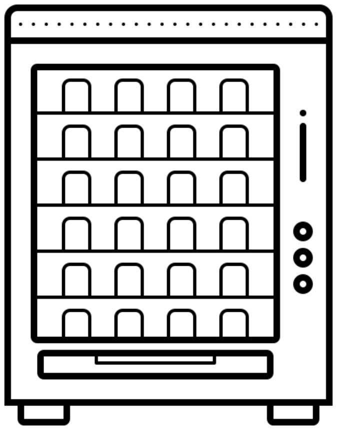
# HW#1:  Vending Machine



For HW#1, you will **use object oriented concepts and Java data structures to model a vending machine**.

Our vending machine will serve two different kinds of products: **Drinks** and **Snacks**.

Our vending machine will have an unlimited number of **slots**.  A slot has a location, for example “A1”, and it will contain N products of the same type.  For example, **“A1” may contain 10 diet cokes**.

Your **vending machine** must support the following operations:

- `stockItems()`: with three parameters
  - `String location // e.g. “A1”`
  - `Stack<Product> productList`
  - `double unitPrice`

- `printInventory()`: prints out all the slots with product details.  Sample output provided below.

- `getTotalSales()`:  returns the total sales at the vending machine.

- `purchase()`: with two parameters:
  - location: a slot location, e.g. “A1”
  - amount: amount of money the user deposits, e.g. 2.0
  - purchase will then return a **VendingMachineOutput** class with two properties:
    - product: for example, if the user selects “A1”, they will get back a Diet Coke Object.
    - change: for example, if a user deposits $2 and a Diet Coke costs $1.50, the user will receive .50 in change.
  - Your purchase() method must handle a number of edge cases.  For example:
    - User may specify an invalid slot ID.
    - The slot may not have any remaining products.
    - The user may not deposit enough money to make a purchase.
  - If any of these edge cases occur, please throw an IllegalArgumentException.

You will need to model an abstract **Product** Class, a **Drink** Class and a **Snack** Class.  Each subclass should implement:

- **consume**():
  - For the Drink Class, you can output, e.g. “Yum, you drink the Diet Coke"

- For the Snack Class, you can output, e.g  “Yum, you eat the Cliff Bar”.

You may also need other intermediate classes that use other data structures, but it really depends on how you solve this.

Here is my sample code to get your started:

```java
import java.util.Stack;

/**
* Run the Vending Machine.
*/
public class RunVendingMachine {

   /**
    *  Illustrates the Vending Machine.
    */
   public static void main(String[] args) {

       // Create a new Empty Vending Machine.
       VendingMachine machine = new VendingMachine();

       // Stock up of Diet Cokes
       Stack<Product> dietCokeStack = new Stack<Product>();
       for (int i=0; i<5; i++) {
           Drink dietCoke = new Drink("Diet Coke");
           dietCokeStack.push(dietCoke);
       }
       machine.stockItems("A1", dietCokeStack, 1.25);

       //  Stock up on Cliff Bars
       Stack<Product> cliffBarStack = new Stack<Product>();
       for (int i=0; i<3; i++) {
           Snack cliffBar = new Snack("Cliff Bar");
           cliffBarStack.push(cliffBar);
       }
       machine.stockItems("A2", cliffBarStack, 4.00);

       //  What do we have now?
       machine.printInventory();

       //  Try to purchase and drink a diet coke.
       VendingMachineOutput output = machine.purchase("A1", 2.00);
       System.out.println("Got:  " + output.product.productName);
       System.out.println("Received change:  " + output.change);
       output.product.consume();
       System.out.println("Total Sales:  " + machine.getTotalSales());

       //  Try to purchase and eat a cliff bar
```

```
        output = machine.purchase("A2", 5.00);
        System.out.println("Got:  " + output.product.productName);
        System.out.println("Received change:  " + output.change);
        output.product.consume();
        System.out.println("Total Sales:  " + machine.getTotalSales());
    }
}
```

And, here is my sample output:

```
A1
- Diet Coke
- Diet Coke
- Diet Coke
- Diet Coke
- Diet Coke
A2
- Cliff Bar
- Cliff Bar
- Cliff Bar
Got:  Diet Coke
Received change:  0.75
Yum, you drink the Diet Coke.
Total Sales:  1.25
Got:  Cliff Bar
Received change:  1.0
Yum, you eat the Cliff Bar.
Total Sales:  5.25
```

A few things to consider:

1. There are multiple solutions to this project. You might want to sketch out ideas first.
2. Don't forget error handling in the `purchase()` method! You will only get full credit if you implement all the edge cases and exception handling correctly.
3. Make sure to include Javadocs on all your public methods. For example:

```
/**
* Consume the Drink.
*/
@Override
public void consume() {
   System.out.println("Yum, you drink the " + this.productName + ".");
}
```

4. Lines of code should be <= 100 characters long.  Avoid really long lines of code!  Please do a visual inspection before submitting your homework.  If you see lines of code that go beyond the editor window, your code is probably too long.
5. Before submitting your homework, remember to reformat your code via IntelliJ.  Hit SHIFT-SHIFT, enter “format”, and select Reformat File.