




Application Program Development

Segment : Model Classes for the GUI

Mahboob Ali

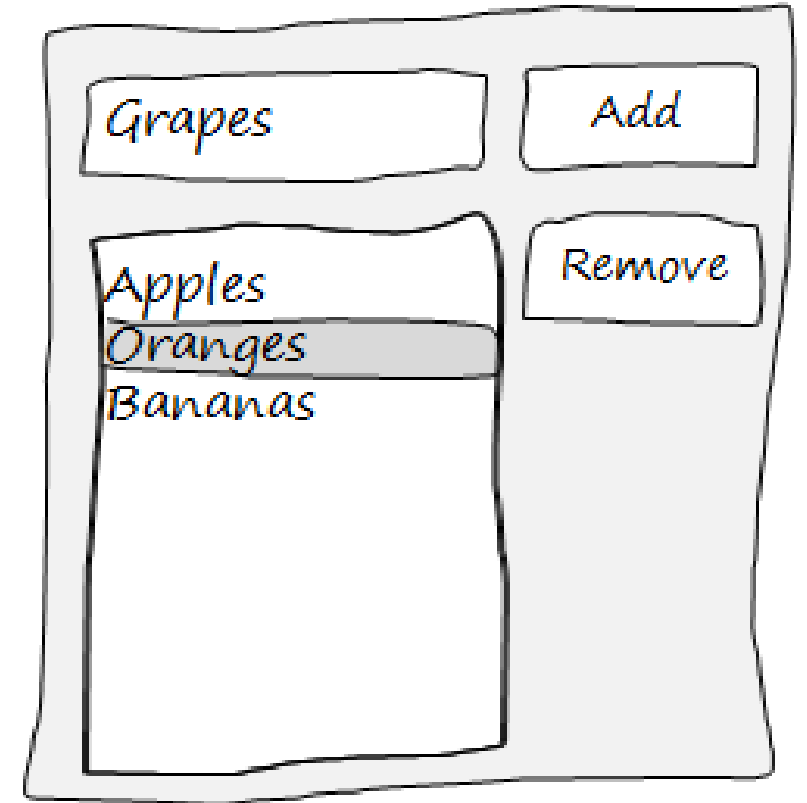


Outcomes

- Preparing Model classes
 - Proper design need and implementation
- 

- We already know how to build model classes ... they are the classes that make up your application apart from the user interface components.
- It is a good idea to prepare your model so that you can interact with it in a simple and clean manner from your user interface.
- A bank machine, for example, is somewhat pointless unless the underlying **Bank** model is fully operational.
- To finalize our model classes, we should decide what kinds of methods should be publicly available so that the main application's user interface can access, modify and manipulate the model in meaningful ways.
- For example, suppose that we wanted to develop the application that we described earlier that allowed us to make a list of things to purchase at the grocery store.
- What is the ***model*** in this application ?

- To figure this out, we just have to understand what "lies beneath" the user interface.
- What is it that we are displaying and changing ?
- It is the list of items.
- Let us develop a proper model for this interface →
- We can call it **ItemList** and it can keep track of an array of **Strings** that represent the list.



- Here is the basic code for this simple model:

```
public class ItemList {  
    public final int MAXIMUM_SIZE = 100;  
    private String[] items;  
    private int size;  
    public ItemList() {  
        items = new String[MAXIMUM_SIZE];  
        size = 0;  
    }  
    public int getSize() { return size; }  
    public String[] getItems() { return items; }  
}
```

- Looking back at the user interface, it is likely that we will want to add items (based on the text that is entered through the text field) to the list when the **Add** button is pressed.
- The item to be added will likely go at the bottom of the list. So, we should make a public method to do this:

```
public void add(String item) {  
    // Make sure that we do not go past the limit  
    if (size < MAXIMUM_SIZE) items[size++] = item;  
}
```

- Likewise, the **Remove** button will likely cause the currently selected item in the list to be removed from the list.
- We will probably remove it according to its index into the list. Here is a public method to do this:

```
public void remove(int index) {  
    // Make sure that the given index is valid  
    if ((index >= 0) && (index < size)) {  
        // Move every item after the deleted one up in the list  
        for (int i=index; i<size-1; i++) items[i] = items[i+1];  
        size--; // Reduce the list size by 1  
    }  
}
```

- Once we have the model implemented with useful methods, it is always a good idea to **test it!** The easiest way to do this is to write a simple test program to try out the various methods. We should write a test program that does a **thorough** testing. We should at least try to add a few items to the list, remove one, remove too many and add too many:

```
public class ItemListTestProgram {  
    public static void main(String[] args) {  
        // Make a new list  
        ItemList groceryList = new ItemList();  
        System.out.println("List has " + groceryList.getSize() + "  
                           items");  
  
        // Add a few items  
        System.out.println("\nAdding Apples, Oranges and Bananas ...");  
        groceryList.add("Apples");  
        groceryList.add("Oranges");  
        groceryList.add("Bananas");  
        System.out.println("List has " + groceryList.getSize() + " items");  
        System.out.println("Here are the items in the list:");  
        for (int i=0; i<groceryList.getSize(); i++)  
            System.out.println(groceryList.getItems()[i]);  
    }  
}
```



```
// Remove an item
System.out.println("\nRemoving Apples ...");
groceryList.remove(0);
System.out.println("List has " + groceryList.getSize() + " items");
System.out.println("Here are the items in the list:");
for (int i=0; i<groceryList.getSize(); i++)
    System.out.println(groceryList.getItems()[i]);
// Try to remove too many items
System.out.println("\nTrying to remove too many items ...");
groceryList.remove(0);
groceryList.remove(0);
groceryList.remove(0);
groceryList.remove(0);
System.out.println("List has " + groceryList.getSize() + " items");
System.out.println("Here are the items in the list:");
for (int i=0; i<groceryList.getSize(); i++)
    System.out.println(groceryList.getItems()[i]);
// Try to add too many items
System.out.println("\nTrying to add too many items ...");
for (int i=0; i<200; i++)
    groceryList.add("Item# " + i);
System.out.println("List has " + groceryList.getSize() + " items");
System.out.println("Here are the items in the list:");
for (int i=0; i<groceryList.getSize(); i++)
    System.out.println(groceryList.getItems()[i]);
}
```