Java Refresher

Roadmap

- Inheritance
- Generics
- Anonymous Inner Classes
- Lambdas (and "::" notation)

Inheritance

- Interfaces: a promise that you will implement these methods
  - Interfaces can only implement other interfaces
  - A class can implement many interfaces
  - Examples: Comparable interface
- Abstract classes: like interfaces but has some fully implemented methods as well
  - Can have abstract functions that are only defined in subclasses like interfaces
  - Also allows you to define shared member variables and functions for all subclasses
  - Examples: Pets all have a name (inherited member variable), are adopted the same way (function defined in abstract class) but eat different foods (abstract function defined only in subclasses)
- Regular class: fully defined behaviors that you want to add to
  - All functions in the parent class have been implemented and are inherited
  - Usually would use this to add more specific behavior by changing implementation or adding new methods

Inheritance common errors and tips

- Be careful:
  - Make sure not to redefine a variable you inherited from a parent class
  - Check and make sure that you are using the same method signature (return types and parameter types) when overriding inherited methods, otherwise this is actually overloading
  - These might lead to undefined and weird behaviors! :(
- Remember:
  - You can only subclass one class, but you can implement as many interfaces as you want
  - Subclasses are able to access and change public and protected member variables of parent
  - You must implement interface methods and all abstract superclass methods

Switch statements

- A form of a conditional with different execution paths

```
public enum EssentialGeometry { INSIDE, ON_EDGE, OUTSIDE };
...
EssentialGeometry where = EssentialGeometry.INSIDE;
switch (where) {
 case ON_EDGE:
   // do the edgy things
   break;
 case INSIDE:
   // do the inside things but also fall through
   // and do the OUTSIDE things because no break statement;
 case OUTSIDE:
   // do the outside things
   break;
 default:
   // do default things
   // automatically falls through
}
```

Private class fields are often labelled with a lowercase “m” at the front

This notation comes from AOSP (Android Open Source Project) Code Style Guidelines for Contributors:

Follow Field Naming Conventions

- Non-public, non-static field names start with ‘m’.
- Static field names start with ‘s’.
- Other fields start with a lower case letter.
- Public static final fields (constants) are ALL_CAPS_WITH_UNDERSCORES.

For example:

```
private float mCircleRadius, mThumbRadius;

private final Paint mPaintStart, mPaintEnd;
```

Enums

An enum type is a special data type that restricts a variable to be a set of predefined constants

```
public enum EssentialGeometry { INSIDE, OUTSIDE };

...

EssentialGeometry where = EssentialGeometry.INSIDE;
```

Generics

Basically, abstraction over types:

Point<Integer>, Point<Double>

```
// Type abstraction: abstract over element type
Interface List<E> {
        Boolean add(E n);
        E get(int index);
}
```

// Lets us use types such as:

// List<Integer>

// List<String>

// List<List<Double>>

Anonymous Inner Classes (1/3)

- In Java, Anonymous Inner Classes are inner classes (or a non-static class that's nested inside another class)
- Anonymous classes *don't have a name* and are often used to make an instance of an object that has slightly different methods of another class or interface. This way, you don't have to actually make a subclass of a class.
- You're going to see this type of class in some of our homework when implementing something called "listeners"

Anonymous Inner Classes (2/3)

```
public class ExActivity extends AppCompatActivity {
    private View.OnClickListener mClickListener = new View.OnClickListener() {
        public void onClick(View v) {
            if (mButton!=v) {
                return;
            }
        }
    }; // remember to end this statement with a semicolon
}
```

Anonymous Inner Classes (3/3)

Digging deeper: Create an Anonymous Class

Let's take some time to parse this...

```
private View.OnClickListener mClickListener = new View.OnClickListener() {
```

- `private` - it's only available inside the class that contains it (i.e. `ExampleActivity`)
- `View.OnClickListener` is the variable type (Documentation) - a nested class in `View`
- `mClickListener` is the variable name which is being set to...
- a `new View.OnClickListener()` which is an anonymous object from an abstract class
  - For those of you who have not taken 331, that means that there are methods that have not been implemented in the class.
  - The on method that you MUST implement (in order to create a new object) is `OnClick` which overrides the abstract method

Lambdas

What are Lambda expressions in Java?

- block of code that can be passed around to execute
- Instances of functional interfaces
- Think of it as using **code** as **data**
- Useful for **anonymous classes** and **functional interfaces**, allows compact instances of one method classes
- This will come up later in the course when dealing with **callbacks!**
- Once instantiated, you can re-use it! Treat it is as a function

Lambda Simple Example

An example functional interface

```
 interface FuncInter1

    {

        int operation(int a, int b);

         int multiplication(int a, int
b);

    }
```

Implementing interface w/ lambda function

```
FuncInter1 add =

(int x, int y) -> x + y;
```

You can reuse this now!

```
add.operation(2, 3) returns 5

add.multiplication(2, 3) return 5
```

Another Example but using : : operator

:: is a method reference, same as using lambda but even shorter and readable

Syntax of :: operator <Class name>::<method name>

Lambda Example

```
numList.forEach(e -> System.out.print(e));
```

This does the same thing!

```
numList.forEach(System.out::print)
```