

# COMP1406 - Assignment #2

(Due: Tuesday, February 14th @ 9:00pm)



In this assignment you will simulate a **Customer** that goes to a grocery store and buys items. The purpose is to get more familiar with programming using inheritance, interfaces and Graphical User Interfaces.

## (1) The **GroceryItem** Class

Create, save and compile the following interface definition in a file called **Carryable.java**:

```
public interface Carryable {
    public GroceryItem[]    getContents();
    public String           getDescription();
    public float            getPrice();
}
```

Here is a class that represents a **GroceryItem** that you buy at the grocery store:

```
public class GroceryItem implements Carryable {
    String      name;
    float       price;
    float       weight; // in kg

    public GroceryItem(String n, float p, float w) {
        name = n;
        price = p;
        weight = w;
    }
}
```



Complete this class by implementing a public **toString()** method that returns a String representing a **GroceryItem** with the format shown below. Note that the RED bolded text should represent the values from the attributes ... and so should vary from item to item (do not worry about the exact formatting of the price amount):

**SnackPack Pudding** weighing **0.396** kg with price **\$0.99**

Since the class implements the **Carryable** interface, you will need also need to write a **getContents()**, **getDescription()** and **getPrice()** method. The contents should return an empty array of size 0. The description should be the item name. The price should be the item price.

## (2) The **PerishableItem** Class

Create a subclass of **GroceryItem** called **PerishableItem** that represents an item that can spoil. Implement a public **toString()** method that returns a String representation of the **PerishableItem** as follows (making sure to make full use of inheritance):

**Smart-Ones Frozen Entrees** weighing **0.311** kg with price **\$1.99** (perishable)

### (3) The **GroceryBag** class

Consider the following **GroceryBag** class that represents a bag which will hold **GroceryItem** objects.

```
public class GroceryBag implements Carryable {
    public static final float MAX_WEIGHT = 5; // max weight allowed (kg)
    public static final int MAX_ITEMS = 50; // max # items allowed

    GroceryItem[] items; // actual GroceryItems in bag
    int numItems; // # of GroceryItems in bag
    float weight; // current weight of bag

    //... continue the code on your own as described below ...
}
```



For the **GroceryBag** class, create the following methods:

- A zero-parameter constructor that sets all three attributes properly.
- A **toString()** method that returns a string showing the number of items in the bag and the total weight as follows: "4.3 kg grocery bag with 8 items". However, if the bag is empty, this method should return "An empty grocery bag".
- **getContents()** which returns the array of items in the bag.
- **getDescription()** which returns "GROCERY BAG (xxx kg)" where xxx is the weight of the bag.
- **getPrice()** which returns the total price of all items in the bag.
- A method called **addItem(GroceryItem g)** which adds the given **GroceryItem** to the bag, provided that the weight of the item will not cause the bag to exceed the MAX\_WEIGHT and the number of items in the bag has not exceeded MAX\_ITEMS (make sure to update weight).

Test your class with the following test program:

```
public class GroceryBagTestProgram {
    public static void main(String args[]) {
        GroceryItem g1, g2, g3, g4, g5;
        GroceryBag b1, b2;

        g1 = new GroceryItem("Jumbo Cherries", 6.59f, 1.0f);
        g2 = new PerishableItem("Smart-Ones Frozen Entrees", 1.99f, 0.311f);
        g3 = new GroceryItem("SnackPack Pudding", 0.99f, 0.396f);
        g4 = new GroceryItem("Nabob Coffee", 3.99f, 0.326f);
        g5 = new PerishableItem("Fresh Salmon", 4.99f, 0.413f);

        b1 = new GroceryBag();
        b2 = new GroceryBag();

        b1.addItem(g1); b1.addItem(g2); b1.addItem(g3);
        b1.addItem(g4); b1.addItem(g5); b1.addItem(g5); b1.addItem(g5);

        System.out.println("BAG 1: " + b1);
        for(int i=0; i<b1.numItems; i++)
            System.out.println(" " + b1.items[i]);
        System.out.println("BAG 2: " + b2);
    }
}
```

```

        b1.addItem(new GroceryItem("Coca-Cola 12-pack", 3.49f, 5));
        b2.addItem(new GroceryItem("Coca-Cola 12-pack", 3.49f, 5));

        System.out.println("BAG 1: " + b1);
        System.out.println("BAG 2: " + b2);
    }
}

