(d) Consider the following statement: (5)

```
public class ABC<S extends Z & Y & X> {
}
```

What can you say about ABC, S, X, Y, and Z?

.....

ABC is a generic class which can operate on such type S which extends class Z, implements interface Y and X or implements interface Z, Y, X.

(b) Write a generic interface named iStack with methods push, pop and isEmpty. Then write a generic class Stack that implements the iStack interface. Please note that iStack interface only supports numeric types. (10

```
interface iStack<T extends Number>
{   void push(T d);
    T pop();
    boolean isEmpty();
}

class Stack<T extends Number>
        implements iStack<T>
```

```
{
    T [] data;
    int top;
  Stack (T [] data) {
      this.data = data;
    } top = -1;

 public void push (T d){
   data [++top] = d;
   }

 public T pop() {
    return data [top--];
   }

public boolean isEmpty()
    } return top == -1;
    }
}
```

(b) A "queue" is a data-structure in which a newly entered element is placed in the back of the queue; however, elements are served or going out of the queue from the front. Write down a generic class named "queue" where all types of data such as integer, character, string, double and object of any kind can be stored and queued. The queue will support following member functions: push(), pop(), size().

Write down the codes for the necessary exception handling during performing different operation on the queue.

```
class QFullException extens Exception{
    QFullException(String alert){
        super(alert);
    }
}
class QEmptyException extens Exception{
    QEmptyException(String alert){
        super(alert);
    }
}
interfac iQueue<T>{
```

```
    void push(T e) throws QFullException;
    T pop() throws QEmptyException;
    int size();
}
class queue<T> implements iQueue<T>
{
    T [] arr;
    int size;
    int top;
    int capacity;

queue(T [] arr) {
    this.arr = arr;
    capacity = arr.length;
    size = 0;
    top = -1;
}
```

```
int size() { return size; }
void push(T e) throws QFullException {
{ if (size == capacity)
throw new QFullException
("Queue is full");
arr[++top] = e; size++;
}

T pop() throws QEmptyException {
if (top == -1)
throw new QEmptyException
("Queue is Empty!");
```

```
T ret = arr[0];
for(int i=0; i<size-1; i++)
{
    arr[i] = arr[i+1];
}
size--;
return ret;
}
}
```

8(b). Write a generic interface named iQueue with methods enqueue, dequeue and isEmpty. Then write a generic class Queue that implements the iQueue interface. Please note that iQueue interface only supports numeric types. (10)

```
interface iQueue<T extends Number>
{
// Same
```

}

class Queue<T extends Number>
implements iQueue<T>

{

// same

}

(c) Write a generic interface named **iStack** with methods **push, pop** and **isEmpty**. Then write a generic class **Stack** that implements the **iStack** interface. **(10)**

same as before;
change <T extends number>
↳ <T>

(d) Consider the following statement.

```
Public class Gen<T extends X & Y & Z>{

}
```

What do you can say about T, X, Y, and Z?

type t extends class X, implements Y, Z

or

type t implements X, Y, Z.

(c) What is a functional interface? Using lambda expression, implement the functional interface shown in Figure 3 to compute factorial of an input integer.

(d) When we say "a class is final" or "a method is final" what does that mean? Show

```
interface SomeFunc<T> {
    T func(T t);
}
```

Figure 3: Functional interface for Question 5(c)

(c) Interface that has one abstract method also knows

as a functional method.

here SomeFunc is a functional interface & func is functional method.

```
SomeFunc<Integer>
  factorial = (n) -> {
          int result = 1;
for(int i = 1; i <= n; i++)
        result *= i;
  return result;
  }
```

given an integer n;

factorial.func(n); // compute factorial

// Handle IllegalArgument Exception