COMP 313/413 Lecture Notes

Release 1.0

Konstantin Läufer

September 02, 2015

Lecture notes for the mostly Scala and Android-based course COMP 313/413: Intermediate Object-Oriented Programming at Loyola University Chicago's Computer Science Department. This version of the course is normally taught by Konstantin Läufer.

Warning: These notes are still being written, so expect a few rough edges. But we're getting closer!

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Introduction

In these lecture notes, we will study intermediate object-oriented development topics from various angles, including design principles and patterns, software architecture, and concurrency.

Please stay tuned for more content!

1.2 Basics of Object-Oriented Programming

The topics discussed in this chapter are considered the basics of object-oriented programming. Some of them are language-independent, others are specific to C#, Java, or Scala. These topics, except perhaps for the last two, are supposed to be covered by the CS 1/2 prerequisite chain for this course.

1.2.1 Reference semantics vs. value semantics

- *Value semantics:* variables directly contain values.
- *Reference semantics:* variables contain addresses that refer (point) to objects.

Examples

- misc/IdentityAndEquality.java
- misc/Comparing.scala

References

- Java
- C# (see also *Effective C#* item 6)

1.2.2 Equality vs. identity

- *Equality:* are two (possibly) different objects equivalent?
- *Identity:* do two references refer to the same objects?
- How are equality and identity related?

- Reconciling value and reference semantics: identity of objects explained as equality of addresses.

Examples

- misc/IdentityAndEquality.java
- misc/Comparing.scala

References

- Java (see also *Effective Java* items 8, 9; see also here)
- C# (see also *Effective C#* items 6, 26)

1.2.3 Parametric polymorphism (generics)

Familiar from data structures course:

- without generics: `Stack` (of objects); loose typing
- without generics: `IntStack`, `StringStack`, `BookStack`; strict typing but lots of duplicate boilerplate code
- with generics: `Stack<Int>`, `Stack<String>`, `Stack<Book>`; strict typing without code duplication

Relatively easy to use, can be challenging to incorporate correctly in the design of one's abstractions.

Examples

- misc/Comparing.java
- misc/Comparing.scala

References

- Java (*Effective Java* chapter 5: items 23-29; see also this tutorial and here for advanced issues)
- C#
- Scala

1.2.4 Relationships among classes and interfaces

These are common relationships among concepts and are part of UML's class diagrams.

Class/interface-level relationships

These relationships are between classes and/or interfaces, so they *cannot* change at run time.

From strong to weak:

- *is-a*, *realizes-a*: generalization/specialization (subtyping)
- *uses-a*: dependency

Instance-level relationships

These relationships are between instances, so they *can* change at run time.

From strong to weak:

- *owns-a*: composition
- *part-of*: aggregation
- *has-a* or other specific relationship: association

Examples

- misc/Animals.java
- misc/Animals.scala
- Figure *A UML class diagram representing a taxonomy of vehicles.*

Fig. 1.1: A UML class diagram representing a taxonomy of vehicles.

1.2.5 Class-interface continuum

- *Concrete class* (C++, C#, Java, Scala): can be instantiated. All specified methods are fully implemented.
- *Abstract class* (C++, C#, Java, Scala): cannot be instantiated. Some or all of the specified methods are not implemented. A class cannot extend more than one abstract class.
- *Trait* (Scala only): cannot be instantiated directly. Some or all of the specified methods are not implemented. A class or trait can extend zero or more traits, and member lookup is automatically disambiguated based on trait order (see traits and mixins for details).
- *Interface* (Java, C# only): limit case of a fully abstract class for specification purposes only. None of the specified methods are implemented, and there are no instance variables.

Related to the single-responsibility and interface-segregation principles.

Examples

- misc/Animals.java
- misc/Animals.scala

References

- Java (see also *Effective Java* items 18, 19)
- C# (see also *Effective C#* items 22, 23)
- Scala

1.2.6 Subtyping vs. subclassing/inheritance

- Subtyping allows substituting a more specific object for a more general one, for example, when passed as an argument or assigned to a variable.
- Inheritance is a mechanism for a subclass to reuse state and behavior from a superclass.
  - inherit methods and fields
  - add fields
  - add or replace/refine methods
- Inheriting from a superclass enables weak syntactic subtyping. (In some languages, this relationship can be public or nonpublic.)
- The Liskov Substitution Principle (LSP) defines strong semantic (behavioral) subtyping.
- Implementing or extending an interface also enables syntactic subtyping (and semantic subtyping because interfaces have no behavior). Extending a trait also enables syntactic subtyping.

