CS 112: Introduction to Last Lab

Computer Science

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);

```
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();
}
```

CS 112: Introduction to Last Lab

Computer Science

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;

```
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();
```

CS 112: Introduction to Last Lab

Computer Science

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)

```
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).

```
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!");
}
```

CS 112: Introduction to Last Lab

Computer Science

```
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);
}
```

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).
5. You can put anything in an exception class that you put any other class, except that usually you only put data relevant to the error condition.
6. You can use a constructor to initialize these data items, and then retrieve them when the exception is caught; in this way you can pass useful information about the error back to the client.
7. Think of the catch as a method which will be called when the exception occurs, and you can pass a parameter (the exception instance) and run code based on the exception's data.