Exception Handling

To do exception handling, we enclose the code that could raise an exception in a begin-end block and use one or more rescue clauses to tell Ruby the types of exceptions we want to handle.

```ruby
def raise_exception_and_rescue
  begin
    puts 'Called before exception is raised'
    raise 'An exception is raised'
    puts 'Called after exception is raised'
  rescue
    puts 'rescue clause'
  end
  puts 'Executed after begin-end clause'
end
```

```ruby
raise_exception_and_rescue
```

```
Called before exception is raised
rescue clause
Executed after begin-end clause
=> nil
```

If you write a rescue clause with no parameter list, the parameter defaults to **StandardError**. Each rescue clause can specify multiple exceptions to catch. At the end of each rescue clause you can give Ruby the name of a local variable to receive the matched exception.

```ruby
begin
  # -
rescue Type1Exception
  # -
rescue Type2Exception
  # -
rescue Exception # Parent Exception

end
```

For each rescue clause in the begin block, Ruby compares the raised Exception against each of the parameters in turn. The match will succeed if the exception named in the rescue clause is the same as the type of the currently thrown exception.

```ruby
begin
  raise ArgumentError, 'A test exception.'
rescue ArgumentError => e
  puts e.message
  puts e.backtrace.inspect
rescue IOError => e
  puts e.message
  puts e.backtrace.inspect
rescue Exception => e
```

```
  puts e.message
  puts e.backtrace.inspect
end
```

The **message** method returns a string that may provide human-readable details about what went wrong. The other important method is **backtrace**. This method returns an array of strings that represent the call stack at the point that the exception was raised.

If you need the guarantee that some processing is done at the end of a block of code, regardless of whether an exception was raised then the ensure clause can be used.

```
begin
  # something which might raise an exception
rescue SomeExceptionClass => some_variable
  # code that deals with some exception
rescue SomeOtherException => some_other_variable
  # code that deals with some other exception
else
  # code that runs only if *no* exception was raised
ensure
  # ensure that this code always runs, no matter what
```

```
end
```