Examples

- misc/Animals.java
- misc/Animals.scala

References

- Java (see also *Effective Java* item 17; see also these pitfalls)
- C# (see also *Effective C#* item 22)

1.2.7 Subtype polymorphism: static vs. dynamic type

- *Static type:* declared type of a variable.
- *Dynamic type:* actual type of the object to which the variable refers.
- *Dynamic method binding:* `x.f(a1, a2, ...)`. Two steps:
  1. Verify whether receiver x supports method f based on static type.
  2. Search for version of f to be invoked starting from dynamic type and proceeding upward until found.
- How are static and dynamic type of a variable related?
- If step 1 succeeds, will step 2 always succeed as well?
- *Casting:* treat an object as if it had a different static type. Three different situations: - *downcast* - *upcast* - *crosscast*
- Overloading versus overriding. - `@Override`/`override` correctness in Java and Scala

Examples

- misc/MethodBinding.java
- misc/InterfaceCast.java

- misc/Super.java
- misc/Super2.java
- misc/MethodBinding.scala
- misc/InterfaceCast.scala

References

- Java (see also *Effective Java* item 52)
- C# (see also *Effective C#* item 3)

1.2.8 Being a good descendant of java.lang.Object/System.Object

Classes are usually required to provide the following methods (these specific ones are for Java):

- `toString` (for displaying instances in a meaningful way)
- `equals` (if an instance can be in an equivalence class that include other instances)
- `hashCode` (ditto)
- `compareTo` (if instances are ordered)
- `clone` (if instances are mutable)
- `finalize` (if instances need to release resources)

Also related to the Liskov substitution principle.

Examples

- misc/IdentityAndEquality.java
- misc/Comparing.java
- misc/Comparing.scala

References

- Java (see also *Effective Java* items 8 through 12; see also here for equals, below for clone; detailed Javadoc is here)
- C# (see also *Effective C#* items 5, 9, 10, 27)

1.2.9 Clone in the context of the Composite pattern

In general, cloning allows you to make a copy of an object. The clone method in Java is similar to the copy constructor in C++, but it is an ordinary method, unlike the copy constructor. Once you have the original object and its clone, then you can modify each one independently. Accordingly, cloning is necessary only if the objects are mutable.

Cloning models the real-life situation where you build a prototype of something, say a car or a piece of furniture, and once you like it, you clone it as many times as you want. These things are composites, and the need to be cloned deeply (recursively).

As another example, imagine a parking garage with a list of cars that have access to it. To build another garage to handle the growing demand, you can clone the garage and the customer access list. But the (physical) cars should not get cloned. That's because the garage is not composed of the cars.

As we can see, the conceptual distinction between aggregation and composition has significant consequences for the implementation of the relationship. True, both relationships are represented as references in Java. However, composites usually require a deep clone (if cloning is supported) where each parent is responsible for cloning its own state and recursively cloning its children.

As mentioned above, you don't need to support cloning at all if your objects are immutable because you wouldn't be able to distinguish the original from the clone anyway.

References

- Java (see also *Effective Java* item 11; see also here for more detail)
- C# (see also *Effective C#* items 14, 27)

1.2.10 Packages/namespaces

- Mechanism for grouping related or collaborating classes (cf. default package-level member access).
- In Java, implemented as mapping from fully qualified class names to file system. In Scala, this is much looser.

Examples

- misc/Outer.java

References

- Java (see also here)
- C#

1.2.11 Member access

- public
- protected
- default (package)
- private

Related to the information hiding and open-closed principles.

References

- Java (see also *Effective Java* items 13, 14, 15; see also here)
- C# (see also *Effective C# item 1)

1.3 Overview of a Lightweight Development Process

A successful development process usually comprises these minimal elements:

- automated regression testing
- refactoring
- continuous integration

1.4 Android App Development Using Scala

In this chapter, we discuss the tools and processes for developing Android apps using Scala in detail.

