# Lottery Machine Programming Challenge

| | |
|---|---|
| **Name:** | **Lincoln Powell** |
| **Date:** | **27 March 2016** |
| **Intent:** | **To display programming proficiency for employment consideration.** |

**Design:**
Acme Lottery Co. is developing a new Lottery Machine that will dispense different types of lottery tickets to customers. Your job is to write code to simulate one complete sale cycle and drawing of the Lottery Machine's functionality to provide insight for company executives. Program deliverables can be found here: https://github.com/lincolnpowell/challenge.

**Approach:**
I decided to be innovative with my design, employing a name generator for a way to easily identify customers. Additionally, keeping to the design of the program's deliverables, I chose to keep the methodology simple through use of class creation, array processing, and using the Random class to generate random numbers.

My decision to use random numbers to automate the selection of ticket numbers and array indexes versus preloaded input to direct the program to one finite conclusion lied on offering Acme Lottery Co. uniqueness on each program run. Moreover, I felt the best way to simulate that a customer would be equally likely to purchase any type of lottery ticket is through randomization on a switch statement.

I wanted to also display the depletion of the lottery tickets within the report to show the end user the emptying of the lottery machine. I felt this design choice helped in examining when customers would get to the point of not being able to purchase select type tickets, helping in predictability and readability of the report.

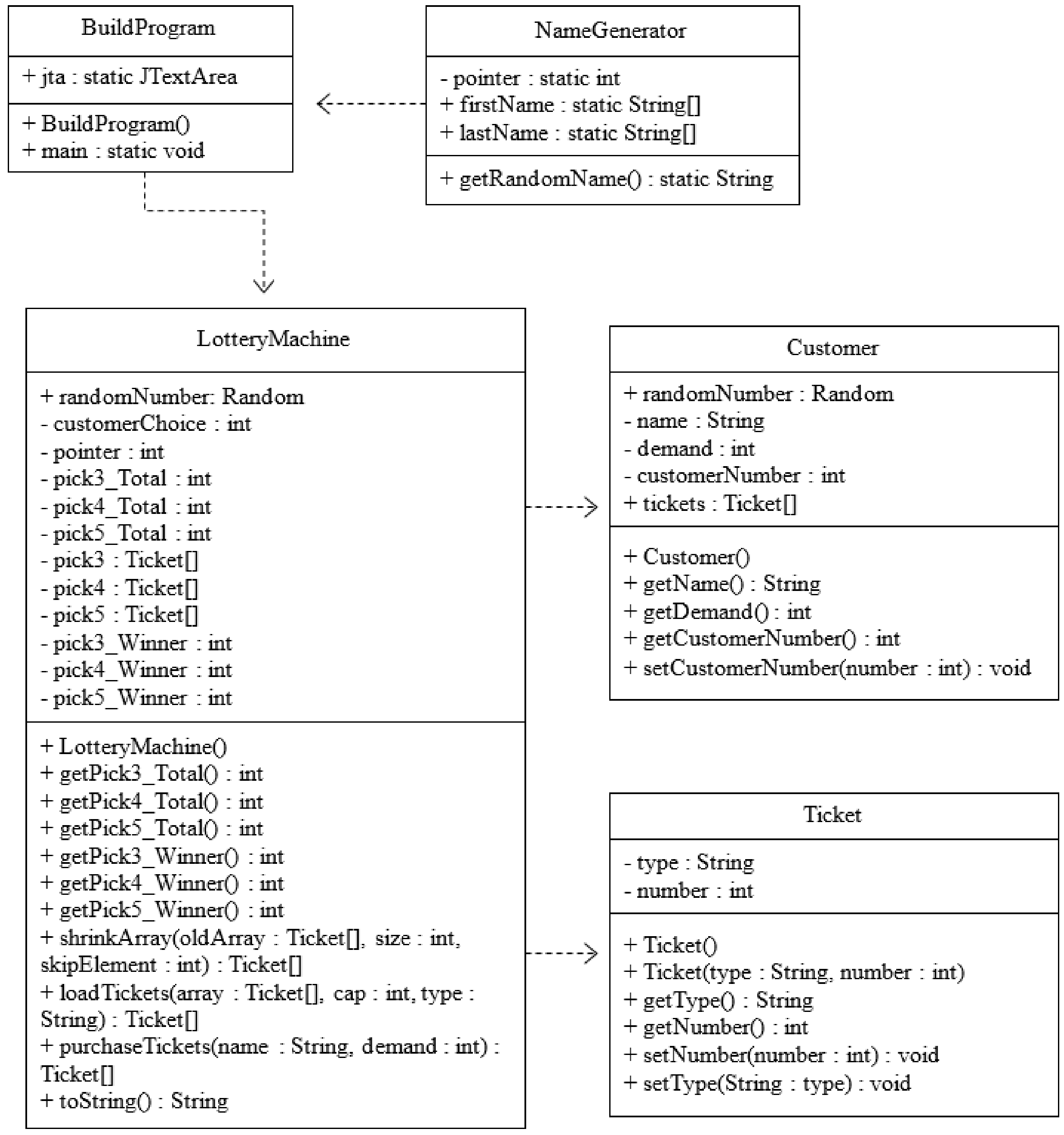
I also decided to create a Ticket class in order to avoid the case where a Pick 3, Pick 4, and Pick 5 would all be the same number (e.g. Pick 3: 100, Pick 4: 0100, and Pick 5: 00100 would all be considered integer 100 by the compiler, thus there would have to be a secondary condition to determine if the customer has the correct type as well). Addition of a type field allowed this situation to be prevented.

