# Interfaces

Aaron Friesen

# What is an Interface?

- A contract!

# What is an Interface?

- A contract: it defines a *list of methods* that implemented classes are required to have.
- Can only contain constants and method signatures
- Similar to an abstract class, but cannot concretely define any methods.

# The Format of the Interface

```
visibilityModifier interface interfaceName { // can
extends other interfaces


    returnType methodName(parameterList); //all methods in
interfaces are implicitly public and abstract


}
```

# Quick Example

- Hero Interface
  - What are some things that all heroes have to be able to do? (Save Lives)
- Now let’s make implementing classes.
  - Firefighter
  - PoliceOfficer
  - Superman

# Why use Interfaces?

- Java only allows single inheritance (only one super class)
- By contrast, a class can implement as many interfaces as you want, so long as you provide all the methods they require.
- Useful when you want to define functions common among some classes, but don’t want to restrict yourself to an awkward class hierarchy

# Uses w/ Polymorphism

- A class can use any of its interfaces as its **static/reference type.**
- This lets us do most of the familiar polymorphism tricks...
- `Hero hankSchrader = new PoliceOfficer();`
- `Hero[] heroSquad = new Hero[5];`
- `heroSquad[0] = new FireFighter();`
- `heroSquad[1] = hankShrader;`

# Common Interfaces

- Comparable
  - Requires a compareTo(Object other) method
  - This method handles all three “comparing” cases
    - a.compareTo(b) returns a positive number if a > b
    - a.compareTo(b) returns 0 if a == b
    - a.compareTo(b) returns negative number if a < b
- Example: Fattest Cat is BEST Cat

# Uses in Software Design

Consider the following scenario:

- Team A is working with Team B on a large project. Team A is focused on the user interface(UI), while Team B is focused on the actual program logic
- To avoid interdependency and to work more efficiently, both teams decide on a mutual Interface that they then both code to. Team A takes in objects of the interface's type, and Team B makes the relevant objects implement that interface.

FLAWLESS VICTORY!