Android Development Immersive

Jay Hiza

Table Of Contents

Android Development Intensive

Jay Hiza (2016)

Chapter 1 - Workflow and Developer Tools

Chapter 2 - Programming Fundamentals in Java

Chapter 3 - User Interface

Chapter 4 - Databases

Title	type	duration	creator		competencies
			name	city	
Chapter 1:1	lesson	1:30	Gerry Mathe	London	Workflow

Terminal Basics + Navigating the Filesystem

Objectives

After this lesson, students will be able to:

- Summarize a basic filesystem structure, including absolute and relative paths
- Use the most common commands to navigate and modify files / directories via the Terminal window (`cd`, `pwd`, `mkdir`, `rm -r`, `mv`, `cp`, `touch`)
- Describe a basic UNIX permissions system
- Differentiate between navigating the file system using the CLI vs. the GUI

Preparation

Before this lesson, students should already be able to:

- Open the terminal
- Issue one or more commands on the command line
- Be comfortable navigating between folders on the command line
- Take a look at some simple keyboard shortcuts to practice: CLI Shortcuts

> Note: Much of this content will be review from the pre-work students completed before the course began. We'll be practicing and reviewing some of the things they've learned, and diving into some additional ways to customize their command line.

Opening: What is a GUI (pronounced gooey) (5 mins)

There was a point when computers didn't come with a **G**raphical **U**ser **I**nterface (GUI). Instead, everyone interacted with the computer using text commands in what we call a **C**ommand **L**ine **I**nterface (CLI).

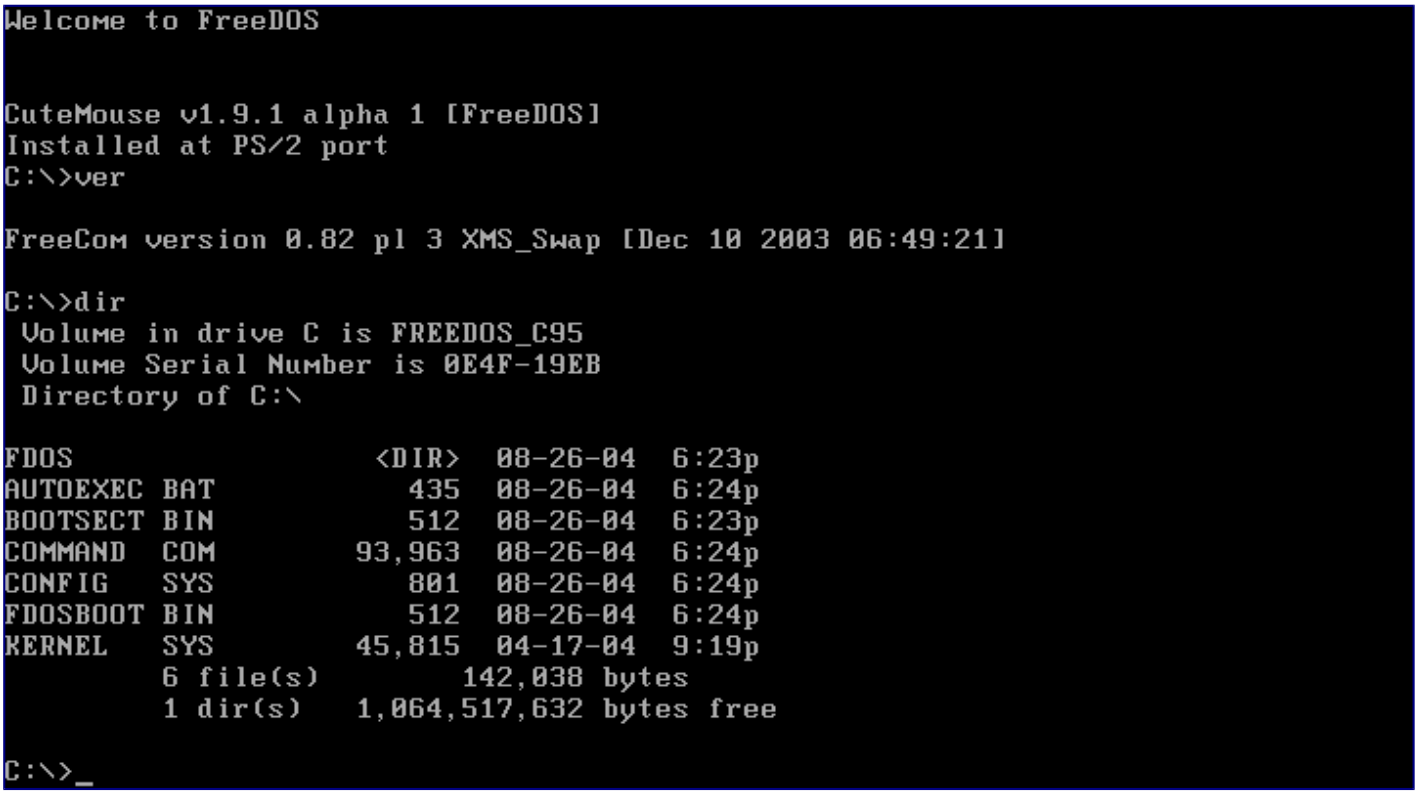


Image from Wikimedia

Today, the command line still exists, even though you may have never seen it as a casual computer user. In this course, we are going to spend a lot of time in the command line, and we will use it every day to manage our files and tell our computer how to run our programs. It will greatly speed up our development process and help us take ownership of our computer at a deeper level. There is so much that your computer will do for you if you know how to speak its language!

What is a shell?

A shell is simply a type of command line program, which contains a very simple, text-based user interface enabling us to access all of an operating system's services. It is, very simply, a program that accepts text as input and translates that text into the appropriate functions that you want your computer to run.

Taken from Just for fun: Type like a hacker

Demo: Forget Finder, get fast at using your laptop (25 mins)

LESSON GUIDE

TIMING	TYPE	TOPIC
pronounced gooey) (5 min	Opening	Discuss lesson objectives
25 min	Demo	Forget Finder, get fast at using your laptop
5 min	Independent Practice	Look around your OS

TIMING	TYPE	TOPIC
20 min	Demo	Creating, copying, and removing files and folders
15 min	Introduction	UNIX permissions and Chmod
15 min	Demo	Customize The terminal
5 min	Conclusion	Review / Recap

Opening & Closing Terminal

First, we need to launch the command prompt. We do this by using spotlight:

- ⌘ (Command) + Space
- "*Te*rminal"
- Enter

Notice that you can actually hit enter as soon as the field autocompletes. Get used to taking shortcuts – don't type the whole word out if you don't have to and avoid using your mouse if you can open or use an app with just keyboard shortcuts. It may seem harder now, but when you get used to it, it will save you literally hours of cumulative time.

Getting Comfortable

1. For many programs, you can open multiple tabs by pressing **⌘-T**.
    - Try it in your terminal!
2. You can close the current tab or window with **⌘-W**. This goes for most applications on a Mac.
    - Try *that* in your terminal!
3. If you have a process running, you can quit it by pressing **Ctrl-C**. Let's try that now.

    - At the command line, type `ping 127.0.0.1`. This basically sends a message to your own computer asking if it's awake.
    - Notice that it will keep pinging, even if you type something.
    - To stop the currently-running script, press **Ctrl-C**.
4. To quit the command line altogether, you can press **⌘-Q**.

Paths

Every file or folder in a file system can be read, written, and deleted by referencing its position inside the filesystem. When we talk about the position of a file or a folder in a file system, we refer to its "path". There are a couple of different kinds of paths we can use to refer to a file – **absolute paths** and **relative paths**.

Directory is an important term that's used interchangeably with *folder*. Though they are not exactly the same thing, when we say "navigate to your project directory" think of this as "navigate to your project folder". Here's a little more information:

Strictly speaking, there is a difference between a directory which is a file system concept, and the graphical user interface metaphor that is used to represent it (a folder)...If one is referring to a container of documents,

the term folder is more appropriate. The term directory refers to the way a structured list of document files and folders is stored on the computer. It is comparable to a telephone directory that contains lists of names, numbers and addresses and does not contain the actual documents themselves.

Taken from Close-To-Open Cache Consistency in the Linux NFS Client

What is an absolute path?

An absolute path is defined as the specific location of a file or folder from the root directory, typically shown as `/`. The root directory is the starting point from which all other folders are defined and is not normally the same as your **Home** directory, which is normally found at `/Users/[Your Username]`.

Working with UNIX commands and file paths

Typing `cd` - a command for "change directory" with no parameters takes us to our home directory.

```
cd
```

If we type in `pwd` - a command for "print working directory" from that folder, we can see where we are in relation to the root directory.

Some examples of absolute path:

```
/usr/local/bin/git
/etc/example.ext
/var/data/database.db
```

Notice, all these paths started from `/` directory which is a root directory for every Linux/Unix machines.

What is a relative path?

> *Note:* Instructors – customize this lesson to demo with folders that you have present on your filesystem, or alternatively create a similar file structure on the fly.

A relative path is a reference to a file or folder **relative** to the current position, or the present working directory(pwd).

Navigating through the command prompt

The tilde `~` character is an alias to your home directory. Use it to quickly return home.

```
cd ~/
```

Another shortcut, if we want to go up one directory from where we currently are, we can use `..`

```
cd ..
```

If we are in the folder `~/a/b/` and we want to open the file that has the absolute path `~/a/b/c/file.txt`, we can just type:

```
open c/file.txt
```

or

```
open ./c/file.txt
```

At any time, we can also use the absolute path, by adding a slash to the beginning of the path. The absolute path is the same for a file or a folder regardless of the current working directory, but relative paths are different, depending on what directory we are in. Directory structures are laid out like `directory/subdirectory/subsubdirectory`.

Below are some examples of using relative and absolute path for the same action:

1. My present location is `~/command-line-lesson/ADI/labs` and now I want to change directory to `/ADI`.
    - Using relative path: `cd ..`
    - Using absolute path: `cd ~/command-line-lesson/ADI`
2. My present location is `~/command-line-lesson/ADI/labs` and I want to change the location to `/ADI/lessons`
    - Using relative path: `cd ../labs`
    - Using absolute path: `cd ~/command-line-lesson/ADI/lessons`

By the way, your terminal is located in:

```
/Applications/Utilities/Terminal.app
```

Pressing **Up** scrolls through previously entered commands.

What if you're wondering what's in a directory? Typing `ls` and hitting enter is like asking the computer the question "what stuff is in this directory?" It stands for `list directory contents`. Try it in your current directory and share what you see!

Independent Practice: Look around your OS (5 mins)

Practice the UNIX commands we just learned with the bullets below:

- Use the `cd` command to go to your home folder.
- Use the `ls` command see what is in your home directory.
- Use the `cd [directory]` command to go into any folder that you spot.
- In the above command, replace `[directory]` with the directory name you intend to move to.
- Use the `ls` command to see what files and directories exist there.
- Use the `cd` command to go to your home folder.

Demo: Creating, copying, and removing files and folders (20 mins)

What if we want to create files and folders? The command `mkdir directory-name` creates a new directory with the name “directory-name”.

Try it out. From your current directory, `ls` to look around and then `mkdir my-project` to create a new directory with the name “my-project”. This new directory will be created within the current parent directory. `ls` again to see it's there.

From the same directory, to create a file, `touch file-name` creates a new file with the name “file-name”. Try it out - `touch my-file` creates a new file with the name “my-file”. Again, this new file will be created within the current parent directory. You can also use different file paths to create files within directories:

```
touch my-project/this-new-file.txt
```

Then if you `ls my-project` you'll notice the `this-new-file.txt` is within.

Copying files and folders? No problem - `cp file new-file1` creates a copy of the “file” and calls it “new-file” in your current directory. If you're looking to copy directories you'll have to pass in a `-r`, which stands for "recursive" - to copy the directory and everything inside of it:

```
cp -r my-project my-project-copy
```

Create a copy of the entire "my-project” directory and call it “my-project-copy". You could always chain a file path onto the second argument and create the copy elsewhere:

```
cp -r my-project my-project-copy/copy-within-a-copy
```

How about removing files and folders? Well, we talked about that earlier, but let's practice - `cd` to your root directory and do the following:

```
touch some-file.txt
mkdir some-directory
touch some-directory/inner-file.txt
ls
```

Now, you should see a `some-file.txt` and a `some-directory` in your root directory. But let's get rid of them:

```
rm some-file.txt
rm -rf some-directory
```

The first command removes the file `junk-file.txt`; the second removes the directory `junk-directory` and everything inside of it - `inner-junk-file.txt` - because we passed the `-r`, which stands for "recursive". You could also accomplish this with `-rf` - the `f` stands for force but use with caution and make sure you are in the right place. You can imagine how bad the results could be if you did that to your home folder!

Modifying multiple files at the same time

Now, if you're making a whole bunch of files/folders, `mkdir` `rm` and `touch` can be used to create and remove more than one file/directory at the same time.

Try it out. First, `ls` to see what you have in there and then:

- `mkdir directory01 directory02 directory03`
- `touch file01 file02 file03`
- `rm file01 file03`

Thinking about this command with relative and absolute paths:

1. If my present location is `~/command-line-lesson/a/b/`, and I am want to remove `~/command-line-lesson/a/b/folder/file.txt` file

    - Using relative path: `rm folder/file.txt`
    - Using absolute path: `rm ~/command-line-lesson/a/b/folder/file.txt`
2. If my present location is `~/command-line-lesson/ADI/labs`, and I want to remove a `a.txt` file located in this directory

    - Using relative path: `rm a.txt`
    - Using absolute path: `rm ~/command-line-lesson/ADI/labs/a.txt`

Finally, we can rename and move files and folders with this syntax:

```
mv file-name file-name2
```

The first argument is the file or folder being moved or renamed, and the second argument is the directory destination you can use to also rename the file/folder if you want. For example, from your root directory:

```
mkdir this-folder that-folder
touch this-file.txt that-file.txt
mv this-file.txt this-folder
mv that-file.txt that-folder/that-file.txt
```

Notice now, `this-folder` has a `this-file.txt` within it and `that-folder` has a `that-file.txt`.

A few other helpful commands you can try on your own:

Command	Explanation
`ls -a`	Lists all items in current directory including hidden files
`ls -l`	Gives a long list of items in current directory including permissions

Introduction: UNIX permissions and Chmod (15 mins)

Note: Update the narrative to reflect your own OS.

An OS is meant to serve many users, A user may correspond to a real-world person, but also a program that acts as a specific user. In my laptop OS, I am "bob" and with bob goes a set of permissions and restrictions on all files and folders, But I can also act for some specific program as the user "www" which corresponds to the privileges necessary to operate a local web server. Every User on the OS has a User ID - the name "bob" or "www" is just an alias for a User ID.

Users can be organized in groups. A user may be in one or several groups. A group will have the same set of permissions for every user assigned to this group.

Every file has an owner user and an owner group. So, for any file in the system, user 'bob' may have one of the following ownership relations:

- bob owns the file, i.e. the file's owner is 'bob'.

- bob is a member of the group that owns the file, i.e. the file's owner group is 'ADI'.
- bob is neither the owner, nor belonging to the group that owns the file.

Every file on the system has associated with it a set of permissions. Permissions tell UNIX what can be done with that file and by whom. There are three things you can do with a given file:

- read it.
- write it.
- execute it.

Permissions for a file will specify which of the three actions above can be performed for groups an users in an OS.

For every file, there is 3 types of permissions: permissions for the owner, for the group owner and for everyone else. Remember that there are three possibles actions a user can take on a file - read (r), write (w), execute (x) - so for each of the three permissions we need three actions.

If you change the present working directory to the root (`cd /`) and then show the folder content with the details by typing `ls -l` on the command prompt, you will get something like this:

```
 0 drwxrwxr-x+ 93 root  admin  3162 Aug 15 19:37 Applications
 0 drwxr-xr-x+ 66 root  wheel  2244 Jan 18  2015 Library
 0 drwxr-xr-x@  2 root  wheel    68 Sep  9  2014 Network
 0 drwxr-xr-x+  4 root  wheel   136 Jan 18  2015 System
 8 lrwxr-xr-x   1 root  wheel    49 Apr 14  2014 User Information ->
/Library/Documentation/User Information.localized
 0 drwxr-xr-x   6 root  admin   204 Jan 18  2015 Users
 0 drwxrwxrwt@  5 root  admin   170 Aug 15 16:35 Volumes
 0 drwxr-xr-x@ 39 root  wheel  1326 Apr 28 21:05 bin
 0 drwxrwxr-t@  2 root  admin    68 Sep  9  2014 cores
 0 drwxr-xr-x   3 root  wheel   102 Sep  8  2014 data
10 dr-xr-xr-x   3 root  wheel  4805 Aug 14 11:45 dev
 8 lrwxr-xr-x@  1 root  wheel    11 Jan 18  2015 etc -> private/etc
 2 dr-xr-xr-x   2 root  wheel     1 Aug 15 19:42 home
 2 dr-xr-xr-x   2 root  wheel     1 Aug 15 19:42 net
 0 drwxr-xr-x@  6 root  wheel   204 Jan 18  2015 private
 0 drwxr-xr-x@ 59 root  wheel  2006 Apr 28 21:05 sbin
 8 lrwxr-xr-x@  1 root  wheel    11 Jan 18  2015 tmp -> private/tmp
 0 drwxr-xr-x@ 12 root  wheel   408 Jan 18  2015 usr
 8 lrwxr-xr-x@  1 root  wheel    11 Jan 18  2015 var -> private/var
```

The second column corresponds to the permissions details for each file/folder. For the first line:

```
drwxrwxr-x+
```

We're not going to talk about the `d` letter at the start or the `+` at the end of each line for the moment. Just pay attention to the nine letters between.

The important part in this set of characters is:

```
rwxrwxr-x
```

This can be read:

```
rwx rwx r-x
```

This breaks down like this:

- first group of three letters corresponds to the owner permission,
- the second group of three letters corresponds to the owner group permission,
- the last group of three letters corresponds to others permission,

chmod

To modify a file's permissions you need to use the `chmod` command.

Only the owner of a file may use chmod to alter a file's permissions.

chmod has the following syntax:

```
chmod [options] mode file(s)
```

The 'mode' part specifies the new permissions for the file(s) that follow as arguments. A mode specifies which user's permissions should be changed, and afterwards which access types should be changed. Let's say for example:

```
chmod a-x file.txt
```

This means that the execute bit should be cleared (-) for all users - owner, group and the rest of the world. The permissions start with a letter specifying what users should be affected by the change, and this might be any of the following:

- u the owner user
- g the owner group
- o others (neither u, nor g)
- a all users

You might have encountered things like `chmod 755 a_file.txt`.

You can change the entire permission pattern of a file in one go using one number like the one in this example. Every mode has a corresponding code number, and as we shall see, there is a very simple way to figure out what number corresponds to any mode.

Every single digit in the triplet corresponds to the level of authorization for a group (user, group and others). Every digit is the addition of the rights for this group, and every level of permission corresponds to a number:

- 4 for r.
- 2 for w.
- 1 for x.

So if a file has rwxr-xr-x permissions we do the following calculation:

- Triplet for u: rwx => 4 + 2 + 1 = *7*
- Triplet for g: r-x => 4 + 0 + 1 = *5*
- Triplet for o: r-x => 4 + 0 + 1 = *5* Which makes : 755

So, 755 in UNIX permissions means 'I don't mind if other people read or run this file, but only I should be able to modify it' while 777 means 'everyone has full access to this file'

For practice, what would I do if I only wanted a file to be executable, but not readable or writeable?

Independent Practice: Get comfortable with chmod

Take 5 minutes to do the following:

1. Create a new directory.
2. Create 3 files in that directory.
3. Make one file readable, writeable, and executable by everyone
4. Make the second file readable by everyone, executable by the user and group, and writeable by the user
5. Make the third file readable and executable by the user and group, and only writeable by the user.

Conclusion (5 mins)

We will use the command line several hours every day, because is makes all files and folders manipulations more easy. A lot of software programs that we will use during the course also only have a CLI interface and can only be used with commands. Always remember that every action you'll do in a GUI can be done in the CLI.

1:2

Git and GitHub Intro

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Explain basic git commands like init, add, commit, push, pull and clone
- Distinguish between local and remote repositories
- Create, copy, and delete repositories locally, or on GitHub
- Fork and clone remote repositories

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Install a text editor (Sublime or Atom works well for this one!)
- Use the command line
- Use a text editor

Introduction: Git vs GitHub and version control (10 mins)

As developers, we often have to make changes to many files, each of which can have a major impact on the project. In order to keep track of these changes, we use version control systems. There are many different version control systems; we will be concentrating on Git.

> Describe Subversion

What is Git?

Git is:

- A program you run from the command line
- A distributed version control system

Programmers use Git so that they can keep the history of all the changes to their code. This means that they can rollback changes (or switch to older versions) as far back in time as they started using Git on their project.

You know how Google Docs allow you to have a "version history" and move between different versions whenever you want? Git enables you to do that with any folder - and its contents - on your computer!

A codebase in Git is referred to as a **repository**, or **repo**, for short.

Git was created by Linus Torvalds, the principal developer of Linux.

What is GitHub?

GitHub is:

- A hosting service for Git repositories
- A web interface to explore Git repositories
- A social network of programmers
- We all have individual accounts and put our codebases on our GitHub account
- You can follow users and star your favorite projects
- Developers can access codebases on other public accounts
- GitHub uses Git

Can you use git without GitHub?

Git is a software that enables version control on local folders on your machine. GitHub is a place to host your Git repositories, remotely. You can certainly have local files, that are using Git, that are not sent to or stored on GitHub.

Read this when you have time!

Demo: Why is Git tricky to understand? (15 mins)

Git is tricky to understand because describing 'how' it works would require the use of strange and technical-sounding words like:

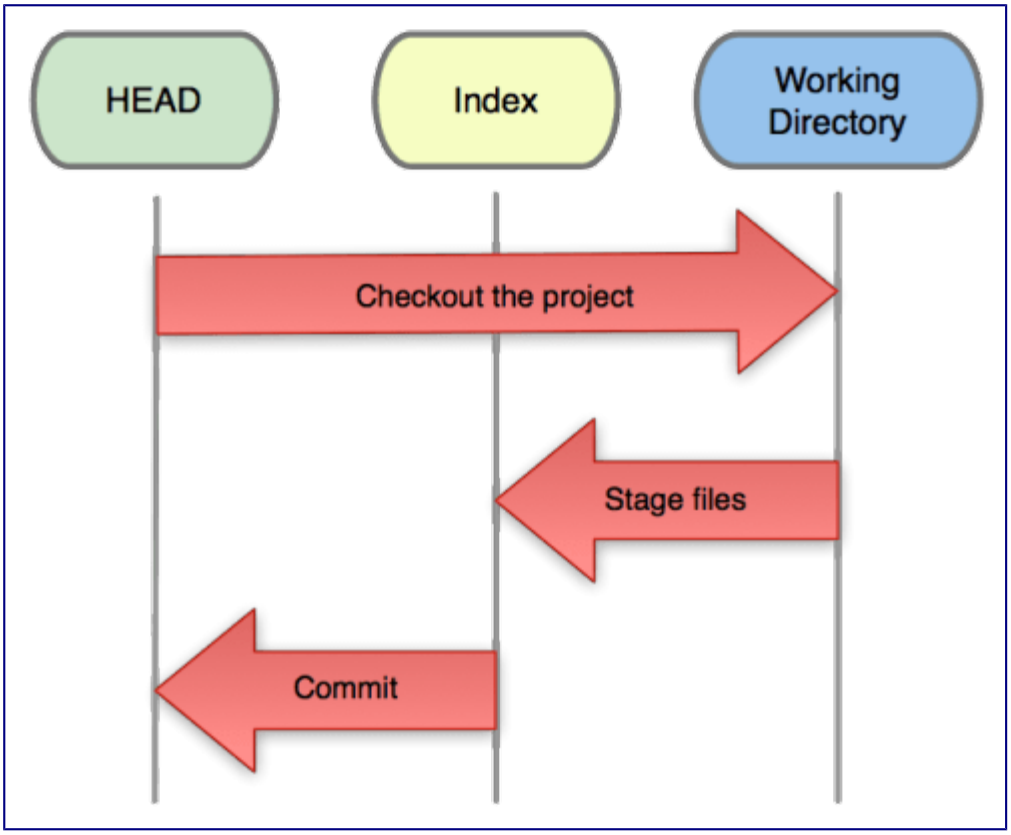
- Directed acyclic graph
- SHA-1
- blob
- tree

However, you don't actually need to know how it works under the hood in order to use it.

Trees?!

Even though you don't need to know how they work, it is useful to know that your local repository consists of three "trees" maintained by Git.

- **Working Directory**: which holds the actual files.
- **Index**: which acts as a staging area
- **HEAD**: which points to the last commit you've made.



So many commands?!

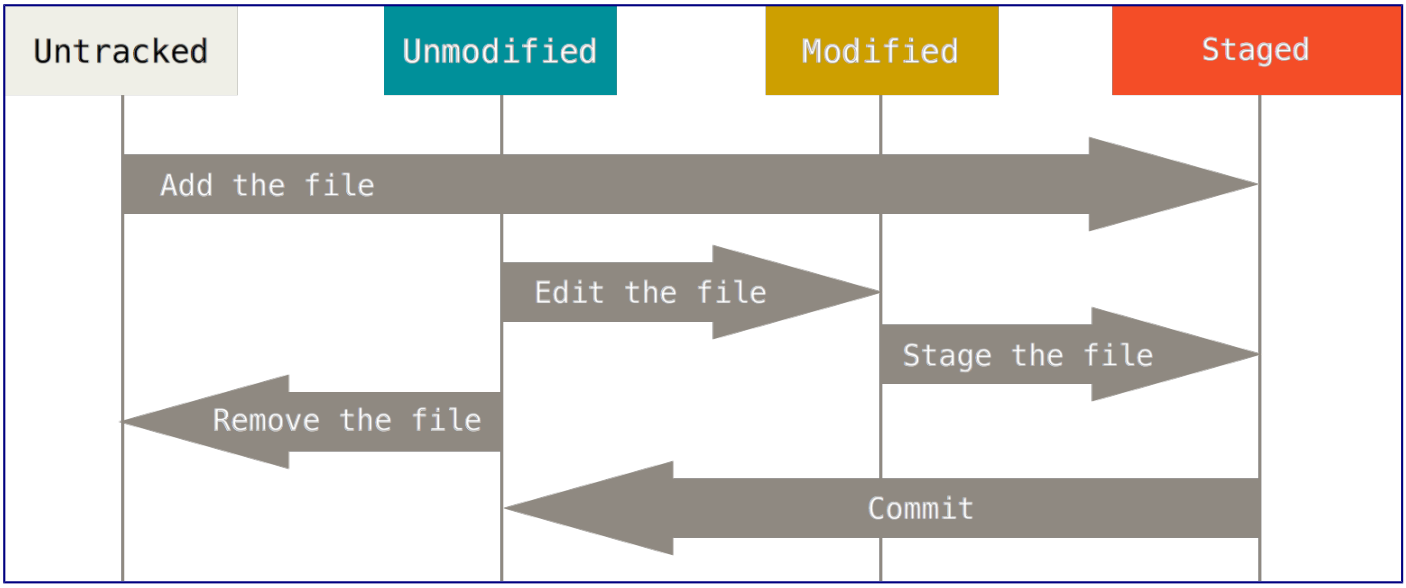
There are also a lot of commands you can use in Git. You can take a look at a list of the available commands by running:

```
$ git help -a
```

Even though there are lots of commands, on the course we will really only need about 10.

Git File Lifecycle

To understand how Git works, we need to talk about the lifecycle of a Git-tracked file.



Schema From [git-scm.com]

There are 4 main stages of Git version controlled file:

1. **Untracked**: The file will not be added in the next commit
2. **Staged**: Staged files have not yet been committed to memory but they are "on deck" so to speak for your next commit
3. **Unmodified**: The file has already been committed and has not changed since the last commit
4. **Modified**: You have changes in the file since it was last committed, you will need to stage them again for the changes to be added in the next commit

Once you have committed a file and it becomes "unmodified" then it's contents are saved in Git's memory.

- **Not saved in git memory**: Your file is not saved until you commit the file to Git's memory
- **Saved in git memory**: Only once you have committed a file, it becomes saved in Git's memory

Guided Practice: Let's use Git (15 mins)

First, we need to set a few things. We need to have a name and email address set so any changes we make show up under our name!

Type in the following commands:

1. git config --global user.name "YOUR NAME"
2. git config --global user.email "YOUR EMAIL ADDRESS"

Make sure the email address is the same as the one you used to sign up with on Github!

Now, create a directory on your Desktop:

```
$ cd ~/Desktop
$ mkdir hello-world
```

You can place this directory under Git revision control using the command:

```
$ git init
```

Git will reply:

```
Initialized empty Git repository in <location>
```

You've now initialized the working directory.

The .git folder

If we look at the contents of this empty folder using:

```
ls -A
```

We should see that there is now a hidden folder called `.git` this is where all of the information about your repository is stored. There is no need for you to make any changes to this folder. You can control all the git flow using `git` commands.

Add a file

Let's create a new file:

```
$ touch a.txt
```

If we run `git status` we should get:

```
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    a.txt

nothing added to commit but untracked files present (use "git add" to track)
```

This means that there is a new **untracked** file. Next, tell Git to take a snapshot of the contents of all files under the current directory (note the .)

```
$ git add .
```

This snapshot is now stored in a temporary staging area which Git calls the "index".

Commit

To permanently store the contents of the index in the repository, (commit these changes to the HEAD), you need to run:

```
$ git commit -m "Please remember this file at this time"
```

You should now get:

```
[master (root-commit) b4faebd] Please remember this file at this time
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt
```

Checking the log

If we want to view the commit history, we can run:

```
git log
```

You should see:

```
* b4faebd (HEAD, master) Please remember this file at this time
```

To exit this view, you need to press:

```
q
```

Make changes to the file

Now let's open a.txt in a text editor.

Inside the file, write something.

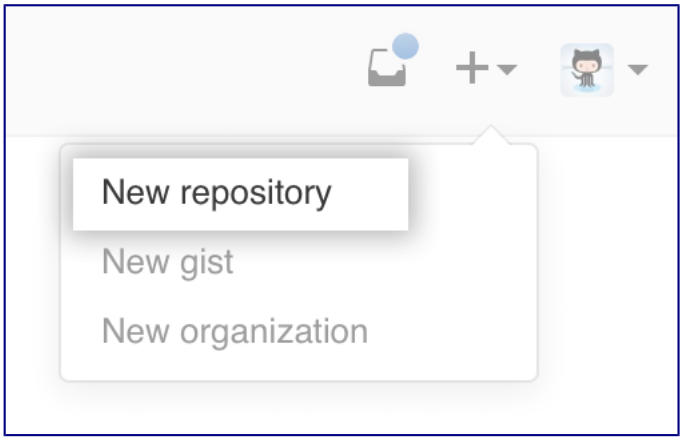
If you press `return` in the terminal, you will now see that you have untracked changes.

Running `git status` again will show you that a.txt has been **modified**.

