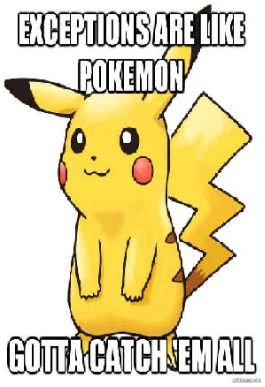
Assignment 8
Due March 25, 2021



1. A thrown exception can be handled in many different ways. For example, suppose method myMethod() throws an Exception. It might be

   - caught by myMethod(),
   - caught by a method that calls myMethod(),
   - not caught at all in the program (i.e. caught eventually by the system).

Write three classes:

- Reverse1,
- Reverse2, and
- Reverse3,

each of which has a method

public static String reverse(String s)

that accepts a string and returns the string in reverse.

If the String contains any characters other than letters or digits, however, your program should throw an IllegalCharacterException. The catch block should

- print: "Illegal Character in String",
- the actual string,
- and the name of the class.

An IllegalCharacterException is thrown if a character is not in the set {'a'..'z', '0'..'9', 'A'..'Z}. **You have to write the IllegalCharacterException class and it extends Exception.**

- Reverse1 throws the exception in reverse(...) but is not caught by reverse(...) or main() (i.e.,,no explicit catch blocks). The exception is caught by the system.
- Reverse2 throws the exception in reverse(...) and catches the exception in main. .
- Reverse3 throws and catches the exception in reverse(...).

For each class, include a main method that reads a string from input, passes it to reverse(...), and prints the reverse string. Output should be the original String reversed or an error message.

So there are three programs to deposit: Revers1.java, Reverse2.java, and Reverse3.java. The only difference among the classes is where the exception is caught. You will also need to deposit IllegalCharacterEXception.java

2. Here is a program that you wrote before. This time you should do it with the ArrayList class. In other words, **do not use arrays at all.** In fact, it is much easier to implement these classes using ArrayLists than it is using arrays.

Here is the assignment again:

There are many different types of lists into which data may be inserted and removed.

In this problem you will implement three types of lists:
A LIFO list, a FIFO list, and a PRIORITY list

Each of these list have similar but different methods

```
insert(x) // inserts x into the list
 remove() // removes and returns an item from the list
```

as well as a common method

```
getSize() // returns the number of items in the list
```

****

A **LIFO list** is a "Last In -- First Out" list.

So

```
insert(x) places x at the end of the list
remove() removes and returns the last item in the list
```

For example, if a is a LIFO list the code

```
a.insert(4);
a.insert(7);
a.insert(3);
```

produces a list of size 3 that looks like

```
4  7   3        // the three was the last one put into the list
```

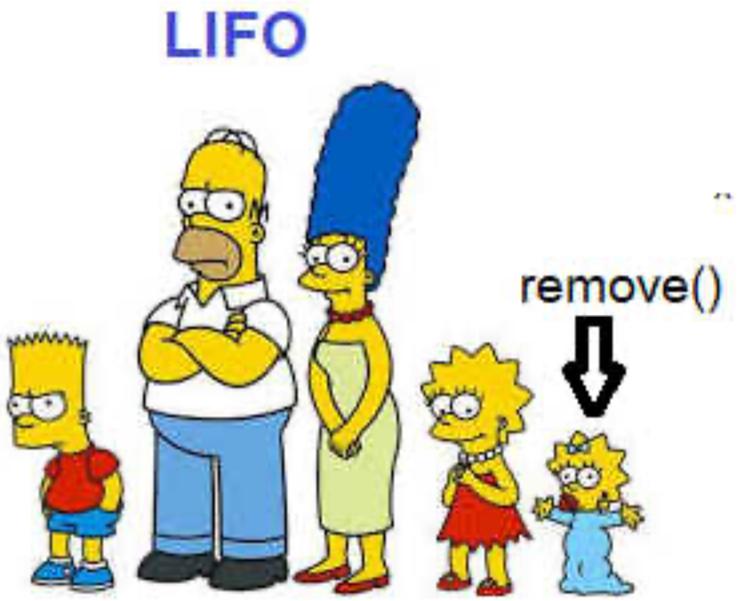
and the code

```
System.out.println(a.remove());
System.out.println(a.remove());
System.out.println(a.remove());
```

produces output

```
3
7
4
```

Last value inserted is the first value removed.



