Catching and throwing an exception further

Earlier, we wrote function mod.

```
/** = the value r that satisfies x = q*y + r   and  0 <= r < abs(y) for some q.
  *    Throw an ArithmeticException if y = 0. */
 public static int mod(int x, int y) {
     if (y == 0) throw new ArithmeticException("mod(x, 0) is undefined");
     int r= x % Math.abs(y);
     return  r >= 0 ? r : y + r;

}
```

The main purpose of the explicit test for y = 0 was so that we could throw an exception with a particular detail message. However, there is another way to get the same result. We remove the if-statement and place the whole body in a try statement that catches ArithmeticExceptions. Then, in the catch-block, we throw a new exception with our desired message.

```
/** = the value r that satisfies x = q*y + r   and  0 <= r < abs(y) for some q.
 *    Throw an ArithmeticException if y = 0. */
public static int mod(int x, int y) {
     try {
          int r= x % Math.abs(y);
          return  r >= 0 ? r : y + r;
     }
     catch (ArithmeticException ae) {
          throw new ArithmeticException((("mod(x, 0) is undefined");
     }
}
```

The new method body does not rethrow object ae; it creates a new object and throws it. This is because it is not possible to change the detail message of a throwable object.

But there are cases where rethrowing ae makes sense. For example, one might catch the exception only to dispose of some resources —which is beyond the scope of CS2110— and then rethrow the same exception.

This second way of detecting that y is 0 is more in keeping with the exception-mechanism philosophy. Rather than intersperse lots of tests for errors, which might double the size of the code, let the exception-handling facilities do that work. Of course, in this case, this second way yields a longer program, but in general, using the exception-handling facilities can help.