Try/Success/Failure

A cleaner way to handle exceptions is to use Try/Success/Failure. If the code succeeds, we return a Success object with the result, and if it fails, we pass the error in a Failure object.

Let's implement divideByZero with Success/Failure:

```
def divideWithTry(dividend: Int, divisor: Int): Try[Int] = Try(divide(dividend, divisor))
```

When we call divideWithTry, we get a Failure object that contains the original error:

```
assert(divideWithTry(10, 0) == Failure(new DivideByZero))
```

Callers of divideWithTry can pattern match using Success and Failure objects, like so:

```
val result = divideWithTry(10, 0) match {
  case Success(i) => i
  case Failure(DivideByZero()) => None
}
```