LIFO --Last in First Out

Maggie was the last to get in line, so the first to be removed

****

A **FIFO list** is a "First in - First Out" list. A FIFO list is like an ordinary waiting line. The first person in line is the first person served.

For example, if b is a FIFO list the cose

```
b.insert(4);
b.insert(7);
b.insert(3);
```

produces a list of size 2 that looks like

```
4  7  3
```

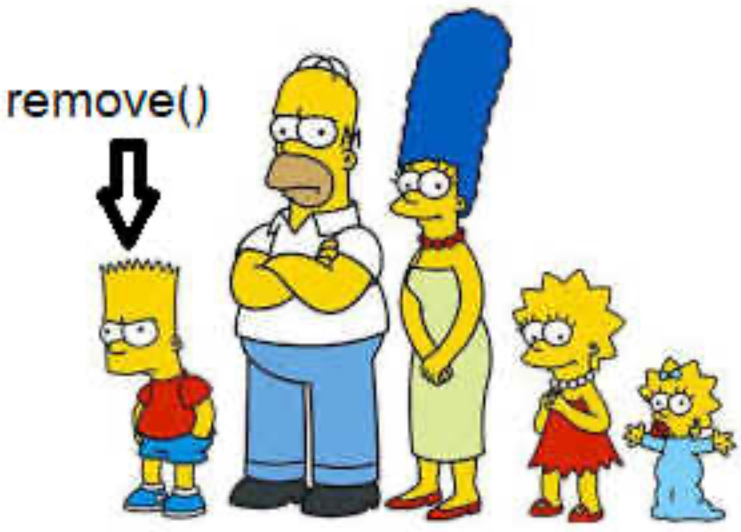
and the code

```
System.out.println(b.remove());
System.out.println(b.remove());
System.out.println(b.remove());
```

produces output

```
4
7
3
```

FIFO



FIFO -   First in   First Out
Bart was the first one in the line
and the first one to be removed

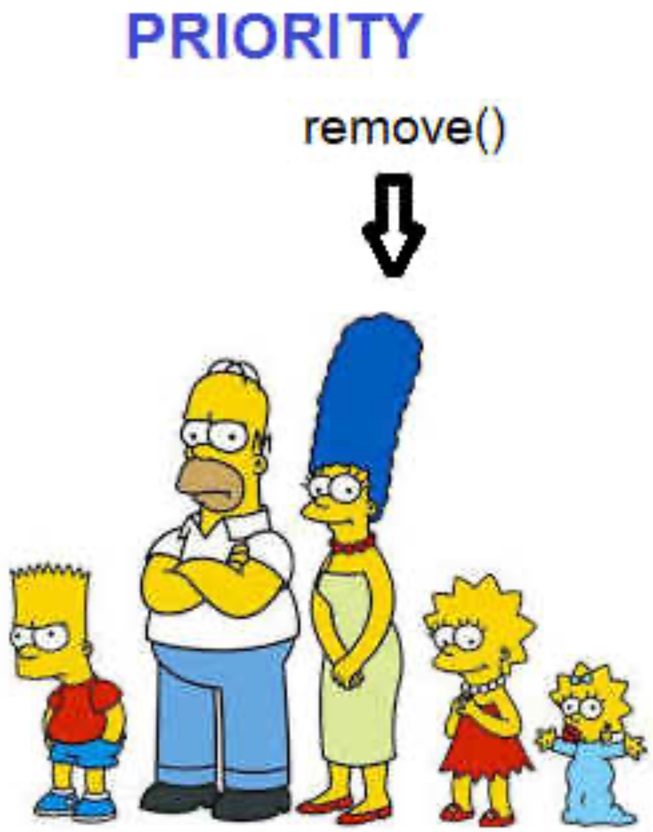
****

With a **(Maximum) PRIORITY** list a call to remove() removes and returns the item of highest priority. If the list is a list of integers, the remove() removes and returns the maximum integer.

For example, if c is an PRIORITY list the code

```
c.insert(4);
b.insert(7);
b.insert(3);
```

remove() removes and returns 7.



(maximum) Priority List
First one removed is the largest (Here the tallest)

Notice that each list has its own insert and remove method.

That is how the three lists work. The assignment is pretty much the same as before

1. Implement an abstract class Lists:

- Use **ArrayList<E>** to store Strings in the list and
- Let default constructor create an initial ArrayList with initial capacity 25
- Include abstract methods

i. int remove()

ii. void insert(int x)

iii. implement int getSize() // not abstract

– this just calls the ArrayList method size()

Unlike the array implementation, you do not have to keep track of size. The ArrayList class does that for you.

