Problem # 1:

Create a Class **MenuItem,** which has three instances

1. **name:** name of the item
2. **type:** whether *food* or a *drink*
3. **price:** price of the item

Write a class called **CoffeeShop**, which has three instance variables:

1. **name** : a string (basically, of the shop)
2. **menu** : an list of items (of object type), with each item containing the item (name of the item), type (whether *food* or a *drink*) and price.
3. **orders** : an empty list of string type.

And a parameterized constructor which takes the name of the CoffeeShop as a parameter.

and eight methods:

1. **addMenuItem:** adds the menu item in the list of menu
2. **addOrder:** adds the name of the item to the end of the orders list if it exists on the menu. Otherwise, return "This item is currently unavailable!"
3. **fulfillOrder:** if the orders list is not empty, return "The {item} is ready!" and make the list empty. If the order list is empty, return "All orders have been fulfilled!"
4. **listOrders:** returns the list of orders taken, otherwise null.
5. **dueAmount:** returns the total amount due for the orders taken.
6. **cheapestItem:** returns the name of the cheapest item on the menu.
7. **drinksOnly:** returns only the *item* names of *type* drink from the menu.
8. **foodOnly:** returns only the *item* names of *type* food from the menu.

IMPORTANT: Orders are fulfilled in a FIFO (first-in, first-out) order.

Following menu needs to be added for the Tesha's Coffee Shop

Name	**Type**	**Price**
"orange juice"	Drink	60

"lemonade"	Drink	50
"cranberry juice"	Drink	100
"pineapple juice"	Drink	100
"lemon iced tea"	Drink	120
"vanilla chai latte"	Drink	150
"hot chocolate"	Drink	140
"iced coffee"	Drink	140
"tuna sandwich"	Food	300
"ham and cheese sandwich"	Food	300
"egg sandwich"	Food	200
"steak"	Food	900
"hamburger"	Food	600
"cinnamon roll"	Food	150

Driver Program: (Following options should be shown to the user)

Welcome to the Tesha's Coffee Shop

1. Add a Menu item
2. View the Cheapest Item in the menu
3. View the Drink's Menu
4. View the Food's Menu
5. Add Order
6. Fulfill the Order
7. View the Orders's List
8. Total Payable Amount
9. Exit

Test Cases:

// After creating an object of CoffeeShop and initializing its attributes

tcs.addOrder("hot cocoa") ➞ "This item is currently unavailable!"
// Tesha's coffee shop does not sell hot cocoa
tcs.addOrder("iced tea") ➞ "This item is currently unavailable!"

// specifying the variant of "iced tea" will help the process

tcs.addOrder("cinnamon roll") ➞ "Order added!"
tcs.addOrder("iced coffee") ➞ "Order added!"
tcs.listOrders ➞ ["cinnamon roll", "iced coffee"]
// the list of all the items in the current order

tcs.dueAmount() ➞ 290

tcs.fulfillOrder() ➞ "The cinnamon roll is ready!"
tcs.fulfillOrder() ➞ "The iced coffee is ready!"
tcs.fulfillOrder() ➞ "All orders have been fulfilled!"
// all orders have been presumably served

tcs.listOrders() ➞ []
// an empty list is returned if all orders have been exhausted

tcs.dueAmount() ➞ 0.0
// no new orders taken, expect a zero payable

tcs.cheapestItem() ➞ "lemonade"
tcs.drinksOnly() ➞ ["orange juice", "lemonade", "cranberry juice", "pineapple juice", "lemon iced tea", "vanilla chai latte", "hot chocolate", "iced coffee"]
tcs.foodOnly() ➞ ["tuna sandwich", "ham and cheese sandwich", "bacon and egg", "steak", "hamburger", "cinnamon roll"]

Problem # 2:

In this problem, you have to create a class called **Point,** which models a 2D point with x and y coordinates.
It contains:

- Two instance variables x (int) and y (int).
- A default (or "no-argument" or "no-arg") constructor that constructs a point at the default location of (0, 0).
- A parameterized constructor that constructs a point with the given x and y coordinates.
- Getter and setter for the instance variables x and y.
- A method setXY() to set both x and y.

Next, create a class named **Boundary**.

It contains:

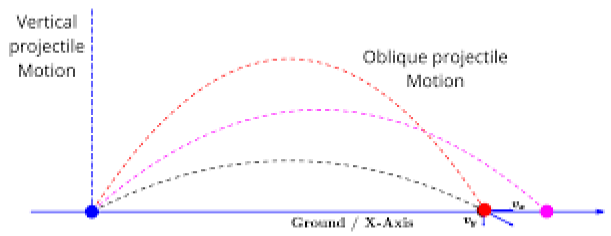
- Four attributes of Point type

    - TopLeft
    - TopRight
    - BottomLeft
    - BottomRight
- A default (or "no-argument" or "no-arg") constructor that constructs a boundary with default location of TopLeft(0, 0), TopRight(0,90), BottomLeft(90,0) and BottomRight(90,90).
- A parameterized constructor that constructs a boundary with the given TopLeft, TopRight, BottomLeft and BottomRight points.

Next, create a class named **GameObject.**

It contains 4 attributes:

- One attribute **Shape** (2D Array char type).
- A **StartingPoint** (Point type).
- A **Premises** (Boundary type).
- A **Direction** (String type).

