Exceptions are an essential way to deal with errors in Java, most commonly, you only have to deal with the simple case of an ADT throwing some exception that must be caught by the client. You have to remember a couple of things:

1. You define an exception as a class extending Exception; it can have data members, or not (the name is often enough);

```java
public class ResizingQueue<Item> implements Queue<Item> {
    ..........
    public Item dequeue() throws QueueUnderflowException {
        if( isEmpty() )
            throw new QueueUnderflowException();
        ..........
    }
}

class QueueUnderflowException extends Exception {
    // put any data here you want, or nothing!
}

interface Queue<Item> {
    void enqueue(Item item);
    Item dequeue() throws QueueUnderflowException;
    boolean isEmpty(); int size();
}
```
Exceptions are an essential way to deal with errors in Java, most commonly, you only have to deal with the simple case of an ADT throwing some exception that must be caught by the client. You have to remember a couple of things:

1. You define an exception as a class extending Exception; it can have members, or not (the name is often enough);
2. You throw an exception when you encounter the condition/error by calling the constructor for the exception;

```java
class QueueUnderflowException extends Exception {
}
```

```java
interface Queue<Item> {
    void enqueue(Item item);
    Item dequeue() throws QueueUnderflowException;
    boolean isEmpty();   int size();
}
```

```java
public class ResizingQueue<Item> implements Queue<Item> {
    ..........
    public Item dequeue() throws QueueUnderflowException {
        if( isEmpty() )
            throw new QueueUnderflowException();
    ........
}
```
**Exceptions** are an essential way to deal with errors in Java, most commonly, you only have to deal with the simple case of an ADT throwing some exception that must be caught by the client. You have to remember a couple of things:

1. You define an exception as a class extending `Exception`; it can have members, or not (the name is often enough);
2. You throw an exception when you encounter the condition/error;
3. Any call to that method must be inside a try block which catches that exception (or a superclass, such as just `Exception`)

```java
public class ResizingQueue<Item> implements Queue<Item> {
    
    public Item dequeue() throws QueueUnderflowException {
        if( isEmpty() )
            throw new QueueUnderflowException();  
.......
    }

    class QueueUnderflowException extends Exception {
    }

    interface Queue<Item> {
       void enqueue(Item item);
       Item dequeue() throws QueueUnderflowException;
       boolean isEmpty();  int size();

    }

    try {
        
        .......
        int n = Q.dequeue();
        
    }
    catch ( QueueUnderflowException e ) {
        System.out.println("Q underflew!");
    }
```
Exceptions are an essential way to deal with errors in Java, most commonly, you only have to deal with the simple case of an ADT throwing some exception that must be caught by the client. You have to remember a couple of things:

1. You define an exception as a class extending Exception; it can have members, or not (the name is often enough);
2. You throw an exception when you encounter the condition/error;
3. Any call to that method must be inside a try block which catches that exception (or a superclass, such as just Exception);
4. The header of the method must list all exceptions that it throws (also in the interface).

```java
public class ResizingQueue<Item> implements Queue<Item> {
    ..........       
    public Item dequeue() throws QueueUnderflowException 
    {
        if( isEmpty() )
            throw new QueueUnderflowException();  .......... 
    }
}

class QueueUnderflowException extends Exception {
}

interface Queue<Item> {
    void enqueue(Item item);
    Item dequeue() throws QueueUnderflowException;
    boolean isEmpty();  int size();
}
```

```java
try {
    ......
    int n = Q.dequeue();
    ......
}
catch ( QueueUnderflowException e ) {
    System.out.println("Q underflew!");
}
```
public class ResizingQueue<Item> implements Queue<Item> {
        ........

public Item dequeue() throws QueueUnderflowException {
        if( isEmpty() )
            throw new QueueUnderflowException("Q Underflew!");
}
}}

class QueueUnderflowException extends Exception {
        public String text;
        public QueueUnderflowException(String text) {
            this.text = text;
        }
    }

    // in the client code:
    try {
        int n = Q.dequeue();
    } catch ( QueueUnderflowException e ) {
        System.out.println(e.text);
    }