```

Here is the output that you should see:

```

BAG 1: A 3.2720003 kg grocery bag with 7 items
  Jumbo Cherries weighing 1.0 kg with price $6.59
  Smart-Ones Frozen Entrees weighing 0.311 kg with price $1.99 (perishable)
  SnackPack Pudding weighing 0.396 kg with price $0.99
  Nabob Coffee weighing 0.326 kg with price $3.99
  Fresh Salmon weighing 0.413 kg with price $4.99 (perishable)
  Fresh Salmon weighing 0.413 kg with price $4.99 (perishable)
  Fresh Salmon weighing 0.413 kg with price $4.99 (perishable)
BAG 2: An empty grocery bag
BAG 1: A 3.2720003 kg grocery bag with 7 items
BAG 2: A 5.0 kg grocery bag with 1 items

```

## (4) The **Customer** class

Consider the **Customer** class below which represents a person that buys grocery items.



```

public class Customer {
    public static final int    MAX_ITEMS = 100; // max # items allowed

    Carryable[]  shoppingCart; // items to be purchased
    int          numItems;     // #items to be purchased

    public String toString() {
        return "Customer with shopping cart of " + numItems + " items";
    }
}

```

Create the following interesting **Customer** methods. A thorough test case is given at the end of this section ... use it to test your program as you go by commenting out the parts that you did not complete yet:

- A proper zero-parameter constructor that sets all attributes.
- An **addItem(GroceryItem g)** method that adds a given **GroceryItem** to the shopping cart.
- Write a public **packBags()** method that simulates the contents of the shopping cart being packed into bags. The method should create a new array (e.g., called **packedItems**) that contains both **GroceryBag** objects and **GroceryItem** objects. The method should use the following packing algorithm: Create a **GroceryBag** object and then take items in order from the shopping cart and place them into the bag. If an item is encountered that itself exceeds **MAX\_WEIGHT** (e.g., a case of coke), then just add that item to **packedItems** and continue packing the same bag. If an item is encountered that would normally fit in a bag but that could not be added to the current bag because it would make the bag's weight exceed

MAX\_WEIGHT, then stop packing the bag, add it to **packedItems**, and start a new bag with that single item in it. You will need to use a typecast in order to access the weight attribute. The items should be taken sequentially from the shopping cart in the order that they were added to the cart. Hence, **packedItems** will contain both **GroceryBag** and **GroceryItem** objects. Once completed, this method should replace the shopping cart with this new array of packed items (simulating the fact that the customer emptied the shopping cart to buy the items and then put the packed bags and no-packable items back into the cart to leave).

- A public **displayPackedItems()** method that takes the **packedItems** and displays them nicely as shown below. You ARE NOT allowed to use **instanceof** in your solution, nor **getClass()**. Note the 2 space indentation for easy readability. There is a minor bug in JCreator that sometimes leaves a blank line in the output at weird locations ... do not lose any sleep over this as there is nothing you can do about it.

```
GROCERY BAG (3.516 kg)
  Smart-Ones Frozen Entrees weighing 0.311 kg with price $1.99 (perishable)
  SnackPack Pudding weighing 0.396 kg with price $0.99
  Breyers Chocolate Icecream weighing 2.27 kg with price $2.99 (perishable)
  Nabob Coffee weighing 0.326 kg with price $3.99
  Gold Seal Salmon weighing 0.213 kg with price $1.99

Coca-Cola 12-pack

GROCERY BAG (4.224 kg)
  Ocean Spray Cranberry Cocktail weighing 2.26 kg with price $2.99
  Heinz Beans Original weighing 0.477 kg with price $0.79
  etc...
```

