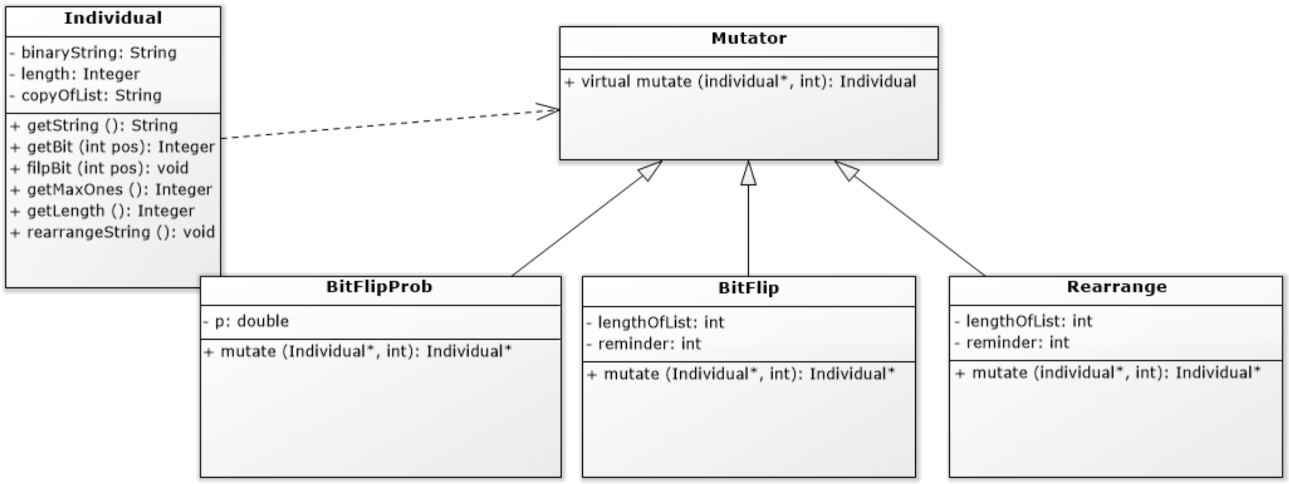
Class Diagram



Individual:

Attribute:

binaryString(string): initialize a variable that store the value of genes.

Length(int): initialize the length to store the size of the binaryString

copyOfList(string): initialize the copy of the binaryString

Behaviour:

getString(): string   // return the binaryString

getBit(): int // the function return the bit value at the position and return -1 if pos is out of bound.

void flipBit(int) // take in the position of the certain bit and flip the bit value. We use if statement to implement that function. For example, if the bit value is ‘0’, we need to aasign a ’1’ to the bit value otherwise ‘0’.

int MaxOnes() // return the longest consecutive sequence of ‘1’ digits in the list. Firstly, we need to define a MaxOnes variable to store the maximum ‘1’s, and then we use for loop to go through all the elements in the list. Inside the for loop, I define a sumOnes. There are couple of if statement in the for loop. If the binaryString[i] is ‘1’, sumOnes ++, if the maxOnes is less than sumOnes, I will assign the sumOnes to MaxOnes. If the binaryString[i] is ‘0’, we need to reset the sumOnes to 0.

Int getLength() // return the length of the binaryString

void rearrangeString(int pos) // I use for loop and if statement to set up the new order of list. If the K is 2 and length is 4, that means I need to put the last 3 elements 2,3,4 to the very beginning and get the first one to last position.

Mutator:

//this is abstract class

Virtual Indiviual* mutate (Individual* list, int k)// this is the virtual function and it will implement in the sub-class.

Subclass:

BitFlip:

Attribute:

lengthOfList: int //using getLength() to store the length of the list.

Reminder: int // counting in the circle, I use reminder that divide the k by the lengthOfList.

Behaviour:

Mutate(Indiviual*. int) // use the reminder as the parameter in the flipBit(int). we also need to return the list.

BitFlipPRob:

Atrribute:

P: int // the probability of the bitflip.

Mutate(Individual*, int)// return the P

Rearrange:

Attribute:

lengthOfList: int //using getLength() to store the length of the list.

Reminder: int // counting in the circle, I use reminder that divide the k by the lengthOfList.

Behaviour:

Mutate(Indiviual*. int) // use the reminder as the parameter in the rearrangeString(int). we also need to return the list.

Testing:

Input	Description	Output
0000 2 0111 2	Flip the second element in the first list and rearrange the second list and will return the longest sequence of 1	0100 1110 3
100111 12 0111 12	Count in circle 12%6=0, it should be the last element	110011 1011 2
00000 -1 01010 -1	Return false if the k is the negative number	-1
135456 1 1232 2	It not the binary digit	0
Asda112 1 eqwe12 3	It is not purely digit	0