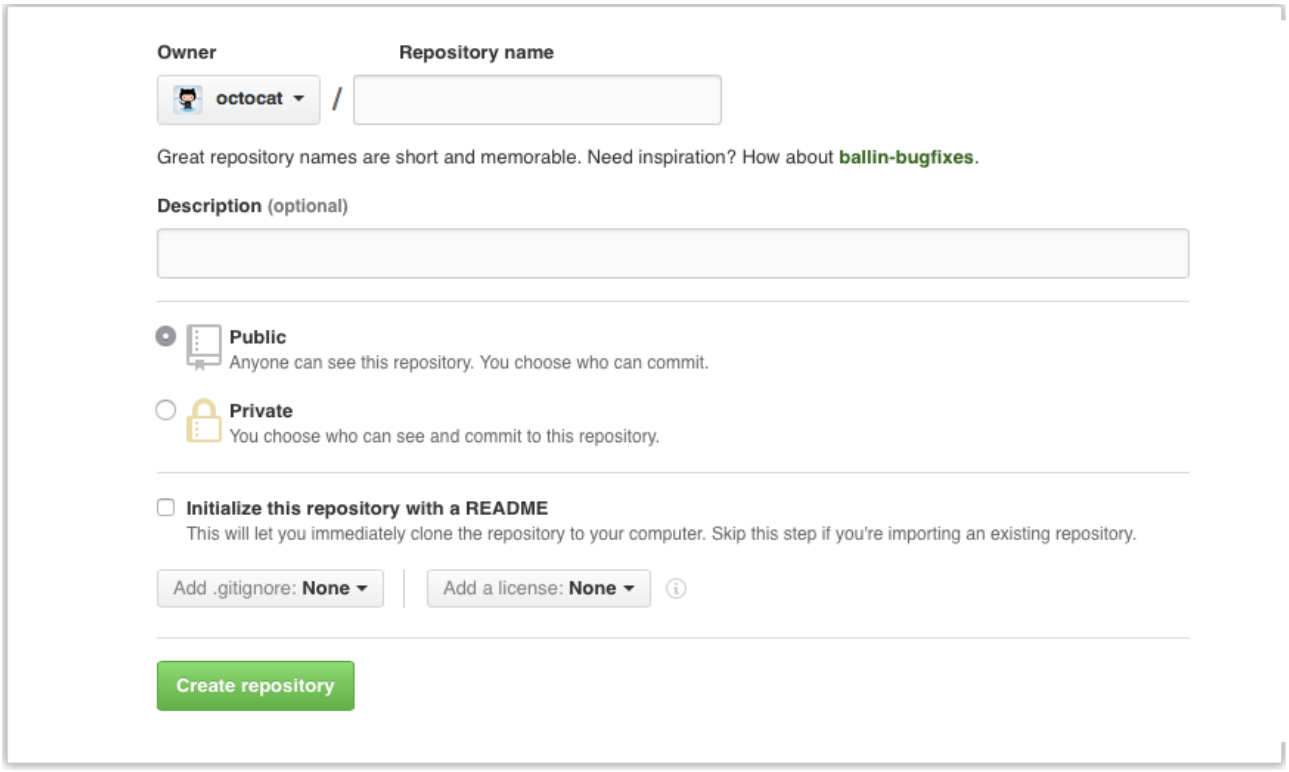
Guided Practice: Making and cloning repositories (10 mins)

Let's do this together:

1. Go to your GitHub account
2. In the top left, hit the + button and select `New repository`



3. Name your repository `hello-world`



4. Click the big green Create Repository button

We now need to connect our local Git repo with our remote repository on GitHub. We have to add a "remote" repository, an address where we can send our local files to be stored.

```
git remote add origin https://github.com/github-name/hello-world.git
```

Pushing to GitHub

In order to send files from our local machine to our remote repository on GitHub, we need to use the command `git push`. However, you also need to add the name of the remote, in this case we called it `origin` and the name of the branch, in this case `master`.

```
git push origin master
```

Pulling from GitHub

Let's add a README.md in our repo on github. We need to first `pull` that file to our local repository to check that we haven't got a 'conflict'.

```
git pull origin master
```

** If a document opened in your terminal, press Shift + : , then type q and press Enter.**

Once we have done this, you should see the README file on your computer. Now you can push your changes:

```
git push origin master
```

Refresh your GitHub webpage, and the files should be there.

Cloning your first repository

Now that everyone has their first repository on GitHub, let's clone our first repository!

Cloning allows you to get a local copy of a remote repository.

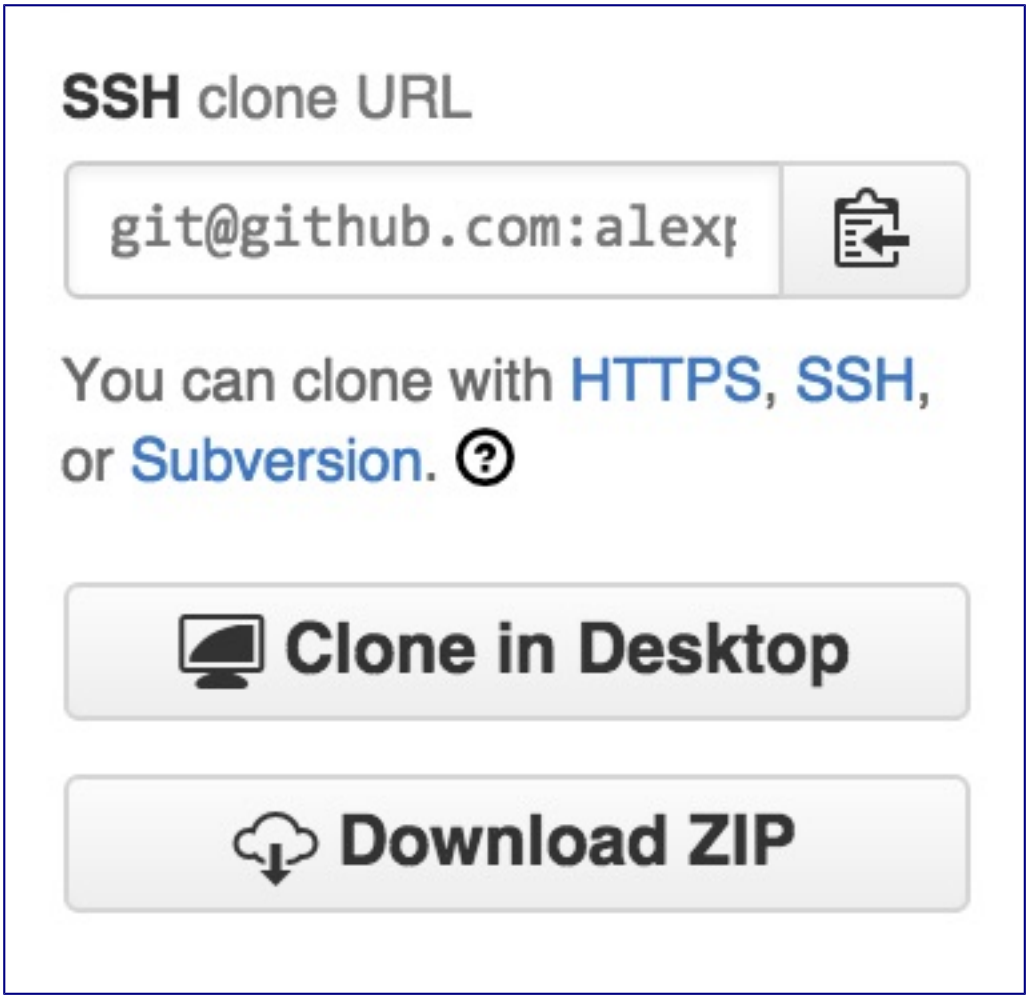
Navigate back to your Desktop and **delete your hello-world repository**:

```
cd ~/Desktop
rm -rf hello-world
```

Now ask the person sitting next to you for their GitHub name and navigate to their repository on GitHub:

```
https://www.github.com/<github-username>/hello-world
```

On the right hand side you will see:



Ensure that you have SSH checked and copy this url.

Clone their repo!

To retrieve the contents of their repo, all you need to do is:

```
$ git clone https://github.com/alexpchin/hello-world.git
```

Git should reply:

```
Cloning into 'hello-world'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
Checking connectivity... done.
```

You now have cloned your first repository!

Introduction: What is forking? (5 mins)

The `fork` and `pull` model lets anyone fork an existing repository and push changes to their personal

fork without requiring access be granted to the source repository.

Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.

> Check: Could someone explain the difference between forking and cloning?

Cloning vs Forking

When you fork a repository, you make a new **remote** repository that is exactly the same as the original, except you are the owner. You can then `clone` your new fork and `push` and `pull` to it without needing any special permissions.

When you clone a repository, unless you have been added as a contributor, you will not be able to push your changes to the original remote repository.

Pull requests

When you want to propose a change to a repository (the original project) that you have forked, you can issue a pull request. This basically is you saying:

"I've made some changes to your repository, if you want to include them in your original one then you can pull them from my fork!"

Guided Practice: Working with Forks and Creating Pull Requests on GitHub (15 mins)

Let's practice the flow that you'll be using in all our classes to get assignments and submit your work. Let's do it:

The goal is to get the work from this repo onto your local machine, make a change, and submit a pull request.

1. First, click on a link to this repository: https://github.com/ga-students/forking-exercise
2. Click on the fork button in the top right corner (After you do, notice that you have a new repo that you own forked from ga-students/forking-exercise!)
3. Find the "Copy to clipboard" button to get the clone URL
4. Jump back to the terminal, and from your root directory, type in: `git clone URL`
5. `cd` into the cloned repository and open the `forking-exercise` folder in a text editor
6. Type in your name in the `readme.md` file
7. `add`, `commit`, and `git push origin master`

Remember, before you can open a pull request from your fork, you must commit new code to your local clone of your fork, and push that code to your fork on GitHub. Now, to create a pull request:

1. Visit the repository you pushed to
2. Click the "Compare, review, create a pull request" button in the repository 
3. You'll land right onto the compare page - you can click Edit at the top to pick a new branch to merge in, using the Head Branch dropdown.
4. Select the target branch your branch should be merged to, using the Base Branch dropdown
5. Review your proposed change
6. Click "Click to create a pull request" for this comparison
7. Enter a title and description for your pull request
8. Click 'Send pull request'

Think about this flow because we'll be using it throughout the course!

Independent Practice: Assess (10 mins)

Use the internet and what you've learned today to answer the following questions with a partner:

- How do I send changes to the staging area?
- How do I check what is going to be committed?
- How do I send the commits to GitHub?
- How do I go back to the previous commit?
- How do I check the configuration on a specific machine?
- How does GitHub know that I am allowed to push to a specific repo?

Introduction: Git Ignore file (10 mins)

Before we wrap things up, let's have a chat about the .gitignore file.

When you create a new project, most IDEs usually generate files specific to your computer (i.e. setup files, temporary files, compiled code, etc). These kind of files should not be pushed to the remote Git repository, as they are specific to you alone and might affect other peoples' ability to use the project.

This is where the .gitignore file comes in.

The .gitignore file lists the type of files that should not be uploaded to your Git repo (i.e., what to *ignore*).

You can put .gitignore files in your repo, so whoever clones your project will ignore unnecessary files. You can also set up your computer so you always ignore certain files for all of your projects - a "global gitignore". Let's do the latter now.

Go to gitignore.io, a website that generates .gitignore files. Type in the types of projects you'll be working with (Android, Intellij, OSX, Windows), and press *Generate*. Copy all of the generated text.

Okay, now open Terminal and create the .gitignore file wherever you want; I tend to run `touch ~/.gitignore`. Then, open it and paste the generated text into the file. Make sure to save it!

Now, you have to register the file with Git. In Terminal, run:

```
git config --global core.excludesfile ~/.gitignore
```

All of your future projects will ignore the files listed.

Note: For local .gitignore files, you don't have to register them with Git. Just put them in the root folder of your Git project.

Independent Practice: Turning in an assignment (5 mins)

Using your new Github skills, turn in the command line lab from today. Fork and clone the lab repo, move the folders/files you created into the local repo, then push the changes. Finally, make a pull request.

Conclusion (5 mins)

As a developer, you'll have to use Git pretty much everyday - the learning curve is steep and all the principles of version control can be a bit blurry sometimes, so we ask students to push their homework everyday and to commit regularly during project time.

Don't be frustrated by all the new commands because we will definitely have the time to practice during this course.

- Explain the difference between forking and cloning.
- Describe the steps to initialize a Git repository and link your local repository to a GitHub remote location.

1:3

Command Line Lab

Introduction

Developing any type of application requires a degree of comfort navigating and interacting with your operating system through the command line, and similar to how you'll be practicing writing and running Java files and Android apps later in the course, we'll be practicing creating, modifying, and moving files and folders in your terminal to get you practicing Unix commands.

For your first lab, you're going to create files and folders to organize your favorite books, movies, and music - then, you're going to reorganize them.

Be sure to use the cheatsheets in the "Additional Resources" section in case you get stuck.

Exercise

Requirements

- From your home directory, create a folder called "my-favorite-things"; you'll use that folder to do the exercises below

- Organize your favorite books

    - in the "my-favorite-things" folder, create a folder called "books"
    - create a folder in books named after your favorite author (e.g. "mark-twain", or "john-grisham", but avoid spaces!)
    - create files named after some of the author's books in the author's folder
    - open the books folder in your text editor
    - edit each file to put a brief description of the book
- Organize your favorite movies

    - in the "my-favorite-things" folder, create a folder called "movies"
    - create a folder in movies named after your favorite actor
    - create a folder in the actor folder named after the actor's breakthrough movie
    - create a text file named after the actor's character in the breakthrough movie in the top level "movies" directory
    - move the text file to the breakthrough movie's folder
    - Use a text editor and edit that text file with a description of the character's role in the movie
- Organize your favorite music

    - in the my-favorite-things folder, create a folder called "music"
    - move into the "music folder"
    - create a folder called "disco"
    - create a text file in "disco" called "ymca"

        - delete the "disco" folder
        - create a folder called "creed"
        - delete the "creed" folder
        - create folders called "one-direction", "the-strokes", and "rihanna"
        - create a text file in "one-direction" called "what-makes-you-beautiful.txt"
        - make two copies "what-makes-you-beautiful.txt" - one into "the-strokes" and one into "rihanna" and rename those files with songs by those artists
- Reorganize *everything*

    - in the my-favorite-things folder, create a folder called "media"
    - move "books", "movies", and "music" into the "media" folder
- Organize the top music, movies, and books of 2016

    - move to the my-favorite-things folder and copy the "media" folder, then, rename it "2016-media"
    - in the 2016-media folder, rename each folder to have "2016-" before the title
    - delete the contents of "2016-music", "2016-movies", and "2016-books"
    - create a file called "top-ten-movies.txt" in "2016-movies"
    - create a file called "top-ten-songs.txt" in "2016-music"
    - create a file called "top-ten-books.txt" in "2016-books"
    - create a list of the top 10 movies, songs, and books in each of the appropriate files

Bonus

- Look through the additional resources and do the following

    - look at the top/bottom 10 lines of each file
    - figure out how search through a file from the command line - without opening the file - for a string of text

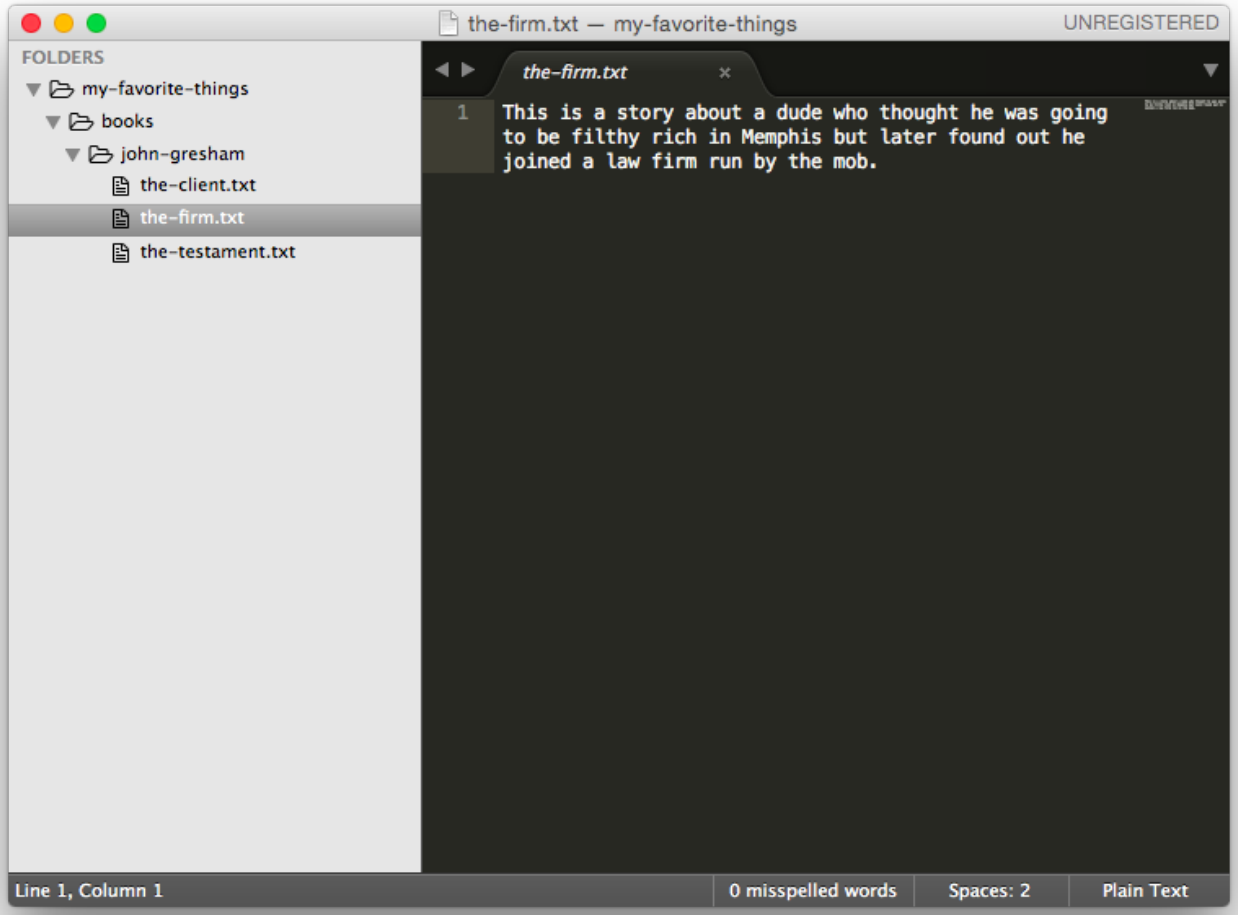
Starter code

No starter code needed for this lab!

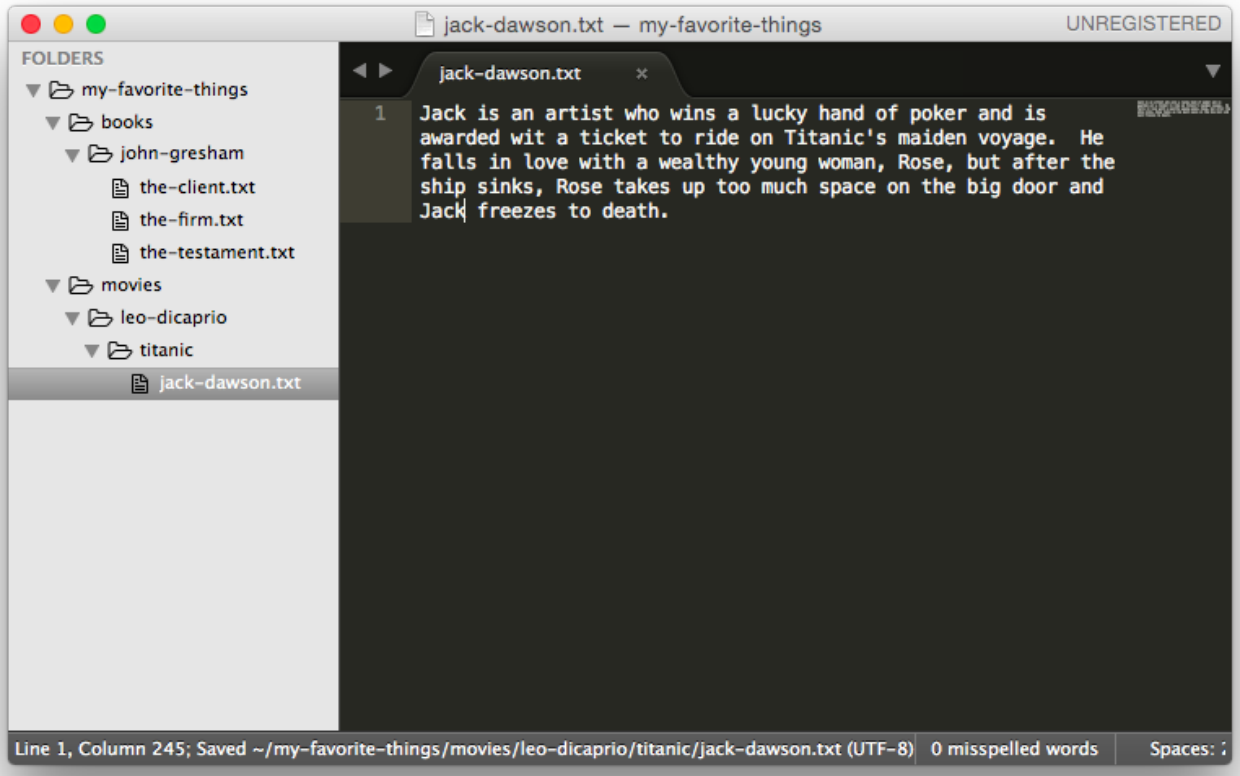
Deliverable

Here's a look at what your files/folders should look like after each big step in the exercise:

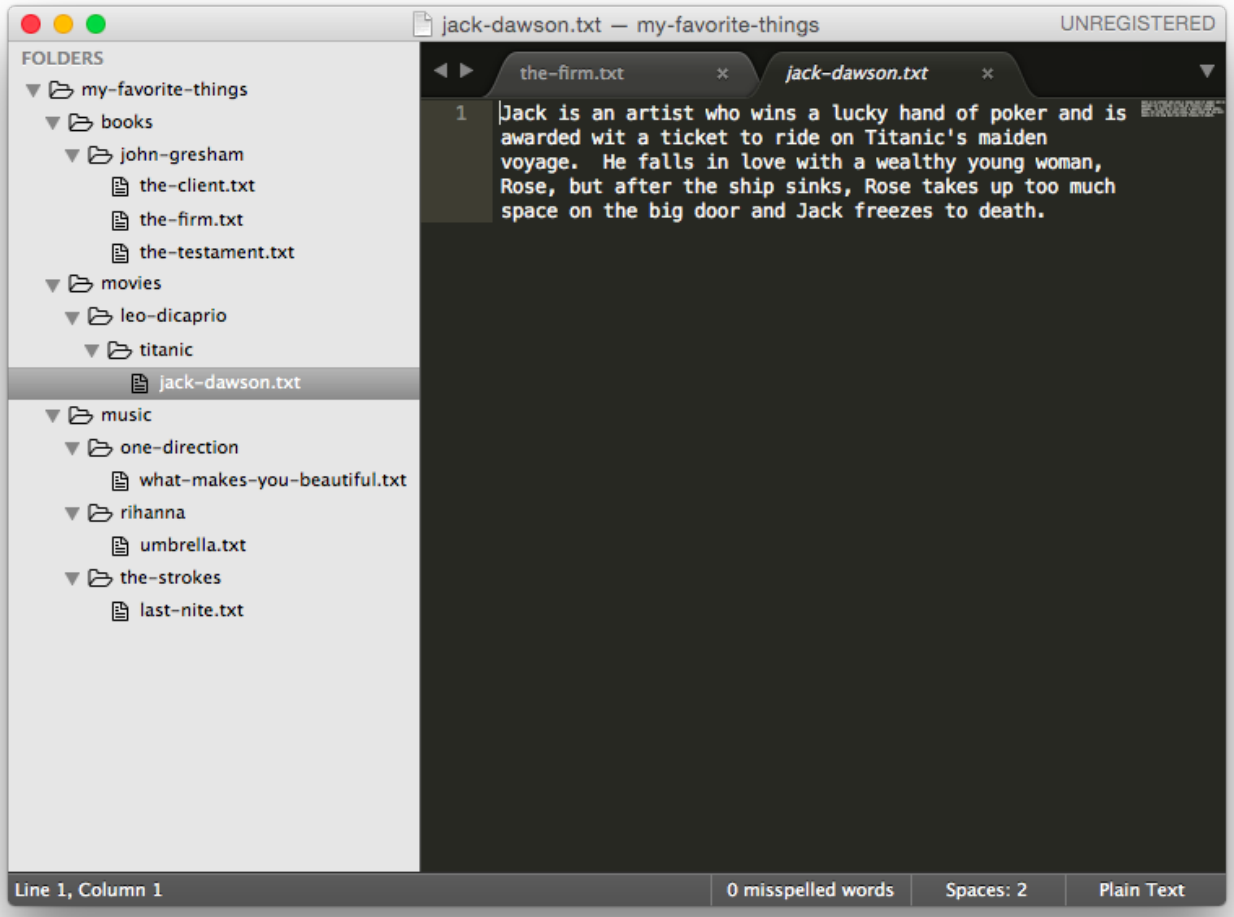
- After "Organize your favorite books":


the-firm.txt — my-favorite-things
UNREGISTERED
FOLDERS
my-favorite-things
books
john-gresham
the-client.txt
the-firm.txt
the-testament.txt
the-firm.txt
1
This is a story about a dude who thought he was going to be filthy rich in Memphis but later found out he joined a law firm run by the mob.
Line 1, Column 1
0 misspelled words
Spaces: 2
Plain Text

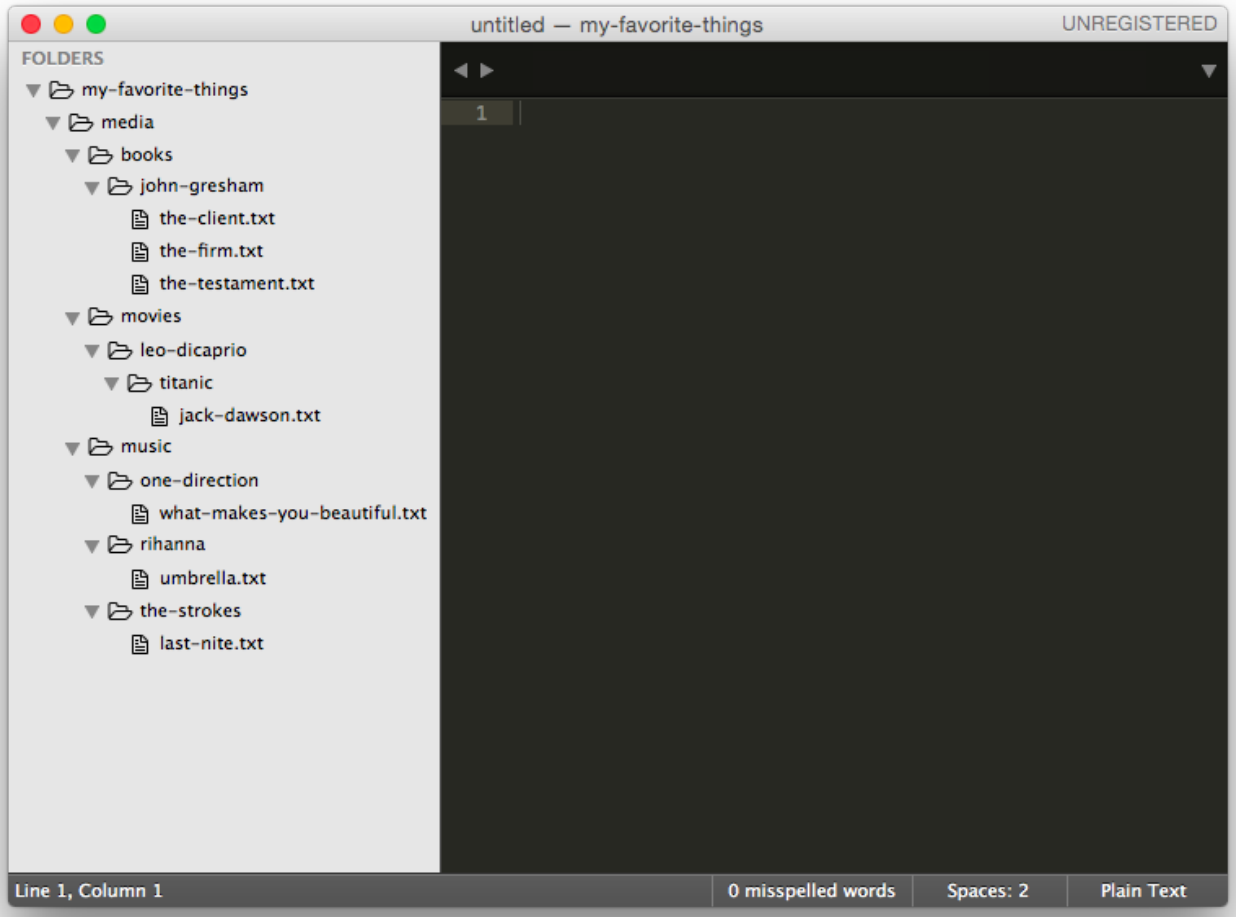
- After "Organize your favorite movies":



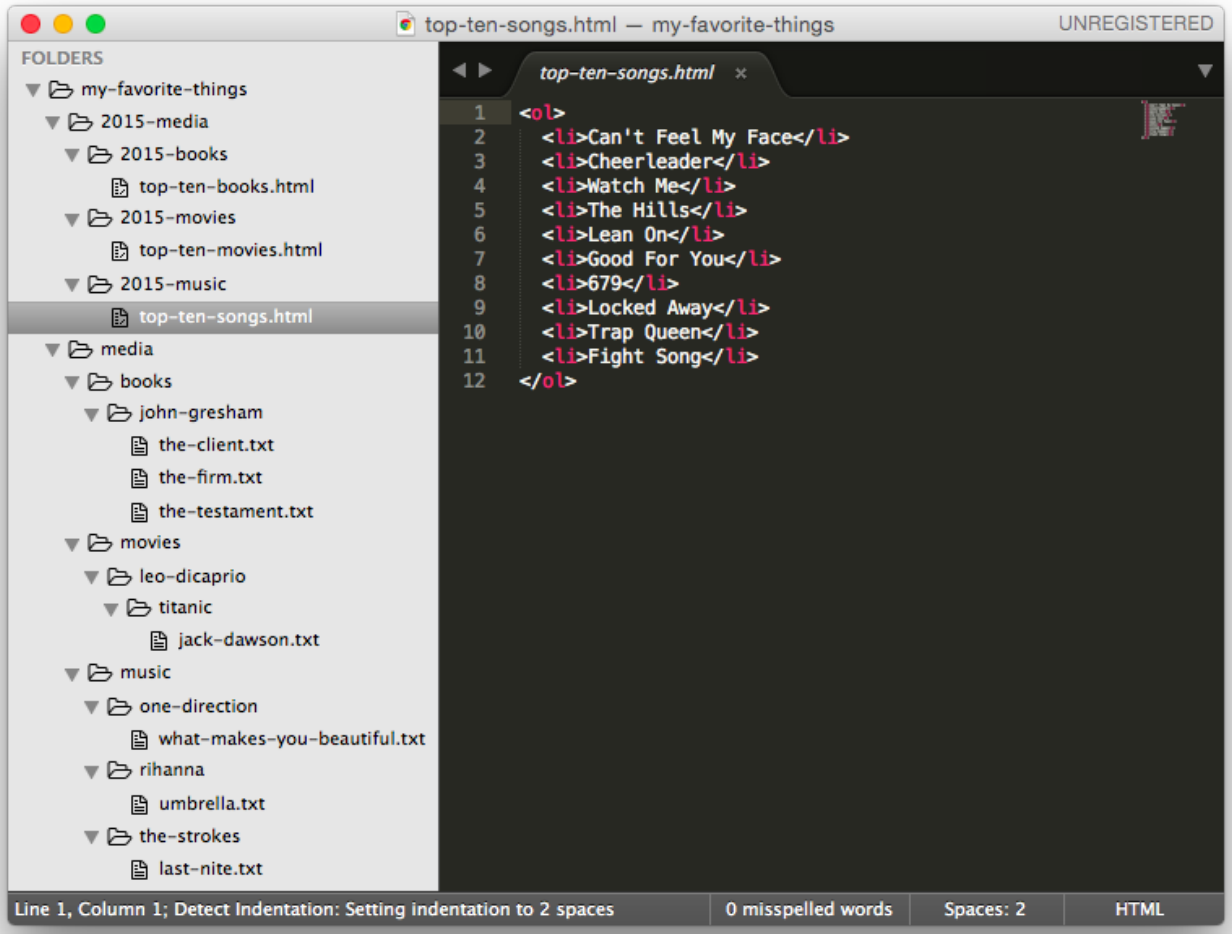
- After "Organize your favorite music":



- After "Reorganize *everything*"



- After "Organize the top music, movies, and books of 2015"



Additional Resources

- A list of CLI Shortcuts
- An awesome Unix command cheatsheet

1:4

Git and GitHub Intro Lab

Introduction

> ***Note:*** *This can be a pair programming activity or done independently.*

Let's apply what we've learned from class to share and update each other's code. With a partner, you're going going to alternate between who 'drives' and who 'navigates' while following the requirements under "Exercise" below. The goal will be to create a text file containing a message, have a partner fork, clone, and add their own changes, submit the changes as a pull request, and then have the changes merged.

Be sure to look at the previous lesson for notes and helpful hints.

Exercise

Partners will be referred to as partner1 and partner2.

Part 1

With partner1 driving:

- create a folder called `git-and-github-practice`
- within that folder create a new text file
- in that file, write a message to your partner.
- initiate a git repository, commit your changes, and push to GitHub

With partner2 driving, from their computer:

- get your partners link to the GitHub repository and fork and clone it
- edit the file to add a new message back to your partner
- commit your changes and submit a pull request back to partner1

With partner1 driving:

- merge the pull request from the GitHub interface

Part 2

With partner2 driving:

- create a folder called `git-and-github-practice-two`

- within that folder create a new text file
- copy and paste the code from the messages from the first part, and add an additional message for your partner.
- initiate a git repository, commit your changes, and push to GitHub

With partner1 driving:

- get your partner's link to the new GitHub repository - fork and clone it
- open the text file, and add a message for your partner.
- commit your changes and submit a pull request back to partner2

With partner2 driving:

- merge the pull request from the GitHub interface

Bonus:

- use the syncing a fork documentation to update partner2's local version of `git-and-github-practice` after partner 1 makes additional changes.