Here is a test program that you should use to test your code:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        GroceryItem g1, g2, g3, g4, g5, g6, g7, g8, g9, g10, g11;

        g1 = new PerishableItem("Smart-Ones Frozen Entrees",1.99f,0.311f);
        g2 = new GroceryItem("SnackPack Pudding",0.99f,0.396f);
        g3 = new PerishableItem("Breyers Chocolate Icecream",2.99f,2.27f);
        g4 = new GroceryItem("Nabob Coffee",3.99f,0.326f);
        g5 = new GroceryItem("Gold Seal Salmon",1.99f,0.213f);
        g6 = new GroceryItem("Ocean Spray Cranberry Cocktail",2.99f,2.26f);
        g7 = new GroceryItem("Heinz Beans Original",0.79f,0.477f);
        g8 = new PerishableItem("Lean Ground Beef",4.94f,0.75f);
        g9 = new PerishableItem("5-Alive Frozen Juice",0.75f,0.426f);
        g10 = new GroceryItem("Coca-Cola 12-pack",3.49f,5.112f);
        g11 = new GroceryItem("Toilet Paper - 48 pack",40.96f,10.89f);

        // Make a new customer and add some items to his/her shopping cart
        Customer c = new Customer();
        c.addItem(g1); c.addItem(g2); c.addItem(g3); c.addItem(g4);
        c.addItem(g5); c.addItem(g6); c.addItem(g7); c.addItem(g8);
        c.addItem(g9); c.addItem(g10); c.addItem(g1); c.addItem(g6);
        c.addItem(g2); c.addItem(g2); c.addItem(g3); c.addItem(g3);
        c.addItem(g3); c.addItem(g3); c.addItem(g3); c.addItem(g10);
        c.addItem(g11); c.addItem(g9); c.addItem(g5); c.addItem(g6);
        c.addItem(g7); c.addItem(g8); c.addItem(g8); c.addItem(g8);
```

```

    c.addItem(g5);

    // Pack the bags and show the contents
    c.packBags();
    c.displayPackedItems();
}
}

```

Here is the expected result:

```

GROCERY BAG (3.516kg)
  Smart-Ones Frozen Entrees weighing 0.311 kg with price $1.99 (perishable)
  SnackPack Pudding weighing 0.396 kg with price $0.99
  Breyers Chocolate Icecream weighing 2.27 kg with price $2.99 (perishable)
  Nabob Coffee weighing 0.326 kg with price $3.99
  Gold Seal Salmon weighing 0.213 kg with price $1.99

Coca-Cola 12-pack

GROCERY BAG (4.224kg)
  Ocean Spray Cranberry Cocktail weighing 2.26 kg with price $2.99
  Heinz Beans Original weighing 0.477 kg with price $0.79
  Lean Ground Beef weighing 0.75 kg with price $4.94 (perishable)
  5-Alive Frozen Juice weighing 0.426 kg with price $0.75 (perishable)
  Smart-Ones Frozen Entrees weighing 0.311 kg with price $1.99 (perishable)

GROCERY BAG (3.0519998kg)
  Ocean Spray Cranberry Cocktail weighing 2.26 kg with price $2.99
  SnackPack Pudding weighing 0.396 kg with price $0.99
  SnackPack Pudding weighing 0.396 kg with price $0.99

GROCERY BAG (4.54kg)
  Breyers Chocolate Icecream weighing 2.27 kg with price $2.99 (perishable)
  Breyers Chocolate Icecream weighing 2.27 kg with price $2.99 (perishable)

GROCERY BAG (4.54kg)
  Breyers Chocolate Icecream weighing 2.27 kg with price $2.99 (perishable)
  Breyers Chocolate Icecream weighing 2.27 kg with price $2.99 (perishable)

