE myCollection.

```
Public void myMethod(Collection c) {
        :
}
```

THEN IN main SAY

```
myCollection x = new myCollection(-);
:
myMethod(x);
```

THUS TO SAY THAT A class implements AN INTERFACE IS ALOT LIKE SAYING THAT THE CLASS IS A SUBCLASS OF THE INTERFACE, EXCEPT THAT

- EACH OF THE FUNCTIONS IN THE INTERFACE MUST BE DEFINED IN ANY CLASS WHICH IMPLEMENTS it, WHEREAS A SUBCLASS OF A CLASS CAN CHOOSE WHETHER OR NOT TO OVERRIDE A method IN ITS Super class.

- AN interface is -not- A CLASS AND cannot BE INSTANTIATED (i.e. IT HAS NO CONSTRUCTOR.)

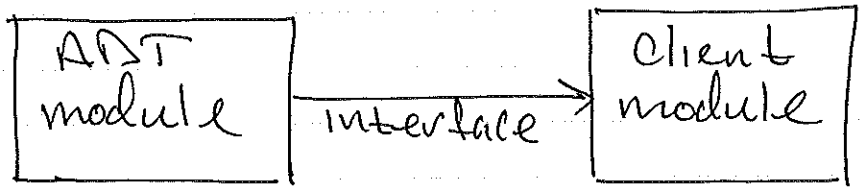
- A class CAN IMPLEMENT ANY NUMBER OF INTERFACES, WHEREAS IT CAN EXTEND ONLY ONE CLASS.

(THUS INTERFACES OFFER THE NEED FOR MULTIPLE INHERITANCE IN C++.)

TO DEFINE YOUR OWN INTERFACE USE THE KEYWORD INTERFACE INSTEAD OF Class

```
Public interface myInterface {
    // DEFINE CONSTANTS (static final)
    // method PROTOTYPES
}
```

AS THE NAME IMPLIES A JAVA INTERFACE CAN SERVE AS AN ADT interface



Ex Interface for IntegerList

```
Public interface IntegerListInterface{

    Public boolean isEmpty();
    Public int size();
    Public int get(int index)
        throws ListIndexOutOfBoundsException;
    Public void remove(int index)
        throws ListIndexOutOfBoundsException;
    Public void add(int index, int newItem)
        throws ListFullException,
               ListIndexOutOfBoundsException;
    Public void removeAll();

}
```