Deliverable

Each student should add a text file called "my_repo.txt" that contains a link to the new repo on their own Github account.

Additional Resources

- Git documentation
- Forking on github
- Syncing a fork

1:5

Data Types and Variables

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Identify what Java data type to use in specific situation and why you chose the specific data type over another.
- Describe the different types of variables (locals, instance, constants) and when to use them
- Use variables to perform a simple task with Math and String classes

---

Opening (5 min)

In our daily lives, we have different methods of storing things for use later on. Some form of organization.

These things could be anything from toys, to clothes, to money, and many other things;

In software development these are considered `data types`.

In the real world we store these `data types` in either, boxes, or garages, or some sort of container.

Computers store these `data types` in what are known as `variables`. Who has moved recently? Did you put things in boxes and label them? Well then you've been dealing with data types and variables all along.

Today we are going to be exploring how data types and variables are the building blocks of basic programming.

Introduction: Data types in Java (10 mins)

Describe a real world situation where students have been using data types and variables:

- Putting things in boxes(data types): Data types are simply the computer's way of categorizing the data we use in programming. Just like putting things in a box, there is a set size that you cannot go over with data types. Each data type has a limit to the value we can assign it.

- Labeling something(variables): Variables are the labels we put on the boxes so we know which box has the data we want. Depending on the data type we can do certain things to the variable. (Frying pan vs Cardboard box)

Category	DataType	Description	Example
True/False	boolean, Boolean	Represents either true or false	true, false
Integers	short, int, Integer, long, Long	Whole numbers, with no delimiter. Can optionally have underscores to make large numbers easier to read	42, 1024, 1_000_000
Decimals	float, Float, double, Double	Decimals, with no delimiter	'42.123', 2.5'
Characters	char	Single character, surrounded by single quotes	'a', 'A'
Strings	String	Anything, surrounded by double quotes, with some exceptions	"lots of kittens", "a lazy lizard", "", "10", "$"

There are also a few odd ones:

- Byte, which is one 8-bits of data. You don't need to worry about it right now.
- Collections (we'll talk more about this Week 3)

We'll dig deeper into all of the categories on the board, and show you how to manipulate them.

> Check: What are a few examples of data and what data type do they fall into?

Demo: Lets start with Numbers (15 mins)

Starting an IntelliJ Project

Steps to Create a new IntelliJ Project: Create New Project > Next > Check the "Create project from template" box > Select "Command Line App" > Next > Pick name & location > Finish. You are given a class that is named the same as the file with a `main` method inside it.

Note: What does the `//` mean? This represents a comment. You can also replace `//` with a multi-line comment `/* write your code here */`. Comments are used to clearly articulate what your code is doing so that other developers can easily jump into a project and understand what's going on.

We'll talk more about all of these pieces later, for now, write your code directly within the main method, where the comment says, `//Write your code here`.

Declaring and Assigning

First lets talk a bit about syntax. Programming languages, and languages in general, have a certain syntax that should be followed in order to create a valid "sentence". In Java there is a specific syntax for creating variables.

```
int num1 = 5;
```

This "sentence" does two things:

- Declare a variable named `num1` with type `int`
- Assign the value `5` to that variable

It's important to know that `=` does not mean "equals" like it would in a math class, but rather it is the *assignment operator*. It is used to assign the value on its right to the variable on its left.

Decimals vs Integers

Ok, lets talk a bit about those number data types. What do you expect to be printed to the console?

```
int num1 = 5;
System.out.println("num1, type int = " + num1);
---
$ num1 = 2;
```

How about here?

```
int num2 = 5 / 2;
System.out.println("num2, type int = 5/2 = " + num2);
---
$ num2 = 2
```

But Why is `num2` not 2.5? Well, in Java (unlike JavaScript, Ruby or PHP) numbers are strictly typed, and a type is either an integer or decimal. An int stores a integer, not a decimal, as demonstrated in the previous function.

So, what type of variable would we use if we wanted to assign a decimal number? How about a float?

```
float num3 = 5 / 2;
System.out.println("num3, type float = 5/2 = " + num3);
---
$ num3 = 2
```

> Check: That didn't work quite as expected. Can anyone guess why?

Because both 5 and 2 are automatically assigned data type int, when the calculation is done the answer is also an int: `float a = (float) (int a = int b / int c);`. We must tell the computer that the divisors are of a decimal type, not an integer type.

```
float num4 = 5f / 2f;
System.out.println("num4, type float = 5f/2f = " + num4);
---
$ num4 = 2.5

double num5 = 5d / 2d;
System.out.println("num5, type double = 5d/2d " + num5);
---
$ num5 = 2.5
```

Note: In the previous example, we used both a float and a double data type to save decimal numbers.

> Check: What is the difference between float and double and which should you use?

Number data types and Bits

To answer this question, it is helpful to understand that a data type defines not only the type of data but also

the methods that can be used to manipulate that data. The *primitive* data types in Java also has a certain pre-assigned size in memory. This is represented in a number of bits.

Name	Width in bits	Range
float	32	3.4e–038 to 3.4e+038
double	64	1.7e–308 to 1.7e+308

More memory means more information can fit into that variable. Double's are much larger than floats. What does that mean for working with decimals? Floats are more memory efficient, and doubles provide more accuracy.

Double's are recommended for currency and where accuracy is important. There is also a `BigDecimal` class, used when even more decimal points are needed.

The same data type differentiation exists in Integers between shorts (did you notice it in our list), ints and longs.

Name	Width in bits	Range
byte	8	-128 to 127
short	16	-32,768 to 32,768
int	32	-(2^31) to 2^31 (approx 2 billion)
long	64	-(2^63) to 2^63

`int` will cover almost all of your Integer needs.

> Check: What is the most common data type for decimals? What is the most common data type for integers?

Using Standard Arithmetic Operators

Now that we understand a bit more about the Number data types, lets look a bit at what we can do with them.

Here are the standard arithmetic operators that you've been learning since grade school:

```
System.out.println(2 + 2);
System.out.println(2 - 2);
System.out.println(2 / 2);
System.out.println(2 * 2);
System.out.println(2 % 2); // What does this do??
```

Demo: Using Math Methods (5 mins)

Programming languages can be a little stingy with the number of operations they allow you to do. For example, how do you square or cube a number? There is a special 'Math' Object, provided by Java, that has some very useful methods.

Raising a number to a power? Use `Math.pow(num1,num2)`.

```
// 3^2 becomes
System.out.println( Math.pow(3,2) );
---
$ 4
```

Taking a square root? Use `Math.sqrt(num1)`.

```
// √(4) becomes
Math.sqrt(4);
---
$ 2
```

Need a random number? Use `Math.random()`.

```
// returns double value with positive sign greater than or equal to 0.0 and less than
1.0
Math.random() // returns random number in range
int max = 10;
int min = 20;
int range = Math.abs(max - min) + 1;
(Math.random() * range) + min;
```

> Check: Who provides the Math object? Where do you think you might be able to find more information? (Oracle Math Documentation)

Introduction: Primitives vs. Objects (10 mins)

Before we get into Strings, let's take a step back. Have you noticed that all the data types we've used so far are lowercase? What do you notice about the `String` data type?

Do you notice that it is capitalized? This is a naming convention that is used to distinguish between primitive and Object data types.

Primitive data types are just a value, nothing more. A variable of a primitive type holds a value, but has no methods or "actions" you can tell it to do.

Object data types contain attributes and methods, and start with a capital letter. *Attributes* are values that live inside the object. A single Object is capable of holding many different values. *Methods* are actions the object can take. An Object includes instructions for how to carry out various actions.

A primitive variable holds the primitive value itself. An Object variable, however, doesn't hold all the attributes and methods of the Object. Instead, an Object variable holds a *reference* to the place in memory where the Object is stored.

> Check: Discuss with the person next to you: What does a primitive contain? What does an object contain? What's one easy way to tell the difference between an Object and a primitive data type? Be ready to share out!

Words: char and Strings

With that basic introduction to the two sorts of data types, primitives and Objects, lets talk about words.

A `char` is a primitive data type. What is an example of a `char`?

```
char myChar = 'a';
myChar = 'b';
myChar = '5';
myChar = ';';
```

A String is capitalized because a String is an Object.

Strings are collections of letters and symbols known as *characters*, and we use them to deal with words and text.

Strings are special - String is actually a array of 'char' data:

```
String str = "abc";
// is actually
char data[] = {'a', 'b', 'c'};
```

Demo: Creating a new string (15 mins)

Strings are a weird type of Object. You can instantiate (i.e. create an instance of) a String in a few ways:

```
//variable can be assigned like a primitive
String a = "I'm a string."
```

Which is really short for:

```
//variable assigned like an Object
String a = new String("I'm a string too!")
```

String helper methods

Because a String is an Object, it has pre-defined methods we can use.

To find the length of a string, use it's `length` property:

```
"hello".length();
---
$ 5
```

To get the first letter of a String:

```
"hello".charAt(0);
---
$ "h"
```

To replace part of a String:

```
"hello world".replace("hello", "goodbye");
---
$ "goodbye world"
```

To make a String uppercase:

```
"hello".toUpperCase();
---
```

```
$ "HELLO"
```

To add two Strings together:

```
"hello".concat(" world");
---
$ "hello world"
```

Remember, Strings are special Objects that look like primitives. Use the `str1.concat(str2)` function. Or concatenate (add together) using `+` :

```
String twoStringsTogether = "Hello" + " World";
---
$ "Hello World"
```

> Instructor Note: Introduce other methods as seen fit. May want to explain when concatenation might be used.

A special note on Equality among Strings:

What if you want to compare two strings?

Can you remember from the pre-work how to compare variables?

```
boolean areEqual = (1 == 2);
---
$ false
```

What's special about strings?

```
String blue = "blue";
boolean withSign = (blue == "blue");            //=> true
boolean withWords = (blue).equals("blue");      //=> true
```

Do you know which one of these would be preferred? Well, lets do another example to show you which and why:

```
String blue = "blue";

String bl = "bl";
String ue = "ue";
System.out.println(bl+ue);                    //=> blue

boolean withSign = (bl+ue == blue);
System.out.println(withSign);                 //=> false

boolean withWords = (bl+ue).equals(blue);
System.out.println(withWords);                //=> true
```

Why isn't `withSigns` true? The output of `println()` looks the same. Remember, Strings are Objects, so what a String variable actually holds is a *reference* to where the String value is stored in memory.

`==` compares the values held by the variables to its left and right. `(bl+ue) == blue` is comparing the reference held by 'blue' to the result of evaluating `(bl + ue)`. The latter references a different location in memory than the former, so the comparison yields false.

`equals()`, on the other hand, doesn't just look at the reference held in each variable, it *follows* each

reference and looks at the attributes within the Objects that the variables reference. In the case of Strings, it compares the underlying `char` arrays to see if they're the same.

The long and short of it is: use `equals()` when comparing Strings or any other non-primitive Objects. Use `==` when comparing primitives.

> Check: Why can we call methods on a variable with data type String but not on an int?

Demo: Converting between data types (10 mins)

Sometimes it is necessary to convert between data types. User input is *always* a String - like when you enter your email address, age, income, etc. If you'd like to operate on those numbers though, you'll have convert it to a type of number.

Remember how we talked about the size of primitive data types? An float is smaller than a double, and a double smaller than a long?

When converting from smaller types to larger types, for example, int(4 byte) to double(8 byte), conversion is done automatically. This is called **implicit casting**.

```
int a = 100;
double b = a;
System.out.println(b);    // 100.0
```

If, on the other hand, you are converting from a bigger data type to a smaller data type, for example, a double(8 byte) to an int(4 byte), the change in data type must be clearly marked. This is called **explicit casting**.

```
double a = 100.7;
int b = (int) a;
System.out.println(b);  // 100
```

While that is useful for numbers, to cast successfully, a variable must be an **instance of** that second Object. What do you think would happen if you tried to cast a String to an int?

There is a different way to convert Strings to numbers.

Did you notice that there is both an `int` and an `Integer` data type? The `Integer` data type is an Object "wrapper" around an int that provides certain methods. For example, to convert an String to an Integer, one can use:

```
String strValue = "42";
int intValue = new Integer(strValue).intValue();
```

Similar methods exist for all of the wrappers.

NaN

If a String is converted to a number but contains an invalid character, the result of the conversion will be **NaN**, which stands for: Not A Number.

NaN is toxic, and if a calculation is attempted on that variable or a method called subsequently, your program will break.

Test for NaN using `isNaN()`.

Null

A null value is an empty value. Taken from a StackOverflow post:

"Zero is a value. It is the unique, known quantity of zero, which is meaningful in arithmetic and other math."

"Null is a non-value. It is a placeholder for a data value that is not known or not specified. It is only meaningful in this context."

> Check: When might you get NaN value? What is a null value, by the way?

Independent Practice: Practice! (15 minutes)

> Instructor Note: This can be a pair programming activity or done independently.

Open the IntelliJ project found in `starter-code` and complete all tasks specified in the comments. We will go over the answers in 12 minutes.

> Check: Were students able to create the desired deliverable(s)? Did it meet all necessary requirements / constraints?

Conclusion (5 mins)

- Identify the different types of data
- What type of data do you think is passed as a web request?

ADDITIONAL RESOURCES

- Oracle Java Docs on Primitive Data Types
- Oracle Java Docs on Math Object

1:6

Functions and Scope

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Identify the parts of a method
- Describe how parameters relate to functions
- Use naming conventions for methods and variables
- Create and call a function that accepts parameters to solve a problem

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Work with basic primitive data types and strings, and assign variables
- Compile and run Java code from IntelliJ

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Read through the lesson
- Add additional instructor notes as needed
- Edit language or examples to fit your ideas and teaching style
- Open, read, run, and edit (optional) the starter and solution code to ensure it's working and that you agree with how the code was written

Opening (5 mins)

Previously we covered variables and data types, two extremely important building blocks of all programming languages. Today, we are going to learn about two more topics: function and scope. Functions are essentially re-usable chunks of code that complete some task, and scope is defining where the variables actually have relevance in your program.

Introduction: Writing Functions and Languages (15 mins)

Programming, and Java in particular can be a tough skill to master. There are many different topics to cover, but we are going to concentrate on the basics first. After learning the basics over the next few days, you will be able to easily produce all of the code needed to produce something similar to the following fun little program in a manner of minutes.

All of this will prepare you for writing Android apps, which require extensive amounts of Java code.

How computer programs work

Before we talk about functions and scope in detail, there are a few very important things to understand about how programming languages actually work. We are first going to explore how the code you write is actually read and run by the computer.

The code that you write *must* be translated into a form that the computer can understand.

Source code is human readable, *we hope*. This source code may be translated into a set of 1's and 0's that a computer's CPU can understand. Yep, the CPU is a chip on the computer that does all the processing. There is a reason it's called the Central Processing Unit, or CPU.

`Source Code` ==> `1's and 0's`

...or, the source code may be translated into a another type of language, byte code, that can be understood by a Virtual Machine(VM). A Virtual Machine executes a program, by translating each VM readable statement into a sequence of one or more subroutines already compiled into machine code. In the case of Java, the JVM understands bytecode.

`Source Code` ==> `byte code`(understood by JVM and mapped to `1's and 0's` that the computer's CPU can understand)

Compiled Languages

Some languages are *explicitly* compiled. In other words, the programmer must run particular commands to invoke compilation.

Java is one of these languages. To compile java code, the programmer must run a command like:

```
javac MainActivity java MainActivity.class
```

The `javac` command translates the Java code in the YourFile.java file into an *executable* or *binary* file (a YourFile.class file) that contains the `bytecode` understood by the JVM. This is what is done under the hood by a IDE like IntelliJ.

The *JVM* is the **Java** compiler.

So here's what happens:

`Source Code` ==> `1's and 0's`

`MainActivity.java` ==> `MainActivity.class`

Interpreted Languages

Some languages do *not* require the programmer to explicitly run a compiler. **JavaScript**, and **Ruby** are a couple of interpreted languages. There is still compilation being done, but it's done automatically.

`Source Code` ==> `byte code`

From Source to Running Code

There are two basic phases to go through when going from code in a file to a program running.

- Compile Time - a phase when the source code is translated to another form. For example, when we run the `javac HelloWorld` command, java program will compile java to an intermediate language/bytecode that the Java Virtual Machine(JVM) understands.

- Runtime - a phase when the computer actually runs each statement in the program. For example, when we run the `java HelloWorld.class` this is when the computer runs the java program bytecode.

Main Method

Let's start by looking at our first method! We've actually used this before, but haven't had the chance to talk about what it actually means.

The main method is where everything starts. From the Oracle Java Documentation:

In the Java programming language, every application must contain a main method whose signature is:

```
public static void main(String[] args)
```

The modifiers public and static can be written in either order (public static or static public), but the convention is to use public static as shown above. You can name the argument anything you want, but most programmers choose "args" or "argv".

The main method is similar to the main function in C and C++; it's the entry point for your application and will subsequently invoke all the other methods required by your program.

The main method accepts a single argument: an array of elements of type String.

```
public static void main(String[] args)
```

This array is the mechanism through which the runtime system passes information to your application. For example:

```
java MyApp arg1 arg2
```

Each string in the array is called a command-line argument. Command-line arguments let users affect the operation of the application without recompiling it.

Demo: Let's break it down (15 mins)

Let's look at what the parts of this method do. Let's start with the basics, which we covered a bit in explaining the main method..

```
public                      void            interestingMethod(  String input  )
throws IOException
//<modifiers/visibility>  <return type>    <method name>   (  <parameters>  )
<Exception list>
{
<opening brace>
    System.out.println("I am making" + input + "interesting!")
```

```
    <method body>
}
<closing brace>
```

Modifiers

Modifiers are used to modify how a method can be called.

Access Modifiers include:

- private *(visible only to the class)*
- protected *(visible to the package and all subclasses)*
- public *(visible to the world)*
- package private *(when none is specified, this is the default)*

Non-Access Modifiers include:

- static *(for creating class methods and variables)*
- final *(for making something permanent)*
- abstract *(to create abstract classes and methods)*
- synchronized / volatile *(used for threads)*

We'll explain more of what all these keywords mean in later lessons. For now, use `public`!

Any method that is called from within a static context must also be static. So, for all methods, for now use, `public static`. Again, we'll explain more what this means later.

Return Type

A method can return a value, but the type of that returned data must be specified so that the calling function knows what to do with it.

The problem:

```
class Main {
    int mSum;
    public static void main(String[] args) {
        getSum();
        // System.out.println(sum); // not available
    }
    public static void getSum() {
        int sum = 2 + 2;
        System.out.println(sum);
    }
  }
```

The solution:

```
class Main {
    int mSum;
    public static void main(String[] args) {
        int returned = returnSum();
        System.out.println(returned);
    }
    // public static void getSum() {
        // int sum = 2 + 2;
```

```
        // System.out.println(sum);
    // }
    public static int returnSum() {
        int sum = 2 + 2;
        System.out.println(sum);
        return sum;
    }
  }
```

A function executes until it reaches a `return` statement or a closing curly brace. If a data type has been specified, that sort of data (or null) must be returned or the code will not compile.

Another solution, if it wasn't appropriate to use a return type, would be to use a global variable.

Global variables are defined at the top of a class, and by convention are named using `mVariableName`.

```
class Main {
    int mSum;
    public static void main(String[] args) {
        getSum();
        System.out.println(mSum);
    }
    public static void getSum() {
        int sum = 2 + 2;
        System.out.println(sum);
        System.out.println(mSum);
        mSum = sum;
    }
  }
```

Independent Practice: Discuss (5 mins)

Take 3 minutes, and discuss these questions with the person next to you:

- Describe the different components that make up a method.

Be ready to share!

Demo: More Functions! (10 mins)

Method Name

This is what the method is called.

It's important to be explicit in the naming of your method so that just by looking at the title - a new developer can come in and can understand what the method will do.

By convention, a method name should be a **verb** in *camel case* that starts in *lowercase*.

Ex: `getName()`, not `GetName()`, nor `getname()`, nor `get_name()`.

Parameters (Enclosed within parenthesis)

Parameters are arguments passed into the function when it is called. This makes a function much more dynamic.

Let's take a look back at the sum method.

Check: What would you need to do if you wanted to pass in a number to this method?

```
public static int returnSum(int num1) {
    int sum = num1 + num1;
    return sum;
}
```

Check: How about two numbers?

```
public static int returnSum(int num1, int num2) {
    int sum = num1 + num2;
    return sum;
}
```

Now, note, the method can be called like so:

```
public static void main(String[] args) {
    int returned = returnSum(2,4);
    System.out.println(returned);
    int returned = returnSum(10,52);
    System.out.println(returned);
}
```

In java, if a method declares a parameter, that *parameter* is required to be sent as an *argument* from the calling method.

Method Body (Enclosed within curly braces)

This is where the main functionality of your method will be called.

Guided Practice: Writing Functions (15 mins)

Let's work through the following example. The Scanner class we'll be creating will be required in the lab.

```
public class Main {
    public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      String userInput = input.nextLine();
        System.out.println(userInput);
    }
}
```

Some things to mention:

- methods must be within a class definition
- no nesting method in method
- a bit about scope, and mGlobal variables

Check: Take a minute and try to come up syntax needed to print the message: "\nAsk: who, what, why, when, or where".

So here we have a basic main class:

```
public class Main {
```

```
    public static void main(String[] args) {
        System.out.println("\nAsk: who, what, why, when, or where");
    }
}
```

Let's add a method that creates a gets user input and responds.

```
public static void askAQuestion() {
    Scanner input = new Scanner(System.in);
    String userInput = input.nextLine();
    if (userString.equals("who")) {
        System.out.println("We're the ADI class.");
    }
}
```

Actually, let's allow the user to put in a more complicated question, such as 'Who are you?'

```
...
if (userString.contains("who")) {
  ...
}
```

Let's add a default:

```
  else {
     System.out.println("I don't know how to answer that question.");
     System.out.println("Try again...");
  }
```

Wait, what if we actually want to be able to try again?

```
    ...System.out.println("Try again...");
    }
    askAQuestion();
}
```

Note, this is called **recursion** - a Recursive method calls itself. For those who are interested in Math, a resource is included that talks about some of the other ways to use recursion to solve basic algorithms.

What if we want to exit out of the program?

```
    else if (userString.contains("exit")) {
            askAgain();
        }
    }
    public static void askAgain() {
        System.out.println("\nAre you sure you have no more questions? y or n");
        String userInput = grabUserInput();
        if (userInput.equals("y")) {
            System.out.println("Thanks for playing. Goodbye.");
            System.exit(0);
        }
        else if (userInput.equals("n")) {
            System.out.println("Ask another then:");
            askAQuestion();
        }
    }
}
```

Independent Practice: Write a few functions (15 min)

Please create a new Java project in IntelliJ and work through as many as these exercises as you can within the next 15 mins. Use the official Oracle Java Docs to help you through these exercises and look up the different class methods you can use.

1. Write a method called `divide152By`. This method should accept one argument, a number, and should divide 152 by the given number. For example, the divide152By result of 1 is `152/1` is 152. Your function should return the result.

   Use your function to find the following:

   ```
   divide152By(3);
   divide152By(43);
   divide152By(8);
   ```

2. Write a method called `transmogrifier`. This method should accept three arguments, which you can assume will be numbers. Your function should return the "transmogrified" result.

   The transmogrified result of three numbers is the product of the first two numbers, raised to the power of the third number.

   For example, the transmogrified result of 5, 3, and 2 is `(5 times 3) to the power of 2` is 225. Use your function to find the following answers.

   ```
   transmogrifier(5, 4, 3);
   transmogrifier(13, 12, 5);
   transmogrifier(42, 13, 7);
   ```

3. Write a method called `thirdAndFirst`. This method accepts two strings, and checks if the third letter of the first string is the same as the first letter of the second string. It should be case insensitive, meaning `thirdAndFirst("Apple","Peon")` should return true.

   Check the following:

   ```
   thirdAndFirst("billygoat","LION");
   thirdAndFirst("drEAMcaTCher","statue");
   thirdAndFirst("Times","thyme");
   ```

4. **BONUS**: Write a method called 'reverseString'. This method should take one argument, a String. The method should return a string with the order of the words reversed. Don't worry about punctuation

   ```
   reverseString("black cat"); => "tac kcalb"
   reverseString("the cow jumped over the moon"); => "noom eht revo depmuj woc eht"
   reverseString("I can ride my bike with no handlebars"); => "srabeldnah on htiw
   ekib ym edir nac I"
   ```

Conclusion (5 min)

- Why do we use methods?
- When might you use a method?

Resources:

- Oracle Java Docs - Defining Methods
- Oracle Java Docs - A Closer Look at the "Hello World!" Application
- Princeton Java Cheat sheet
- Java Modifier Types
- Princeton Recursion

1:7

Functions and Scope

Introduction

> ***Note:*** *This can be a pair programming activity or done independently.*

This lab provides an opportunity to practice creating functions!

Exercise

Requirements

Please write code that implements various functions, following the requirements in the list below:

- Define a method `stringLength`, that accepts a `String` parameter, and prints out the length of the String.
- Define a method `isEven`, that takes an `int` parameter, and returns `true` if the number is even, `false` if it is not. Print this returned value to the command line. *Note how do we know a number is even?*
- Define a method `userInput`, asks the user to type something into the command line, uses the Scanner class we used during the lesson to take the `String` the user provides in the command line and prints it back to the command line.
- Define a method `typeQuit`, asks the user to type something into the command line, prints the `String`, then continues to ask the user to type something until the user types in `quit`. Use the Scanner class we used during the lesson, to take the `String` the user provides in the command line. *hint: this requires recursion*

Bonus

- Define a method, `parameterCount`, that accepts an unknown number of `String` parameters, and prints out the the number of parameters.
- Define a method `longestString`, that accepts two `String` parameters of different length, and returns the `String` with the longest length. Print the returned value to the command line.
- Define a method `fibonacci`, that takes an `int` parameter, and prints out the fibonacci sequence up to that value. *Note: Fibonacci sequence: given a number N, the fibonacci sequence equals the sum of (N-1) + (N-2) hint: this requires recursion*

Starter Code

The code snippet below will be useful for the isEven.

```
/**
     * Converts a string into an integer
     * @return An integer version of the string, or 0 if the string is not a number.
     */
private int convertToInteger(String numberString){
    int value = 0;

    if (!TextUtils.isEmpty(numberString)){
        try {
            value = Integer.parseInt(numberString);
        } catch (NumberFormatException ex){ }
    }

        return value;
}
```

Deliverable

An IntelliJ project containing the methods described above.

Additional Resources:

- Oracle Java Docs - Defining Methods
- Oracle Java Docs - A Closer Look at the "Hello World!" Application
- Princeton Java Cheat sheet
- Java Modifier Types
- Princeton Recursion

1:8

Data Types and Variables

Homework

Requirements

- Fork this repo: *click the Fork button in GitHub*
- Clone your fork to your computer
    - *navigate to the directory where you want git create the copy of your fork*
    - *use `git clone <url>` to do the cloning*
- Open the IntelliJ project found in the `starter-code` folder
    - *select the `starter-code` folder to open, not one if its subfolders*
    - *in general, you want to select the folder that contains the `.iml` file and the `src` directory as the one to open - in this case, `starter-code` meets those criteria*
- Complete the code in `Main.java` based on the instructions in the comments

Starter code

The starter code is in `starter-code/src/ly/generalassemb/Main.java`. Curious about the "ly/generalassemb" part? Read here about package names in Java.

Deliverable

Once you complete the code in `Main.java`, do the following:

- Commit your changes
    - *do `git add <relative path to file you changed>` to stage the file you updated, or just `git add .` to stage everything in your current working directory*
    - *run `git commit -m "put your commit message here"` to commit the staged file(s)*
    - *if you forget the `-m` and the Vim editor opens, you can hit `:q` then enter to close it*
- Push your local commits to your fork on GitHub
    - *do `git push origin master`*
    - *go to your fork on github, refresh the page, and you should see your new commit(s)*
- Submit a pull request on GitHub from your fork back to the original repo
    - *please put your name in the pull request title*
    - *from the original repo, click on the "Pull Requests" tab and double check that your request appears in the list*

1:9

Functions and Scope

Exercise

Requirements

- *Fork* this repository so you have a copy in your github account
- *Clone* your forked repo onto your machine
- Open the `starter-code` folder as a project in IntelliJ
- Modify the starter code per the instructions for problems 1-5 in the `main()` method of `Main.java`
- Save your code and *commit* it to git
- After you've committed all your code, *push* it to github
- Finally, submit a *pull request* back to the original repo so we can see your code
- You are welcome to help each other, but everyone must submit their own code

Bonus:

- Problems 6 and 7 in the `main()` method are bonus problems

Starter code (if needed)

`starter-code` contains a simple IntelliJ project. The specific instructions for each problem are written in the comments of `Main.java`.

Deliverable

Your *pull request* with your completed code is the deliverable. Please put your name in the title of your pull request.

1:10

Mastering Control Flow

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Use if/else if/else conditionals to control program flow based on boolean conditions
- Use comparison operators to evaluate and compare statements
- Use boolean logic (!, &&, ||) to combine and manipulate conditionals
- Use switch conditionals to control program flow based on explicit conditions
- Loop over a code block one or more times

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Create variables in Java using basic data types
- Use a text editor

Opening (5 mins)

From Culttt.com: "Control Flow Structures are an important aspect of programming languages that allow your code to take certain actions based on a variety of scenarios. Control Flow is a fundamental concept in programming that allows you to dictate how your code runs under different conditions or until a certain condition is met."

Introduction: Logical operators and control flow (10 mins)

Java supports a compact set of statements, specifically control flow statements, that you can use to incorporate a great deal of interactivity in your application.

Block Statements

Statements meant to be executed after a control flow operation will be grouped into what is called a **block statement**. These statements are wrapped into a pair of curly braces:

```
{
  System.out.println("hello");
  System.out.println("roar");
}
```

Block scope

We've seen that scope changes depending on whether a variable is defined in the class (we use the `mVariableName` convention for these), or in a method (these variables have local scope only and are not available outside that method).

In Java, variables defined in **block statements** modify scope, meaning those variables are not available outside of the block.

For example:

```
boolean beautiful = true;
if (beautiful)
{
  String name = "jay";
}
System.out.println(name); // symbol 'name' cannot be resolved
```

Variables defined in **block statements** are not available outside of the curly braces. How might we resolve this issue?

```
boolean beautiful = true;
String name = "pepe";

if (beautiful)
{
  name = "robin"; // use the predefined variable
}
System.out.println(name);
//=> robin
```

Demo: Conditional statements (10 mins)

Conditional statements are a way of essentially skipping over a block of code if it does not pass a boolean expression. Java supports two conditional statements: `if`...`else` and `switch`.

if...else statement

`if(expr) { code }`

... means run the `code` block if `expr` is `true`

```
if (1 > 0) {
  System.out.println("hi");
}
//=> hi
```

When you need to test more than one case, you may use `else if`:

```
String name = "kittens";
if (name.equals("puppies")) {
    name += "!!!";
} else if (name.equals("kittens")) {
    name += "!!";
} else {
    name = "!" + name;
}
System.out.println(name);
//=> "kittens!!"
```

Ternary Operator

Java has a ternary operator for conditional expressions. You can think about the ternary operator as a concise

"if-else in one line":

```
int age = 12;

String allowed = (age > 18) ? "yes" : "no";

System.out.println(allowed);
//=> "no"
```

Truth-y & False-y

It's important to know that all of the following become `false` when converted to a Boolean:

- `false`
- `0`
- `""` (empty string)
- `NaN`
- `null`
- `undefined`

For example:

```
Boolean b = new Boolean("");
System.out.println(b);
//=> false
```

This can be vary helpful when checking if conditions exist, are undefined, or if variables don't hold value.

Demo: Comparison Operators (10 mins)

Comparisons in Java can be made on primitives using <, >, <=, and >=.

```
'A' > 'a'
//=> false

'b' > 'a'
//=> true

12 > 12
//=> false
```

Note that you *cannot* do:

```
12 >= "12"

// or

"Apple" > "Oranges"
```

Equality Operator ==

Check: Can you remember from the pre-work how to compare variables?

When verifying equality between primitives use double equal ==:

```
System.out.println(1 == 2);
=> false
```

But what about with Objects like strings?

A special note on Equality among Strings:

There are actually two ways to compare the equality of strings.

```
String blue = "blue";
boolean withSign = (blue == "blue");            //=> true
boolean withWords = (blue).equals("blue");      //=> true
```

Do you know which one of these would be preferred?