Coca-Cola 12-pack

Toilet Paper - 48 pack

GROCERY BAG (2.9090002kg)
  Breyers Chocolate Icecream weighing 2.27 kg with price $2.99 (perishable)
  5-Alive Frozen Juice weighing 0.426 kg with price $0.75 (perishable)
  Gold Seal Salmon weighing 0.213 kg with price $1.99

GROCERY BAG (4.987kg)
  Ocean Spray Cranberry Cocktail weighing 2.26 kg with price $2.99
  Heinz Beans Original weighing 0.477 kg with price $0.79
  Lean Ground Beef weighing 0.75 kg with price $4.94 (perishable)
  Lean Ground Beef weighing 0.75 kg with price $4.94 (perishable)
  Lean Ground Beef weighing 0.75 kg with price $4.94 (perishable)

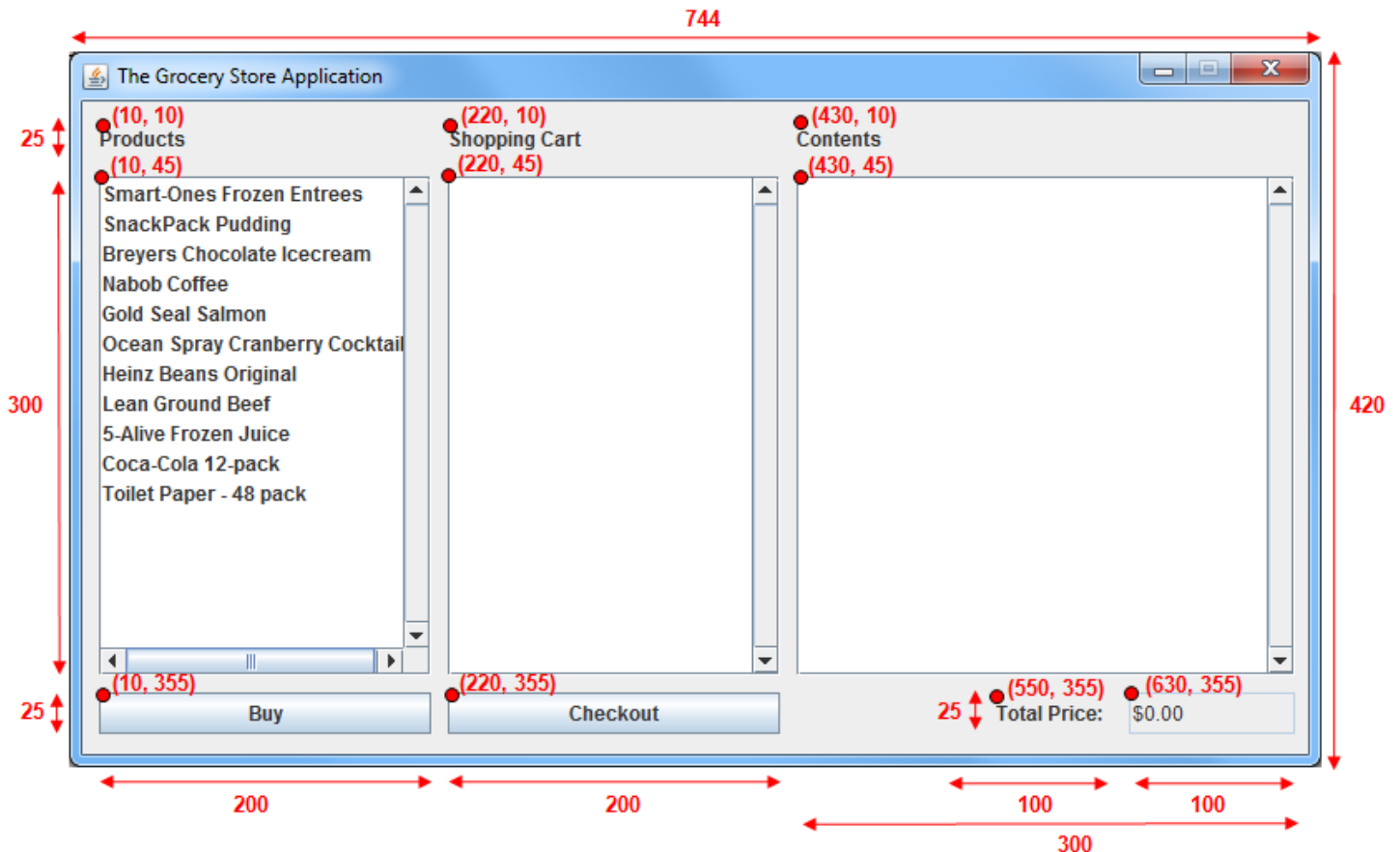
GROCERY BAG (0.213kg)
  Gold Seal Salmon weighing 0.213 kg with price $1.99

```

---

## (5) The GUI

Finally, we will create a simple application (called **GroceryStoreApp.java**) to manipulate the objects that we created. Create the following GUI with 4 **JLabels**, 3 **JLists**, 2 **JButtons** and 1 non-editable **JTextField**. The red letters show the locations and dimensions:



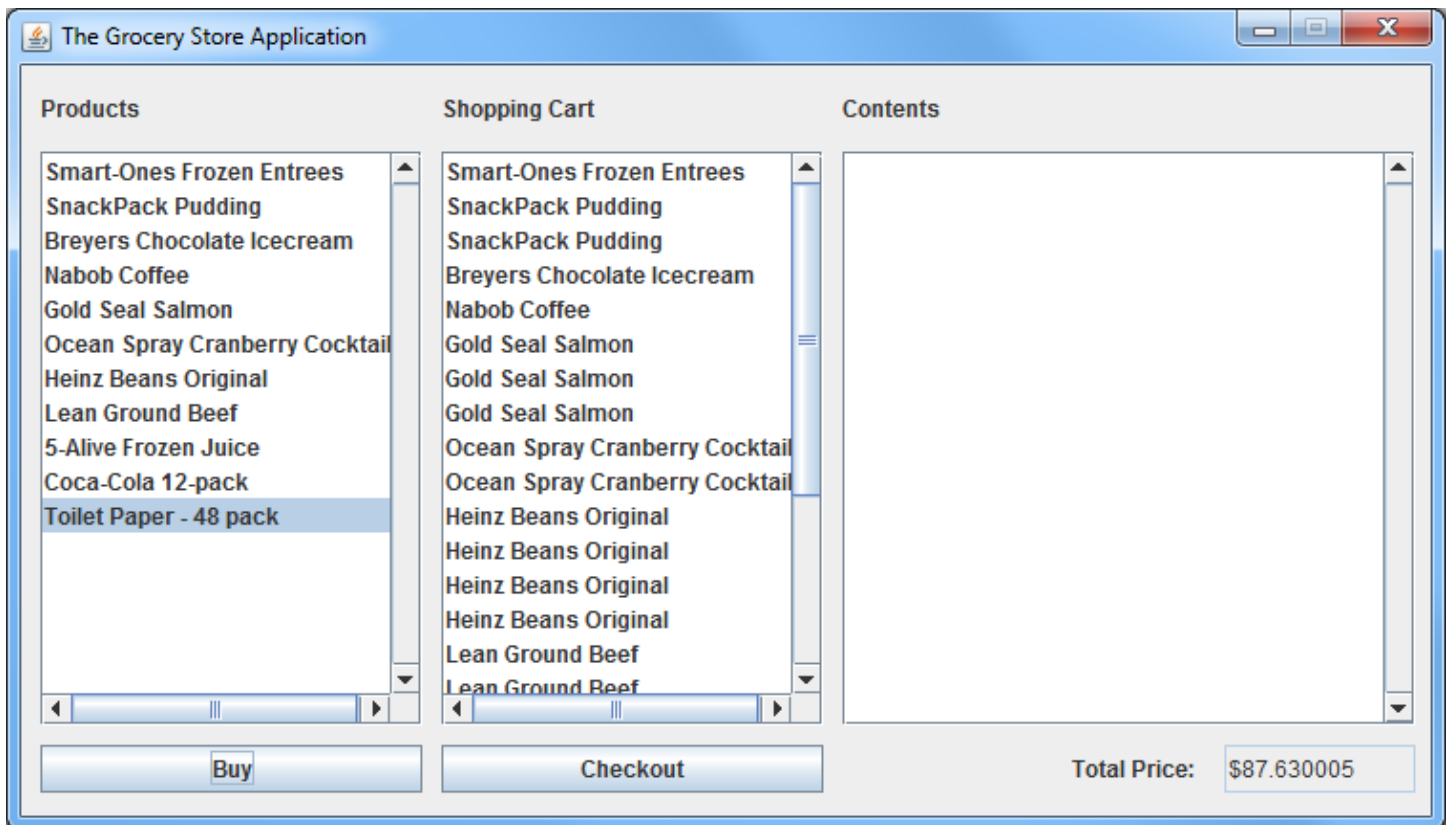
The following array was created and it should be the contents of the **Products** list (you will need to add only the names of the grocery items to the JList):

