Create a package CIE which has two classes- Student and Internals. The class Student has
members like usn, name, sem. The class Internals has an array that stores the internal marks
scored in five courses of the current semester of the student. Create another package SEE
which has the class External which is a derived class of Student. This class has an array
that
stores the SEE marks scored in five courses of the current semester of the student. Import the
two packages in a file that declares the final marks of n students in all five courses.

```
package CIE;

public class Int {
   public int[] im = new int[5];

   public Int(int[] im) {
       System.arraycopy(im,0,this.im,0,im.length);
   }
}
```

```
package CIE;

public class Stu {
   public String usn, name;
   public int sem;

   public Stu(String usn, String name, int sem) {
       this.usn = usn;
       this.name = name;
       this.sem = sem;
   }
}
```

```
package SEE;

import CIE.Stu;

public class Ext extends Stu {
```

```
    public int[] sm = new int[5];

    public Ext(String usn, String name, int sem, int[] sm) {
        super(usn, name, sem);
        System.arraycopy(sm, 0, this.sm, 0, sm.length);
    }
}

import CIE.Int;
import SEE.Ext;

public class MAA {
    public static void main(String[] args) {
        int[] im = {80, 80, 80, 80, 80};
        int[] sm = {80, 80, 80, 80, 80};

        Ext ext = new Ext("1WA23CS015", "Tarun S", 5, sm);
        Int inter = new Int(im);

        System.out.println("Name: " + ext.name + ", USN: " + ext.usn + ", Sem: " +
ext.sem);

        System.out.println("\nInternal Marks:");
        for (int i = 0; i < 5; i++) {
            System.out.println("sub "+(i+1)+":"+ inter.im[i]);
        }

        System.out.println("\nSEE Marks:");
        for (int i = 0; i < 5; i++) {
            System.out.println("Sub " + (i + 1) + ": " + ext.sm[i]);
        }

        System.out.println("\nFinal Marks:");
        for (int i = 0; i < 5; i++) {
            int fm = (inter.im[i] + ext.sm[i])/2;
            System.out.println("Sub " + (i + 1) + ": " + fm);
        }
    }
}

Name: Tarun S, USN: 1WA23CS015, Sem: 3
```

```
Internal Marks:
sub 1:80
sub 2:80
sub 3:80
sub 4:80
sub 5:80

SEE Marks:
Sub 1: 80
Sub 2: 80
Sub 3: 80
Sub 4: 80
Sub 5: 80

Final Marks:
Sub 1: 80
Sub 2: 80
Sub 3: 80
Sub 4: 80
Sub 5: 80
```

3. Demonstrate Fixed and Dynamic Stack using Interfaces

```
interface FixedStack {
void push(int value);
int pop();
boolean isFull();
}
interface DynamicStack {
void push(int value);
int pop();
boolean isEmpty();
}
class FixedStackImpl implements FixedStack {
int[] stack;
int top;
int capacity;
public FixedStackImpl(int capacity) {
this.capacity = capacity;
this.stack = new int[capacity];
this.top = -1;
```

```
}
public void push(int value) {
if (isFull()) {
System.out.println(&amp;quot;Stack Overflow&amp;quot;);
} else {
stack[++top] = value;
}
}
public int pop() {
if (top == -1) {
System.out.println(&amp;quot;Stack Underflow&amp;quot;);
return -1;
} else {

return stack[top--];
}
}
public boolean isFull() {
return top == capacity - 1;
}
}
class DynamicStackImpl implements DynamicStack {
int[] stack;
int top;
int size;
public DynamicStackImpl() {
stack = new int[10];
top = -1;
size = 10;
}
public void push(int value) {
if (top == size - 1) {
resize();
}
stack[++top] = value;
}
public int pop() {
if (top == -1) {
System.out.println(&amp;quot;Stack Underflow&amp;quot;);
return -1;
} else {
return stack[top--];
```

```
}
}
public boolean isEmpty() {
return top == -1;
}
private void resize() {
size = size * 2;
int[] newStack = new int[size];
System.arraycopy(stack, 0, newStack, 0, stack.length);

stack = newStack;
}
}
Stack Overflow
3
2
1
Stack Underflow
```