Well, lets do another example to show you which and why:

```
String blue = "blue";
String bl = "bl";
String ue = "ue";
System.out.println(bl+ue);                      //=> blue
boolean withSigns = (bl+ue == blue);            //=> false
boolean withWords = (bl+ue).equals(blue);       //=> true
```

Why isn't `withSigns` true? The print out looks the same.

Well, Objects and arrays are complex collections of values, and when we refer to them, we're actually referencing where they live in memory.`==` compares the place where the object was stored on the computer.

What this means is that Java doesn't care if they look similar. It only compares whether or not they are the exact same object in memory. In each of the cases above, when checking for equality, we're actually comparing two objects that are in two different places in memory.

`String blue` has a reference to where it is stored on the computer, and that is a different place than `String bl` is stored. They're not exactly "the same" according to `==`.

`equals()`, on the other hand, is a method that can be called on an instance(`str1`) of a String Object. And this method checks whether the `char` arrays in each String are the same, not whether the references are the same.

The long and short of it, use `equals` when comparing strings.

!=

There is also an `!=` operator, which is the inverse `==`.

Guided Practice: Boolean/Logical Operators (15 mins)

Logical operators will always return a boolean value `true` or `false`.

There are two "binary" operators that require two values:

- **AND**, denoted &&

- **OR**, denoted `||`

A third "unary" operator requires only one value:

- **NOT**, denoted `!`

&& (AND)

Do these with me!

The `&&` operator requires both values to the left and right of the operator to be `true` in order to return the entire statement as `true`:

```
boolean result = false;

if(true && true) {
  result = true;
}
System.out.println(result);
//=> true
```

Any other combination using the `&&` operator is `false`. What happens if I check `true && false`?

```
boolean result = false;

if(true && false) {
  result = true;
}
System.out.println(result);
//=> false

boolean result = false;

if(false && false) {
  result = true;
}
System.out.println(result);
//=> false
```

|| (OR)

The `||` operator requires just one of the left or right values to be `true` in order to return true.

So, now, if I do `true || false`, what will be returned?

```
if(true || false) {
  System.out.println("true");
}
//=> true

if(false || true) {
  System.out.println("true");
}
//=> true

if(false || false) {
    System.out.println("true");
}
```

```
//=> ... silence ...
```

Only `false || false` will return `false`

The `!` takes a value and returns the opposite boolean value, i.e.

```
!(true)
//=> false
```

The && and || operators use short-circuit logic, which means whether they will execute their second operand is dependent on the first.

This is useful for checking for null objects before accessing their attributes:

```
if(instructor != null && instructor.getName().equals("drew")) {
  System.out.println("davis")
}
```

In this case, if the first operand `instructor != null` is false, then the second operand `instructor.getName().equals("drew")` will not be evaluated. The expression is basically saying "we already know the whole && expression is false, because `instructor != null` is false. Why bother dealing with the second operand?"

This is also important because a `Null Pointer Exception` will be thrown if we try to call a method using "dot notation" on a null Object reference.

Introduction: Switch Statement (10 mins)

The switch statement can be used for multiple branches based on a number or string:

```
  String food = "apple";

  switch(food) {
    case "pear":
      System.out.println("I like pears");
      break;
    case "apple":
      System.out.println("I like apples");
      break;
    default:
      System.out.println("No favourite");
  }
//=> I like apples
```

In this case, the `switch` statement compares `food` to each of the cases (`pear` and `apple`) and evaluates the expressions beneath them if there is a match. It uses `String.equals` method in this case, or == in the case of primitives, to evaluate equality.

The default clause is optional.

Note: Breaks are important! If you don't put a break statement, the expression will continue to be evaluated for each following case. This might cause unintended consequences.

For example if you eliminate the break statements:

```
  String food = "apple";

  switch(food) {
    case "pear":
      System.out.println("I like pears");
    case "apple":
      System.out.println("I like apples");
    default:
      System.out.println("No favourite");
  }
```

The result would be:

```
I like apples
No favourite
```

If `food = "pear"` then the output would be:

```
I like pears
I like apples
No favourite
```

Thi is not exactly what we had intended.

Demo: Loops (15 mins)

In just about all programming languages, loop-ing exist. A loop is a statement or block of code that will continue to execute while or until a condition exists.

`while` loops, for example, will run a block of code **while** a condition is `true`.

Java has `while` loops and `do-while` loops.

The `while` loop is good for basic looping, but there's a possibility it will never get run.

```
while (true) {
  // an infinite loop!
}
```

Using a `do-while` loop makes sure that the body of the loop is executed at least once, because `while()` isn't evaluated until after the block of code runs.

```
int input = 0;
do {
  System.out.println(input++);
} while (input < 10);
```

You can use looping in combination with iteration: a way of incrementally repeating a task.

For example, using a for loop:

```
int iterations = 10;
for (int i = 0; i < iterations; i++) {
  System.out.println(i);
}
```

Notice the placement of the comma and semi-colons, and let's take a look at what each of the parts do:

1. `int i = 0;` is the **initialization** phase.

    - This is executed once, before the loop begins.
    - Note that `int i` is declared within this phase. This means that the lifespan of `i` is limited to within the for loop, which is a much cleaner, and leads to less problems down the line.
2. `i < iterations;` is the **termination** phase.

    - Every time the loop evaluates, it checks this statement.
    - If this statement evaluates to `false`, the loop terminates.
    - This is equivalent to the `while` section of the `do...while` loop.
3. `i++` is the **increment** expression.

    - This happens every time the loop evaluates.
    - This is equivalent to the `do` section of the `do...while` loop.
    - In this case, each loop, `i` is incremented by 1.

In android studio, you can use `fori+TAB` to automatically create an empty `for loop`.

Fizz Buzz - Independent Practice (15 minutes)

Fizz buzz is a game about division. Create a program that will iterate through numbers from 1 to 101 and log each number in the console.

- In the loop every time a number is divisible by **3**, instead of logging the number itself, the word "fizz" should appear.
- If the number is divisible by **5**, the word "buzz" should be logged.
- If the number is divisible by both **3** and **5**, then the word "fizzbuzz" should be logged.

Hint: Remember the *modulus* operator?

A typical output would look like this:

Solution

```
for (int i = 1; i < 101; i++) {

  if((i % 3 == 0) && (i % 5 == 0)) {
    System.out.println("fizzbuzz");
  } else if(i % 3 == 0) {
    System.out.println("fizz");
```

```
  } else if(i % 5 == 0) {
    System.out.println("buzz");
  } else {
    System.out.println(i);
  }
}
```

Conclusion (5 mins)

These are some of the foundational tools you'll use in many of your applications. You'll probably need to refresh yourself on the exact syntax a few times before you memorize it, but it's important to be able to remember, these core "control flow" concepts, in general, because they'll come up in pretty much every programming language you'll ever encounter.

1:11

Writing Advanced Functions

Note: *You can help each other, but everyone must turn in their own code*

Exercise

Requirements

You are supposed to write 10 functions that accept parameters. You can use control flow and multiple conditionals. When a function is created, please call it (from inside the `main` method) and test all the cases including *edge cases* (e.g. empty strings, null values, extremely big and small values). It is of high importance to practice and master Java fundamentals, which are the core of Android development. Java functions create the solid foundation, without which it is impossible to build high quality bug free apps.

- Fork this repository & clone your fork to your computer
- Inside the repo folder create a new IntelliJ project
- Add code to the IntelliJ project to complete the requirements
- Commit your changes & push them up to Github
- Submit a *pull request* and put your name and "Functions Advanced Lab" in the pull request title

Functions

1. Write a function that takes in a number and returns true if the number is even or false if the number is odd. You may use control flow and multiple conditionals.

2. Write a function that takes in two words and compares them. If the spelling is the same, the functions returns true, otherwise false.

3. Write a function that takes in a word and prints out its every letter on a new line before it sees the letter "w". If it encounters "w" the function should stop executing.

4. Write a function that generates a random number. Print it out. If the number is greater that 50, return it, otherwise return -1. You may use control flow and multiple conditionals.

5. Write a function that takes in 3 integers: day, month, and year and computes day, month, and year of the next day. The function should return a string (e.g. day:12 month:11 year:1988)

6. Write a function that takes in a word, omits every other letter in that word and returns a new word. You can use control flow and multiple conditionals.

7. Write a function that takes in a number greater than 5 and prints out all the numbers from 1 to that number except number 4. Please use a while statement.

8. Write a function that takes in 2 numbers, and checks if they are the same. If they are not the same, it adds 1 to every number and returns its sum. If the numbers are the same, it adds 2 to every number and returns its sum.

9. Write a function that asks the user to enter a country domain. If the user types in "us", the function

prints out "United States", if the user enters "de", the function prints out "Germany", if the user types in "hu" the response should be "Hungary". In all other cases the function should print out "Unknown". Please try to use a switch statement and make sure the function works with a user's input of lower and upper case strings.

10. Write a function that asks the user to type in a letter and prints out whether the letter is a vowel or a consonant. You may use control flow and multiple conditionals.

Deliverable

You are expected to create one Java file with 10 functions and a `main` method that calls all the other functions, testing all cases. This file should be in an IntelliJ project, and you should submit it via pull request.

1:12

Functions Practice Homework

> ***Note:*** *You can help each other, but everyone must submit their own code*

Exercise

Requirements

- Complete these problems on CodingBat:
  - http://codingbat.com/prob/p187868
  - http://codingbat.com/prob/p154485
  - http://codingbat.com/prob/p172021
- Test your code on the website
  - When you click "Go" it automatically runs a set of test cases and lets you know which passed and which failed
  - If you register (optional and free) and log in, CodingBat will save your progress
- Fork & clone this repo, create an IntelliJ project, and paste your completed code into the project
- Finally, push your code and submit a pull request; put your name in the title of the pull request

Bonus:

- http://codingbat.com/prob/p123384
- http://codingbat.com/prob/p191914

Deliverable

Your pull request including your code for each of the 3 required problems. If you have time, try to add the 2 bonus problems as well.

1:13

Storing Data in Arrays and Collections

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Create and manipulate arrays and Lists
- Access specific values in arrays and lists using `for loops` to iterate over them

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Work with basic data types and assign variables
- Create basic functions

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Gather materials needed for class
- Complete prep work required
- Prepare any specific instructions

Opening (5 mins)

So far, we have stored all of the information for our apps in individual variables. That works for a small amount of information, but what if we had to manage larger sets of data? For instance, what if we were a store and needed to print out a receipt of all the items a customer has purchased? Would we create a variable for every item? How would we know how many items the customer is purchasing before they even came in?

> In pairs, come up with a few more examples of where storing our data in a collection would be beneficial (data situations or example apps). Write them down on the desk. (1-2 minutes)

Introduction: Basic Arrays (5 mins)

Remember the box analogy I gave you? Data types are boxes that you put data in and variables are the labels for each box.

```
int var1 = 1;
int var2 = 2;
int var3 = 3;
int var4 = 4;
int var5 = 5;
int var6 = 6;
int var7 = 7;
```

```
int var8 = 8;
int var9 = 9;
int var10 = 10;
```

What if I had 200 `int` values? Would I have to create 200 variables? That's crazy. We can use an array to hold all these values.

An array is a container that holds a **fixed** number of values of **a single data type**. You've already seen an example in the `main` method.

An important concept to understand is an array is not necessarily a data type, it just holds data types in a structured way. This is also referred to a data structure. In a nutshell, a data structure manages how we interact with larger amounts of data. Java provides us data structures as `Objects`, so we don't need to worry too much about how they work, just yet.

Each item in an array is called an *element*, and each element can be accessed by it's *index*. The index of elements starts at 0. That means visually that:

INDEX	**0**	**1**	**2**	**3**	**4**
ELEMENT	x	x	x	x	x

Demo: Creating Arrays (5 mins)

Let's create an Array together:

Data Type	**Brackets**	**Variable name**	**Assign Value To**	**Instantiation**
int	[]	anIntArray	=	new int[]
double	[]	aDoubleArray	=	new double[]
String	[]	aStringArray	=	new String[]
boolean	[]	aBooleanArray	=	new boolean[]

```
class ArrayDemo {
  public static void main(String[] args) {
    //declares an array of integers

     // note datatype, followed by [] indicating array.
    int[] anArray;

    //allocates memory for 10 integers
    //note the need for the 'new' keyword and the need for a 'size'
    anArray = new int[10];

    //assign elements

    /*
    The best thing about arrays is that you can access very specific parts
    of the array by using the 'index' of the item you want. We've covered indexes
    during yesterday's lesson with loops. We'll be using what you guys learned
    about loops to help us go through the data inside arrays.
```

```
      So how do we put values inside this array?
      Similar to how we access values inside the array.
      > Important to note that arrayName[index] is the syntax we need to open
      that specific box.
      */
      anArray[0] = 111;
      anArray[1] = 222;
      anArray[2] = 123;

      //What about if i wanted to put the value 12345 in the 8th box?
      //How would I do that?


      //Lets print out the values that we just put in.
      //access elements
      System.out.println("Element at index 0: "+ anArray[0]);
      System.out.println("Element at index 1: "+ anArray[1]);
      System.out.println("Element at index 2: "+ anArray[2]);

      //What happens when I try to print out the value in the 5th position.

      System.out.println("Element at index 1: "+ anArray[4]);
      System.out.println("The array has a size of " + anArray.length);
    }
  }
```

There is also a shortcut for initializing arrays if you know the values ahead of time.

```
int[] myArray = {22,33,44};
```

Manipulating Basic Arrays

Let's take a look at some basic things you can do with an array.

You guys have already seen me do some of these things.

> Check: Let's say I wanted to declare an array variable myArray that holds 10 integers. What is the syntax for that? Write it down and I'll call on some people to explain what they did. Can I set the array's size using a variable?
>
> Check: Now, what if I wanted to assign an integer value in the 7th position in that array? Write down the syntax and I'll call on someone to explain.
>
> Check: Let's say we filled all the positions in the array with some number. I want to print out the value at position 3, 9, 10. Write down the syntax of that and we'll discuss.

Demo: Manipulating Arrays (5 mins)

Now you: Create a String array of *three* of your favorite things.

Out of bounds

What will I get if I wrote, `myArray[3]`? Write down your response, we'll discuss.

We get an exception called `OutOfBoundsException`

An Exception is thrown when the computer is asked to do something it can't do, like accessing index -1 or 4 in a 4-element array. Remember in Java, we start from 0. Exceptions bubble up, and unless caught (which we will talk about in a later lesson) can cause the program to quit.

It is important to know how to read and handle these exceptions when they arise. (You'll learn more about this later)

Demo: Problems with Arrays (10 min)

But what if I decide that actually, I want this list to include 4 things instead of 3? For example, let's go back to our favoriteList; I decide I really like "whiskers on kittens" and want it to be the second favorite thing, and just move everything down one position.

```
class Main{

  public static void main(String[] args){

    String[] favoriteThings = new String[3];
    System.out.println(favoriteThings[0]);
    System.out.println(favoriteThings[1]);
    System.out.println(favoriteThings[2]);

    //I want to change the size of my array to hold 4 things, but I already set it to
hold 3 things.

    String[] tempArray = new String[4];
    tempArray[0] = favoriteThings[0];
    tempArray[1] = "whiskers on kittens"
    tempArray[2] = favoriteThings[2];
    tempArray[3] = favoriteThings[3];

    //What's another way to do it?
    for(int i = 0; i < favoriteThings.length; i++){
      if(i == 1){
        tempArray[i] = "whiskers on kittens";
        continue; //what does this do?
      }
      else{
      tempArray[i] = favoriteThings[i];
      }
    }
    tempArray[4] = "bright";

    //But that's not our favoriteThings array, its a tempArray.
    favoriteThings = tempArray;

  }

}
```

Luckily, Java has provided something that does the hard work for us: collections.

Demo: Collections (15 mins)

Collections will not only provide us with convenience methods, allowing us to do more things with the data we have, but will also *automatically increase the size* if we need!

There are many different collection classes that we use just like data types, each of which provides some

distinct functionality, but for today, we're going to focus on just one of them.

ArrayList

Let's take the array we made of favorite things and convert it into an `ArrayList`.

```
  public static void main(String[] args) {
    // initialize ArrayList
      List<String> favoriteThings = new ArrayList<>();
    // add items
      favoriteThings.add("bright copper kettles");
      favoriteThings.add("warm woolen mittens");
      favoriteThings.add("brown paper packages tied up with strings");
  }
```

Note, the data type of each element is defined in angle brackets <>. This data type could be any object type. So, if you'd created a `Person` object, as you may have in the pre-work, you could create an ArrayList of Persons! (i.e. an `ArrayList<Person>`)

If you want to make an ArrayList of a primitive type, you need to use a "boxed" version of that type as the thing in the angle brackets. For example, if you want to store a bunch of `int`s, you would use an `ArrayList<Integer>`.

Manipulating a ArrayList

An ArrayList is an object, with methods, which makes it much easier to manipulate than a simple array. Check out the following methods.

```
  //to print. No need to explicitly convert it to a string first!
  //(because ArrayList has a toString() method, which automatically
  //gets called here)
    System.out.println("favoriteThings = " + favoriteThings);

  //add(item) -- adds to the end of the list
    favoriteThings.add("chocolate");
    System.out.println("favoriteThings = " + favoriteThings);

  //add(index, item) -- adds to the list at specified index and moves all other entries
over. Think: what you would have to do with a simple array to do that?
    favoriteThings.add(1, "warm woolen mittens");
    System.out.println("favoriteThings = " + favoriteThings);

  //set(index, item) -- replace the item at the given index with a new one
    favoriteThings.set(0, "tarnished copper kettles");
    System.out.println("favoriteThings = " + favoriteThings);

  //to search for an entry
    int indexOfIceCream = favoriteThings.indexOf("icecream");
    if(indexOfIceCream != -1) {
      String ic = favoriteThings.get(indexOfIceCream);
      System.out.println("ic = " + ic);
    }
    else {
      System.out.println("icecream not found");
    }

 //here's another good one: get number of things in the ArrayList
    favoriteThings.size();
```

Guided Practice: Iterating Through a List with For Loops (15 mins)

One more thing before we start talking about lists in Android: How do you iterate through a list?

Using the For Loop with Lists and arrays

Do you remember the syntax used in a for loop? We used it in the Control Flow lesson to print something to the command line a set number of times.

The for loop is also commonly used with arrays and collections to iterate through each of the elements and do something with each entry.

For example, let's create a list of 5 movies and iterate through it, printing each one to the command line.

```
public static void main(String[] args) {
  // initialize ArrayList
    List<String> favoriteMovies = new ArrayList<>();
  // add items
    favoriteMovies.add("Finding Nemo");
    favoriteMovies.add("The Theory of Everything");
    favoriteMovies.add("Eternal Sunshine of A Spotless Mind");
    favoriteMovies.add("Crash");
    favoriteMovies.add("Ip Man");

    for(int i = 0; i < favoriteMovies.size(); i++){
      System.out.println(favoriteMovies.get(i));
    }

    //Another way to do it, called a for-each loop
    for(String movies : favoriteMovies){
      System.out.println(movies);
    }
}
```

Independent Practice (15 mins)

Complete as many of the following challenges as you can in the next 15 minutes. Each challenge should be completed in its own method.

1. Find the size: a. Create an array of ints. b. Print the length of the array to the command line.

2. Concrete Jungle a. Create a ArrayList of New York City wildlife. b. Create a function that, given an List of Strings, prints for each element: "Today, I spotted a /*Thing here*/ in the concrete jungle!"

3. Create a method that, given an array of ints, return the sum of the first 2 elements in the array. If the array length is 1, just return the single element, and return 0 if the array is empty (has length 0).

Bonus 4. Create an method that, given an List of words (Strings), turns each word into Pig Latin. Our rules of Pig Latin are as follows: a. If the first character is a consonant, it is moved it to the end of the word and suffixed with an `ay`. b. If a word begins with a vowel you just add `way` to the end.

For example, `pig` becomes `igpay`, `banana` becomes `ananabay`, `twig` becomes `wigtay`, and `aardvark` becomes `aardvarkway`.

Conclusion (5 mins)

Quick review:

- List some differences between arrays and Array lists

Arrays and Lists are fundamental tools needed by anything that wants to store and manipulate collections of data. Now that you know how to use them, we can start creating apps that use those collections. Excited?

ADDITIONAL RESOURCES

- Oracle Java Docs - Arrays
- Android Docs - ArrayList
- Oracle Java Docs - The for Statement
- CodingBat - Array-1 examples
- Android Docs - Building Layouts with an Adapter
- Android Docs - Adapter

1:14

Debugging Fundamentals in Java

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Given sample code, set a breakpoint and look at the current state
- Given a stack trace, identify which lines are within your application
- Describe what a stack trace is saying

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Work with data types and variables
- Write functions that use control flow

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Read through the lesson
- Add additional instructor notes as needed
- Edit language or examples to fit your ideas and teaching style
- Open, read, run, and edit (optional) the starter and solution code to ensure it's working and that you agree with how the code was written

Opening (5 mins)

Even the best programmers make mistakes when writing code, and a large amount of development time is often spent fixing mistakes in code, also known as bugs. Therefore, this is called debugging! There are many tools that can be used for debugging, but today we will be concentrating on **reading a stack trace** and **using breakpoints**.

Introduction: Stack Trace (10 mins)

Making mistakes is a natural part of programming, so an absolutely crucial skill set to have is the ability to debug your code. The first thing we are going to look at is the stack trace. You have probably seen the a stack trace in the past few days without realizing it.

Let's take the following example.

```
public class Main {

    public static void main(String[] args) {
```

```
        printSomething(null);
    }

    public static void printSomething(String s){
        System.out.println(s.length());
    }
}
```

If we run this code, we get the following error in the console:

```
Exception in thread "main" java.lang.NullPointerException
    at Main.printSomething(Main.java:11)
    at Main.main(Main.java:7)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:497)
    at com.intellij.rt.execution.application.AppMain.main(AppMain.java:144)
```

While this might seem intimidating at first, you really only need to focus on a few key parts. First is the very top line, it tells you what the error actually was. In this case, we had a Null Pointer Exception, meaning we tried to call a method on a variable that was null.

The second thing to concentrate on are the lines that have our file name(s) in it, Main.java. This is called the Stack Trace because it shows the path of execution for your program. Since the errors in the code are in the files we write, those are the lines we concentrate on. The higher up in the stack trace, the more recent the execution.

The number after the colon next to the file name is the line number the error occurred on. In our example, the error was in Main.java on line 11.

Demo: Stack Trace (5 mins)

Let's take a look at another example.

```
public class Main {

    public static void main(String[] args) {
        int[] arr = new int[0];

        printSomethingArray(arr);
    }

    public static void printSomethingArray(int[] arr){
        System.out.println(arr[0]);
    }
}
```

If we run this code, we get the following error.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at Main.printSomethingArray(Main.java:13)
    at Main.main(Main.java:9)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
```

```
    at java.lang.reflect.Method.invoke(Method.java:497)
    at com.intellij.rt.execution.application.AppMain.main(AppMain.java:144)
```

The two errors we just saw, NullPointerException and ArrayIndexOutOfBoundsException are very common, so they are something you will probably see come up in your stack traces.

Introduction: Breakpoints (10 mins)

The second topic we are going to cover today is the Breakpoint. In general terms, a breakpoint is a position in the code you want to halt execution at.

Adding a breakpoint allows us to do a few very basic things:

- Trace execution of the code, step by step
- Inspect the value of all variables in scope

That second point can make debugging your code much faster. Instead of adding a lot of print statements to monitor your variables, you can just stop your code in place!

Demo: Breakpoints (10 mins)

Let's continue with the array example from the previous demo. Enabling breakpoints in IntelliJ (as well as Android Studio) is very simple. Simply click in the margin next to the line of code you want to halt the program at. You should see a red dot appear next to it. Go ahead and click the margin next to the line where we access index 0 in the array.

Next, you need to run the program in debugging mode. Instead of pressing the Run icon like normal, you need to press the Debug button.

The program should run, but a new window will pop up on the bottom, and you should see your code change slightly.

As you can see, the console now shows us the values of the variables, as well as any errors that might show up. Also, in the actual code it shows us what the values of the variables are!

Let's add one more breakpoint in main where we call the printSomethingArray method. If we initialize the array with a few values and re-run the program, you should see these values change, and the error should go away.

Independent Practice: Using Breakpoints (5 mins)

Modify the program to use a for loop to print all elements in the array. Try using the different tools that are provided with breakpoints to get used to using them.

Independent Practice: Bug Fixing (10 mins)

You have been provided the starter code for a simple integer calculator. Given that the user enters two numbers, a result is shown depending on what choice the user made from the main menu.

However, the division method is throwing an error in certain cases.

Hint: The problem occurs when the divisor is a certain number

Find the bug using the techniques we learned today, and fix the bug.

Conclusion (5 mins)

Learning to debug well is a crucial skill, and what we covered today is a great way to start. Later in the course we will be covering some Android-specific tools, but what we learned today can apply to almost any programming language you might learn in the future.

1:15

Writing Functions that use Iteration and Collections

> ***Note:*** *This should be done independently.*

Objective:

- Demonstrate understanding of arrays and Lists

Exercise

Requirements

Write 9 functions that accept parameters that use combinations of control flow and multiple conditionals. Some of these functions will also require you to iterate over data collections (Arrays, ArrayLists). When a function is created, please call it in the main method and test all the cases including edge ones (e.g. empty strings, null values, big and small values). It is of high importance to practice and master Java fundamentals, which are the core of Android development.

Functions

1. Write a function `stringLengthOrValue` that accepts one `String` parameter. This function should print the value of the `String` parameter to the command line if the length of the `String` is greater than 5. If the length of the `String` is less than 5, print the length of the `String` parameter.

2. Write a function `reversedOrder` that accepts no parameters. This function should create an `int` array of size 10 and assign values 0-9 to each index in the array by using a `for loop`. It should then print out the values in reverse order using a separate `for loop` inside the function.

3. Write a function `maxValue` that accepts one `int` array parameter. This function should loop through the array to return the max value in that array. If the array is of size 1, the max value is the only item in the array. If the array is of size 10, how do we keep a record of the current max value when looping through the array?

4. Write a function `sumOfValues` that accepts a `double` array parameter. This function should loop through the array and ADD all the values in the array. It should then return the sum of the values, and print the returned value from within the calling function.

5. Write a function `charsToString` that takes in a `char` array parameter. This function should loop through the array and concatenate each `char` value into a `String`. It should then return the `String` that was created, and print the `String` from within the calling function.

6. Create a List of type `String` with the variable name `myStringList`. Add at least 5 unique `String` values to the list, and return it. Then print the values in the List from the calling function.

(Think about how to generate these strings using a loop)

7. Write a function `reversedStringOrder` that accepts a List parameter of type `String`. The function should loop through the List and print each `String` in reverse order to the command line. Use the List you created in problem 6 as the parameter you give to the function.

8. Write in a function `printOrAdd` that accepts a List parameter of type `String`. The function should print all values in the list if the size of the List is equal to 10. If the size of the List is less than 10, add `Strings` of your choice to the list until the size equals 10. Finally, return the List and print its values from the calling function.

9. Write a function `findMiddle` that accepts an `int` array parameter. The function should then return the value from the array located at the middle index, and print it from the calling function.

Deliverable

You are expected to turn in a Java program with 9 functions. Be sure to test your functions thoroughly!

Resources

- Collections

1:16

Project-0: Rock Paper Scissors

Overview

Let's put your Java knowledge to use! You're going to be creating a basic version of Rock Paper Scissors that you play against the computer in the console. The game consists of two main features:

- Play Rock Paper Scissors against a computer player
- View previous game history

Hint: Use a random number generator to pick the computer's choice.

Requirements

Your work must:

- Have a main menu with options to enter "play" or "history":
    - If the user enters "play", they should be able to play Rock Paper Scissors against the computer
    - If the user enters "history", the program should display previous game history
- Handle invalid user input
- Use Arrays or ArrayLists to store game history

Bonus:

- Handle incorrect capitalization of otherwise valid user input (rock, Rock, RoCk, ROCK, etc.)
- Store game history across sessions

Code of Conduct

As always, your app must adhere to General Assembly's student code of conduct guidelines.

If you have questions about whether or not your work adheres to these guidelines, please speak with a member of your instructional team.

Necessary Deliverables

Submit a pull request with a Java program that meets the above requirements.

Below, you can see sample output:

```
Welcome to Rock Paper Scissors!
```

```
MAIN MENU
=====
1. Type 'play' to play
2. Type 'history' to view your game history
Type 'quit' to stop playing

play


Type in 'rock' 'paper' or 'scissors' to play.
 Type 'quit' to go back to the Main Menu

rock
Computer picks: scissors
User picks: rock
You win!

Welcome to Rock Paper Scissors!

MAIN MENU
=====
1. Type 'play' to play
2. Type 'history' to view your game history
Type 'quit' to stop playing

play


Type in 'rock' 'paper' or 'scissors' to play.
 Type 'quit' to go back to the Main Menu

paper
Computer picks: scissors
User picks: paper
You lose!

Welcome to Rock Paper Scissors!

MAIN MENU
=====
1. Type 'play' to play
2. Type 'history' to view your game history
Type 'quit' to stop playing

history
=== GAME HISTORY ===
WIN: Player-rock computer-scissors
LOSS: Player-paper computer-scissors

Welcome to Rock Paper Scissors!

MAIN MENU
=====
1. Type 'play' to play
2. Type 'history' to view your game history
Type 'quit' to stop playing

quit
```

Suggested Ways to Get Started

- Don’t hesitate to write throwaway code to solve short term problems
- Read the docs for whatever technologies you use.** Most of the time, there is a tutorial that you can follow, but not always, and learning to read documentation is crucial to your success as a developer
- Write pseudocode before you write actual code.** Thinking through the logic of something helps.

Useful Resources

- Random number generator
- Rules, history of Rock Paper Scissors

1:17

More Functions Practice Homework

Introduction

Sometimes we all need a bit of guidance in our life. For this assignment, you will create a simple "magic eight ball" that runs on the command line. Like a real magic eight ball, it will give a random answer to the questions on our minds.

The user won't actually type their question into the command line, but the game should instruct the user to think about their question. Then the game will give a randomly selected response, as if it knew what question they were thinking. Then the game will ask the user if they want to play again, and then the user will type yes or no, and the game will repeat or end as needed.

Exercise

Requirements

- Create a method (for example, `getRandomNumber()`) that returns a random number between 0 and 8.

- Create a method (e.g. `shakeTheBall()`) that calls the first method to get a random number, uses a `switch` statement to pick a response based on that number, and prints the response to the console.

  Here are the traditional magic eight ball responses. Feel free to pick 9 of these to use, or come up with your own:

  - It is certain
  - It is decidedly so
  - Without a doubt
  - Yes, definitely
  - You may rely on it
  - As I see it, yes
  - Most likely
  - Outlook good
  - Yes
  - Signs point to yes
  - Reply hazy try again
  - Ask again later
  - Better not tell you now
  - Cannot predict now
  - Concentrate and ask again
  - Don't count on it
  - My reply is no
  - My sources say no

        - Outlook not so good
        - Very doubtful
- Create a method (e.g. shakeAgain()) that asks if the user wishes to play again, and responds accordingly. Use the Scanner class. User input should be case insensitive, and spaces before or after the words should not matter.

- Start the "shake" from the main method.

Bonus:

- Add a countdown, with delay, so that the trepidation is high when the answer is given.
- Create a visual representation of the magic eight ball - maybe some ascii art.
- Make it more interactive by adding a second level of questioning.

Starter code

There is no starter code! Please create a new IntelliJ project inside your forked/cloned repo.

Deliverable

A pull request with your completed code. We expect to be able to ask a question and receive a response from the powers that be, well, the magic algorithms anyway :)

Additional Resources

- A link to the Wikipedia page on Magic 8 Balls

2:1

Organizing Information

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Compare array and linked lists
- Create and manipulate Java ArrayLists and LinkedLists
- Describe and implement a Hash Map

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Understand basic Java data types
- Be familiar with the `new` keyword

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Gather materials needed for class
- Complete Prep work required
- Prepare any specific instructions

Opening (5 mins)

One of the most commonly used libraries used in Java is the Collections class. We have already used Arrays and Lists. Today we will be covering in more detail the following popular collections: Lists and Maps. We will be learning about and comparing some of these today to understand their strengths and weaknesses, and how you should decide which to use. Once you have a better understanding of how to organize the information in your apps, you will be able to write much cleaner and more efficient code.

Introduction: Lists(10 mins)

First we will cover Lists, one of the most popular collection types. We know the basics of lists. They hold groupings of data, and provide a large set of helper functions that make manipulating the data much easier. You can search, sort, add and remove with ease. One of the major drawbacks of Lists are that they contain a larger overhead than a simple array (more memory, slightly slower due to all the work behind the scenes).

The two most commonly used types of lists are the ArrayList and the LinkedList. As the name implies, ArrayLists are backed by an array to hold the data. As we know, one major advantage over arrays is that you

don't need to manually create larger arrays if your data set grows. Keep in mind that the ArrayList is taking care of this for you, so the operation isn't free. Therefore ArrayLists aren't the best choice for rapidly growing or shrinking sets of data.

A LinkedList is backed by a collection of objects linked together by pointers to each memory location. While we won't be going over how these are implemented, it is important to know that each element in the list is stored in empty spots in memory, so there is no need to worry about the changing size of the data set. Data is simply inserted and removed like links in a chain. Accessing a specific element in a linked list is generally slower than an array, because they aren't stored in sequential memory.

Demo: Lists (5 mins)

We're going to begin with reviewing how to define a list.