NOTE: COMMENTS SHOULD NOT BE LEFT OUT.

```
// IntegerListInterface.java
// interface for the IntegerList ADT

public interface IntegerListInterface{

   // isEmpty
   // pre: none
   // post: returns true if this IntgerList is empty, false otherwise
   public boolean isEmpty();

   // size
   // pre: none
   // post: returns the number of elements in this IntegerList
   public int size();

   // get
   // pre: 1 <= index <= size()
   // post: returns item at position index
   public int get(int index)
         throws ListIndexOutOfBoundsException;

   // add
   // inserts newItem in this IntegerList at position index
   // pre: size() < MAX_LENGTH, 1 <= index <= size()+1
   // post: !isEmpty(), items to the right of newItem are renumbered
   public void add(int index, int newItem)
         throws ListFullException, ListIndexOutOfBoundsException;

   // remove
   // deletes item from position index
   // pre: 1 <= index <= size()
   // post: items to the right of deleted item are renumbered
   public void remove(int index)
         throws ListIndexOutOfBoundsException;

   // removeAll
   // pre: none
   // post: isEmpty()
   public void removeAll();

}
```

```
// IntegerList.java
// Array based implementation of IntegerList ADT with Exceptions & Interface

public class IntegerList implements IntegerListInterface{

   public static final int MAX_LENGTH = 1000; // Maximum IntegerList length
   private int[] item;                        // array of IntegerList items
   private int numItems;                      // number of items in this
   IntegerList

   // arrayIndex
   // transforms a List index to an Array index
   private int arrayIndex(int listIndex){
      return listIndex-1;
   }

   // IntegerList
   // default constructor for the IntegerList class
   public IntegerList(){
      item = new int[MAX_LENGTH];
      numItems = 0;
   }

   // isEmpty
   // pre: none
   // post: returns true if this IntgerList is empty, false otherwise
   public boolean isEmpty(){
      return(numItems == 0);
   }

   // size
   // pre: none
   // post: returns the number of elements in this IntegerList
   public int size() {
      return numItems;
   }

   // get
   // pre: 1 <= index <= size()
   // post: returns item at position index
   public int get(int index)
      throws ListIndexOutOfBoundsException {

      if( index<1 || index>numItems ){
         throw new ListIndexOutOfBoundsException("get() precondition
         violated");
      }

      return item[arrayIndex(index)];
   }
```

```
// add
// inserts newItem in this IntegerList at position index
// pre: size() < MAX_LENGTH, 1 <= index <= size()+1
// post: !isEmpty(), items to the right of newItem are renumbered
public void add(int index, int newItem)
   throws ListFullException, ListIndexOutOfBoundsException{

   if( numItems == MAX_LENGTH ){
      throw new ListFullException("add() precondition violated");
   }

   if( index<1 || index>(numItems+1) ){
      throw new ListIndexOutOfBoundsException("add() precondition
      violated");
   }

   for(int i=numItems; i>=index; i--) {
      item[arrayIndex(i+1)] = item[arrayIndex(i)];
   }
   item[arrayIndex(index)] = newItem;
   numItems++;
}

// remove
// deletes item from position index
// pre: 1 <= index <= size()
// post: items to the right of deleted item are renumbered
public void remove(int index)
   throws ListIndexOutOfBoundsException{

   if( index<1 || index>numItems ){
      throw new ListIndexOutOfBoundsException("remove() precondition
      violated");
   }

   for(int i=index+1; i<=numItems; i++){
      item[arrayIndex(i-1)] = item[arrayIndex(i)];
   }
   numItems--;
}

// removeAll
// pre: none
// post: isEmpty()
public void removeAll(){
   numItems = 0;
}

// toString
// pre: none
// post:  prints current state to stdout
// Overrides Object's toString() method
public String toString(){
   int i;
   String s = "";

   for(i=0; i<numItems; i++) s += item[i] + " ";
   return s;
}
```

```
    // equals
    // pre: none
    // post: returns true if this IntegerList matches rhs, false otherwise
    // Overrides Object's equals() method
    public boolean equals(Object rhs){
       int i = 0;
       boolean eq = false;
       IntegerList R = null;

       if(rhs instanceof IntegerList){
          R = (IntegerList)rhs;
          eq = (this.numItems == R.numItems);
          while(eq && i<numItems){
             eq = (this.item[i] == R.item[i]);
             i++;
          }
       }
       return eq;
    }

}
```

```
// IntegerListTest.java
// A test client for the IntegerList ADT

public class IntegerListTest{

   public static void main(String[] args){
      IntegerList A = new IntegerList();
      IntegerList B = new IntegerList();
      int i, j;

      for(i=1; i<=100; i++){
         j = i*i;
         A.add(i, j);
         B.add(i, (j+i)/2);
      }

      System.out.println(A);
      System.out.println();
      System.out.println(B);
      System.out.println();
      System.out.println(A.equals(B));
      System.out.println();
      System.out.println(A.size());
      System.out.println();
      System.out.println(B.size());
      System.out.println();

      for(i=1; i<=10; i++){
         A.remove(9*i);
         B.remove(8*i-3);
      }

      System.out.println(A.size());
      System.out.println();
      System.out.println(B.size());
      System.out.println();
      System.out.println(B.get(37));
      System.out.println();
      try{
         System.out.println(A.get(200));
      }catch(ListIndexOutOfBoundsException e){
         System.out.println("Caught Exception " + e);
         System.out.println("Continuing without interuption");
      }
      System.out.println();
      System.out.println(A.get(20));
   }
}
```

```
// ListFullException.java

public class ListFullException extends RuntimeException{
   public ListFullException(String s){
      super(s);
   }
}
```

```
// ListIndexOutOfBoundsException.java

public class ListIndexOutOfBoundsException extends IndexOutOfBoundsException{
   public ListIndexOutOfBoundsException(String s){
      super(s);
   }
}
```

```
#
# makefile for IntegerList ADT
#

JAVASRC    = IntegerList.java IntegerListInterface.java IntegerListTest.java\
             ListFullException.java ListIndexOutOfBoundsException.java
MAINCLASS  = IntegerListTest
CLASSES    = IntegerList.class IntegerListInterface.class IntegerListTest.
class\
             ListFullException.class ListIndexOutOfBoundsException.class
JARFILE    = IntegerListTest
JARCLASSES = $(CLASSES)


all: $(JARFILE)

$(JARFILE): $(CLASSES)
   echo Main-class: $(MAINCLASS) > Manifest
   jar cvfm $(JARFILE) Manifest $(JARCLASSES)
   rm Manifest
   chmod +x $(JARFILE)

$(CLASSES): $(JAVASRC)
   javac $(JAVASRC)
# Note: no -Xlint option

clean:
   rm -f $(CLASSES) $(JARFILE)
```