1.4.1 Prerequisites

Required Development Tools

- Java Development Kit (JDK) 6 or higher through your package management system or from Oracle; to verify, visit this site and, if necessary, download Java from the same place. (On a Mac, be sure to do this in Safari.)

  Warning: Check specific prerequisite details for your platform; in particular, on the Mac, a Java 6 JDK is required to run IDEA, though you can (and should) use a Java 8 JDK as development target.

  Warning: On Windows, it is usually best to install the JDK in a location that does not contain spaces.

- Android SDK
- Simple Build Tool (sbt)
- android-sdk-plugin
  - *this is included in each of the code examples, so no explicit installation is required*
  - detailed *usage instructions* are half way down past the change log
- Git distributed version control system (DVCS)

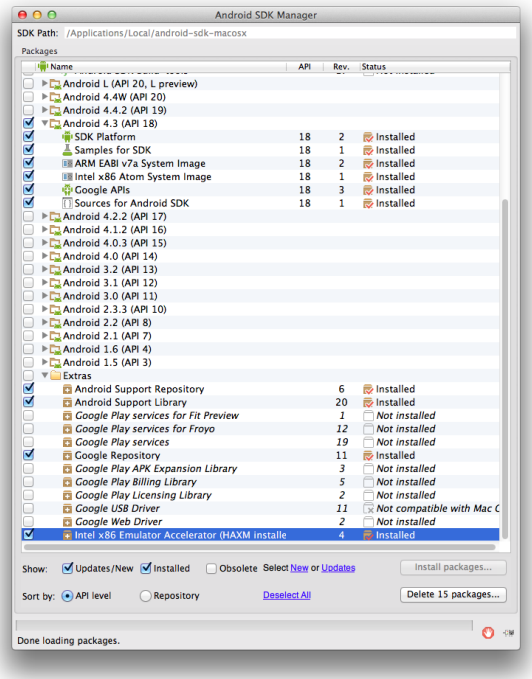
Recommended Tools

- JetBrains IntelliJ IDEA CE (the latest stable version is currently 13.1)
- IDEA Scala plugin installed through the IntelliJ IDEA plugin manager
  - *installation:* `IntelliJ IDEA > Preferences > Plugins > Browse repositories`, then find and right-click `Scala`, choose `download and install`, close repository browser, and hit `OK` to restart IDEA
  - *documentation:* Scala development in IDEA
- IDEA SBT plugin installed through the IntelliJ IDEA plugin manager
  - *installation:* `IntelliJ IDEA > Preferences > Plugins > Browse repositories`, then find and right-click `SBT`, choose `download and install`, close repository browser, and hit `OK` to restart IDEA

  - *documentation:* idea-sbt-plugin

Preparation

- Download a *standalone* Android SDK.
  1. visit http://developer.android.com/sdk/index.html
  2. expand "sdk for existing ide"
  3. press download button
  4. unzip to a suitable location, e.g., `/Applications/Local/android-sdk-macosx`, which we will refer to as `ANDROID_HOME`
- Set up the Android SDK and download SDK components.
  1. run the Android SDK manager, `$ANDROID_HOME/tools/android`
  2. check Android SDK platform tools and build tools version 20, 19.1.0, 19.0.1
  3. check Android 4.3 (API 18) (the API 19 emulator has a bug related to screen rotation)
  4. check Android support repository and Google repository (see also here)
  5. check HAXM (hardware acceleration for the emulator)
  6. on Windows, check USB driver
  7. install selected packages

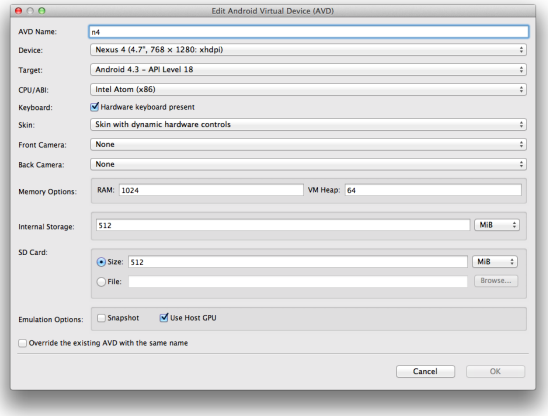


- Configure virtual machine hardware acceleration per these instructions.
  - on Windows or Mac, look in `$ANDROID_HOME/extras/intel`
  - on Linux, follow these instructions
- Create an Android Virtual Device (AVD) of your choice (perhaps resembling your phone) per these instructions.