```
List<String> myList = new ArrayList<>();
```

The process for a LinkedList and ArrayList is the same, just with the class name changed.

```
List<String> myList = new LinkedList<>();
```

That's it! Notice the type `String` between the brackets must match the type of data you want to store. Now you have an empty list ready to work with. To insert items, we do the following.

```
myList.add("Hello");
myList.add("Test");
```

There are many methods you can use to manipulate the list now. Let's take a look at a few.

```
myList.sort();
myList.contains("Test"); //Returns true because "Test" is in the list
myList.indexOf("Test"); //Returns 1, the index of "Test"
myList.get(1); //Returns "Test", because it's at index 1
myList.size() //Returns 3
```

As you can see, Lists make working with collections of data much more straightforward than arrays.

As a side note, you can actually convert between arrays and lists very easily.

```
String[] arr = {"one","two"};
List<String> stringList = Arrays.asList(arr);
```

Independent Practice: Lists (10 mins)

The students must create a LinkedList containing at least 10 Integers of their choice. Afterwards, they must test to see if even values exist in the list, and remove the elements that exist. Finally, print the size of the list.

Introduction: Maps(10 mins)

The next Collections type we will cover is the Map. A map is similar to a list, but each entry contains two parts: A key and a value. The key is unique in the map, think of it like the index in an array. Each key maps to a certain value, but there can be duplicate values in a map. Keys can be any kind of Object, but they are often Strings.

One of the most popular implementations of a map is called the HashMap. A HashMap is a map, as described above, where the data is stored in an array. The index where the data is stored is generated by something called a hash function. Basically, you give the HashMap the Object you want to use as a key, and it returns an integer to use as the index. When you try to retrieve the value from the map, it uses the same function to generate the same index. Strings are often used as the keys for a map.

Collision (OPTIONAL 10 minutes)

The hash function isn't flawless, what happens if it gives us the same index for two keys? This is called a collision.

Java handles this by using a structure like a LinkedList in place of the actual Objects in the HashMap. If a collision occurs, the object is added onto the end of that list.

Demo: Maps (10 mins)

Let's try implementing a HashMap. Pretend we have an inventory system for a store. Each item name has a quantity associated with it.

```
HashMap<String,Integer> inventoryMap = new HashMap<String,Integer>();

inventoryMap.put("paper towels",55);
inventoryMap.put("light bulbs",100);

Integer retrievedNum1 = inventoryMap.get("paper towels"); //55
Integer retrievedNum2 = inventoryMap.get("light bulbs"); //100

inventoryMap.remove("paper towels");
```

As you can see, a Map isn't that different from a List. You can add and retrieve objects. One major difference is that you have no direct control over the underlying data structure. There is no way for you to sort or control the order of the elements in the Map.

Independent Practice: Maps (10 mins)

Let's create a HashMap that represents a dictionary. The key will be the word, and the value will be the definition. We must add the following value to the map, retrieve it, and print it to the console.

For example:

"apple":"A fruit from a tree" "lake":"A large body of water"

Add a few more words of your choice.

Independent Practice: Organizing Information (15 minutes)

The students will do the following:

- Create an **array** of **Integers** containing the values: 1,2,3,4
- Create a **LinkedList** of **Strings** with the values: "One","Two","Three","Four"
- Create a **HashMap**, using a loop, that uses the keys {"One","Two","Three","Four"} taken from the list, and the Integer values 1,2,3,4 taken from the array
- Print out the HashMap size after adding the above items to it.

Conclusion (5 mins)

We've covered the major components of the Collections class and how to use them. From sorting to searching, collections make organizing your data in the app very fast and easy. Arrays, lists and maps will become integral parts of your apps as you continue development, and working with them will become second nature. As you have seen in previous lessons, Android provides built-in tools to help integrate your list and array data directly into the Views on your apps.

ADDITIONAL RESOURCES

- Oracle: Array
- Oracle: List
- Oracle: HashMap

2:2

Classes

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Describe what classes and objects are
- Describe what object properties and methods are
- Demonstrate and explain instantiation
- Write getter and setter methods for a given class

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Recall basic knowledge of variables
- Recall basic knowledge of Java data types
- Recall basic knowledge of method creation

Opening: What are classes? (5 minutes)

In computer programming, there is this concept called Object Oriented Programming. It is where we as programmers want to break up the logic for some complex idea into smaller more manageable, meaningful and reusable parts. These reusable parts are what we call *Objects*. In Java, almost every data type we've used is an extension of the *Object* class.

Introduction: What are classes? (15 minutes)

Objects are special pieces of data that have properties and functions contained inside of them. We use these objects to divide our code into separate responsibilities. We can then use these objects from other code to execute their responsibility. How the object accomplishes its responsibility is irrelevant to the calling code. We'll call this a black box. We don't care how it works, as long as it reliably gives us the right answer.

We only have to worry about the inputs and outputs of the object, and let the object worry about itself and how it works. For example, with a `String` object we are able to call the `.split()` method to break it into substrings (e.g. `myString.split(",")` would break the string apart at each comma). We don't care *how* it breaks apart the String, just that it reliably returns the correct result.

This has an advantage for whomever is implementing the object as well. They can change the internals, perhaps make it more efficient, without breaking compatibility with any code that uses it already.

We give these objects a template using the concept of `classes`. A `class` is a special *type* that can be user-defined to ensure every object of its type contain all the properties and methods of that class. Unlike data

types which have intrinsic value - i.e an `int` is a number, `char` are characters - classes have programmer-defined values. For example, a **Car** class could hold the `double` value of **speed** and also `String` value of **model**.

Check: In your own words, explain what is an object? What is a class?

Introduction: Instantiating an Object (10 minutes)

Creating an object of a class is called *instantiation*. This means we are creating a new object in memory. This is done by using the `new` keyword and a special method called a ***constructor***. Constructors are methods whose name *exactly* matches the name of the class. They can include some parameters that we can pass to our new object. They set up the initial state of the object.

Constructors are used only when an object is instantiated. Their sole purpose is to help create an instance of a class. In contrast, the purpose of other methods is much more general: execute Java code. Note the difference between constructors and methods.

Demo: Defining a Class (10 minutes)

The paradigm of Object Oriented Programming allows programmers to compartmentalize specific bits and pieces of code so that we can reuse this code while also hiding parts of it within each class. In the **Car** example, we can create new Cars and give them different model names but have an internal concept of *speed* that only the Car itself is aware.

Variables and other objects defined within an class are known as attributes or ***member variables***. Java convention dictates that these variables be named in camel case starting with an 'm' denoting that it is a ***member variable***.

Functions defined within a class are known as ***methods***. Java convention dictates that methods also be named in camel case. Since methods are the actions your object can perform, best practice is to put a verb in your method names which clearly explains what the method does.

```
public class Animal {
    String mName;
    float mWeight;

    public Animal(String name,float weight) {
        mName = name;
        mWeight = weight;
    }

    public float getWeight(){
      return mWeight;
    }

    public void setWeight(float weight){
      mWeight = weight;
    }

}

// creating a new object of type Animal
Animal aMoose = new Animal("Moose", 500.0f);
```

Check: What are methods? How should they be defined and written?

Guided Practice: Defining a Class (10 minutes)

Ok, time for you to write your own class!

Create a new class using the syntax your learned above. Your class should be modeling a **Color** object. Your **Color** must have a name. All of these must be set during construction.

```
public class Color {

    String mName;

    public Color(String name) {
        mName = name;

    }

}
```

Note: do a few more examples.

In basic Java programs, you need a `static main()` method to be able to run the program, instantiate a `Color` object inside the main method in your Main.java class.

```
public static void main(String [] args) {

}
```

Inside of this method, instantiate a Color object with a name:

```
    public static void main(String [] args)
{
        Color pink = new Color("pink");

}
```

Introduction: Getters and setters (5 minutes)

In Java, there is a convention used called *getters and setters*. It is not always a good idea to make *member variables* public and so instead methods are defined to allow access.

By definition, getters "get" or return stored information from an instance of a class; setters assign information / data to an instance of a class.

Demo: Getters and setters (5 minutes)

In our color object, we simply want to allow read access to the mName variable and so we would define a method called `getName()` within our `class`.

```
public String getName() {
  return "The name of this color is " + mName;

}
```

Java naming convention dictates that getters should start with 'get' and then the name of the variable (minus any leading letters). The only exception is for getters with return type `boolean`, it is convention to name

these as `isBoolean()` or `hasBoolean()`. For example our car might have a method: `hasLowFuel()`.

Perhaps we also to allow public access to change the `mName` variable, we would define a setter. The naming convention here is similar to the getter methods but with starting with 'set'

```
public void setName(String name) {

  // With a setter, we have the opportunity to check/verify the input
  if (newName.isEmpty()) {
    System.out.println("Name can't be empty!!!")
  }
  else {
    mName = name;
  }
}
```

Check: In your own words, write down the difference between getters and setters.

Independent Practice: Getters and Setters (10 minutes)

Create a few more classes with a constructor, getters, and setters.

Conclusion (5 minutes)

Throughout much of our work so far in this course, we have seen classes being used, and object being instantiated without understanding the reason behind all of it. Hopefully today's lesson helped clarify the mystery behind the code being automatically generated by IntelliJ. In an upcoming lesson, we will be discussing the `extends` keyword which allows us to copy a class and it's accessible variables and methods to allow us to modify a class we might otherwise not be able to modify. This is a concept known as subclassing.

ADDITIONAL RESOURCES

- Classes & Objects
- Java Getters vs Setters

2:3

Organizing Information - Challenge Questions

Introduction

> ***Note:*** *This should be completed independently.*

In this lab, you will be completing a series of 3 challenge questions that involve data collections. These questions are very similar to what you would see in an interview, so this is great practice. Think through each problem before you start creating code, possibly writing out pseudocode if it will help you. Run your code against the included test cases when you are finished. **Feel free to look up any Java documentation to help you (ie arrays, Lists, etc.)**

Exercise

Requirements

- Complete each question in its own provided method
- Your methods must pass all of the included test cases

Bonus:

- Complete the bonus question (Marked as Bonus)
- Optimize your methods to be more efficient

Starter code

An **intelliJ project** called ChallengeQuestions has been provided for you that includes a Main.java file. Each method has a comment above it that contains the problem. Complete each problem in that method (although you can write additional methods if it will help in your solution). Test each method in your code!

Deliverable

The completed Java file with solutions to the questions.

2:4

Creating Classes Lab

Exercise

> ***Note:*** *You can help each other, but everyone must submit their own code.*

In this lab you will be using your knowledge of classes to create two classes that will be used in the main method. To calculate

Requirements

StopLight Class

Define a `StopLight` class that includes the following:

- Member variables - just one: `mLightColor`
- Constructor: takes no input and sets `mLightColor` to whatever default value you choose
- Methods:
    - Three that take no input and return a boolean based on the current value of `mLightColor`: `isRed()`, `isYellow()`, `isGreen()`
    - One *setter* method named `setLightColor()` that takes an input, and assigns the value of that input to the `mLightColor` variable. Make sure to handle when invalid input is provided in a call to this method.

Car Class

Define a `Car` class that includes the following:

- Member variables: `mColor`, `mBrand`, `mTopSpeed`
- Constructor: takes 3 inputs for color, brand, and top speed, and assigns the values to the corresponding member variables
- Methods:
    - `go()`, `slow()`, `stop()` - each should take no input, return nothing, and print something to the command line that describes what the car is doing
    - *Getter* methods that return the color, brand, and topSpeed of the car (one for each variable)
    - *Setter* methods that set the color, brand, and topSpeed of the car (one for each variable)

Use Your Classes!

When you create your IntelliJ project, check the "Create project from template" box and select "Command Line App". This will create a Main.java file for you which contains a `Main` class with a `publice static void main()` method. This is the method that will execute when you run your project.

Inside your `main()` method, do the following:

- Instantiate a `StopLight` object using the constructor you wrote, and assign it to a variable of type

`StopLight`.
- Instantiate a `Car` object using the constructor you wrote, and assign it to a variable of type `Car`.
- Create a loop that will run 30 times. Inside the body of the loop:
    - Change the color of the stop light. It should follow the real-life sequence - if the light is currently green, change it to yellow. If it's yellow, change it to red. If it's red, change it to green.
    - Make the car react properly to the new light color using the methods you defined for the `Car` class.

Bonus:

- Create an array or ArrayList of 20 cars and make them all react to the stop light.
- Randomly have a car run the red light.

Starter code

There is no starter code. Create a new project in IntelliJ for this lab inside your forked/cloned repo.

Deliverable

Pull request from your forked repo on GitHub that contains your implementation of the `StopLight` class, the `Car` class, and the `main()` method that uses the classes you created.

Additional Resources

- Oracle: Defining a class
- Oracle: Instantiating a class

2:5

Classes Homework

> ***Note:*** *You can help each other, but everyone must submit their own code*

Exercise

You're going to create some classes that represent parts of a simple music player app.

Requirements

- Fork & clone, then start a new IntelliJ project in the repo (no starter code)
- Create 4 classes: **Song, Playlist, User, Main**
    - Right click on the "src" folder in Project menu > New > Java Class
    - Each class must go in its own separate Java file
    - Remember that by convention, class names are capitalized
    - For all the member variables you need to create, select an access modifier that will prevent code from other classes from changing them
- **Song** class requirements
    - Define 3 member variables: `mSongName`, `mArtistName`, `mAlbumName`
    - Define a *constructor* method that takes song name, artist name, and album name as inputs, and assigns those values to the corresponding member variables
    - Define a method `play()` that just prints the name, artist, & album to the command line (since we don't know how to actually play media files yet)
- **Playlist** class requirements
    - Define 2 member variables: `mName` and `mSongs`, where the latter is an `ArrayList` of `Song` objects
    - Define a *constructor* method that does the following:
        - Takes playlist name as an input and assigns it to the corresponding member variable
        - Initializes `mSongs` as a new, empty `ArrayList`
    - Define a method `getName()` that returns the playlist's name as a String
    - Define a method `addSong(Song song)` that takes a `Song` object as input and adds it to the list
    - Define a `playAll()` method that calls the `play()` method from each `Song` object in the list
- **User** class requirements
    - Define 2 member variables: `mName` and `mPlaylists`, where the latter is a `HashMap` that uses playlist names as keys and the corresponding `Playlist` objects as values
    - Define a *constructor* method that does the following:
        - Takes user name as an input and assigns it to the corresponding member variable
        - Instantiates `mPlaylists` as a new, empty `HashMap`

    - Define a method `addPlaylist(Playlist playlist)` that takes a `Playlist` object as an input and adds it to the `mPlaylists` map (hint: use the `getName()` method from the `Playlist` object to get the key to use for your map)
    - Define a method `getAllPlaylistNames()` that returns a collection of all the user's playlist names
    - Define a method `getPlaylistByName(String name)` that returns the `Playlist` from the collection that matches the `name` parameter. If nothing in the collection matches that input, return `null`.
- **Main** class requirements

    - Create your `main()` method here - this is where you create and manipulate instances of the other classes; do the following steps inside `main()`
    - Create a new instance of `User`
    - Create at least 5 new instances of `Song` and assign each to a variable (You can come up with creative values or just use "song1", "artist1", etc.)
    - Create at least 2 instances of `Playlist`, add multiple `Song` objects to each, then add those `Playlist` objects to your `User` object
    - Use the methods you defined and print out the names of your user's playlists, then "play" each song in each playlist so the songs print out as well

Bonus:

- Add methods to each class which can be used to edit the values of the member variables
- Add a method to **User** to remove a `Playlist` by name
- Call your bonus methods from `main()` to make sure they work properly (hint: if you change a playlist's name, you'll need to first remove it from `User` by specifying the old name as a key, then re-add it using the new name as the new key)

Deliverable

Make sure your code runs! Push to Github and submit a pull request.

2:6

Singleton Design Pattern Mini-Lesson

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Explain the motivation for using the Singleton Design Pattern
- Implement the Singleton Design Pattern

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Create classes and subclasses

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Gather materials needed for class
- Complete Prep work required
- Prepare any specific instructions

Opening (5 mins)

We've learned the basic syntax of Java and the mechanisms of inheritance, but now it's time to dive into *design patterns*, which are not a language-specific syntax, but instead a conceptual approach that developers can implement in any object-oriented language.

The *singleton* design pattern is useful when you need to access an instance of a class from multiple scopes, or different sections of your app, but you want changes made in one scope to persist to the other scope. It is also useful when you are dealing with a class that includes a large amount of data that must be loaded into memory each time an instance of the class is created. What might happen to your phones memory and performance if you create many instances of such a class? Would it be beneficial to limit the number of instances?

Introduction: Static vs Non-Static (10 mins)

In Java, you will occasionally see a modifier on variables and methods called `static`. This means two different things depending on if it's used on a method or variable.

Static methods

A static method makes that method accessible without instantiating an object first. We have seen an example of this when using methods from the Math class:

```
double myAnswer = Math.sqrt("2");
```

`sqrt()` is a static method within Math.

Static variables

Static variables are variables that are only created once in memory across all instances of a class. We will see an example of this in the next demo.

> Check: Ask the students to give an example of a static variable we've used.

Demo: Singleton Design Pattern (20 mins)

One example of using both static methods and variables is in a design pattern called a Singleton. A singleton is a class whose data is only loaded once into memory, but is accessible anywhere an instance of the class is loaded.

Let's pretend a town only has one School, a class we will make that contains teachers and students. We only want one instance of the school.

```
public class School{
  private static School school = null;
  private static LinkedList<String> teachers;
  private static LinkedList<String> students;

  private School(){
    teachers = new LinkedList<String>();
    students = new LinkedList<String>();


  }

  public static School getInstance(){
    if(school == null){
      school = new School();


    }
    return school;


  }

  public void addTeacher(String teacher){
    teachers.add(teacher);


  }

  public void addStudent(String student){
    students.add(student);


  }
```

```
  public LinkedList<String> getStudents(){
    return students;


  }

  public LinkedList<String> getTeachers(){
      return teachers;


  }


}

public static void main(String[] args) {
    School school = School.getInstance();
    school.addStudent("Bobby");

    addAnotherStudent();

    System.out.println(school.getStudents());


}

public static void addAnotherStudent(){
    School school = School.getInstance();
    school.addStudent("Joe");


}
```

Check: Ask the students what would happen if we made addTeacher or addStudent static.

Independent Practice: Make your own Singleton (20 minutes)

Now we are going to practice what we learned today and implement the singleton design pattern. Work alone or in small groups, and create a Singleton class - you can be creative and have it represent a real-world concept, or simply call it "MySingleton" - that's up to you.

In your singleton class:

- Follow the pattern of the *constructor* and `getInstance()` methods in the demo code above
- You don't need to add linked list member variables like the demo, but add a private String member variable and include a getter and setter method for it

Create a Main class:

- Create a `main()` method
- Create a method named `addMessage()`
    - In this method, create an instance of the singleton class
    - Use the object's setter method to save a message to its String variable
- Create a method named `viewMessage()`
    - In this method, create another instance of the singleton class
    - Use the object's getter method to access, then print the value from its String variable

- Finally, in your `main()` method, call both `addMessage()` and `viewMessage()`. Remember that those methods each have their own *scope*. Does the value printed in `viewMessage()` match what you added in `addMessage()` even though they operate in different scopes?

Here, we demonstrated how a singleton can persist data across different *scopes*. In Android, we will use the same approach to persist data across different screens or sections within the app.

> Check: Were students able to create the desired deliverable?

Conclusion (5 mins)

You will use the singleton concept frequently in your Android apps. There are many times you will want only 1 instance of a particular class in order to persist data across screens, or sections of your app. In addition, some objects in Android are very large and take up a lot of memory, so limiting them to 1 instance can make your app perform better.

2:7

Debugging in Android Studio

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Add logging statements to code
- Show toast messages

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Have Android Studio installed
- Have at least one emulator image downloaded

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Gather materials needed for class
- Complete Prep work required
- Prepare any specific instructions

Introduction: Logs and LogCat (5 minutes)

Android provided a way to view messages sent from a device. This is called logcat.

An app can send messages to the logcat system to provide useful information to a developer. For instance, if an app crashed, the app would send a message saying that it crashed and that it crashed at a particular line of code!

Demo: What do Log messages look like? (5 minutes)

Like this:

```
W/EmailActivity: Could not fetch emails successfully.
```

Let's break this down.

`W`: The type of message (discussed in the next section). This can be V, D, I, W, or E.

`EmailActivity:`: This is the tag of the message. A tag is just a way of saying where you are in the code. Usually, the tag says what screen is currently visible and/or what method is currently being called. There's no science to it; it should be descriptive so you know what the message relates to.

`Could not fetch emails successfully.`: This is the message. This describes what is being logged, and why.

Introduction: Types of Log messages (5 minutes)

There are 5 types of Log messages. In order of severity:

- Verbose
- Debug
- Info
- Warning
- Error

Verbose: Generally, the most descriptive. Used to log as much info needed for a particular section.

Debug: Probably the most useful. Used to show debugging messages, like variable values or noting when you hit certain parts of the app.

Debug messages are visible on debug versions of an app. Release versions do not show debug messages.

Info: Used to show informative messages. Messages like "successfully connected to server" or "Could not delete item from list."

Warning: Use this when something that should not happen just happens (or may happen), but it is not a full fledged error. An example would be warning the developer that they are using too much memory, and are in danger of crashing the app.

Error: Used when an error happens. These show up as red text in the logcat.

Demo: Logcat (15 minutes)

Let's show how to open LogCat and how to filter messages to be useful to them.

Using Log statements

Sometimes, for any reason, you will need show your own messages while you are running your app.

To do this, you would use Log Statements.

A Log Statement is a line of code that, when run, it displays a given message to the logcat. Here's how to send a debug log to logcat:

```
Log.d("MainActivity", "The onCreate() method was called");
```

Let's break this up:

- All Log statements belong to the static class, Log.

- The Log class contains methods that match the types of Log messages:
    - Log.v() for Verbose
    - Log.d() for Debug
    - Log.i() for Info
    - Log.w() for Warning
    - Log.e() for Error
- Each of these methods take two String parameters: the Tag and the message

Guided Practice: Adding log statements (10 minutes)

Using the provided sample code, which is filled with functions calling each other, let's add log statements to the code.

> Instructor Note: The teacher will ask the students about expected output, what logs show, etc.

Introduction: Displaying messages to the User (10 minutes)

Toasts

Toasts are messages that pop up to the user, showing the user information that is important to them.

For an email client, for example, the toast "Email sent" would be shown after the email is sent successfully. A lot of times, you would see a toast pop up, saying "No Network Connection" if you try to do something that requires internet.

To display toasts, you do the following:

```
Toast
  .makeText(MainActivity.this, "Saved email to drafts folder.", Toast.LENGTH_SHORT)
  .show();
```

As usual, let's break this up:

- The class Toast has a static method called `makeText`. This method takes three parameters:
    - Context: Usually the Activity.
    - The message to show
    - The amount to time the toast should be shown. Either LENGTH_SHORT or LENGTH_LONG, which is 2 seconds or 3.5 seconds, respectively.
- The `makeText` method creates a new Toast object, but does now show it on the screen. So, you have to call Toast's `show` method.

Guided Practice: Adding toasts (10 minutes)

Using the provided sample code, which is a screen with four buttons, lets add a few toasts!

Conclusion (5 mins)

- What is a log statement?
- What is a toast?

Additional Resources

- Toasts
- Log Statements

2:8

Intro to XML

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Identify what XML is and what it is used for
- Describe the basics parts of XML: elements, attributes, XML namespace
- Describe how an element can contain child elements
- Describe how empty elements work
- Create a basic XML file

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Edit files in a text editor

Opening (5 min)

To create an Android app you need not only Java, but XML as well. XML is the language that is used to write:

- a Manifest file (the contract where all components of the app are registered), which is located in the folder manifests/
- layout and values files that can be found in the folder res/

Let's go over some simple vocabulary that we'll need to know:

Parser - a program, usually part of a compiler, that receives input in the form of sequential source program instructions, interactive online commands, markup tags, or some other defined interface and breaks them up into parts (for example, the nouns (objects), verbs (methods), and their attributes or options) that can then be managed by other programming (for example, other components in a compiler).

Extensible - a quality of something, such as a program, programming language, or protocol, that is designed so that users or developers can expand or add to its capabilities.

Introduction: What is XML? (15 min)

- XML stands for Extensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive

Elements

The simplest XML elements contain an opening tag, a closing tag, and some content. The opening tag begins with a left angle bracket (<), followed by an element name that contains letters and numbers (but no spaces), and finishes with a right angle bracket (>) In XML, content is usually parsed character data. It could consist of plain text, other XML elements, and more exotic things like XML entities, comments, and processing instructions. Following the content is the closing tag, which exhibits the same spelling and capitalization as your opening tag, but with one tiny change: a `/` appears right before the element name.

```
<elements>
    <myelement>one</myelement>
    <myelement>two</myelement>
 </elements>
```

A tag – either opening or closing – is used to mark the start or end of an element. In the above example we have 3 elements: "elements" and 2 "myelement". Pay attention that the element names are case sensitive and in our example mean different things.

Attributes

Inside the tag, following the element name, there can be some data (e.g. myAttribute="Main element"). This is called an attribute. You can think of attributes as adjectives – they provide additional information about the element that may not make any sense as content. If we modify our example and add attributes to each element, our example will look like this:

```
<elements="parentElement">
    <subElement elementId="childElement1">one</myelement>
    <subElement elementId="childElement2">two</myelement>
 </elements>
```

Namespaces

One of the features of xml is that you can define your own elements, which provides flexibility and scope. But it also creates the strong possibility that, when combining XML content from different sources, you'll experience clashes between code in which the same element names serve very different purposes. For example, if you're running a bookstore, your use of tags in XML may be used to track books. A mortgage broker would use `<element>` in a different way – perhaps to track a deed. A dentist or doctor might use `<element>` to refer to chemical solution. Here is where problems can arise. To solve this ambiguity XML namespaces attempt to keep different semantic usages of the same XML elements separate. In our example, each person could define their own namespace and then prepend the name of their namespace to specific tags: `<book:element>` is different from `<broker:element>` and `<medrec:element>`. A Namespace is a set of unique names. A namespace is declared as follows:

```
<elements xmlns:pfx="http://www.example.com"></elements>
```

In the attribute xmlns:pfx, xmlns is like a reserved word, which is used only to declare a namespace. In other words, xmlns is used for binding namespaces. A namespace has a prefix, in our example it is "pfx", and URI, "http://www.example.com". Although a namespace usually looks like a URL, that doesn't mean that one must be connected to the Internet to actually declare and use namespaces. Our full example would look like this:

```
<elements xmlns:pfx="http://www.example.com">
    <subElement pfx:elementId="childElement1">one</subElement>
    <subElement pfx:elementId="childElement2">two</subElement>
</elements>
```

The attributes `elementId` are associated with the namespace "http://www.example.com".

Nesting

As you might notice from our example there is a special structure of the xml code and indentation is different. The tag `<elements>` closes after nested elements `<subElement>`. Thus, `<subElement>` parts are considered to be children of the parent `<elements>`.

Some XML elements are said to be empty – they contain no content whatsoever. Remember that in XML all opening tags must be matched by a closing tag. For empty elements, you can use a single empty-element tag to replace this:

```
<myEmptyElement></myEmptyElement>
```

with this:

```
 <myEmptyElement/>
```

The `/` at the end of this tag basically tells the parser that the element starts and ends right here. It's an efficient shorthand method that you can use to mark up empty elements quickly.

Independent Practice: Look at an XML file (5 min)

Look at the following code/file and answer the questions below:

```
 <?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="generalassembly.yuliyakaleda.supportdifferentdevices">

  <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".InformationActivity" />

    <service
        android:name=".QuickService"
        android:enabled="true">
    </service>
  </application>

</manifest>
```

1. How many elements does this xml file have?
2. What is a namespace in this example?
3. Name parent and children elements.
4. Find and name empty elements (the ones that do not have content).

Conclusion (5 mins)

XML provides a structured way to specify data, with elements, child elements, attributes, and name spaces. XML files are important in Android to specify various types of data about how the system should treat your app, and how the app should look on the screen.

2:9

Intro to XML

Introduction

> ***Note*** *You can help each other, but everyone must hand in their own code.*

Imagine you are a data specialist for a movie theater. You have been asked to store the information about the movies you show in an XML file for the office manager. The file must meet particular criteria so read below for guidance on creating your data file.

Exercise

Requirements

Create a file called `movies.xml` that includes the following:

- The parent element of the file should be "movie_collection" with the prefix "collection" and the namespace "http://movies.com"
- The "movie_collection" element should have at least 4 "movie" child elements
- Each movie must have attributes for id, title, and description
- Each movie must have a child element called "showtimes", and it should contains at least 3 relevant showtime showtime elements (make something up)
- Each movie should also have 2 empty elements (name them anything you like)

Starter code

There is no starter code! Please create a new file and add your xml code. You can edit in IntelliJ if you like, or just use a simple text editor.

Deliverable

A pull request including your xml file.

Additional Resources

- Sitepoint XML introduction

2:10

Functions Practice Morning Exercise

> ***Note:*** *This can be a pair programming activity or done independently.*

Exercise

Requirements

For more practice writing methods, try these problems from CodingBat:

- Blackjack
- Build an array
- Word count
- Shift left

Use CodingBat's built-in tests to check if your code works correctly. We'll go over these problems together at the end of the exercise.

Deliverable

Nothing to hand in! Just work on your code in CodingBat until all the tests pass.

2:11

Subclassing

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Describe what subclassing means
- Explain how subclassing works in Java
- Extend a class using Java

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Create a basic class with getters and setters
- Instantiate a user-defined class

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Gather materials needed for class
- Complete Prep work required
- Prepare any specific instructions

Opening (5 mins)

Today we will be expanding upon our last lesson about classes. We learned about the basic components of a class and how they are created, but they can do so much more. We will be learning about how classes interact with each other, and more specifically, how they are related to each other.

> Check: Ask students to define what a class is, and how to instantiate one.

Introduction: Subclassing (20 mins)

One of the key ideas behind Object-Oriented Programming is defining relationships between the classes you create. In OOP, we create templates that can be reproduced and interacted with. A subclass can be thought of as a more detailed version of a class you have already created.

For instance, a NormalUser and an Admin can both be considered a subclass of a User. We can say that a NormalUser `is a` User, and that an Admin `is a` User. We could simply make the properties and methods

inside of User for every type of User, but our code becomes much clearer if we make a separate subclasses to represent each specific type of User.

```
List<String> myList = new ArrayList<>();
```

Demo: Subclassing (15 mins)

Do this with me!

We're going to start with the example of shapes. First, let's define our Shape class with the property "mColor":

```
public class Shape {
  private String mColor;

  public Shape(String color){
    mColor = color;
  }

  public String getColor(){return mColor;}
}
```

When designing classes and subclasses, you need to ask yourself what properties or methods are unique to the subclass and what are common across all possible subclasses. In the case of our example, every type of shape has a color, so we include it in the superclass.

Now we're going to make a subclass, Square. A Square is a Shape, so we can make it a subclass:

There are some important keywords to notice:

- The `extends` keyword denotes that we are subclassing **Shape** for this class. (Making **Shape** our superclass)
- The `super` keyword is used to access members from the superclass, such as the constructor

Note that we must call the super constructor so the parent class can do the setup work required. This must be the first statement in the constructor.

```
public class Square extends Shape {
  private int mSideLength;

  public Square(int length, String color){
    super(color);
    mSideLength = length;
  }

  public int getArea(){
    return mSideLength * mSideLength;
  }
}
```

Now we will make a Triangle class that will still be a Shape, but have different behavior.

```
public class Triangle extends Shape {
  private int mSideLength1;
  private int mSideLength2;
  private int mSideLength3;
```

```
  public Triangle(int length1, int length2, int length3, String color){
    super(color);
    mSideLength1 = length1;
    mSideLength2 = length2;
    mSideLength3 = length3;
  }

  public double getArea(){
    double p = 0.5 * (mSideLength1 + mSideLength2 + mSideLength3);
    return Math.sqrt(p*(p-mSideLength1)*(p-mSideLength2)*(p-mSideLength3));
  }
}
```

What we have covered so far are all of the basics you need to build a class and create a subclass using it.

Guided Practice: Subclassing (30 mins)

This is a tricky topic, so let's get some more guided practice. Follow along: Let's write a **Vehicle** class with the `mModel` and `mSpeed` member variable which are assigned on instantiation of the **Vehicle**. How might we do that?

