Translation of For

For-Expressions and Higher-Order Functions

- The syntax of for is closely related to the higher-order functions map, flatMap and filter
- First of all, these functions can all be defined in terms of for:

```
def mapFun[T, U](xs: List[T], f: T => U): List[U] =
  for (x <- xs) yield f(x)

def flatMap[T, U](xs: List[T], f: T => Iterable[U]): List[U] =
  for (x <- xs; y <- f(x)) yield y

def filter[T](xs: List[T], p: T => Boolean): List[T] =
  for (x <- xs if p(x)) yield x
```

Translation of For

- In reality, the Scala compiler expresses for-expressions in terms of map, flatMap and a lazy variant of filter.
- Here is the translation scheme used by the compiler (we limit ourselves here to simple variables in generators)

1. A simple for-expression ***for (x <- e1) yield e2*** is translated to ***e1.map(x => e2)***
2. A for-expression ***for (x <- e1 if f; s) yield e2*** where f is a filter and s is a (potentially empty) sequence of generators and filters, is translated to:

   for (x <- e1.withFilter(x => f); s) yield e2

   In fact with filter is a lazy variant of filter, which means it doesn't immediately produce a new data structure of all the filtered element; instead, it remembers that any following call to map or flatMap has to be filtered by the function f.
3. A for-expression ***for (x <- e1; y <- e2; s) yield e3*** where s is a (potentially empty) sequence of generators and filters, is translated into

   e1.flatMap(x => for (y <- e2; s) yield e3)

   We use the flatMap function because we need the results of the generators concatenated into a result list.

Example: *Take the for-expression that computed pairs whose sum is prime:*

```
for {
  i <- 1 until n
  j <- 1 until i
  if isPrime(i + j)
} yield (i, j)
```

Applying the translation scheme to this expression gives:

```
(1 until n).flatMap(i =>
  (1 until i).withFilter(j => isPrime(i+j))
  .map(j => (i, j)))
```

Exercise: *Translate* ***for (b <- books; a <- b.authors if a startsWith "Bird") yield b.title*** *into higher-order functions.*
Step 1: ***books.flatMap(b => for (a <- b.authors if a.startsWith("Bird")) yield b.title)***
Step 2: ***books.flatMap(b => for(a <- b.authors.withFilter(a => a.startsWith("Bird") yield b.title)***
Step 3: ***books.flatMap(b => b.authors.withFilter(a => a.startsWith("Bird").map(x => x.title))***

Generalization of For

- Interestingly, the translation of for is not limited to lists or sequences, or even collections
- It is based solely on the presence of the methods map, flatMap and withFilter
- This lets you use the for syntax for your own types as well – you must only define map, flatMap and withFilter for these types
- There are many types for which this is useful: arrays, iterators, databases, XML data, optional values, parsers, etc.

For and Databases

- For example, books might not be a list, but a database stored on some server
- As long as the client interface to the database defines the methods map, flatMap and withFilter, we can use the for syntax for querying the database
- This is the basis of the Scala data base connection frameworks ScalaQuery and Slick. Similar ideas underly Microsoft's LINQ.