Linear Queue:

```
import java.util.Arrays;

public class LinearQueue {
	private int maxSize, front, rear;
	private int[] queue;

	LinearQueue(int size) {
		maxSize = size;
		queue = new int[maxSize];
		front = rear = -1;
	}

	boolean isFull() {
		if (rear == maxSize - 1) {
			return true;
		} else {
			return false;
		}
	}

	boolean isEmpty() {
		if (front == rear) {
			return true;
		} else {
			return false;
		}
	}

	public void add(int x) {
		if (isFull()) {
			System.out.println("Queue Overflow");
			System.exit(-1);
		}
		queue[++rear] = x;
	}

	void delete() {
		if (isEmpty()) {
			System.out.println("Queue Underflow");
			System.exit(-1);
		}
		int x = queue[++front];
		System.out.println("Deleting: " + x);
```

```
    }

    void displayQueue() {
        System.out.println(Arrays.toString(queue));
    }

    public static void main(String[] args) {
        LinearQueue myQueue = new LinearQueue(5);

        myQueue.add(5);
        myQueue.add(4);
        myQueue.add(3);
        myQueue.add(2);
        myQueue.add(1);

        myQueue.displayQueue();

        myQueue.delete();
        myQueue.delete();
        myQueue.delete();
        myQueue.delete();
        myQueue.delete();
    }
}
```

Circular Queue:

```
import java.util.Arrays;

public class CircularQueue {
    private int front;
    private int rear;
    private final int maxSize;
    private final int[] queue;

    CircularQueue(int size) {
        maxSize = size;
        queue = new int[maxSize];
        front = rear = maxSize - 1;
    }

    void add(int x) {
        int k = (rear + 1) % maxSize;
        if (front == k) {
            System.out.println("Queue Overflow");
```

```
                System.exit(-1);
            } else {
                queue[k] = x;
                rear = k;
            }
        }

        void delete() {
            if (front == rear) {
                System.out.println("Queue Underflow");
                System.exit(-1);
            }
            front = (front + 1) % maxSize;
            System.out.println("Deleting: " + queue[front]);
            queue[front] = 0;
        }

        void displayQueue() {
            System.out.println(Arrays.toString(queue));
        }

        public static void main(String[] args) {
            CircularQueue myQueue = new CircularQueue(5);
            myQueue.add(5);
            myQueue.add(1);
            myQueue.add(6);
            myQueue.add(2);

            myQueue.displayQueue();

            myQueue.delete();
            myQueue.delete();
            myQueue.delete();

            myQueue.add(5);
            myQueue.add(1);
            myQueue.add(11);
            myQueue.delete();
            myQueue.delete();
            myQueue.delete();
            myQueue.displayQueue();

        }
}
```