```
public class Vehicle {
    private float mSpeed;
    private String mOwnerName;

    public Vehicle(String ownerName) {
        mOwnerName = ownerName;
        mSpeed = 0.0f;
    }

    public void goFaster(){
      mSpeed += 5.0;
    }

    public void setSpeed(float speed){
      mSpeed = speed;
    }

    public float getSpeed(){return mSpeed;}
}

// creating a new object of type Car
Vehicle mFirstCar = new Vehicle("Civic");
```

That is a good start, but we can create a subclass of **Vehicle** to get a more specific Vehicle.

So instead lets create a `class` **Car** with all the properties we want:

```
public class Car extends Vehicle {
    private boolean mIsManual;
    private String mModel;

    public Car(String modelName, String ownerName, boolean isManual) {
        super(ownerName);
        mModel = modelName;
        mIsManual = isManual;
    }

    public int getManual(){return mIsManual;}
```

```
    public String getModel(){return mModel;}

    @Override
    public void goFaster(){
      setSpeed(getSpeed() + 10);
    }
}
```

Why can we call the method goFaster in Car the same thing as Vehicle? This is called overriding a method, and is a key concept in OOP. Sometimes we want subclasses to have behavior that is different from its parent. If we call goFaster on a car, it will increase the speed by 10, but if we call it on a Vehicle it will only increase the speed by 5.

The Override annotation isn't required, but is a useful check to make sure you are properly overriding the parent method (same name and parameters).

Java provides us with an important concept in OOP. Inheritance. When we subclass the **Vehicle** our **Car** class will **inherit** all the methods and variables contained within:

```
public class Main {
  public static void main(String[] args){
    Car myCar = new Car("Civic","Drew",false);
    myCar.goFaster();
    System.out.println(myCar.getSpeed());
  }
}
```

We call the constructor of **Car** with our model name and owner name using the method call `super()`. This way now every `new` Car object will have its model name and owner name set correctly.

In practice, we can now use Car everywhere we use Vehicle but not the other way around. Think of it in the way all Squares are Shapes but not all Shapes are Squares.

```
Vehicle myCar2 = new Car("CR-V","Drew");
```

To use Car's methods, we need to cast it.

```
System.out.println("Is Manual: "+((Car)myCar2).isManual());

List<Vehicle> myVehicles = new ArrayList<>();

Car car1 = new Car("Civic","Drew");
myVehicles.add(car1);

Vehicle vehicle1 = new Vehicle("Bob");
myVehicles.add(vehicle1);

for (int i = 0; i < myVehicles.size(); i++) {
    myVehicles.get(i).isManual();
}
```

Independent Practice: Subclassing Cards (15 minutes)

Instructor Note: This can be a pair programming activity.

Now, you are going to try implementing your own class and subclasses. Create a new project:

- create a superclass Card and subclasses DebitCard and CreditCard
- give your superclass the properties "nameOnCard" and "cardBrand" (Visa, Mastercard, etc.)
- CreditCard should contain the property "cardLimit", and DebitCard should have the property "accountBalance"
- Add any other properties you want
- Instantiate your Classes and print out their member variables in a main method

Conclusion (5 mins)

Today we gained a further understanding of how multiple classes can fit together to make a useful system that reduces code duplication and makes your code much easier to understand. You can have many levels of subclasses beyond the simple two-level examples we saw today. Hopefully the concepts you learned today helps you to create more complex and better structured apps.

ADDITIONAL RESOURCES

- Inheritance

2:12

Classes

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Describe the difference between interfaces and abstract classes
- Describe why we would choose an interface over an abstract class and vice versa.
- Declare and extend an abstract class
- Declare an interface and implement it's required methods.

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Recall basic knowledge of Classes
- Recall basic knowledge of method creation

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Gather materials needed for class
- Complete Prep work required
- Prepare any specific instructions

Opening: Interfaces and Abstract Classes; what are they? (5 minutes)

When we created our classes in the previous lesson we had to implement everything. We had to set up the template of member variables, the constructors, and the methods. What if we wanted to just set up the template and not have to do all the work right then and there? What if we wanted to set it up in a way that allows us to implement multiple classes that do some logic, just in a different way for each class? Interfaces and Abstract Classes allow us to do that.

Introduction: What are Abstract Classes? (20 minutes)

Abstract classes are basically classes that do a bunch of work for us but also asks the developer to do some work as well.

Let's take a look at an example:

```
public class Animal{
  protected String mName = "";
  protected int mNumOfLegs = 4;

  public Animal(String name){
    mName = name;
  }

  public int getNumOfLegs(){
    return mNumOfLegs;
  }

  public void setNumOfLegs(int numOfLegs){
    mNumOfLegs = numOfLegs;
  }

}
```

Let's instantiate this in the main method.

```
public class Main{

  public static void main(String[] args){

    Animal aCat = new Animal("cat");
    int numOfLegs = aCat.getNumOfLegs();

  }

}
```

What if we wanted to also find out the sound that the animal makes? We could just add a method `setSound(String sound)` then have another method to print out the sound. A better way would be to make the class an abstract class and allow other classes to handle what sound the animal makes.

```
public abstract class Animal{

  protected String mName;
  protected int mNumOfLegs;

  public Animal(String name){
    mName = name;
  }

  public int getNumOfLegs(){
    return mNumOfLegs;
  }

  public void setNumOfLegs(int numOfLegs){
    mNumOfLegs = numOfLegs;
  }

  public abstract void speak();

}
```

If we extend this abstract class we will see that we need to implement those abstract methods. This is the

subclassing you learned earlier today, but this makes it slightly more robust.

```
public class Dog extends Animal {

    public Dog(){
      super("dog");
    }

    public Dog(String name){
      super(name);
    }

    @Override
    public void speak(){
      System.out.println("woof");
    }

}
```

Independent Practice(15 min)

Let's go back to the main method and instantiate this Cat class.

```
public class Main{

  public static void main(String[] args){

    Animal theAnimal = new Cat();
    int numOfLegs = theAnimal.getNumOfLegs();
    theAnimal.speak(); //This will print out meow
    theAnimal = new Dog();
    theAnimal.speak(); //This can print out woof
  }

}
```

Let's create a few more classes that extend `Animal`.

What are Interfaces(20 min)

Interfaces are a way to make another programmer do all the work for you. This makes heavy use of a concept called polymorphism.

Unlike classes, interfaces **generally** only contain method stubs, not any actual implementation. By definition, all methods in an interface are abstract. In addition, any fields in an interface must be declared as static and final.

```
public interface MyInterface {
  void sayHello();
  void sayGoodbye();
}

public class MyClass implements MyInterface{
  //constructors
  ...

  public void sayHello(){
    System.out.println("Hi!");
  }
```

```
  public void sayGoodbye(){
    System.out.println("Bye!");
  }
}
```

In Java 8, the concept of default implementations was added. Suppose you created an interface that is used by many classes in a large program, but later on you wanted to add a new method to the interface. If you did this, every single class using this interface would break because they don't have the new method. Instead, you could add a basic default implementation in case the other classes decide not to implement it.

So why we don't just use classes and make all of the methods abstract? One major reason is that you can only extend one class, but you can implement as many interfaces as you want.

The other reason stems from the idea behind why interfaces exist. Subclasses are meant to share common functionality with its parent classes, but have some differentiation between them. Interfaces are a contract for functionality of what a class should be able to do, but two different classes implementing the same interface can be completely unrelated.

You cannot instantiate an interface, but a class that implements one can be considered to be that type of object, just like a subclass can be considered the same type of object as its parent.

Let's take a look at an example:

```
public interface Flyable{

 boolean canFly();

}

public interface Swimmable{

  boolean canSwim();

}

public abstract class Animal implements Flyable, Swimmable{
  ...
}

public class Dog extends Animal {
  ...
  public boolean canSwim(){
    return true;
  }

  public boolean canFly(){
    return false;
  }
}
```

Polymorphism is a concept in computer programming that allows a program to ignore what class exactly is calling a method. It does not care what that class is, just that it implements certain methods.

Let's look at another example:

```
public class Main{

  public static void main(String[] args){
```

```
    List<Animal> animals = new ArrayList<>();

    animals.add(new Dog());
    animals.add(new Cat());
    animals.add(new Bird());
    animals.add(new Mouse());
    animals.add(new Cow());
    animals.add(new Frog());
    animals.add(new Elephant());
    animals.add(new Duck());
    animals.add(new Fish());
    animals.add(new Seal());
    animals.add(new Fox());

    for(Animal animal: animals){
      animal.speak();
      animal.canSwim();
      animal.canFly();
    }

  }

}
```

Independent practice (15 min)

Conclusion (5 min)

We created classes, and learned how to subclass other classes for our own purposes in the previous lesson. In this lesson we learned how to architect our classes in a way that makes subclassing more robust. We also learned when and why we would use an abstract class over an interface. These are important concepts for everyone to understand and be able to explain the differences between them. Questions regarding polymorphism are standard practice in most programming interviews.

2:13

Subclasses with Animals Lab

Introduction

In this lab you will be using your knowledge of classes to build an Animal class, complete with properties (see below), getters and setters. After completing the class, you will then create subclasses of Animal called Mammal and Reptile. Finally, you will create at least one subclass of Mammal, and one subclass of Reptile, making them whatever animals you want.

Each subclass should have some unique property that differentiates it from its parent class. For instance, a reptile could have a boolean to determine if it has a shell, or a snake could have a boolean to determine if it is poisonous.

Then, you will write a main method that instantiates one of each of your custom classes, and prints out the properties of each.

Exercise

Requirements

- Create an Animal class with the following properties set in the constructor: mNumLegs, mTopSpeed, mIsEndangered, mName.
- Create subclasses of Animal called Mammal and Reptile
- Create at least one subclass of Mammal, and one subclass of Reptile
- Each subclass must have something that makes it unique from its parent class.
- Create a main method that instantiates each class and prints out its details.

Bonus:

- Create more than the minimum number of mammal or reptile subclasses

Starter code

There is no starter code for this lab.

Deliverable

A Java program that meets the requirements above.

2:14

Working with Interfaces and Abstract Classes

Starter code

There is no starter code. Create a new project in IntelliJ for this lab

Introduction

> ***Note:*** *This can be a pair programming activity or done independently.*

In this lab you will be using your knowledge of classes, subclassing, interface and abstract class creation and implementation to create a few classes that will be used in the main method.

This looks like a lot. It is, but it is a lot of repetition.

Exercise

Requirements

Abstract Class

- Create an abstract class called `Remote`.
- Give the abstract class the property needsBatteries and supportsUsb.
- Create a constructor that accepts parameters to set the properties you just created.
- Create getters and setters for the properties.
- Declare the following abstract methods, `channelUp`, `channelDown`, `volumeUp`, `volumeDown`; they all take no parameters and return nothing.
- Implement `powerButtonPressed` that does not take any parameters and just prints out that the power button was pressed.
- Implement `numberButtonPressed`, that takes in an `int` parameter and appends/concatenates the `int` parameter to the `String` `"Changing channel to "` then prints it out to the terminal.

Interfaces

- Create an interface class named `WaterProof`.
- Inside this class, declare a method `isWaterProof`. **What do you think it should return?**
- Create an interface class named `Rechargeable`.

- Inside this class, declare a method `isRechargeable`. **What do you think it should return?**
- Create an interface class named `Universal`.
- Inside this class, declare a method `supportsBrands`. It should return nothing.

Concrete Class

- Create three concrete classes that extends the abstract class `Remote`, name the concrete class one of your favorite brands. (Samsung, LG, Panasonic, Sony, etc.)
- Implement the methods you declared in the `Remote` abstract class which are required to be implemented in each concrete class.
- Make sure the implementations output something different for each concrete class.
- Implement the `WaterProof`, `Rechargeable`, and `Universal` interfaces.
- Implement the methods defined by those interfaces. You can choose what to do, but at the minimum print something out to the terminal.

Main Java Class

- Create a `List` of `Remote` objects.
- Populate the `List` of `Remote` objects with new instances of the concrete classes you created.
- Create a loop that loops as many times as there are items in the `List` of `Remote` objects. (You choose the kind of loop)
- Inside the loop call the `numberButtonPressed` method on each object and pass it the current iteration value of the loop.
- Inside the loop call the `channelUp`, `channelDown`, `volumeUp`, `volumeDown` methods on each object.

Bonus:

- Create 3 more concrete classes that extend `Remote` and use them in the loop.

Deliverable

Pull request from forked Project on GitHub that contains your implementation of the `Remote` abstract class, the `WaterProof`, `Rechargeable`, and `Universal` interface classes, and the main program that uses the classes you created. Additional Resources below.

- Oracle: Creating abstract classes
- Oracle: Creating an Interface
- Oracle: Defining a class
- Oracle: Instantiating a class

2:15

Subclassing, Abstract Classes, and Interfaces

Note: *You can discuss with classmates, but everyone must submit their own answers*

Exercise

Requirements

- Fork this repo, then add your answers direcly to this **readme.md** file
    - After forking, you can clone, edit the readme in the text editor of your choice, and push back to Github...
    - Or, you can edit the readme right from the browser
- After adding your answers, submit a **pull request**

Questions

1. What is the difference between *member variables* (also called *instance variables*) and *class variables* (w/ keyword `static`)? Which can be accessed without creating an instance of the class?

2. Does it make sense to write *getter* and *setter* methods for a `public` member variable? What about `private` variables?

3. What are some benefits of making member variables `private`?

4. If class A extends class B, which is the super class and which is the sub class? Which would you call parent, and which would you call child?

5. What does it mean for a class to *inherit* methods and/or variables from its parent class?

6. Consider the following code, where class Refrigerator extends class Appliance, and `getTemperature()` is a method in Refrigertor, but NOT in Appliance:

    ```
    Appliance myAppliance = new Refrigerator();
    double temperature = myAppliance.getTemperature();
    ```

    Why will this call to `getTemperature()` cause an error? How will *casting* help solve this issue?

7. In a normal class (also called a *concrete* class), do you need to *implement* all of the methods, or can your simply *declare* some? What about in an `abstract` class?

8. What about an `interface`? Can you implement any methods in an interface? Can you declare methods in an interface?

9. Can you create an instance of an `abstract` class? Also, look up the Java keyword `final` and see if you can explain why a class CANNOT be both `abstract` and `final`.

10. What happens when a method *overrides* another method? If a parent and child class have methods with the same name, when you call that method on an instance of the child class, which implementation of the method will be executed?

11. What is the relationship between `List`, `LinkedList`, and `ArrayList`? Why do we call a method *polymorphic* if it takes an input of type `List` rather than an input of type `LinkedList` or `Arraylist`, and why is that useful?

Deliverable

This file, with your answers added

Additional Resources

Refer to the Classes lesson, the Subclassing lesson, and the Interfaces & Abstract Classes lesson.

Feel free to google these concepts as well. There are plenty of Java tutorial websites and StackOverflow posts that can help you. But be sure to write up your answers in your own words - copying and pasting some text does NOT help you actually learn!

2:16

Views 101

Objectives

After this lesson, students will be able to:

- Identify what a view is and what it’s used for
- Match sections of the view XML to what’s on the screen

Preparation

Before this lesson, students should already be able to:

- Create or import a project in Android Studio

Introduction: Views (15 minutes)

What is a view?

A **View** is the basic building block for any app's user interface (UI). Views define components that can be seen by the user, such as text fields, buttons, and images.

What are the types of views?

Some of the most commonly used views are:

- TextView
- EditText
- Button
- ImageView
- CheckBox and Switch
- ProgressBar
- WebView

TextViews

Simply, they display text that's provided to them. You can change the text, text color, typeface, size, etc. Think of it like changing fonts in a word processor.

Buttons

Buttons are fancy TextViews. Just like a TextView, you can set its text and change its font attributes.

Buttons, by default, have a background that react to a user's touch. (i.e., it looks like you are pressing a physical button). You can create your own custom button backgrounds. That is covered in a future lesson.

How are views laid out on screen?

- A Layout defines how other views are shown on screen.
- There is a parent/child relationship. Layouts are parent views that contain child views.
- **RelativeLayout** arranges views relative to each other. Examples:
    - This TextView is below this ImageView
    - This Button is to the right of another button
    - This ImageView is centered, relative to the RelativeLayout itself.

For the purposes of this lesson, we will only look at RelativeLayout. We talk about the other layouts more in depth in the Layouts lesson.

Refer to this online cheat sheet for help remembering different types of views and the corresponding syntax.

Guided Practice: Let's draw a few Layouts (10 minutes)

In pairs, on your desks, draw and identify the components of the following UIs using the views we just learned:

- A post on Facebook or Twitter
- A dating card on Tinder
- The description on a YouTube video

You should have an idea of popular components for well-known applications.

Introduction: Layouts and XML (20 minutes)

How do I create my UI using layouts?

Here's a quick refresher on XML files: XML Lesson

The easiest way to place views are by using XML based layout files. Each element represents a different view or layout; attributes define how the views and layouts are displayed to the user.

Here's an example of how a TextView is represented in xml:

```
<TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
```

Elements that don't hold other views, like TextView, can be defined as empty tags. Most attributes are in the XML namespace, "android:". This denotes default attributes provided by Android.

Every element ***is required*** to have **layout_width** and **layout_height** attributes. Otherwise, the app will not compile.

`layout_width` and `layout_height` can be defined in 3 ways:

- An exact dimension, in pixels (talked about in depth in Views 102)
- **wrap_content**, where it takes up only the amount of space it needs. For instance, with a TextView,

setting its width to wrap_content makes it as wide as its text.
- **match_parent**. Remember, layouts are considered parents. Using this makes the width or height match the parent's width or height, respectively.

The **id** attribute is not mandatory, but is important. It is used whenever you want to reference a view or layout, either within a layout or in Java.

Here's a more detailed example of a full xml layout:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="Top" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/textView1"
        android:layout_centerHorizontal="true"
        android:text="Bottom"
        android:textSize="20sp"
        android:textColor="#000000"
        android:textStyle="bold" />

</RelativeLayout>
```

In this example, take note of the following:

- The layout is defined as a RelativeLayout that is the width and height of the device's screen
- The RelativeLayout is the parent of two children: Two TextViews
- The top TextView, with id "textView1", is centered horizontally in the relative layout. It's width and height wraps around its text, "Top".
- The bottom TextView, with no id, is also centered horizontally in the relative layout. It's width and height wraps around its text, "Bottom".

    - Because it is a TextView, it also defines its optional parameters: text size is 20 scaled pixels (Views 102), text color is the hex code for black, and the text style is bold.
    - Because its parent is a RelativeLayout, the text view can be placed relative to other views. In this case, it is directly below the view with id "textView1".

Guided Practice: Let's program some layouts (15 minutes)

Let's go over adding views and layouts in Android Studio. We will start off dragging views in the Android Studio Designer Tool, then switch over to the XML view to see how they are related. Then, we will add views via xml and see how the look in the preview.

Independent Practice: Define your own layouts (20 minutes)

Work with the person next to you and try to recreate the screenshot below:

Bonus: Add more customization to the screen.

Conclusion (5 mins)

- What is a view?
- How does a RelativeLayout work?

2:17

Views 102

Objectives

After this lesson, students will be able to:

- Reference views in Java
- Change view properties in Java and XML
- Attach OnClickListeners to views

Preparation

Before this lesson, students should review the following lesson

- Views 101

Introduction (10 minutes)

In the last lesson, you were able to layout views and show them on the screen. Building on top of that, this lesson will show you how to change the visual properties of views. Afterwards, you will be able to make views react to when a user clicks on them.

By the end of this lesson, you will be able to create your first interactive app!

Basics - Views vs. Properties vs. Values

Think of a view as a component of the interface: buttons, text boxes, input fields. All are views!

Every view has a number of properties specific to it. Some examples:

- In **TextView**, you are able to set the view's text, text color, font, etc.
    - In fact there are a number of other views that are "descendants" (or subclasses) of TextView and have the same properties you can set: Button, EditText
    - Notice how views that subclass TextView have all of its properties?
- In **ImageView**, you can set the view's image and how it is scaled within the view.
- In **ProgressBar**, you can set the current progress or if the view is indeterminate (where it spins indefinitely).

All views are a *subclass* (or "descendant") of **View.java**. "All views are views." So, exactly how a Button has all of the properties of a TextView, all views have core properties that are provided by the View "class". We'll go over classes in depth at a later date.

The commonly used properties for all view types are:

- width
- height
- padding
- margins
- background
- alpha (opacity)
- id
- visibility

There are properties and attributes for each view. The Android website has a detailed reference to all of the popular views, properties, and how to change them. It's found here: http://developer.android.com/develop/index.html. From there, you can use the search icon on the top right to search any view.

Also, you can just Google it! Usually Googling "Android *TypeOfView*" would have the #1 link lead directly to the view's reference page. Try it: Google "Android TextView".

Now, that you know about views having attributes/properties, you can probably guess that you can assign values to these properties to affect how they look and behave on the screen.

Think of the following comparison:

You sit down for a meal and the meal has different components: pizza, diet coke, and wings. Depending on the component, there are particular properties you can ask to change:

- What toppings can you have on your pizza?
- Can or fountain for your diet coke?
- Flavor of the wings?

We'll come back to this example in a moment.

Demo: Changing View Properties in XML (10 minutes)

You, hopefully, have already changed some properties in XML, especially when it comes to TextViews, but let's go through the process anyway to solidify the concept.

First, to access all of a view's properties, the layout must be inside the android xml namespace. This is usually added to the topmost layout automatically, but here's how it looks:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    ...
</RelativeLayout>
```

The app would not work unless you have access to this namespace!

Now that you have access to the android namespace added, you can access all view properties using the prefix `android:`. For example, when you set the text attribute to a TextView, you would add `android:text="Some Text"`.

Here's how a TextView with large red bold text would look:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello!"
    android:textSize="30sp"
    android:textColor="#FF0000"
    android:textStyle="bold" />
```

What's the deal with the "30sp" value for text size? This will be explained in detail later. But for now, know that they stand for "scaled pixels," and that **sp (Scale-independent Pixels)**, **dp (Density-independent Pixels)**, and **px** are units of pixel measurement.

Going back to our meal example:

Example	Comparison to Android
Meal	Activity
Pizza	TextView (View)
Pizza Topping	textColor (Property/Attribute)
Cheese and Chicken	#FF0000 (Value)

Here's another example of a button with the id "button1", margins, that is the width of it's parent view, and has an opacity of 50%:

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="I'm a button!"
    android:layout_margin="16dp"
    android:alpha="0.5"/>
```

Demo: Changing Views Part 1 (5 minutes)

Watch me as I create a layout and add a few views to it. In XML, the views are changed to look differently.

Intro: Referencing views in Java (10 minutes)

Before we change view properties in Java, you have to learn how to reference the views you create.

Things to be aware of before you continue:

- Every view and layout has a java class that matches its XML counterpart.
    - ex., is defined by a Java, TextView.java.
- If you give a view or layout an id, it can then be used in Java for reference.
    - ex, **@+id/textView1** can be referenced in java as **R.id.textView1**.
    - **@+id/** is used in XML layout files, and **R.id** is used in Java.

To interact with views in your layouts, you have to get a reference to that view. Activities provide a helpful method, **findViewById()**.

```
TextView textView = (TextView) findViewById(R.id.textView1);
```

`findViewById(R.id.textView1)` returns a view that has the id textView1. Notice how I said **"returns a view"** and not "returns a text view". TextViews, Buttons, RelativeLayouts, etc., are all Views (and subclass View.java). So, findViewById() returns a instance of View.

Views have to be cast if you want access to that view's specific functionality.

- ex. TextViews allow you to set a view's text, so you have to cast the result of findViewById(): **(TextView) findViewById(R.id.textView1);**

Now that you have a reference to your specific views, you can change the views to your liking!

Demo: Changing view properties in Java (10 minutes)

Remember how you can set attributes to views in XML? Every attribute defined in XML can also be accessed and changed in Java. Usually, the attribute has a **get** and **set** method that matches the XML attribute. Example:

XML

```
<TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Text"
        android:textColor="#000000" />
```

Java

```
TextView textView = (TextView) findViewById(R.id.textView1);

textView.setText("New Text");
textView.setTextColor(Color.BLACK);
```

Guided Practice: Changing Views Part 2 (10 minutes)

Let's use the layout created in the first demo, reference the views in Java, and change those view properties according the specifics provided. Finally we'll build the project to see their results. I want you, though, to guide my progress and instruct me on what I should be doing to make this happen. Do this with me.

Demo: Making views do "things" (20 minutes)

You can make an app that shows a static layout and nothing else. However, that would be boring. Apps do stuff; they show information, and they react to information provided by the user.

Usually, the cool stuff is done in Java.

Users Interacting with Views

Every good app has some form of interaction. Usually, that means that the user can click the views on the screen. Examples:

- Tapping the + icon in Gmail to compose a new email
- Clicking the *Like* button on a Facebook post
- Clicking someone's profile photo to see more information in LinkedIn

All views are clickable by default. However, clicking a view will not do anything; they don't know what to do. So, in Java, you have to tell them what to do.

To do this, you have to create an OnClickListener and assign it to a view. As it sounds, it's an interface that listens for when the view is clicked. It has only one method, **onClick()**; this is where you put the code you want to run when the view is clicked. Here's how it looks:

```
OnClickListener myOnClickListener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // do stuff here
    }
});

Button button = (Button) findViewById(R.id.button1);

button.setOnClickListener(myOnClickListener);
```

With the above code, we created a new OnClickListener and set it to a button. Now, whenever that button is clicked, it will call the listener's onClick() method and run the code we writ in `// do stuff here`.

Note: This is worth repeating. ***All*** views are clickable, not just buttons. You can click a TextView, a ProgressBar, etc. Buttons are special because they are visually clickable; i.e., a button looks pressed when a user touches it and raised when it's not.

I bring this up because sometimes you want to click things that are not buttons. For instance, clicking a photo.

Putting it all together

Let's assume we have a layout, *activity_blue.xml*, with two views: a TextView and a Button. They have the ids *textView* and *button*, respectively.

When we click the button, it sets the color of the TextView to blue.

Here's how the Activity would look. All of the code is defined in the activity's **onCreate()** method, which is talked about more in detail at a future lesson. Just know that onCreate is called when the activity starts, and that it loads your created layout with setContentView().

```
public class BlueTextActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_blue);

        // Create references to the views inside of activity_blue.xml

        Button blueButton = (Button) findViewById(R.id.button);

        // Create the onClickListener for the button

        OnClickListener buttonOnClickListener = new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                TextView textView = (TextView) findViewById(R.id.textView);
                textView.setTextColor(Color.BLUE);
            }
        };

        // Set your onClickListener to the button
```

```
            blueButton.setOnClickListener(buttonOnClickListener);
        }
    }
```

Independent Practice (20 minutes)

> ***Note:*** *This can be a pair programming activity or done independently.*

Create a Hello World app! I will share the code above to use as a reference.

The app should do the following:

- Layout with two views: A TextView and a Button
    - The TextView should start off with no text, but have a text size of 30sp
    - The Button should have the text, "Say hello"
- When you click the button, the text in the TextView should say "Hello!"

Bonus: Change the size of the text to 40sp when the button is pressed.

Conclusion (5 mins)

- How do you reference a view in Java?
- How does a OnClickListener work?
- How do you change view properties in XML? In Java?

2:18

Activities and Intents

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Define what an Activity is
- Understand the components of the Manifest
- Given a running application, identify the activities in that application
- Identify the same activities in the app manifest
- Use Intents to move between Activities
- Pass data between activities

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Create an activity
- Use nested layouts to create more complicated views
- Explain view recycling and why it matters

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Read through the lesson
- Add additional instructor notes as needed
- Edit language or examples to fit your ideas and teaching style
- Open, read, run, and edit (optional) the starter and solution code to ensure it's working and that you agree with how the code was written
- **Note**: With the addition of `startActivityForResult`, this lesson is unusually long. Consider giving the students a break mid-way through.

Opening (5 mins)

Previously, we learned about the UI elements that make up a screen in an app, and how we interact with it in the Java code. Notice how the file was called MainActivity.java? Most of the time, you can think of an Activity as a screen of your app - that's all!

> Check: If Activities are screens, why do you think they are called "Activities"?

Introduction: What is an Activity? What is a Manifest? (10 mins)

The definition of an activity is something that is done for a particular purpose.

Think about the activity that the user is doing on a screen. If you are looking at a screen whose purpose is to log in the user, for example, it should be called the Login Activity. If the activity shows a user's social network profile , it should be called the User Profile Activity.

Let's add a new Activity to this app. Right click on the folder where you want to put the activity, then go to *New > Activity*, and then click on the type of base activity you want (usually, *Empty Activity* is what you want if you want to build it from scratch).

An Activity is a plain ol' Java class, so you already know how to add it to a project.

Doing this also adds the Activity to the Android Manifest.

> Check: So what is an activity again?

What is a Manifest?

The dictionary defines a ship's manifest to be "a document giving comprehensive details of a ship and its cargo and other contents, passengers, and crew for the use of customs officers."

The ship, in our case, is the app you are building.

The Android Manifest xml file presents important information about your app to the Android system. If something isn't defined in the manifest, then the system just ignores it.

Notably, the manifest is known for describing the main components of your app; the Activities, Services, Content Providers, etc. It is also the place to define permissions (e.g., giving your app permission to access the internet or to access a device's camera).

Whenever you create a new Activity through the New Activity menu, Android Studio will automatically add it to the manifest.

> Check: How do activities relate to the manifest?

Demo: Creating Activities (15 mins)

In this demo, let's walk through the following:

- Creating a new project in Android Studio
- Examining the manifest file, describing components like the XML elements, attributes, and package
- Describe what a launcher activity is (The activity that opens when the app is launched), and make comparisons to Java's *public static void main* method
- Add 2 more activities to the project, and go back to the manifest and see them added

> Instructor Note: A complete example of this is found in the solution code folder.
>
> Check: Take 2 minutes, with the person next to you, and discuss what a launcher activity is, and how it compares to public static void main. Be ready to share out!

Introduction: What are Intents? (10 mins)

Intent, as defined in the dictionary, means: purpose, goal, objective. *Something intends to do some goal*.

This translates to your app and activities; every activity has a goal.

For example, a `ComposeEmailActivity` allows the user to compose and send an email. If you click a "compose new email" button, you are actively saying "I intend to compose an email".

This is the idea behind Intents in Android. Intents are messages you send between app components, like Activities, usually with the goal of doing something.

> Check: So, imagine you are in your app's `EmailListActivity`, and you click on one of your emails. In plain ol' English, could you describe what is happening between the user and the EmailListActivity? Take 10 seconds to think about it.

The following "dialogue" is happening:

- EmailListActivity: "Hey, you clicked one of your emails. What's up?"
- You: "I intend on reading that email. Is that okay?"
- EmailListActivity: "Yeah, sure! I'll start the ReadEmailActivity now."
- You: "Thank you."

Let's create an EmailListActivity and a ComposeEmailActivity.

So, how does an Intent look like in code?

