

### Choices

- **when calling a function that may throw an exception**
  - Catch and handle the exception.
  - Catch the exception, then re-throw it or throw another exception.
  - Ignore the exception (let it "pass up" the call chain).

---

---

---

---

---

---

---

### Guideline

- If your code *throws a checked exception, or your code ignores a checked exception that might be thrown by a called function*
- *then your function should include a throws clause listing all such exceptions.*
  
- *Only uncaught checked exceptions need to be listed in a function's throws clause.*

---

---

---

---

---

---

---

### Example

- `public void doDa() throws FileNotFoundException, EOFException`
- `{ // an uncaught FileNotFoundException or EOFException may be thrown here }`

---

---

---

---

---

---

---

## Custom Exception Classes

Create a custom *unchecked* exception class simply by extending `RuntimeException` and providing two constructors. Everything else you need is inherited from `Throwable`!

```
package gray.adts.shapes;

/**
 * The exception that is thrown whenever an operation on a
 * shape is in violation of a method pre-condition.
 */
public class ShapeException extends RuntimeException {

    public ShapeException() {
        super();
    }

    public ShapeException( String errMsg ) {
        super(" " + errMsg );
    }
}
```

---

---

---

---

---

---

---

---

## Best Practices for Designing the API

- **deciding on checked exceptions vs. unchecked exceptions**
  - ask yourself, "What action can the client code take when the exception occurs?"
  - If the client can take some alternate action to recover from the exception, make it a checked exception.
  - If the client cannot do anything useful, then make the exception unchecked.

---

---

---

---

---

---

---

---

## Best Practices for Designing the API

- **Preserve encapsulation.**
  - Never let implementation-specific checked exceptions escalate to the higher layers.
  - Convert into another checked exception, if the client code is expected to recuperate from the exception.
  - Convert into an unchecked exception, if the client code cannot do anything about it.

---

---

---

---

---

---

---

---

**Best Practices for Designing the API**

- Try not to create new custom exceptions if they do not have useful information for client code.

---

---

---

---

---

---

---

---

**Best Practices for Designing the API**

- Document exceptions.

---

---

---

---

---

---

---

---

**Best Practices for Using Exceptions**

- Always clean up after yourself
- Never use exceptions for flow control
- Do not suppress or ignore exceptions
- Do not catch top-level exceptions

---

---

---

---

---

---

---

---