Double-Ended Queue:

```java
import java.util.Arrays;

public class DeQueue {
	private int front, rear, maxSize;
	private int[] queue;

	DeQueue(int size) {
		maxSize = size;
		queue = new int[maxSize];
		front = rear = -1;
	}

	 void enQueueFront(int x) {
		if ((front == 0 && rear == maxSize - 1) || (front == rear + 1)) {
			System.out.println("Queue Overflow");
			System.exit(-1);
		} else if (front == -1 && rear == -1) {
			front = rear = 0;
			queue[front] = x;
		} else if (front == 0) {
			front = maxSize - 1;
			queue[front] = x;
		} else {
			--front;
			queue[front] = x;
		}
	}

	void enQueueRear(int x) {
		if ((front == 0 && rear == maxSize - 1) || (front == rear + 1)) {
			System.out.println("Queue Overflow");
			System.exit(-1);
		} else if (front == -1 && rear == -1) {
			front = rear = 0;
			queue[rear] = x;
		} else if (rear == maxSize - 1) {
			rear = 0;
			queue[rear] = x;
		} else {
			++rear;
			queue[rear] = x;
		}
	}

	void deQueueFront() {
		if (front == -1 && rear == -1) {
			System.out.println("Queue Underflow");
```

```
                System.exit(-1);
            } else if (front == rear) {
                System.out.println(queue[front]);
                front = rear = -1;
            } else if (front == maxSize - 1) {
                System.out.println(queue[front]);
                front = 0;
            } else {
                System.out.println(queue[front]);
                ++front;
            }
        }

        void deQueueRear() {
            if (front == -1 && rear == -1) {
                System.out.println("Queue Underflow");
                System.exit(-1);
            } else if (front == rear) {
                System.out.println(queue[rear]);
                front = rear = -1;
            } else if (rear == 0) {
                System.out.println(queue[rear]);
                rear = maxSize - 1;
            } else {
                System.out.println(queue[rear]);
                --rear;
            }
        }

        void displayQueue() {
            System.out.println(Arrays.toString(queue));
        }

        public static void main(String[] args) {
            DeQueue myQueue = new DeQueue(5);
            myQueue.enQueueRear(2);
            myQueue.enQueueFront(5);
            myQueue.enQueueRear(-1);
            myQueue.enQueueRear(0);
            myQueue.enQueueFront(7);
            //myQueue.enQueueFront(4);
            myQueue.displayQueue();
            myQueue.deQueueFront();
            myQueue.deQueueRear();
            myQueue.deQueueFront();
            myQueue.deQueueRear();
            myQueue.deQueueFront();
        }
    }
```

Priority Queue:

```
import java.util.Arrays;

public class PriorityQueue {
	private int[] queue;
	private int maxSize;
	private int n = 0;

	PriorityQueue(int size) {
		maxSize = size;
		queue = new int[maxSize];
	}

	void enQueue(int item) {

		if (n == maxSize) {
			System.out.println("Queue Overflow");
			System.exit(-1);
		}

		int i = n - 1;
		while (i >= 0 && item < queue[i]) {
			queue[i + 1] = queue[i];
			--i;
		}
		queue[i + 1] = item;
		++n;
	}

	void deQueue() {
		int item;
		if (n == 0) {
			System.out.println("Queue Underflow");
			System.exit(-1);
		}
		item = queue[n - 1];
		n = n - 1;
		System.out.println(item);
	}

	void displayQueue() {
		System.out.println(Arrays.toString(queue));
	}
```

```
    public static void main(String[] args) {
        PriorityQueue myQueue = new PriorityQueue(5);
        myQueue.enQueue(5);
        myQueue.enQueue(4);
        myQueue.enQueue(10);
        myQueue.enQueue(2);
        myQueue.enQueue(1);

        myQueue.displayQueue();

        myQueue.deQueue();
        myQueue.deQueue();
        myQueue.deQueue();
        myQueue.deQueue();
        myQueue.deQueue();

    }
}
```

Simple Stack:

```
import java.util.Arrays;

public class SimpleStack {
    private int[] stack;
    private final int maxSize;
    private int top;

    SimpleStack(int size) {
        maxSize = size;
        stack = new int[maxSize];
        top = -1;
    }

    boolean isFull() {
        if (top == maxSize - 1) {
            return true;
        } else {
            return false;
        }
    }

    boolean isEmpty() {
        if (top == -1) {
            return true;
```

```
        } else {
            return false;
        }
    }

    void push(int x) {
        if (isFull()) {
            System.out.println("Stack Overflow");
            System.exit(-1);
        } else {
            stack[++top] = x;
        }
    }

    void delete() {
        if (isEmpty()) {
            System.out.println("Stack Underflow");
            System.exit(-1);
        }
        System.out.println("Deleting: " + stack[top]);
        --top;
    }

    void displayStack() {
        System.out.println(Arrays.toString(stack));
    }

    public static void main(String[] args) {
        SimpleStack myStack = new SimpleStack(5);
        myStack.push(5);
        myStack.push(4);
        myStack.push(3);
        myStack.push(2);
        myStack.push(1);

        myStack.displayStack();

        myStack.delete();
        myStack.delete();
        myStack.delete();
        myStack.delete();
        myStack.delete();
    }
}
```