```
interface IntStack {
	void push(int item); // store an item
	int pop(); // retrieve an item
}


// An implementation of IntStack that uses fixed storage.
class FixedStack implements IntStack {
	private int stck[];
	private int tos;
	// allocate and initialize stack
	FixedStack(int size) {
		stck = new int[size];
		tos = -1;
	}
	// Push an item onto the stack
	public void push(int item) {
		if(tos==stck.length-1) // use length member
			System.out.println("Stack is full.");
		else
			stck[++tos] = item;
	}
```

```
    // Pop an item from the stack
    public int pop() {
        if(tos < 0) {
            System.out.println("Stack underflow.");
            return 0;
        }
        else
            return stck[tos--];
        }
    }


class IFTest {
    public static void main(String args[]) {
        FixedStack mystack1 = new FixedStack(5);
        FixedStack mystack2 = new FixedStack(8);
        // push some numbers onto the stack
        for(int i=0; i<5; i++) mystack1.push(i);
        for(int i=0; i<8; i++) mystack2.push(i);
        // pop those numbers off the stack
        System.out.println("Stack in mystack1:");
        for(int i=0; i<5; i++)
            System.out.println(mystack1.pop());
        System.out.println("Stack in mystack2:");
```

```
            for(int i=0; i<8; i++)
                System.out.println(mystack2.pop());
        }
}
```

Dynamic stack

```
class DynStack implements IntStack {
	private int stck[];
	private int tos;
	// allocate and initialize stack
	DynStack(int size) {
		stck = new int[size];
		tos = -1;
	}

	// Push an item onto the stack
	public void push(int item) {
	// if stack is full, allocate a larger stack
		if(tos==stck.length-1) {
			int temp[] = new int[stck.length * 2]; // double size
			for(int i=0; i<stck.length; i++) temp[i] = stck[i];
			stck = temp;
			stck[++tos] = item;
		}
		else
			stck[++tos] = item;
	}
```

```
        // Pop an item from the stack

        public int pop() {
                if(tos < 0) {
                        System.out.println("Stack underflow.");
                        return 0;
                }
                else
                        return stck[tos--];
        }
}
class IFTest2 {
public static void main(String args[]) {
        DynStack mystack1 = new DynStack(5);
        DynStack mystack2 = new DynStack(8);
        // these loops cause each stack to grow
        for(int i=0; i<12; i++) mystack1.push(i);
        for(int i=0; i<20; i++) mystack2.push(i);
        System.out.println("Stack in mystack1:");
        for(int i=0; i<12; i++)
                System.out.println(mystack1.pop());
        System.out.println("Stack in mystack2:");

        for(int i=0; i<20; i++)
                System.out.println(mystack2.pop());
```

```
    }
}
```

Class uses both Fixed stack and dynamic stack

```
/* Create an interface variable and access stacks through it.*/
class IFTest3 {
    public static void main(String args[]) {
        IntStack mystack; // create an interface reference variable
        DynStack ds = new DynStack(5);
        FixedStack fs = new FixedStack(8);

        mystack = ds; // load dynamic stack
            // push some numbers onto the stack
            for(int i=0; i<12; i++) mystack.push(i);

        mystack = fs; // load fixed stack
            for(int i=0; i<8; i++) mystack.push(i);

        mystack = ds;
            System.out.println("Values in dynamic stack:");
            for(int i=0; i<12; i++)
```

```
                System.out.println(mystack.pop());
        mystack = fs;
            System.out.println("Values in fixed stack:");
            for(int i=0; i<8; i++)
                System.out.println(mystack.pop());
    }
}
```