```
Intent intent = new Intent(EmailListActivity.this, ComposeEmailActivity.class);
startActivity(intent);
```

You create a new Intent object, and you pass it two parameters: The activity you are currently in, and the class of the activity you intend to start. The code snippet above could be read as, *From the Email List Activity, please start the Compose Email Activity*.

The method, `startActivity()`, starts the intended activity immediately.

> Check: What two parameters should you pass your Intent objects?

Independent Practice: Starting an activity with an Intent (10 mins)

Using the code from the previous demo, add a button to the app's main activity. Set an `onClickListener` to that button and have the listener start one of the other activities. Run the app in a virtual device, and click on the button to start the new activity.

Note: A complete example of this is found in the solution code folder.

Introduction: Sending data from one Activity to another Activity (10 mins)

Intents are how Activities communicate with each other. In the previous example, we started an activity to compose an email by clicking an email in the list. However, how does the ReadEmailActivity know what email to show?

> Check: Take 30 seconds to talk with the person next to you about this question.

When you start a new activity, it is shown with the default settings that you give it. However, some activities need to receive a bit more information. This info is sent from the original activity to the one you are starting.

> Let's rename our Activities to match the email example

When creating new intents, you can also give it *extra* data. Here's an example:

```
Intent intent = new Intent(EmailListActivity.this, ReadEmailActivity.class);
intent.putExtra("ID", 123);
intent.putExtra("SENDER", "John Smith");
startActivity(Intent);
```

> Check: Take 20 seconds to study the code and come up with an explanation, in English, about what's happening. Be ready to share!

The Intent class has a handful of helper methods you can call to get and store extra data. The main one is `putExtra()`, which takes two parameters: a String that gives the data a name, and the data itself.

With `intent.putExtra()`, you can put data inside the intent (including Strings, numbers, booleans, certain objects).

Once you start a new activity, you can retrieve the Intent and get the sent data, as follows:

```
// get the intent that started this activity
Intent intent = getIntent();

// get the data from the intent
int id = intent.getIntExtra("ID", 0);
String sender = intent.getStringExtra("SENDER");
```

Again, the Intent class has a handful of getters for extra data, usually formatted like *get_____Extra*. Examples, `getIntExtra()`, `getStringExtra()`, `getBooleanExtra()`, etc.

Here we are hard-coding the Keys, what do you think we could do as an alternative?

Note: You should only send data if you need to do so. If the activity you are starting doesn't need extra data, you don't have to set it.

Guided Practice: Sending Data between Activities (15 mins)

> Instructor Note: Show the class that getting data without setting it or with a type in the extra id would cause incorrect or null data. Then, after you introduce this exercise, be sure to stop the class after each step and reveal the answer.

With the person next to you, go ahead and start a new Android project with a empty main activity. Do the following:

- Add a new Activity, and call it `EchoActivity`.
- In `MainActivity`, add an `EditText` and a Button. In the `EchoActivity`, it will just have a `TextView`.
- In Java, set an `onClickListener` to the button and make it start the `EchoActivity`. The value of the `EditText` is passed in the intent.
- In the `EchoActivity`, the value is plucked from the intent and put in the TextView.

Independent Practice: Add two numbers (15 mins)

> This should be done as a pair programming exercise.

Now, with the person next to you, without stopping every two minutes, do the following:

- Create a new project, with a blank main Activity
- Create a new activity, call it SolutionActivity
- In the main activity, put two EditTexts and one button in the layout. The button's text will say "Add"
- In the solution activity, just have one TextView
- When the button is pressed, it takes the two values and sends them to the solution activity, where the sum of the two numbers are shown. Add the two numbers in the SolutionActivity.

Demo: Passing data back in the result (20 mins)

Passing data works in both directions, and you can receive data when returning from an Activity you previously started. For example, if you start SecondActivity from MainActivity, you can get data back from SecondActivity when it closes.

Returning data from an Activity only requires a few additions to your code.

This can be broken down into changes in your calling Activity (the one you are starting the second activity from), and changes in your secondary Activity.

Open the Starter-Code and follow along.

Second Activity

First, let's look at the second Activity, or the one you are passing the data back from. In this Activity, we press a button, and it passes the values from the two EditTexts back to the Main Activity.

> Check: How do you think we are going to pass data from the second activity to the first when clicking the button? Share out!

Just like when we're starting an Activity, we also pass data back using an Intent and extras.

```
mButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
    Intent resultIntent = new Intent();
    resultIntent.putExtra("first",mFirstNameText.getText().toString());
    resultIntent.putExtra("last",mLastNameText.getText().toString());

    setResult(RESULT_OK,resultIntent);
    finish();
}
});
```

First we create an intent, then put the two Strings in as extras. The next two lines of code are new, though.

- **setResult**: This method takes two parameters. The first is a value that lets our first Activity know that everything went well, and that this Activity finished correctly. The second parameter is simply an Intent holding the data we want to pass back.
- **finish**: This method closes the current Activity and returns to the previous one.

Main Activity

Now that we've finished the Second Activity, let's return to the Main Activity. We have two steps to complete.

> Check: What are they? You know this by now.

We need to start the Second Activity and get the results from the Second Activity.

Starting the Activity is almost identical to how we previously did it, with one exception:

```
Intent intent = new Intent(MainActivity.this,SecondActivity.class);
startActivityForResult(intent,NAME_REQUEST);
```

Notice the `startActivityForResult` method. The first parameter is a normal Intent, but the second is a variable telling the system what we are asking for (we will use this in a minute). We need to add that variable at the top of the Activity.

```
private static final int NAME_REQUEST = 20;
```

You can assign any integer value that is **greater than 0**.

Next, we have to get the results from the Second Activity. Whenever you return from an Activity that is expecting results, the `onActivityResult` is automatically called.

> Check: Give the students 2 minutes to discuss what data they think will be returned to us based on what was passed into the second activity and what was returned in the result after pressing the button.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // Check what request we're responding to...
    if (requestCode == NAME_REQUEST) {
        // Make sure the request was successful...
        if (resultCode == RESULT_OK) {
            String firstName = data.getStringExtra("first");
            String lastName = data.getStringExtra("last");
            mText.setText(firstName+" "+lastName);
        }
    }
}
```

The first parameter is the static variable we passed in when starting the activity, in our case `NAME_REQUEST`. The second parameter is the result status, in our case `RESULT_OK`. The final parameter is the Intent we passed back containing the data.

Since we could be starting different Activities from our Main Activity, we first have to check that our result is coming from the `NAME_REQUEST` activity, then we have to check to make sure the results are valid. After that, we can retrieve the data like normal and use it however we want.

> Check: What are the key differences between how we started activities before and what we just did?

Independent Practice: startActivityForResult (10 mins)

Do this activity with a partner

Instead of passing values from the first activity to the second, we are going to pass the data back from the second activity to the first.

If the user chooses add, they are taken to a calculate activity with two EditTexts where they type 2 numbers, and a calculate button. The sum of the numbers from the calculate activities is displayed on the main activity after pressing the calculate button.

Bonus:

If the user chooses subtract, the same steps occur, except the difference is displayed in the main activity.

> Check: Let's take 2 minutes to review the answer. Were all students able to complete the activity successfully? If not, where did you get stuck?

Conclusion (5 mins)

- What is an activity?
- What is an intent?
- How do we start an activity?
- How do we send data from one activity to another?
- How do we receive data when returning from an activity?

Additional Resources

- Android Developer | Starting Activities - http://developer.android.com/guide/components/activities.html#StartingAnActivity
- Android Developer | Intents - http://developer.android.com/guide/components/intents-filters.html

2:19

Experimenting With a Simple App

In this lab, we will be focusing on creating a simple app. You will practice how to add and modify basic views in an XML layout. Also, using Java, you will practice how to modify views and how to make a view react to a user's click.

Exercise

Requirements

- **Add three buttons**, with the texts "Blue", "Magenta", and "Gray".
- When the buttons are clicked, have each button set the TextView color to the corresponding color.

Bonus: Change the background of the text in addition to changing the text's color.

Starter code

No starter code is provided.

Deliverable

A working Android app that meets the requirements above.

Take a look at what your screen should look like upon clicking "Blue":

Additional Resources

- Official Android Developer Website
- Building a Simple User Interface

2:20

Activities and Intents Lab: Mad Libs

In this lab, you will be creating an app based on the game *Mad Libs*.

Mad Libs is a game where you provide certain words and the words are then inserted into a pre-written paragraph with blank spots for fill-ins. Typically, you read the sentence out loud, which usually creates a random (and hopefully, funny) story. Here's an example of a Mad Lib.

This lab will let you practice starting activities and sending data from one activity to another. Also, it will help you practice getting data input from the user, and showing errors when the user doesn't provide data.

Exercise

Requirements

The app consists of two activities.

The main activity should be laid out as follows:

- The layout should have 6 EditTexts, each with hints that describe what type of word is expected.
    - The app should accept the following: 2 adjectives, 2 nouns, animals (plural), and a name of a game.
- The app should have a button that submits the data and starts the Result Activity.
    - If this is pressed and an EditText is not filled out, the user should be notified that they must fill out all the inputs, and the app should not move to the next activity
    - One option for notifying the user is by setting an error message on the empty EditText via EditText.setError().
- The Result Activity already takes the words you previously entered, and inserts them into the madlibs template of your choice. You have to provide the 6 words, passed as extras to the activity when starting it.

Starter Code

Use the starter-code provided if you would like. It helps set up your layout for you. Or, you can start from scratch by creating a new Android Studio project in your repo.

Deliverable

An Android app that follows the requirements above. There are no design requirements aside from having the six EditTexts and the button; feel free to play around with color, font, backgrounds, etc.

Submit a pull request once you have pushed your code to GitHub. Ensure that the project builds and runs successfully.

Additional Resources

- Official Android Developer Website - http://developer.android.com/training/index.html
  - Building a Simple User Interface
  - Starting Another Activity

2:21

OOP Assessment

Introduction

> ***Note:*** *This should be done independently.*

The goal of this assignment is to test your knowledge of Object Oriented Programming concepts and to make sure you know how to implement them.

There are two main components:

1. A written section, with both open-ended and multiple choice questions
2. A programming section

Exercise

Requirements

- Write your answers to each each question in the oop_questions.txt file
- Complete the TODO items from the Java project in the starter-code folder

Starter code

The starter code contains a Java program that is the beginning of a role playing game! It has a Main class, Monster class, Zombie class, and Dragon class. You must go through each file and complete all of the TODO items.

Deliverable

Answer the questions in the oop_questions.txt and complete the code as per the requirements list in the starter-code.

When you run the code, the output should look like this:

```
I am a Dragon with 3 health and do 8 damage!
I am a Zombie with 4 health and do 10 damage!
```

3:1

Whiteboarding for Interviews

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Describe the purpose of whiteboarding
- Implement whiteboarding techniques

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Write a function in Java using programming fundamentals

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Open and run the solution code to ensure you agree with the solution and it works properly

---

Opening (5 mins)

Whiteboarding is often a very important part of the technical interview process and gives the interviewers an insight into your thought process as well as your knowledge on the subject. As the name implies, whiteboarding is the technique of solving a problem on a whiteboard. While this sounds relatively straightforward, it can actually be quite challenging. Today we're going to learn!

Introduction: Whiteboarding Techniques (10 mins)

While one of the main goals in whiteboarding is to solve the problem you are given, the interviewers are also finding out much more about who you are and how you work.

Here are some extremely important skills to practice while at the whiteboard:

- **Keep talking:** One of the worst things you can do is walk up to the board and start writing your code to solve the problem without communicating your plan of attack or explaining what you are writing to your interviewer. As soon as you have gathered your thoughts, start brainstorming out loud about how you are thinking of solving the problem. As you are writing code, explain what the goal of the code is. **Part of the goal of whiteboarding is to get a feel for how you will communicate with other team members.**

- **Ask questions:** If you are unclear about part of the problem, ask questions. Interviews are your time to shine, and you don't want to waste it by going down the wrong path because of a misunderstanding.

- **Start with an example:** Before you start writing code, write out a few examples of what your code is trying to accomplish. This is extremely useful, because as you are coding, you can refer back to it to make sure you are still on track, and then at the end, you can trace the example through to show the interviewer it is working. If a diagram is appropriate, draw that too!

- **Pseudocode:** Before you dive into the code, write out pseudocode, or at the very least, list out the steps you are going to take. Not only will this give you a guide to refer back to while you code, but it can help you spot mistakes in your logic before you start coding. Again, this ties back to the ability for your interviewer to understand your overall thought process.

If you can't finish a question, that's ok! It's more important that you show the interviewer how you approached the problem and to give them a good idea of how you code.

Guided Practice: Whiteboarding Examples (15 mins)

Now that we've discussed some techniques, let's walk through a real example.

Suppose we had the following question:

```
Given two arrays of integers of equal length, that are identical except for a single
element, write a method to return the index of the number which is different between
the two arrays.
```

First, we should start off with a few quick examples:

```
  findDifference([1,2,3],[1,2,4]) => 2
  findDifference([1,2,3,4],[1,3,3,4]) => 1
  findDifference([0],[1]) => 0
```

Next, we should list the steps we are going to follow in our method.

1. Create a loop that goes through the arrays.
2. At each index, compare the values in both arrays
3. If they are different, return that index, else move on to the next index
4. Return -1 if no difference is found

Finally, we can start writing our code:

```
public int findDifference(int[] arr1, int[] arr2){

  for(int i=0; i < arr1.length; i++){
    if(arr1[i] != arr2[i]){
      return i;
    }
  }

  return -1; //Arrays are identical
}
```

You can see that each step we wrote out corresponds to a certain section of the code. We started off with

concrete examples, then moved on to planning, and then finally wrote out our code.

Independent Practice: Whiteboarding Problems (55 mins)

Now it's your turn! Get into groups of 3-4 people, and each person will take, at most, 10 minutes to pick a problem from the list below to solve. The remaining people will act as the interviewers. Remember to follow the techniques we discussed.

1. Find the character that appears the most time in a given string (i.e. "tomorrow" should return 'o'). Return the character later in the alphabet if there is a tie.
2. An array is supposed to contain the numbers 1-10, but one number is missing. Find and return that number.
3. Write a method that finds the second highest number in an array of integers.
4. Given a non-empty string and an int N, return the string made starting with char 0, and then every Nth char of the string. So if N is 3, use char 0, 3, 6, ... and so on. N is 1 or more.
5. Given an array of ints, return the string "even" if there are more even elements in the array, or "odd" if there are more odd elements in the array.

Conclusion (5 mins)

Whiteboarding can be deceptively difficult and only becomes easier with practice - we're doing this in week 1, so you can immediately start practicing and get better each week. Knowing these basic techniques will help you to relax and concentrate on the problem you are presented with during your interviews.

- Describe the steps we reviewed when approaching a whiteboarding problem?
- Why is it ok to ask questions?

3:2

Layouts

Objectives

After this lesson, students will be able to:

- Identify the layout options provided by Android
- Given a sketch of some components, determine which layout to use
- Use nested layouts to create more complicated views

Preparation

Before this lesson, students should review the following lessons:

- Views 101
- Views 102

Introduction: Layouts (5 mins)

In the past lessons, we have used RelativeLayout as the base of all of our layout file.

In this lesson, we are going to examine three of the main layout types: **RelativeLayout**, **LinearLayout**, and **FrameLayout**.

These layouts are view groups. Exactly how it sounds, they group views and arrange them in a particular way.

Demo: RelativeLayout (15 mins)

`RelativeLayout` is a view group that arranges its child views to be relative to each other or the layout itself. For example, if you had a `RelativeLayout` with buttons inside of it, the following can be true:

- Button 1 is below Button 2
- Button 3 is to the right of Button 4, which is to the right of Button 1
- Button 5 is centered, relative to the RelativeLayout itself

Here's how it looks in xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="16dp">

    <Button
        android:id="@+id/button2"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:layout_below="@+id/button2" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 4"
        android:layout_alignTop="@+id/button1"
        android:layout_toRightOf="@+id/button1"
        android:layout_marginLeft="80dp"/>

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:layout_alignTop="@+id/button4"
        android:layout_toRightOf="@+id/button4" />

    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 5"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Introduction: LinearLayout (5 mins)

`LinearLayouts` show all of its child views linearly.

Think of standing on line somewhere, like a bank or an amusement park. In this analogy, the line is a LinearLayout and the people on line are the views; every person in a straight line, one in front of the other.

LinearLayout, unlike the other layouts, has an orientation attribute. This makes the views line up vertically (from up to down) or horizontally (left to right).

Demo: LinearLayout (5 mins)

Here's how it looks in xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">     // or android:orientation="horizontal"

    <Button
        android:id="@+id/button2"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button one" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 4" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="3"/>

    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="The fifth and last button" />

</LinearLayout>
```

Introduction: FrameLayout (10 minutes)

On the Android Developer website, FrameLayout is described as follows:

FrameLayout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view.

The idea behind FrameLayout is that it occupies a space on the screen, and its child view can be defined within that space. It is suggested that we only put one child view inside the FrameLayout (much like how a picture ***frame*** can only show one picture).

However, if you *really* wanted to, you can put multiple pictures in the same picture frame. Each picture could overlap each other, if need be.

Why use this?

The following screenshot is a real world use of a FrameLayout. From bottom to top, there is an `ImageView` with the background, a `TextView` with the title, and another `ImageView` with the logo.

`FrameLayout` can be used to easily stack views atop one another.

`FrameLayout` also uses the attribute `"android:layout_gravity"`. Gravity defines how a view floats within its layout. A view can occupy the `FrameLayout`'s left, right, top, and bottom edges (or any combination of those), or the center of the layout.

In the above example, the "General Assembly" TextView has a gravity of bottom left, and the logo ImageView has a gravity of bottom right.

Demo: FrameLayout (5 mins)

Here's how the same layout looks in xml:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="200dp">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center"
        android:src="@drawable/background_image"
        android:scaleType="centerCrop" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="General Assembly"
        android:layout_gravity="left|bottom"
        android:textColor="@android:color/white"
        android:textStyle="bold"
        android:layout_margin="12dp" />

    <ImageView
        android:layout_width="75dp"
        android:layout_height="75dp"
        android:layout_gravity="bottom|right"
        android:src="@drawable/logo"
        android:layout_margin="12dp" />

</FrameLayout>
```

Introduction: Nested Layouts (5 minutes)

Many apps have complicated views that cannot be made with one ViewGroup alone. To get around this, you can nest ViewGroups inside other ViewGroups.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        ...

    </FrameLayout>
</LinearLayout>
```

The idea is that the inner layout (in this case, the FrameLayout) is effected by the properties of the parent layout.

Guided Practice: Nested Layouts (15 minutes)

Together, let's examine the image below and figure out its layout. Then, in Android Studio, attempt to build it from scratch. Don't worry about using the correct images, solid backgrounds or placeholders are fine.

Independent Practice: Layouts (20 minutes)

> ***Note:*** *This can be a pair programming activity or done independently.*

Recreate the layout of an app on the Google Play Store - use at least one nested layout. By the way, source images are not required and setting the background as a solid color should suffice.

Provide screenshot

Conclusion (5 mins)

- What are the three layouts went over today?
- How does one use the three layouts?
- What are nested layouts, and why are they used?

3:3

Views 103 - ListViews and ListAdapters

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Given some sample data in a Java list, create a view to show this data
- Create a ListAdapter to connect the ListView and data
- Create a BaseAdapter
- Implement `OnItemClickListener` to make the ListView respond to the user

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Declare and use Java Lists
- Describe the basics of Android Views

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Open and run the starter and solution code
- Modify sections and checks as needed

LESSON GUIDE

TIMING	TYPE	TOPIC
5 min	Opening	Discuss lesson objectives
10 min	Introduction	ListViews
5 min	Demo	ListViews
10 min	Introduction	ListAdapters
10 min	Demo	ListAdapters
15 min	Guided Practice	ListAdapters
10 min	Introduction	View Recycling

TIMING	TYPE	TOPIC
20 min	Independent Practice	ListViews and ListAdapters
5 min	Conclusion	Review / Recap

Opening (5 mins)

Previously, we learned about how to create and work with Lists in Java. These are a very important part of many Android apps, so Android provides some very useful tools for integrating them into your layouts. Today we will be covering three main topics: ListViews, ListAdapters, and how we interact with them. ListViews and ListAdapters are tied very closely together. The first is, as the name implies, it's a View that contains a List. The second is what ties (or adapts) our List data to the ListView.

Introduction: ListViews (10 mins)

ListViews are one of the important Views in Android apps that can be used to display collections of data. So many of your daily apps might use ListViews: Contacts, email, music, text messages, and many more. Anything that contain lists of data with repeated visual elements can use a ListView.

There are two major components to a ListView. First, there is the actual ListView item in your Layout. This is just like any other View, with a width, height, and id.

The second component is the item contained in the ListView. For each piece of your data, a copy of the layout for the item is created, populated with your data, and inserted into the list. Combining these two components, you get the ListViews you see every day.

Demo: ListViews (5 mins)

Starting with an empty Android project, we will add a ListView to the activity. We won't be populating it until we cover ListAdapters in the next section of this lesson.

First, let's open up the layout XML file and add the ListView

```
<ListView
       android:id="@+id/list_view"
       android:layout_width="match_parent"
       android:layout_height="match_parent"></ListView>
```

Next we're going to discuss the ListAdapter so we can insert our data into the list.

```
ListView listView = (ListView)findViewById(R.id.list_view);
```

Introduction: ListAdapters (10 mins)

ListAdapters are the other part of the puzzle for showing your List data on the screen. Your data is passed directly into your adapter. After the adapter has your data, and you have specified how you want to display your data, you connect the adapter to your ListView. Once that happens, your Adapter takes an item from your List, creates a new list item, applies the data to the appropriate layout for the item, and inserts it into the ListView. This process repeats for every item in your List.

One important thing to note is that if the data in the list changes, you must tell the adapter so it can refresh the ListView on the screen. To do this, you call a method named notifyDataSetChanged on the adapter. This will be shown in the example in the next section.

You don't actually make instances of a ListAdapter because it is an interface. Instead, other classes implement the ListAdapter interface, such as BaseAdapter. Then, other Adapters extend the BaseAdapter.

One very common example is the ArrayAdapter, which can take your Lists and directly map them to the ListView with very little effort.

Demo: ListAdapters (10 mins)

To complete our example, we will create a LinkedList, create an ArrayAdapter (a specific type of ListAdapter), then set the ArrayAdapter to the ListView. The following code shows these three steps.

```
LinkedList<String> mStateList;
ArrayAdapter<String> mArrayAdapter;

mStateList = new LinkedList<>();
mStateList.add("Arizona");
mStateList.add("New Mexico");
mStateList.add("New York");
mStateList.add("Rhode Island");

mArrayAdapter = new
ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,mStateList);

listView.setAdapter(mArrayAdapter);
```

Now, say we wanted to add South Carolina to our list. To do this, we simply call the add method on the list, then notify the adapter.

```
exampleList.add("South Carolina");
arrayAdapter.notifyDataSetChanged();
```

Guided Practice: ListAdapters (15 mins)

Use the provided Android project ListViewsAndAdapters, and complete the `MainActivity.java` and `activity_main.xml`. Work with a partner to do do the following:

- Add a ListView where there is a comment in the XML file
- Add an ArrayAdapter to the newly created ListView, using the data in the provided LinkedList
- Implement the onClick method of the provided FloatingActionButton to remove the first element of the list (if one exists), and refresh the ListView on screen.

Introduction: View Recycling (10 mins)

The examples we have seen so far have relatively small lists of data. Imagine, however, if we had a list with 10,000 items. Does Android create 10,000 list items all at once?

In order to conserve memory and improve performance, Android instead only instantiates enough list items to fill the screen. As you scroll up or down on the list, the adapter retrieves the correct data and replaces the data for the next row, replaces the data in the row that just moved off screen, and moves the updated row to the correct spot. The actual data isn't removed from memory, since it is stored in its own collection.

Introduction: Base Adapters (10 mins)

The ArrayAdapter we saw before provides us with a lot of convenience, but it is extremely limited in what it can actually do. Another adapter that requires more implementation, but is much more flexible, is the Base Adapter.

- **getCount**: Return the size of the data set
- **getItem**: Return the item from the data set at the given position
- **getItemId**: Return the row id (or index) of the position clicked in the list
- **getView**: Return a complete view to display in the ListView

```
BaseAdapter b = new BaseAdapter() {
            @Override
            public int getCount() {
                return 0;
            }

            @Override
            public Object getItem(int position) {
                return null;
            }

            @Override
            public long getItemId(int position) {
                return 0;
            }

            @Override
            public View getView(int position, View convertView, ViewGroup parent) {
                return null;
            }
        }
```

Guided Practice: Base Adapters (5 mins)

Let's try replacing the adapter from the previous example with the list of names to use a Base Adapter.

Introduction: OnItemClickListener (5 mins)

So far we've handled clicking on Views using an `OnClickListener`, but in ListViews, we have a bunch of individual Views. What happens if we try to write an `OnClickListener` for a ListView?

Android Studio tells us that we don't want to use this!

The solution is to use something called the `OnItemClickListener`. Like the OnClickListener, the OnItemClickListener also contains a method where you put the code for what happens when you click. In this case, it's called `OnItemClick`.

Demo: `OnItemClickListener` (20 mins)

Let's walk through the following code:

```
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        //Do stuff
    }
});
```

- **parent**: The parent object refers to ListView object. From there we can do things like retrieve items from the data collection, or access the Adapter behind the ListView.
- **view**: This refers to the actual list item that was clicked
- **position**: This refers to the position in the view
- **id**: The position in the underlying data set

Let's make it show a toast.

```
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Toast.makeText(MainActivity.this,"Item clicked",Toast.LENGTH_SHORT).show();
    }
});
```

Now, let's have it say what position we clicked.

```
Toast.makeText(MainActivity.this,"Item clicked at position
"+position,Toast.LENGTH_SHORT).show();
```

Guided Practice: Setting up `OnItemClickListeners` (10 mins)

Let's take our example further - do this with me.

How would we change the text for the item to show "You clicked position X" for each item? We know that one of the parameters of `onItemClick` is a View, but we need to figure out how to use it.

Check it out:

```
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
  @Override
  public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    TextView textView = (TextView) view.findViewById(android.R.id.text1);
    textView.setText("Clicked "+position);
  }
});
```

Independent Practice: ListViews and ListAdapters (20 minutes)

Build an activity that contains a ListView with two types of interaction. Pressing the FloatingActionButton should add a new list item containing whatever text the student wants. If the student presses on a specific list item, that item should be deleted. Basic code for the FloatingActionButton and long click detection will be provided in the project ListViewsIndependent.

Conclusion (5 mins)

ListViews and Adapters are crucial components to many of the Android apps you will build in your career. They can handle listing data from practically any source you need, and do it with relatively little coding. While we only covered implementing ArrayAdapters today, Android offers other useful built in adapters such as the CursorAdapter for listing information from databases. In addition, you can create your own custom adapters! Basically any data source you have can be used in an adapter.

ADDITIONAL RESOURCES

- ListViews Reference
- ListAdapters Reference

3:4

ListViews and ListAdapters - Bookshelf

Introduction

In this lab, you will be building a ListView that shows your collection of books (using Book objects). Complete all of the TODO items marked in the starter code.

The user should be able to mark a book as read when they are done. When you click on a book in the list, the text color should change to red.

Hint: You can use android.R.layout.simple_list_item_2 to display text.

Exercise

Requirements

- Make an activity that contains a ListView which displays the book title and author
- Complete the BaseAdapter to display the book information
- Clicking a Book title changes the text to red

Bonus:

- Add additional book info, and use a custom layout to display it in each list item
- Allow the user to sort books by title and author

Deliverable

The screenshot below shows the completed app.

Additional Resources

- Java Listview
- Jave ListAdapters

3:5

ListViews, List Adapters HW

Introduction

Football season is underway! Now that we've learned about ArrayLists and ListViews, you're going to build your own fantasy football roster. You will be able to type in your player names, click an add button, and have the players appear on your list. You'll also have the ability to drop players from your list. (No knowledge of football is actually required to complete this lab.)

This lab will help you to practice creating and manipulating Java collections as well as creating and using ListViews/Adapters.

Exercise

Requirements

- `EditText` that allows the user to type in a player's name
- `Button` that will add the name currently entered in the `EditText` to your roster (only if the `EditText` isn't blank, otherwise set an error message)
- `ArrayList` to store the players in your roster
- `Button` that will remove the most recent player added to your roster
- A `ListView` to display all players in your roster `ArrayList`
- An `Adapter` to tie together your `ListView` and your `ArrayList`

Bonus:

- Add football/player images or logos next to the player names (or just set a background color on an `ImageView` as a placeholder)
- Add player positions with the players (might help to create a custom `Player` class to hold all the info for each player)
- Don't allow duplicate players on the roster (combination of name and position)

Starter code

There is no starter code for this lab.

Deliverable

An Android app that follows the requirements above. There are no design requirements aside from having the EditText, ListView, and 2 Buttons; feel free to play around with color, font, backgrounds, etc.

Submit a pull request with your project on GitHub. Ensure that your project builds and runs successfully.

Here are screenshots of an app meeting the requirements, and one with the bonuses:

Additional Resources

- Java ArrayList
- ListView

3:6

Whiteboard Practice

Exercise

Team up in groups of 2 or 3 and take turns working on the following problems. Pretend you are in an interview setting:

- Keep talking!
- Ask questions (to the instructors, or to your teammates)
- Consider starting with an example, showing the expected outputs for sample inputs
- Consider writing pseudocode before Java to make your steps clear to both you and the interviewer

Requirements

Take turns working through these problems on the whiteboard. It's OK if you don't finish them all; make as much progress as you can.

1. `centeredAverage`: Return the "centered" average of an array of ints, which we'll say is the mean average of the values, except ignoring the largest and smallest values in the array. If there are multiple copies of the smallest value, ignore just one copy, and likewise for the largest value. Use int division to produce the final average. You may assume that the array is length 3 or more.

2. `repeatSeparator`: Given two strings, word and a separator sep, return a big string made of count occurrences of the word, separated by the separator string. For example, `repeatSeparator("Word", "X", 3)` returns "WordXWordXWord".

3. `makeBricks`: We want to make a row of bricks that is goal inches long. We have a number of small bricks (1 inch each) and big bricks (5 inches each). Return true if it is possible to make the goal by choosing from the given bricks. This is a little harder than it looks and can be done without any loops.

Deliverable

No deliverable - just practice working through problems out loud on a whiteboard

3:7

RecyclerViews

Objectives

- Create and use custom `RecyclerView ViewHolders`
- Create and use custom `RecyclerView Adapters`
- Explain difference between the `RecyclerView` layout managers and when to use them.

Preparation

- You should be familiar with creating custom layouts
- You should be familiar with creating custom objects / classes
- You should be familiar with creating custom `ListView` adapters
- You should be familiar with implementing and setting custom `OnClickListener`

Introduction: RecyclerView

In last week's lesson we learned about ListViews and ListView adapters. The Android framework provided us with built in Adapters that did most of the work for us. We also learned about BaseAdapter, which allowed us to create an adapter that can handle our custom XML layouts. Today we're going to learn about the components that make up a complete implementation of RecyclerView. There are a lot of components involved so let's get started.

Follow these steps

- Add the RecyclerView dependency
- Set up the `RecyclerView` in `XML`
- Get a reference to the `RecyclerView` in `Java`
- Configure `RecyclerView` with a `LayoutManager`
- Create a custom `XML` layout to use with `RecyclerView`
- Create a custom `Java` object to hold information for use in `RecyclerView`
- Create a custom `ViewHolder` that uses the `XML` layout created earlier
- Create a custom `RecyclerView Adapter` and implement the custom `ViewHolder`
- Make sure there is a list of the custom `Java` objects available for you to give the custom `RecyclerView Adapter`
- Set `RecyclerView Adapter` on `RecyclerView` you referenced

Guided Practice: Setting up the RecyclerView components(20 min)

RecyclerView dependency

```
compile 'com.android.support:recyclerview-v7:24.2.1'
```

In XML

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/recyclerview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

In onCreate()

```
public class MainActivity extends AppCompatActivity {

    RecyclerView mRecyclerView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mRecyclerView = (RecyclerView) findViewById(R.id.recyclerview);

        LinearLayoutManager linearLayoutManager =
                new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false);
        mRecyclerView.setLayoutManager(linearLayoutManager);

    }
}
```

Creating custom layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="horizontal"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:background="#d4e1ff"
              android:layout_marginBottom="8dp">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:id="@+id/textview_1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        <TextView
            android:id="@+id/textview_2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        <Button
            android:id="@+id/button_1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="First Button"/>
    </LinearLayout>

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
```

```
            android:id="@+id/textview_3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        <TextView
            android:id="@+id/textview_4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        <Button
            android:id="@+id/button_2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Second Button"/>
    </LinearLayout>

</LinearLayout>
```

Creating custom object

```
public class CustomObject {

    private String mText1;
    private String mText2;
    private String mText3;
    private String mText4;

    public CustomObject(){
        mText1 = "Text 1";
        mText2 = "Text 2";
        mText3 = "Text 3";
        mText4 = "Text 4";
    }

    public CustomObject(String text1, String text2, String text3, String text4) {
        mText1 = text1;
        mText2 = text2;
        mText3 = text3;
        mText4 = text4;
    }

    public String getText1() {
        return mText1;
    }

    public void setText1(String text1) {
        mText1 = text1;
    }

    public String getText2() {
        return mText2;
    }

    public void setText2(String text2) {
        mText2 = text2;
    }

    public String getText3() {
        return mText3;
    }

    public void setText3(String text3) {
        mText3 = text3;
    }
```

```
    public String getText4() {
        return mText4;
    }

    public void setText4(String text4) {
        mText4 = text4;
    }
}
```

Creating Custom ViewHolder

```
public class CustomViewHolder extends RecyclerView.ViewHolder {

    public TextView mTextView1;
    public TextView mTextView2;
    public TextView mTextView3;
    public TextView mTextView4;
    public Button mButton1;
    public Button mButton2;

    public CustomViewHolder(View itemView) {
        super(itemView);

        mTextView1 = (TextView) itemView.findViewById(R.id.textview_1);
        mTextView2 = (TextView) itemView.findViewById(R.id.textview_2);
        mTextView3 = (TextView) itemView.findViewById(R.id.textview_3);
        mTextView4 = (TextView) itemView.findViewById(R.id.textview_4);
        mButton1 = (Button) itemView.findViewById(R.id.button_1);
        mButton2 = (Button) itemView.findViewById(R.id.button_2);

    }
}
```

Creating Custom Adapter

```
public class CustomRecyclerViewAdapter extends RecyclerView.Adapter<CustomViewHolder> {

    List<CustomObject> mCustomObjectsList;

    public CustomRecyclerViewAdapter(final List<CustomObject> customObjectList){
        mCustomObjectsList = customObjectList;
    }


    @Override
    public CustomViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View parentView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.custom_layout, parent,
false);
        CustomViewHolder viewHolder = new CustomViewHolder(parentView);
        return viewHolder;
    }

    @Override
    public void onBindViewHolder(CustomViewHolder holder, final int position) {

        holder.mTextView1.setText(mCustomObjectsList.get(position).getText1());
        holder.mTextView2.setText(mCustomObjectsList.get(position).getText2());
        holder.mTextView3.setText(mCustomObjectsList.get(position).getText3());
        holder.mTextView4.setText(mCustomObjectsList.get(position).getText4());

        View.OnClickListener onClickListener = new View.OnClickListener() {
            @Override
```

```
            public void onClick(View view) {
                switch (view.getId()){
                    case R.id.button_1:
                        Toast.makeText(view.getContext(), "You clicked button 1",
Toast.LENGTH_SHORT).show();
                        break;
                    case R.id.button_2:
                        Toast.makeText(view.getContext(), "You clicked button 2",
Toast.LENGTH_SHORT).show();
                        break;
                    default:
                        Toast.makeText(view.getContext(), "You clicked row " +
position, Toast.LENGTH_SHORT).show();
                }
            }
        };

        holder.mButton1.setOnClickListener(onClickListener);
        holder.mButton2.setOnClickListener(onClickListener);

    }

    @Override
    public int getItemCount() {
        return mCustomObjectsList.size();
    }
}
```

Setting the adapter

```
List<CustomObject> customObjects = new ArrayList<>();
for (int i = 0; i < 20; i++) {
    customObjects.add(new CustomObject("A Text","Another Text","Some other Text","Yet
another Text"));
}

mRecyclerView.setAdapter(new CustomRecyclerViewAdapter(customObjects));
```

Completing the implementation

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mRecyclerView = (RecyclerView) findViewById(R.id.recyclerview_main);

        LinearLayoutManager linearLayoutManager =
                new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false);
        mRecyclerView.setLayoutManager(linearLayoutManager);

        List<CustomObject> customObjects = new ArrayList<>();
        Collections.fill(customObjects, new CustomObject());

        mRecyclerView.setAdapter(new CustomRecyclerViewAdapter(customObjects));

    }
```

Independent Practice (30 min)

Now that we have an implementation under our belts, let's try another one. This time your layout will have two buttons, a "Green Button" and a "Red Button". When the Red button is pressed, the background color for that item changes to red, and pressing green changes it to green. This change must persist when the recycler view is scrolled up and down.

Unlike ListViews, there are more granular ways to notify the adapter that information has changed.