Now make **three classes**: LIFO, FIFO, and PRIORITY that implement the three types of list mentioned above. Each extends the abstract class Lists

LIFO:

Insertion and removal from the LIFO list is easy.

FIFO :

For the FIFO list, insert data at the end and remove the data at position 0.
Remember: ArrayList takes care of shifting data. That makes it easy

For example, if the list is

G	D	P	A	B	C	H													
0	1	2	3	4	5	6	...	...											

a call to remove() removes and returns "G" and shifts all the data {"to the left") so that the "D" is now in position 0..

D	P	A	B	C	5													
0	1	2	3	4	5	...	...											

(size is now 6)
This too is easy, if you use the ArrayList methods()

(There are more efficient ways to implement a FIFO list and we will see them later)

PRIORITY:

For the PRIORITY list, when inserting data x, just place x at the end of the list, as you did with the LIFO list.

DO NOT SORT THE LIST AFTER EVERY INSERTION.

To remove a value

1. search the list for the **MAXIMUM** value as well as its position, ***maxPosition***
   So in this case we assume “Z” > “A” or “M” > “H” etc

2. remove the value at ***maxPosition***

3. return the maximum value

So suppose the list is

G	D	**P**	A	B	C	H													
0	1	2	3	4	5	6	...	...											

The maximum is “P” in position 2. A call to remove() removes and returns the “I.”

G	D	**A**	**B**	**C**	**H8**														
0	1	2	3	4	5	6	...	...											

The value “I” has been removed, values “A” “B” “C” and “H” have been shifted and size is now 6

If any list is empty (i.e. size ==0), a call to remove() is obviously an error.
Throw and catch an Exception with the message “Empty List”

Test your List hierarchy with the following class:

```
public  class TestLists
{
   public static void main(String[] args)
   {
      LIFO s = new LIFO();
      System.out.println("LIFO: ");
      s.insert("B");
      s.insert("L");
```

```
s.insert("M");
s.insert("C");
System.out.println(s.remove());
System.out.println(s.remove());
s.insert("P");
s.insert("N");
System.out.println(s.remove());
System.out.println(s.remove());
s.insert("D");
System.out.println(s.remove());
System.out.println(s.remove());
System.out.println(s.remove());
System.out.println(s.remove());
FIFO q = new FIFO();
System.out.println("FIFO: ");
q.insert("B");
q.insert("L");
q.insert("M");
q.insert("C");
System.out.println(q.remove());
System.out.println(q.remove());
q.insert("P");
q.insert("N");
System.out.println(q.remove());
System.out.println(q.remove());
q.insert("D");
System.out.println(q.remove());
System.out.println(q.remove());
System.out.println(q.remove());
System.out.println(q.remove());
PRIORITY pq = new PRIORITY();
  pq.insert("B");
pq.insert("L");
pq.insert("M");
pq.insert("C");
System.out.println(pq.remove());
System.out.println(pq.remove());
pq.insert("P");
pq.insert("N");
System.out.println(pq.remove());
System.out.println(pq.remove());
pq.insert("D");
System.out.println(pq.remove());
System.out.println(pq.remove());
System.out.println(pq.remove());
System.out.println(pq.remove());
```

```
    }
}
```

The ArrayList methods

- add(x)
- add(index, x)
- remove(index)
- size()
- get(i)

should make this pretty easy

Program 3.

Make a class Shuffle.java with instance variable, list, an ArrayList of Integer.

The default constructor should initialize list with the numbers 1- 10, inclusive.

The one argument constructor
    public Shuffle(int n)
should initialize list with the numbers 1 – n, inclusive

Include methods
    public void displayList() that prints the contents of the list
    public void shuffle() that shuffles the list, as we did with a deck of cards

The shuffle method should shuffle the data **using the same algorithm** that we used to shuffle cards. The algorithm is in your notes.

So you will need a random number generator for the shuffle.

Remember that list.size() returns the size of the list. That may be important in the shuffle() method.

Include a main method that

1. Creates a list *list1* using the default constructor
2. Shuffles the list
3. Displays the shuffled list
4. Creates a list *list2* of size 15
5. Shuffles the list
6. Displays the shuffled list