LeftToRight	Starts from the starting point and keeps on moving towards the right and stops at the extreme right boundary point
RightToLeft	Starts from the starting point and keeps on moving towards the left and stops at the extreme left boundary point
Patrol	Starts from the starting point and keeps on moving towards the left and stops at the extreme left boundary point and reverses the direction.
Projectile	Starts from the starting point and move 5 points towards the top right and then 2 steps towards the right and then 4 steps towards bottom right.  Vertical projectile Motion Oblique projectile Motion Ground / X-Axis v_x v_y
Diagonal	Starts from the starting point and moves towards the bottom right and stops at the extreme right boundary point.

- A default constructor that initializes
    - Shape (1x3 line “---”)
    - StartingPoint (constructs a point at the default location of (0, 0))

    - Premises (constructs a boundary with default location of TopLeft(0, 0), TopRight(0,90), BottomLeft(90,0) and BottomRight(90,90))
    - Direction (“LeftToRight”)

- A parameterized constructor that takes
    - Shape, StartingPoint
    - Whereas **Premises** (constructs a boundary with default location of TopLeft(0, 0), TopRight(0,90), BottomLeft(90,0) and BottomRight(90,90)) and **Direction** with default direction (“LeftToRight”)

- A parameterized constructor that takes
    - Shape
    - StartingPoint
    - Premises
    - Direction

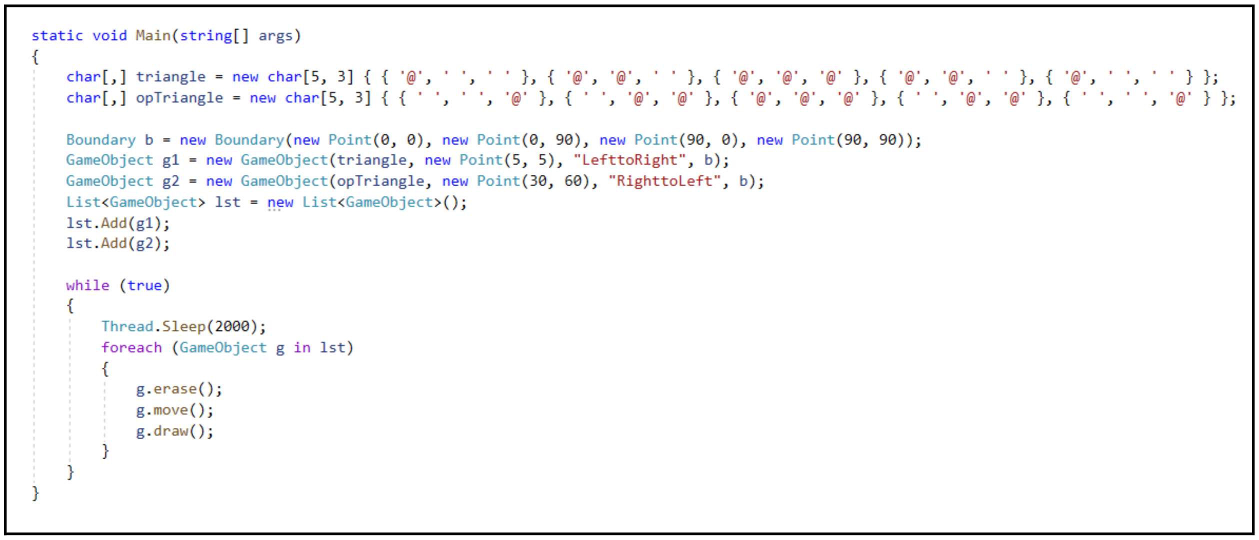
- It will also contain the following methods
    - **Move:** if the direction is “LeftToRight”, the shape will move one step according to its direction. For example, if the direction is from left to right it will move the game object one step toward right.
    - **Erase:** When called, this method will erase the shape on the console.
    - **Draw:** When called, this method will draw the shape on the console.

- Following are some examples of the shapes you can draw (Use your creativity to come up with different shapes)

```
 ¤¤¤¤
¤¤¤¤¤¤  ¤
¤¤¤¤¤¤ ¤¤¤
¤    ¤
```

```
  ☼☼
 ☼☼☼☼
☼☼☼☼☼☼
 ☼☼☼☼
  ☼☼
 ☼☼☼☼
```

Demo Driver Program:



```
static void Main(string[] args)
{
    char[,] triangle = new char[5, 3] { { '@', ' ', ' ' }, { '@', '@', ' ' }, { '@', '@', '@' }, { '@', '@', ' ' }, { '@', ' ', ' ' } };
    char[,] opTriangle = new char[5, 3] { { ' ', ' ', '@' }, { ' ', '@', '@' }, { '@', '@', '@' }, { ' ', '@', '@' }, { ' ', ' ', '@' } };

    Boundary b = new Boundary(new Point(0, 0), new Point(0, 90), new Point(90, 0), new Point(90, 90));
    GameObject g1 = new GameObject(triangle, new Point(5, 5), "LefttoRight", b);
    GameObject g2 = new GameObject(opTriangle, new Point(30, 60), "RighttoLeft", b);
    List<GameObject> lst = new List<GameObject>();
    lst.Add(g1);
    lst.Add(g2);

    while (true)
    {
        Thread.Sleep(2000);
        foreach (GameObject g in lst)
        {
            g.erase();
            g.move();
            g.draw();
        }
    }
}
```