Concepts of Programming Languages
Lecture Notes: Option Types

Stefan Mitsch

School of Computing, DePaul University
smitsch@depaul.edu

Learning Goals

🎓 Identify uses of option types

Option Types

Option types are a principled approach to missing data. Option[T] resembles List[T] with a length of at most 1. Java, for a long time, emphasized programming with exceptions in order to report anything that deviates from a successful result.

```
1 // the Java way
2 def getDirs1 (dirName : String) : List[java.io.File] =
3     val dir = new java.io.File (dirName)
4     val xs = dir.listFiles
5     if xs == null then throw new java.io.FileNotFoundException
6     xs.nn.toList.map (_.nn).filter (_.isDirectory)
```

Exceptions, however, are best used to report exceptional circumstances (such as a broken network connection); missing data is quite an expected result. Programming with optionals allows us to document missing data and allows our users to appropriately react to it (as opposed to the Java habit of just passing on exceptions).

```
1 def getDirs2 (dirName : String) : Option[List[java.io.File]] =
2     val dir = new java.io.File (dirName)
3     val xs = dir.listFiles
4     if xs == null then return None
5     Some(xs.nn.toList.map (_.nn).filter (_.isDirectory))
```

With optionals, clients no longer suffer from the convolutional way of providing a result from multiple execution traces. An option is a type that may have some result or nothing. Scala knows several option types:

- None is the empty option
- Nil is the empty list
- null is a reference to nothing

Unit is not an option type, it only has a single value (always has nothing). Even though Scala has null, we often pretend it does not exist. More recent languages, such as Swift and Kotlin, identify None and null; these languages distinguish nullable and non-nullable types. Java also includes an optional, but its intended use is narrowed to library methods whose return types needed a clear way of communicating the absence of a result and null is overwhelmingly likely to cause errors.