  - you can run the AVD manager from the command line:

```
$ $ANDROID_HOME/tools/android avd
```

  - the virtual device should support API level 18 (Android 4.3) and have an x86 CPU, a skin with hardware controls, and the option *hardware keyboard present* checked



  - to run the Android emulator from the command line, where n4 is the name you chose for the virtual device you just created:

```
$ $ANDROID_HOME/tools/emulator @n4 &
```

- If you have an Android device and wish to use it for development, you can follow these instructions to enable it.
- Fork the desired project from this collection, then clone it to your local workstation using Git.

1.4.2 Developing on the Command Line

We recommend globally and persistently setting `$ANDROID_HOME` (Windows instructions, Mac OS X instructions) as discussed below and putting `$ANDROID_HOME/tools` and `$ANDROID_HOME/platform-tools` in the `$PATH`. These instructions assume that you have done this; if not, you can still invoke the `adb` and `emulator` commands by specifying their full paths.

Specifying the location of the Android SDK

You can either

- set `$ANDROID_HOME` to the directory where you installed your Android SDK *(recommended)*
- create a file `local.properties` in your project root (or copy an existing one) with a single line

```
sdk.dir=/location/of/android/sdk
```

You need to do this step for each project you clone to your local workstation.

Starting the emulator

To start the emulator:

```
$ emulator @YourAVD &
```

It will take the emulator a couple of minutes to boot to your AVD's home or lock screen. If you set up hardware acceleration correctly, you will see

```
HAX is working and emulator runs in fast virt mode
```

To verify that you have a connection with the emulator:

```
$ adb devices
```

The resulting list should look like this:

```
List of devices attached
emulator-5554   device
```

If this is not the case, restart the adb server

```
$ adb kill-server
$ adb start-server
```

and check again.

Viewing the log

In Android, all log messages typically carry a tag. In this example, the tag for the main activity is

```
private def TAG = "hello-android-activity"
```

You can then write tagged log messages like this:

```
Log.i(TAG, "onCreate")
```

You can view the complete log using this command:

```
$ adb logcat
```

This quickly results in too much information. To view only the messages pertaining, say, to a particular tag, you can filter by that tag:

```
$ adb logcat | grep hello
```

Running the application

Once your emulator is running or device connected, you can install and run the app:

```
$ sbt clean android:run
```

The app should now start in the emulator and you should be able to interact with it. *(Cleaning before running ensures that the app gets installed properly on the emulator.)*

> **Warning:** If you get this error `Unsupported class version number [52.0] (maximum 51.0, Java 1.7)))` (usually near the top of a long stack trace), make sure you are *not* using Java 8.

Running the tests

This command runs the unit tests and the Robolectric-based out-of-container functional tests.

```
$ sbt test
```

Warning: In-container Android instrumentation tests are included in some of the examples (sharing a testcase superclass with the Robolectric tests) and work in principle, but not with the current build file for reasons we do not yet understand. We will rely on the Robolectric-based tests instead.

Starting from scratch

We have not been able to get pfn's gen-android task to work even though we tried with a global installation of the plugin.

In addition to the usual sbt directory structure, the key ingredients are

- `build.sbt` like in our examples
  - set project name and version as desired
  - review the library dependencies, e.g., choose between Mockito and ScalaMock
  - review the Proguard options
- `project/plugins.sbt` containing

```
addSbtPlugin("com.hanhuy.sbt" % "android-sdk-plugin" % "1.3.5")

addSbtPlugin("com.hanhuy.sbt" % "sbt-idea" % "1.7.0-SNAPSHOT")

resolvers += Resolver.sbtPluginRepo("snapshots")
```

  (The blank lines are required.)

- `project/build.scala` containing

```
object Build extends android.AutoBuild
```

For details, please refer to the android-sdk-plugin documentation.