Finally, to avoid overcrowding data in a console output window, I decided to create a simple GUI with a text area to display the report. The report structure is as follows:

- On top, a synopsis of customers purchasing tickets and displaying the lottery machine state;

- Afterwards, displayed lines indicating the total number of customers who purchased tickets, which ticket numbers were selected during the drawing, and which customers won the drawing for each ticket type.

**UML:**


BuildProgram
+ jta : static JTextArea
+ BuildProgram()
+ main : static void
NameGenerator
- pointer : static int
+ firstName : static String[]
+ lastName : static String[]
+ getRandomName() : static String
LotteryMachine
+ randomNumber: Random
- customerChoice : int
- pointer : int
- pick3_Total : int
- pick4_Total : int
- pick5_Total : int
- pick3 : Ticket[]
- pick4 : Ticket[]
- pick5 : Ticket[]
- pick3_Winner : int
- pick4_Winner : int
- pick5_Winner : int
+ LotteryMachine()
+ getPick3_Total() : int
+ getPick4_Total() : int
+ getPick5_Total() : int
+ getPick3_Winner() : int
+ getPick4_Winner() : int
+ getPick5_Winner() : int
+ shrinkArray(oldArray : Ticket[], size : int, skipElement : int) : Ticket[]
+ loadTickets(array : Ticket[], cap : int, type : String) : Ticket[]
+ purchaseTickets(name : String, demand : int) : Ticket[]
+ toString() : String
Customer
+ randomNumber : Random
- name : String
- demand : int
- customerNumber : int
+ tickets : Ticket[]
+ Customer()
+ getName() : String
+ getDemand() : int
+ getCustomerNumber() : int
+ setCustomerNumber(number : int) : void
Ticket
- type : String
- number : int
+ Ticket()
+ Ticket(type : String, number : int)
+ getType() : String
+ getNumber() : int
+ setNumber(number : int) : void
+ setType(String : type) : void

**Test Plan:**
Each test run should examine the following four areas to ensure the program produces valid data:

1. Number of customers is consistent and correct
2. Purchased tickets reflect the correct type (e.g. ticket is shown to be part of that particular Ticket type array)
3. Customers who try to purchase sold out tickets eventually decide to purchase remaining tickets of another type
4. Do the tickets that were selected to win match the correct customers who drew the ticket

**Test Cases:**

| Test Case Number | Input Values (bold denotes input) | Expected Output Values | Pass/Fail |
|---|---|---|---|
| **1** | N/A | I expect the program to run successfully, displaying a correct report. | **Pass** (see Test Case 1.txt report readout) |
| **2** | N/A | I expect the program to run successfully, displaying a correct report. | **Pass** (see Test Case 2.txt report readout) |
| **3** | N/A | I expect the program to run successfully, displaying a correct report. | **Pass** (see Test Case 3.txt report readout) |