```
notifyItemChanged(position);
notifyItemInserted(position);
notifyItemRemoved(position);
notifyItemMoved(fromPos,toPos);
```

Conclusion

- What are the different LayoutManagers?
- What are two components required to complete an implementation of RecyclerViews?

3:8

Paper Prototyping

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Explain what prototyping is and why it is important to the UX process.
- Identify different prototyping methods.
- Apply what you have learnt by creating a paper prototype

STUDENT PRE-WORK

None

Opening (5 mins)

Let's discuss: How do you know if your app design or process is working?

Introduction: What is a Prototype? (15 mins)

A prototype is an early sample, model or release of a product built to test a concept or process or to act as a thing to be replicated or learned from. It is important to test early and as often as possible.

So why do we prototype? Well, we use prototypes primarily as a way to refine and explore a concept. We can also use prototypes to communicate with our team and additional stakeholders.

Let's review some prototyping methods and tools...

Guided Practice: Discuss Prototyping (15 mins)

Imagine this prompt: You are designing an app that helps people find the nearest subway station or bus stop.

- What questions do you have as the designer?
- What do you want to test?
- Based on the methods we just discussed, What prototyping methods will you choose? Why?

Work with the person next to you to discuss these questions.

Introduction: Why Paper Prototyping? (10 mins)

Paper is flexible cheap and quick, key aspects to ensure testing happens as frequently and as early as possible. Here's some great context from a StackOverflow post:

- It's faster. Significantly faster than any other prototyping tool for many situations.

- It encourages different kinds of feedback. People respond differently to sketches on paper than they do to things on a computer - even rough looking things on a computer. One is very obviously temporary. People seem to be more open to different kinds of feedback.
- It's adaptive. I can change what a paper prototype does in the middle of an interaction. Somebody clicks on the photo rather than the edit button next to the photo I can go "gosh - they think that will let then edit" and just throw up the edit page. A new interaction model comes to mind during a session - and you can sketch something out and try it live.

Let's practice creating paper prototypes.

Independent Practice: Jump In (30 mins)

Let's practice prototyping using an activity taken from Stanford:

Your goal is to design the interface of an Android app that controls the climate in your house or apartment.

Assume you are able to get the data about your house/apt layout into this app.

The interface should be able to:

- Set the desired temperature and air flow for a room or for the whole house/apt
- See what the temperature and setting is for a room in the house/apt
- Schedule temperature changes according to a fixed plan (day and night, weekends,…)
- See energy use information in a way that helps determine the best settings

Spend 5 minutes on each of the following steps:

- Sketch the different screens of your app (feel free to use cards, Post-Its, tape, etc., and design a “working” paper prototype)
- Test this prototype on a member of another group, making notes on the problems and potential design changes.
- Revise the aspects of the interface that are most in need of change.
- Test again!

A very useful tool for turning paper prototypes into something you can view & interact with on your phone is the POP app.

Conclusion (15 mins)

- Let's reflect and note what you learned about your techniques and specific design from the previous activity.
- Were your designs realistic?
- Could you test the actual interactivity?
- Were your “users” satisfied?

Additional Resources:

- IDEO video: www.youtube.com/watch?v=M66ZU2PCIcM
- Nordstrom video: www.youtube.com/watch?v=szr0ezLyQHY
- POP app: https://popapp.in/

3:9

Fun? With RecyclerViews

Introduction

> ***Note:*** *This can be a pair programming activity or done independently.*

In this lab you will be using your knowledge of RecyclerView to build an app.

Exercise

Requirements

Activities

- You need to create one Activity to hold your RecyclerView

RecyclerView

- Add the RecylerView to your XML
- Get a reference to the `XML` created `RecyclerView` inside the `onCreate()` method in `Java` in both activities.
- Set up the layout manager for the `RecyclerView` in both activities.

Custom View Layout

- Create a custom `XML` layout that replicates the following.

Custom RecyclerView ViewHolder

- Create a custom `RecyclerView ViewHolder`.
- The `ViewHolder` should get a reference to the custom view layout you created in `XML`.

Custom Object

- Create a custom `Java` objects that hold this data:
- The object should hold a `title`, a `description`, a `color`, and a `check` to see if the item was selected.
- Make sure you create `getters/setters` and `constructors` that will help you with the requirements.

Custom RecyclerView Adapter

- Create a custom `RecyclerView Adapter`.
- The adapter should use the custom `ViewHolder`.

OnClickListener

- Have the `OnClickListener` show a `Toast` that changes the checked value of both the object and view. Make sure it maintains that checked state when scrolling.

MainActivity

- Create and populate a `List` that contains ten instances of your custom `Java` object.
- Provide the custom adapter this list to use.
- Set your custom adapter on the `RecyclerView`.

Bonus: How would you change your classes to allow for more than one type of view in one RecyclerView adapter?

Deliverable

A pull request from forked GitHub repo that contains your implementation of the requirements.

Additional Resource:

Android: RecyclerView

3:10

Project #1: To-Do List

Overview

It's time to create your very first Android project - a to-do list app. This might seem intimidating at first, but you have the ability to break down what you see on the screen into easily identifiable parts. Learning to both design an app and implement your designs are crucial skills to being a successful Android developer.

You will be working individually for this project. Think through your design carefully before you implement it in Android Studio. Test both the layout and functionality of each screen as you create them. Don't wait until the end!

Requirements

Your app must:

- **Display a collection of to-do lists**
- **Display items on each to-do list, including descriptions**
- Allow the user to **create a new to-do list**
- Allow the user to **Add items** to each to-do list
- Allow the user to **remove items from a to-do list**
- Allow the user to **remove an entire to-do list**
- Use two custom Java objects to contain your data for each `ToDoList`, and the `ToDoItem`s in each `ToDoList`
- Show an error message if invalid input is given (e.g. blank input)
- Persist data (your `ToDoList`s) while the app is open using a **singleton**
- Use either `ListView` or `RecyclerView` to show your `ToDoList`s and `ToDoItem`s. We recommend `RecyclerView`, but will accept `ListView` for this project.
- Display correctly in both landscape and portrait orientations

Bonus:

- Allow the user to check off completed items
- Allow the user to edit previously added items
- Persist data when the app closes and is re-opened

Code of Conduct

As always, your app must adhere to General Assembly's student code of conduct guidelines.

If you have questions about whether or not your work adheres to these guidelines, please speak with a member of your instructional team.

Necessary Deliverables

- A **git repository hosted on GitHub**, with frequent commits dating back to the **very beginning** of the project. Commit early, commit often.
- **A `readme.md` file** describing what the app does, and any bugs that may exist
- At least one screenshot of your home screen in the `readme.md`
- Pictures of your prototypes in the git repository

Suggested Ways to Get Started

- Complete as much of the layout XML as possible before starting to write your logic
- Use the Android API documentation - it is very thorough and provides useful code samples
- Don’t hesitate to write throwaway code to solve short term problems
- Write pseudocode before you write actual code (remember to think through the logic first!)

Useful Resources

- Android API Reference
- Android API Guides

3:11

Whiteboard Practice - Recursion

Introduction

Today we're going to focus on **recursion**, which is when a method calls itself. You need two things to avoid an infinite loop where the method keeps calling itself forever:

1. A *base case*, which returns a value without any further recursive calls (you can have more than one base case sometimes)
2. A *reduction step*, which changes the input before making the next recursive call, so as to make progress toward the base case

Consider this sample problem:

```
// factorial(5), a.k.a. 5!, is defined as 5 * 4 * 3 * 2 * 1
public static int factorial(int n) {
    if (n <= 1) {
        // this is the base case, when n is 1 (or less than 1)
        return 1;
    } else {
        // otherwise, reduce the input by 1 and make a recursive call
        return n * factorial(n - 1);
    }
}
```

Sketch of the recursive calls on the memory stack for `factorial(4)`:

Exercise

Team up in groups of 2 or 3 and take turns working on the following problems. Pretend you are in an interview setting:

- Keep talking!
- Ask questions (to the instructors, or to your teammates)
- Consider starting with an example, showing the expected outputs for sample inputs
- Consider writing pseudocode before Java to make your steps clear to both you and the interviewer

Requirements

Take turns working through these problems on the whiteboard. It's OK if you don't finish them all; make as much progress as you can.

1. Write a recursive function called `reverse` that accepts a string and returns a reversed string. For example:
    - `reverse("abc")` -> "cba"
    - `reverse("ab")` -> "ba"

    - `reverse("a")` -> "a"
    - `reverse("")` -> ""
2. The fibonacci sequence is 1, 1, 2, 3, 5, 8, etc. where the sequence starts with two 1s, then each following entry is the sum of the previous two entries. Write a recursive function called `fib` that accepts a number `n`, greater than zero, and returns the Nth fibonacci number. For example:

    - `fib(1)` -> 1
    - `fib(2)` -> 1
    - `fib(3)` -> 2
    - `fib(4)` -> 3
    - `fib(5)` -> 5
    - `fib(-1)` and `fib(0)` -> 0 (input shoud be >= 1)
3. A *palindrome* is a string that is spelled the same backwards and forwards. Put another way, a palindrome is a string where the first letter is equal to the last letter, and the second letter is equal to the second to last letter and so on and so forth. An empty string is considered a palindrome. A one letter string is considered a palindrome.

    Write a function called `isPalindrome` that accepts a string and returns `true` if the string is a palindrome, and returns `false` if the string is not. For example:

    - `isPalindrome("")` -> true
    - `isPalindrome("a")` -> true
    - `isPalindrome("ab")` -> false
    - `isPalindrome("abba")` -> true
    - `isPalindrome("catdog")` -> false
    - `isPalindrome("tacocat")` -> true

Deliverable

No deliverable - just practice working through problems out loud on a whiteboard

3:12

Activities and Intents

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Define what an Activity is
- Understand the components of the Manifest
- Given a running application, identify the activities in that application
- Identify the same activities in the app manifest
- Use Intents to move between Activities
- Pass data between activities

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Create an activity
- Use nested layouts to create more complicated views
- Explain view recycling and why it matters

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Read through the lesson
- Add additional instructor notes as needed
- Edit language or examples to fit your ideas and teaching style
- Open, read, run, and edit (optional) the starter and solution code to ensure it's working and that you agree with how the code was written
- **Note**: With the addition of `startActivityForResult`, this lesson is unusually long. Consider giving the students a break mid-way through.

Opening (5 mins)

Previously, we learned about the UI elements that make up a screen in an app, and how we interact with it in the Java code. Notice how the file was called MainActivity.java? Most of the time, you can think of an Activity as a screen of your app - that's all!

> Check: If Activities are screens, why do you think they are called "Activities"?

Introduction: What is an Activity? What is a Manifest? (10 mins)

The definition of an activity is something that is done for a particular purpose.

Think about the activity that the user is doing on a screen. If you are looking at a screen whose purpose is to log in the user, for example, it should be called the Login Activity. If the activity shows a user's social network profile , it should be called the User Profile Activity.

Let's add a new Activity to this app. Right click on the folder where you want to put the activity, then go to *New > Activity*, and then click on the type of base activity you want (usually, *Empty Activity* is what you want if you want to build it from scratch).

An Activity is a plain ol' Java class, so you already know how to add it to a project.

Doing this also adds the Activity to the Android Manifest.

> Check: So what is an activity again?

What is a Manifest?

The dictionary defines a ship's manifest to be "a document giving comprehensive details of a ship and its cargo and other contents, passengers, and crew for the use of customs officers."

The ship, in our case, is the app you are building.

The Android Manifest xml file presents important information about your app to the Android system. If something isn't defined in the manifest, then the system just ignores it.

Notably, the manifest is known for describing the main components of your app; the Activities, Services, Content Providers, etc. It is also the place to define permissions (e.g., giving your app permission to access the internet or to access a device's camera).

Whenever you create a new Activity through the New Activity menu, Android Studio will automatically add it to the manifest.

> Check: How do activities relate to the manifest?

Demo: Creating Activities (15 mins)

In this demo, let's walk through the following:

- Creating a new project in Android Studio
- Examining the manifest file, describing components like the XML elements, attributes, and package
- Describe what a launcher activity is (The activity that opens when the app is launched), and make comparisons to Java's *public static void main* method
- Add 2 more activities to the project, and go back to the manifest and see them added

> Instructor Note: A complete example of this is found in the solution code folder.
>
> Check: Take 2 minutes, with the person next to you, and discuss what a launcher activity is, and how it compares to public static void main. Be ready to share out!

Introduction: What are Intents? (10 mins)

Intent, as defined in the dictionary, means: purpose, goal, objective. *Something intends to do some goal*.

This translates to your app and activities; every activity has a goal.

For example, a `ComposeEmailActivity` allows the user to compose and send an email. If you click a "compose new email" button, you are actively saying "I intend to compose an email".

This is the idea behind Intents in Android. Intents are messages you send between app components, like Activities, usually with the goal of doing something.

> Check: So, imagine you are in your app's `EmailListActivity`, and you click on one of your emails. In plain ol' English, could you describe what is happening between the user and the EmailListActivity? Take 10 seconds to think about it.

The following "dialogue" is happening:

- EmailListActivity: "Hey, you clicked one of your emails. What's up?"
- You: "I intend on reading that email. Is that okay?"
- EmailListActivity: "Yeah, sure! I'll start the ReadEmailActivity now."
- You: "Thank you."

Let's create an EmailListActivity and a ComposeEmailActivity.

So, how does an Intent look like in code?

```
Intent intent = new Intent(EmailListActivity.this, ComposeEmailActivity.class);
startActivity(intent);
```

You create a new Intent object, and you pass it two parameters: The activity you are currently in, and the class of the activity you intend to start. The code snippet above could be read as, *From the Email List Activity, please start the Compose Email Activity*.

The method, `startActivity()`, starts the intended activity immediately.

> Check: What two parameters should you pass your Intent objects?

Independent Practice: Starting an activity with an Intent (10 mins)

Using the code from the previous demo, add a button to the app's main activity. Set an `onClickListener` to that button and have the listener start one of the other activities. Run the app in a virtual device, and click on the button to start the new activity.

Note: A complete example of this is found in the solution code folder.

Introduction: Sending data from one Activity to another Activity (10 mins)

Intents are how Activities communicate with each other. In the previous example, we started an activity to compose an email by clicking an email in the list. However, how does the ReadEmailActivity know what email to show?

> Check: Take 30 seconds to talk with the person next to you about this question.

When you start a new activity, it is shown with the default settings that you give it. However, some activities need to receive a bit more information. This info is sent from the original activity to the one you are starting.

> Let's rename our Activities to match the email example

When creating new intents, you can also give it *extra* data. Here's an example:

```
Intent intent = new Intent(EmailListActivity.this, ReadEmailActivity.class);
intent.putExtra("ID", 123);
intent.putExtra("SENDER", "John Smith");
startActivity(Intent);
```

> Check: Take 20 seconds to study the code and come up with an explanation, in English, about what's happening. Be ready to share!

The Intent class has a handful of helper methods you can call to get and store extra data. The main one is `putExtra()`, which takes two parameters: a String that gives the data a name, and the data itself.

With `intent.putExtra()`, you can put data inside the intent (including Strings, numbers, booleans, certain objects).

Once you start a new activity, you can retrieve the Intent and get the sent data, as follows:

```
// get the intent that started this activity
Intent intent = getIntent();

// get the data from the intent
int id = intent.getIntExtra("ID", 0);
String sender = intent.getStringExtra("SENDER");
```

Again, the Intent class has a handful of getters for extra data, usually formatted like *get_____Extra*. Examples, `getIntExtra()`, `getStringExtra()`, `getBooleanExtra()`, etc.

Here we are hard-coding the Keys, what do you think we could do as an alternative?

Note: You should only send data if you need to do so. If the activity you are starting doesn't need extra data, you don't have to set it.

Guided Practice: Sending Data between Activities (15 mins)

> Instructor Note: Show the class that getting data without setting it or with a type in the extra id would cause incorrect or null data. Then, after you introduce this exercise, be sure to stop the class after each step and reveal the answer.

With the person next to you, go ahead and start a new Android project with a empty main activity. Do the following:

- Add a new Activity, and call it `EchoActivity`.
- In `MainActivity`, add an `EditText` and a Button. In the `EchoActivity`, it will just have a `TextView`.
- In Java, set an `onClickListener` to the button and make it start the `EchoActivity`. The value of the `EditText` is passed in the intent.
- In the `EchoActivity`, the value is plucked from the intent and put in the TextView.

Independent Practice: Add two numbers (15 mins)

> This should be done as a pair programming exercise.

Now, with the person next to you, without stopping every two minutes, do the following:

- Create a new project, with a blank main Activity
- Create a new activity, call it SolutionActivity
- In the main activity, put two EditTexts and one button in the layout. The button's text will say "Add"
- In the solution activity, just have one TextView
- When the button is pressed, it takes the two values and sends them to the solution activity, where the sum of the two numbers are shown. Add the two numbers in the SolutionActivity.

Demo: Passing data back in the result (20 mins)

Passing data works in both directions, and you can receive data when returning from an Activity you previously started. For example, if you start SecondActivity from MainActivity, you can get data back from SecondActivity when it closes.

Returning data from an Activity only requires a few additions to your code.

This can be broken down into changes in your calling Activity (the one you are starting the second activity from), and changes in your secondary Activity.

Open the Starter-Code and follow along.

Second Activity

First, let's look at the second Activity, or the one you are passing the data back from. In this Activity, we press a button, and it passes the values from the two EditTexts back to the Main Activity.

> Check: How do you think we are going to pass data from the second activity to the first when clicking the button? Share out!

Just like when we're starting an Activity, we also pass data back using an Intent and extras.

```
mButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
    Intent resultIntent = new Intent();
    resultIntent.putExtra("first",mFirstNameText.getText().toString());
    resultIntent.putExtra("last",mLastNameText.getText().toString());
```

```
    setResult(RESULT_OK,resultIntent);
    finish();
}
});
```

First we create an intent, then put the two Strings in as extras. The next two lines of code are new, though.

- **setResult**: This method takes two parameters. The first is a value that lets our first Activity know that everything went well, and that this Activity finished correctly. The second parameter is simply an Intent holding the data we want to pass back.
- **finish**: This method closes the current Activity and returns to the previous one.

Main Activity

Now that we've finished the Second Activity, let's return to the Main Activity. We have two steps to complete.

> Check: What are they? You know this by now.

We need to start the Second Activity and get the results from the Second Activity.

Starting the Activity is almost identical to how we previously did it, with one exception:

```
Intent intent = new Intent(MainActivity.this,SecondActivity.class);
startActivityForResult(intent,NAME_REQUEST);
```

Notice the `startActivityForResult` method. The first parameter is a normal Intent, but the second is a variable telling the system what we are asking for (we will use this in a minute). We need to add that variable at the top of the Activity.

```
private static final int NAME_REQUEST = 20;
```

You can assign any integer value that is **greater than 0**.

Next, we have to get the results from the Second Activity. Whenever you return from an Activity that is expecting results, the `onActivityResult` is automatically called.

> Check: Give the students 2 minutes to discuss what data they think will be returned to us based on what was passed into the second activity and what was returned in the result after pressing the button.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // Check what request we're responding to...
    if (requestCode == NAME_REQUEST) {
        // Make sure the request was successful...
        if (resultCode == RESULT_OK) {
            String firstName = data.getStringExtra("first");
            String lastName = data.getStringExtra("last");
            mText.setText(firstName+" "+lastName);
        }
    }
}
```

The first parameter is the static variable we passed in when starting the activity, in our case `NAME_REQUEST`. The second parameter is the result status, in our case `RESULT_OK`. The final parameter is

the Intent we passed back containing the data.

Since we could be starting different Activities from our Main Activity, we first have to check that our result is coming from the `NAME_REQUEST` activity, then we have to check to make sure the results are valid. After that, we can retrieve the data like normal and use it however we want.

> Check: What are the key differences between how we started activities before and what we just did?

Independent Practice: startActivityForResult (10 mins)

Do this activity with a partner

Instead of passing values from the first activity to the second, we are going to pass the data back from the second activity to the first.

If the user chooses add, they are taken to a calculate activity with two EditTexts where they type 2 numbers, and a calculate button. The sum of the numbers from the calculate activities is displayed on the main activity after pressing the calculate button.

Bonus:

If the user chooses subtract, the same steps occur, except the difference is displayed in the main activity.

> Check: Let's take 2 minutes to review the answer. Were all students able to complete the activity successfully? If not, where did you get stuck?

Conclusion (5 mins)

- What is an activity?
- What is an intent?
- How do we start an activity?
- How do we send data from one activity to another?
- How do we receive data when returning from an activity?

Additional Resources

- Android Developer | Starting Activities - http://developer.android.com/guide/components/activities.html#StartingAnActivity
- Android Developer | Intents - http://developer.android.com/guide/components/intents-filters.html

3:13

Using Developer Documentation

LEARNING OBJECTIVES

After this lesson, you will be able to:

- Identify an error in an application
- Research the causes of those errors using developer documentation
- Rely on developer documentation to learn something new
- Find reputable sources of information online other than official documentation

STUDENT PRE-WORK

Before this lesson, you should already be able to:

- Open an Android application in Android Studio
- Run an app on an emulator or physical device
- Use logcat to view runtime error messages

INSTRUCTOR PREP

Before this lesson, instructors will need to:

- Read through the lesson
- Add additional instructor notes as needed
- Edit language or examples to fit your ideas and teaching style
- Open, read, run, and edit (optional) the starter and review the many intentional errors the students will need to investigate
- Read and run the solution code to ensure it's working and that you agree with how the code was written

Opening (5 mins)

Story of the re-usable rocket

> Instructor Note: It is recommended to have students pair up for this lesson. By pairing students together, they will see that other students have the same problems that they have. If your cohort shows signs of frustration from errors, it is recommended to encourage them that everyone has problems and even demonstrate an error or two that you ran into when you started developing. Feel free to replace any examples here with examples that you can personally identify with for cohort-to-cohort consistency.

At this point in your journey to become an Android developer, you have likely run into a problem -- or **exception**. You may be familiar with a variety of these such as `IOException`, `ClassNotFoundException`, or `NullPointerException`. We're going to look into exceptions this

morning and find out how to resolve them. To do that, we're going to use **Developer Documentation**.

When Space-X designed their re-usable rocket for shipments to the international space station their engineers had to read hardware and software documentation to master their craft and build a safe and re-usable rocket. What if something went wrong during lift off? What if the rocket failed to land successfully (which it did on multiple occasions)? Space X's engineers needed to look through their error logs and then research causes to problems they found in documentation. As a developer, when you run into a problem you can use *developer documentation* to properly understand errors and the correct way to resolve them. However, sometimes documentation can be confusing - especially when you are new to something.

> Instructor Note: ask students to brainstorm different types and sources of documentation. List them on the board. Possibilities include:
>
> - developer.android.com
> - StackOverflow
> - error messages in Android Studio
> - comments in source code in Android Studio (show students command + click)

Introduction: What Errors Have You Hit? (10 mins)

> Instructor Note: It is recommended to take 5-10 minutes here to discuss how errors are handled on the job. This will help students see a real-world example of why documentation is important. Remember that students are new developers and may not have been exposed to any programming before this so keep terminology and problems simple.
>
> Check: What do you do right now when you run into an error? (2 minutes)
>
> Check: What is the most confusing source of information you've seen? (2 minutes)

Here's what you'll learn

Today's lesson will show you how identify problems in your applications. Once we identify them together, you'll learn how to read documentation and sift through the mountains of search results that you'll find on your own. At the end of this lesson, you will have seen how to use documentation, we'll have done so together, and you'll have learned to read data from Files thanks to developer documentation that already exists.

Demo: Let's look at a broken app (15 mins)

Let's take a look at a basic Android application. Open the starter-code folder for this lesson in Android Studio. Let's run the app and see if we run into any errors.

```
.../starter-
code/app/src/main/java/com/charlesdrews/documentationlesson/MainActivity.java
Error:(35, 17) error: cannot find symbol variable text
```

Oh no! We have a problem! What ever will we do? What would you do if you ran into this problem? Has anyone ran into this exact problem yet? Let's break this message down together. The compiler is telling us that there is a problem and it is doing the best job that it can to tell us in English.

First, we are told that in `MainActivity.java` on line 35, starting at character 17 of the line, we have an `error`. Yeah, we get that, Java; thanks. Java is telling us that it `cannot find symbol`. So it cannot find

something called a symbol. I'm not entirely sure what that is. Wouldn't it be great if I had **developer documentation** to help me out? I guess I'll just have to look for some help online...

> Instructor Note: Search for 'cannot find symbol' and show the results (or use the result below). Read the question that was asked on Stack overflow. Explain to students that there are open and closed communities where other developers can ask for help and receive it for free. Mention that STACK OVERFLOW is one such site.

This Stack Overflow website is one of the most popular websites on the internet for developers to ask for help. It looks like other people have had the same problem! That makes me feel a little bit better about this already. Stack Overflow allows a lot of people to ask questions and then provide answers. There could be many answers for a particular question. The community then votes for the best answer. After reading that person's question, I can't wait to read the answers!

> Instructor Note: Read the accepted answer and explain it 'in English' to students.

Based on what I've read here, it looks like my app failed to compile because I didn't declare the variable `text`. But didn't I declare that up on line 22? Why can't the program recognize it here?

> Instructor Note: Ask students to answer this question. Make sure everyone understands that a local variable in one on-click listener is **out of scope** from the perspective of a different on-click listener.

Right, I declared `text` in the `OnClickListener` for `firstButton`, so it is NOT in scope in the `OnClickListener` for `secondButton`. Let's move the declaration up into `onCreate` so it's in scope for both `OnClickListener`s. Note the Android Studio error message which says `text` must be declared `final` in order to be accessed from within an inner class.

```
protected void onCreate(Bundle savedInstanceState) {
    final EditText text = (EditText) findViewById(R.id.text);
    Button firstButton = (Button) findViewById(R.id.button1);
}
```

Let's compile and run. Nice! So using *developer documentation*, in this case both StackOverflow and Android Studio error messages, I am able to find answers to problems that I run into. The entire internet can contribute to answers. You should, too.

> Instructor Note: Point out that "accepted" answers on StackOverflow are selected by the user that posted the question, and NOT by a panel of experts. There is no guarantee that an "accepted" answer, or any other answer, is correct. Students must carefully read SO answers and be sure they make sense!

Guided Practice: Let's research and fix the remaining errors (20 mins)

Let's work through the remaining errors in the starter code one by one, using developer documentation to guide us.

> Have students find and fix the remaining errors in small groups, then go through them as a class. Use the solution code as needed.

Independent Practice: Let's learn something new (30 min)

Alert Dialogs are a very handy tool in Android apps. They function like popups on a website - they display over the current activity, partially obscuring it, and they typically include buttons like "Cancel" or "OK" to close the dialog and/or complete an action.

Take some time to read through the official documentation for Alert Dialogs on the Android developer site, focusing on the "Building an Alert Dialog" section:
https://developer.android.com/guide/topics/ui/dialogs.html#AlertDialog

> Instructor Note: This should be performed independently by students; it is a good idea to touch on each Learning Objective right before explaining each step.

Task: Launch an alert dialog

1. Add a third button to MainActivity
2. Create an on-click listener for your new button, and launch your alert dialog from within the `onClick()` method
3. Set a title and message for your alert dialog
4. Set positive and negative buttons, and show a Toast message when each one is clicked
5. Don't forget to call the `show()` method on your instance of `AlertDialog` to make it appear to the user

Solution

```
Button thirdButton = (Button) findViewById(R.id.button3);
thirdButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
        builder.setTitle("Here is my title")
                .setMessage("Here is my message")
                .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {
                        Toast.makeText(MainActivity.this, "you clicked OK",
Toast.LENGTH_SHORT).show();
                    }
                })
                .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {
                        Toast.makeText(MainActivity.this, "you clicked Cancel",
Toast.LENGTH_SHORT).show();
                    }
                });

        AlertDialog dialog = builder.create();
        dialog.show();
    }
});
```

> Check: Write and walk through the code that students should have completed during their independent practice. If students are confused, explain that many of their answers can be found in developer documentation. Everyone will find documentation that they like (nobody likes the same flavor of pancakes).

Conclusion (5 mins)

- Why is it important to learn to use documentation?
- How is StackOverflow different than the official documentation?

3:14

Tic-Tac-Toe

> ***Note:*** *You can help each other, but everyone must submit their own code.*

Exercise

In this exercise, you will be creating tic-tac-toe! The app will consist of two activities. The first activity is the main menu where both players enter their names, and the winner of the last game is displayed.

The second activity is the game itself. It has text that displays whose turn it currently is, and who the winner is once the game is over. Below that is the game board, where each player touches a square to place their letter.

You'll need to send data via an Intent, and pass data back as a result. Please review the Activities & Intents lesson for a reminder of how to accomplish these two tasks. The sample code there will be a good example for how to pass data around in this app.

Requirements

- Pass the player names from `MainActivity` to `GameActivity` in an Intent.
- Add logic to control the game play in `GameActivity`:
    - For each turn, show at the top of the screen whose turn it is.
    - When a player selects a square, populate it with an "X" or "O" as appropriate.
    - After each turn, check if either player has won.
- When the game is over, display the winner at the top of the screen.
- When the user navigates back to the `MainActivity` by hitting the back button, pass the winner of the game from `GameActivity` to `MainActivity` as a *result*. (You might want to override the `onBackPressed()` method of the activity, but you can do it other ways too.)
- Back in `MainActivity`, retrieve the *result* in the `onActivityResult()` method, and display it.

Bonus:

- Rather than just the one winner of the previous game, show a list of previous winners in a RecyclerView in `MainActivity`, with the most recent winner at the top.

Starter code

Starter code is provided which has the layouts for the two activities completed. The focus of this exercise is on the Java code.

Deliverable

Complete the tic-tac-toe game similar to the screenshots below:

Additional Resources

- Passing Data with an Intent
- Passing Data Back as a Result