1.4.3 Developing with IntelliJ IDEA

Configuring IntelliJ IDEA

It is convenient to configure the required SDKs at the global (IDE) level before working on new or existing projects.

- configure the Java SDK at the global (IDE) level using these instructions (you can go through the initial dialog or use `Command ;` on the Mac to open the project structure dialog directly)
- repeat these steps for the Android SDK

Generating the configuration files

This step requires that you have the `sbt-idea` plugin installed per the instructions for pfn's plugin.

```
$ sbt gen-idea
```

You will have to repeat this step after every change to the `build.sbt` or `AndroidManifest.xml` files (see also under "adding dependencies" below.

Opening the project in IDEA

Open *(not import)* the project through the initial dialog or `File > Open`. You should now be able to edit the project with proper syntax-directed editing and code completion.

Right after opening the project, you may be asked to confirm the location of the Android manifest file.

If you ever get a popup saying that this is an sbt-based project and offering to import it, choose ignore.

Running the tests and the application

Some aspects of generated IDEA Android/Scala project do not work out of the box. We have found it easier to open a terminal within IDEA using `View > Tool Windows > Terminal` and running `sbt test` or `sbt android:run` as desired. In the latter case, the app should start in the emulator and you should be able to interact with it.

Integrating IDEA and sbt

For a faster edit-build-run cycle, though, you will want to perform the IntelliJ IDEA integration described in the *Advanced Usage* section of pfn's android-sdk-plugin documentation. This requires IDEA with both the Scala *and* SBT plugins mentioned above.

In our experience, this integration requires the following adjustments *on a per-project basis*:

- edit the default runtime configuration for Android Application to invoke `sbt android:package` instead of `Make`
- edit the default runtime configuration for ScalaTest to invoke `sbt test:products` instead of `Make`

1.4.4 Adding build dependencies

To add a dependency, you can usually

- look it up by name in the Central Repository or MVNrepository
- find the desired version (usually the latest released or stable version)
- select the sbt tab
- copy the portion _after_ `libraryDependencies +=`
- paste it into this section of `build.sbt` (followed by a comma)

```
    libraryDependencies ++= Seq(
```

If you are using IntelliJ IDEA, you will also need to

- rerun

```
    $ sbt gen-idea
```

- back in IDEA, confirm that you want to reload the project
- reconfirm the location of the Android manifest file

1.4.5 Optional Tools

For Windows users

- TortoiseHg (integration of Mercurial with Windows Explorer)
- Ubuntu in a virtual machine (consider this option if you are a Windows user and have trouble getting things to work)

For Windows and Mac users

- SourceTree is a GUI client for Mercurial and Git

For all users

- Genymotion emulator and IDEA plugin
- These are useful additional Android Studio/Intellij IDEA plugins. (Installation procedure is the same as for the Scala plugin.)
  - Code Outline 2
  - Key Promoter (helps you learn keyboard shortcuts)
  - Markdown
  - Scala import organizer
- These are useful additional sbt plugins. You can install them per project or globally.
  - sbt-scoverage: uses Scoverage to produce a test code coverage report
  - ls-sbt: browse available libraries on GitHub using ls.implicit.ly
  - sbt-dependency-graph: creates a visual representation of library dependency tree
  - sbt-updates: checks central repos for dependency updates
  - cpd4sbt: copy/paste detection for Scala *(be sure to set* `cpdSkipDuplicateFiles := true` *in Android projects to avoid a false positive for each source file)*
  - scalastyle: static code checker for Scala
  - sbt-stats: simple, extensible source code statistics/metrics
  - sbt-scalariform: automatic source code formatting using Scalariform

1.4.6 Tips

- IntelliJ IDEA has a built-in native terminal for your OS. This allows you to use, say, hg or sbt conveniently without leaving IDEA.:

```
View > Tool Windows > Terminal
```

- To practice Scala in a light-weight, exploratory way, you can use Scala worksheets in IntelliJ IDEA. These will give you an interactive, console-like environment, but your work is saved and can be put under version control.:

```
File > New > Scala Worksheet
```

You can even make it test-driven by sprinkling assertions throughout your worksheet!

1.5 Summary

In these notes, we studied intermediate object-oriented development topics from various angles, including design principles and patterns, software architecture, and concurrency.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search