```
GroceryItem[]    products = {
    new PerishableItem("Smart-Ones Frozen Entrees", 1.99f, 0.311f),
    new GroceryItem("SnackPack Pudding", 0.99f, 0.396f),
    new PerishableItem("Breyers Chocolate Icecream", 2.99f, 2.27f),
    new GroceryItem("Nabob Coffee", 3.99f, 0.326f),
    new GroceryItem("Gold Seal Salmon", 1.99f, 0.213f),
    new GroceryItem("Ocean Spray Cranberry Cocktail", 2.99f, 2.26f),
    new GroceryItem("Heinz Beans Original", 0.79f, 0.477f),
    new PerishableItem("Lean Ground Beef", 4.94f, 0.75f),
    new PerishableItem("5-Alive Frozen Juice", 0.75f, 0.426f),
    new GroceryItem("Coca-Cola 12-pack", 3.49f, 5.112f),
    new GroceryItem("Toilet Paper - 48 pack", 40.96f, 10.89f)};
```

The interface should work as follows:

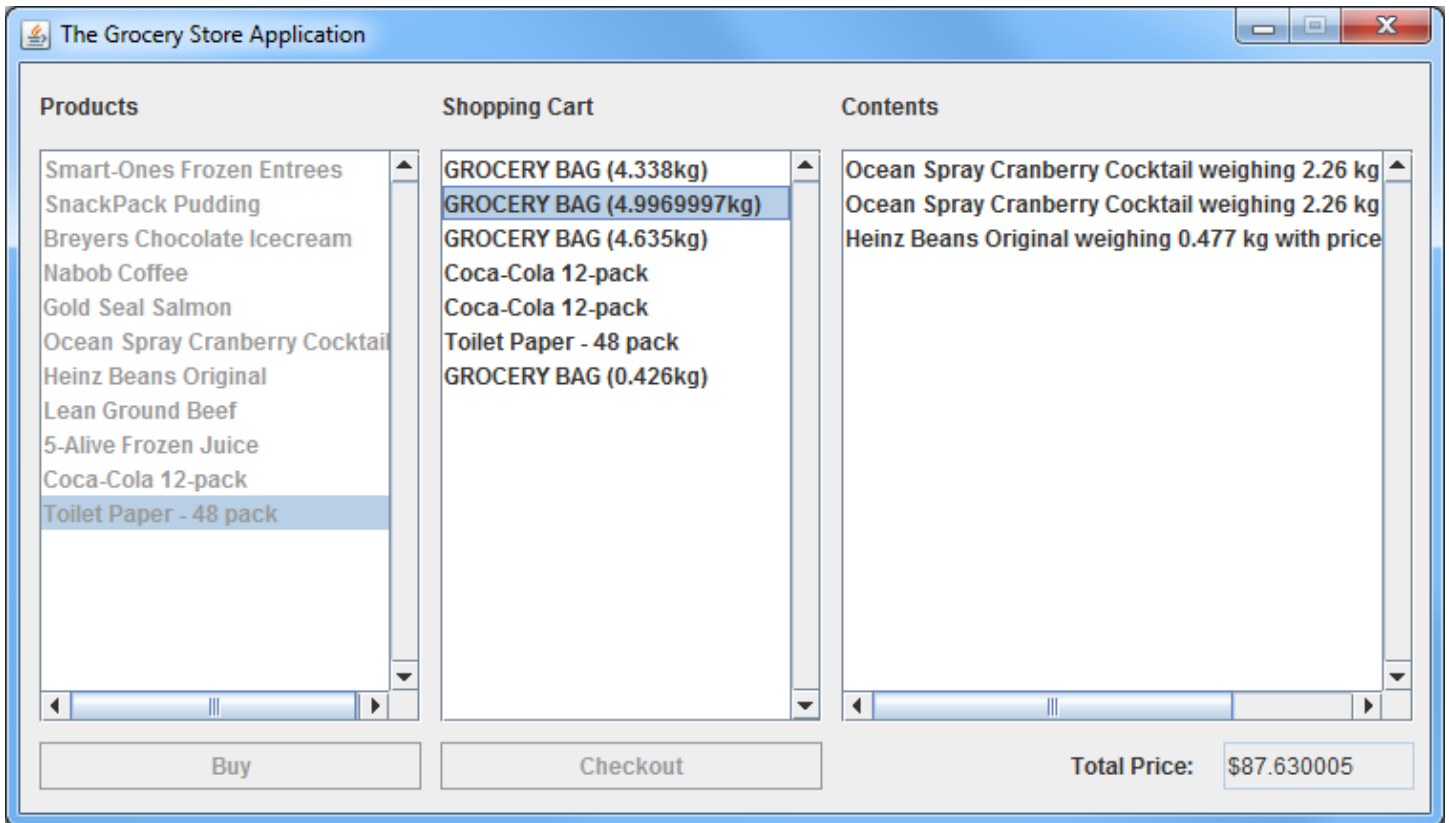
- The window should appear with both buttons enabled, the **Products** list populated with item names and the total price should be 0.
- Each time the **Buy** button is pressed, the name of the item selected from the **Products** list should appear in the **Shopping Cart** list.

- The **Total Price** should always show the total price of all items in the shopping cart



When **Checkout** is pressed, the following should happen:

- the **Buy** and **Checkout** buttons and **Products** list should be disabled
- the **Shopping Cart** contents should be packed using **packBags()**.
- the **Shopping Cart** list should display the packed items, showing the **description** of all **Carryable** items (see below).
- if a **Grocery Bag** is then selected from the **Shopping Cart** list, its contents should appear in the **Contents** list (see below).
- if an unpacked item is selected from the **Shopping Cart** list, nothing should appear in the **Contents** list.



**NOTE:** Submit (on WebCT) all **.java** and **.class** files needed to run. You **MUST NOT use packages** in your code, **nor projects**. Submit ALL of your files in one folder such that they can be opened and compiled individually in JCreator. Some IDEs may create packages and/or projects automatically. You **MUST** export the .java files and remove the package code at the top if it is there. Do NOT submit JCreator projects either. **JUST SUBMIT the JAVA and CLASS FILES**. Note that if your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late ... so make sure to submit the assignment WELL BEFORE it is due !

**Please NOTE that you WILL lose marks on this assignment if any of your files are missing. You will also lose marks if your code is not written neatly with proper indentation. See examples in the notes for proper style.**