4) Demonstrate Fixed and Dynamic Queues using Interfaces

```
interface FixedQueue {
   void enqueue(int value);
   int dequeue();
   boolean isFull();
   }
   interface DynamicQueue {
   void enqueue(int value);
   int dequeue();
   boolean isEmpty();
   }
   class FixedQueueImpl implements FixedQueue {
   int[] queue;
   int front, rear, capacity;
   public FixedQueueImpl(int capacity) {
   this.capacity = capacity;
   this.queue = new int[capacity];
   this.front = -1;
   this.rear = -1;
   }
```

```
public void enqueue(int value) {
if (isFull()) {
System.out.println(&amp;quot;Queue Overflow&amp;quot;);
} else {
if (front == -1) front = 0;
queue[++rear] = value;
}
}
public int dequeue() {
if (front == -1 || front &amp;gt; rear) {
System.out.println(&amp;quot;Queue Underflow&amp;quot;);
return -1;

} else {
return queue[front++];
}
}
public boolean isFull() {
return rear == capacity - 1;
}
}
class DynamicQueueImpl implements DynamicQueue {
int[] queue;
int front, rear, size;
public DynamicQueueImpl() {
queue = new int[10];
front = -1;
rear = -1;
size = 10;
}
public void enqueue(int value) {
if (rear == size - 1) {
resize();
}
if (front == -1) front = 0;
queue[++rear] = value;
}
public int dequeue() {
if (front == -1 || front &amp;gt; rear) {
System.out.println(&amp;quot;Queue Underflow&amp;quot;);
return -1;
} else {
```

```
   return queue[front++];
   }
   }
   public boolean isEmpty() {
   return front == -1 || front &gt; rear;
   }
   private void resize() {
   size = size * 2;

   int[] newQueue = new int[size];
   System.arraycopy(queue, 0, newQueue, 0, queue.length);
   queue = newQueue;
   }
   }
```

```
Queue Overflow
1
2
3
Queue Underflow
```

5) MyDataStructure implements both the SimpleStack and SimpleQueue interfaces. Recall that a class can implement more than one Interface. NOTE: Demonstrate Stack and Queues.

```
interface SimpleStack {
   void push(int value);
   int pop();
   }
   interface SimpleQueue {
   void enqueue(int value);
   int dequeue();
   }
   class MyDataStructure implements SimpleStack, SimpleQueue {
   int[] stackQueue;
   int top, front, rear, capacity;
   public MyDataStructure(int capacity) {
   this.capacity = capacity;
   stackQueue = new int[capacity];
   top = -1;
   front = -1;
   rear = -1;
```

```
}
public void push(int value) {
if (top == capacity - 1) {
resizeStack();
}
stackQueue[++top] = value;
}
public int pop() {
if (top == -1) {
System.out.println(&amp;quot;Stack Underflow&amp;quot;);
return -1;
} else {
return stackQueue[top--];

}
}
public void enqueue(int value) {
if (rear == capacity - 1) {
resizeQueue();
}
if (front == -1) front = 0;
stackQueue[++rear] = value;
}
public int dequeue() {
if (front == -1 || front &amp;gt; rear) {
System.out.println(&amp;quot;Queue Underflow&amp;quot;);
return -1;
} else {
return stackQueue[front++];
}
}
private void resizeStack() {
capacity = capacity * 2;
int[] newStackQueue = new int[capacity];
System.arraycopy(stackQueue, 0, newStackQueue, 0, stackQueue.length);
stackQueue = newStackQueue;
}
private void resizeQueue() {
capacity = capacity * 2;
int[] newStackQueue = new int[capacity];
System.arraycopy(stackQueue, 0, newStackQueue, 0, stackQueue.length);
stackQueue = newStackQueue;
```

```
   }
   }
30
20
10
Stack Underflow

40
50
60
Queue Underflow

300
200
```