Introduction to Java

**Purpose**:

Provide an overview of what java is and the basics of flowcontrol. This lesson should reenforce the concepts of Object Orient program.

Objectives:

[Discuss some basic Software Design Principles](#h.v4quqbl4uhr5)

[Understand what Java is (and what it is NOT)](#h.ahl5dic58w7f)

[Understand the Java Compilation Process](#h.rrixbfrh53u3)

[Discuss the Basic Parts and Keywords of a Class](#h.fcgm3jj7xwum)

[Basic Java Structures](#h.fvt42mhamwkd)

[Describe what object oriented programming means](#h.630zagevywbm)

# Discuss some basic Software Design Principles

**Law of Demeter for Functions/Methods (LoD-F)**:  Each unit should have only limited knowledge about other units. Each unit should only talk to its friends; don't talk to strangers. Only talk to your immediate friends

**Separation of Concerns**: Separation of Concerns (SoC) is the process of separating a computer program into distinct features that overlap in functionality as little as possible. A concern is any piece of interest or focus in a program.

**Favor Composition over Inheritance**: Inheritance must be decided at design time. Composition allows you to change behavior at runtime provided the the object you’re using implements the correct interface.

**Open/Closed Principle**: In [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), the open/closed principle states "*software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification*";[[1]](https://en.wikipedia.org/wiki/Open/closed_principle#cite_note-1) that is, such an entity can allow its behaviour to be extended without modifying its [source code](https://en.wikipedia.org/wiki/Source_code).

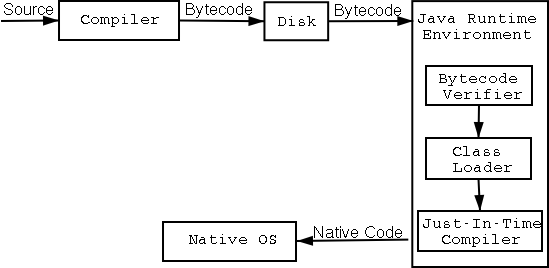
# Understand what Java is (and what it is NOT)

* How does Java relates to Android
  + Java SE, Java ME, Java EE, & Android
* Language vs the Platform

# Understand the Java Compilation Process

Java is a general purpose computer programming language this is concurrent, class-based, and object oriented. It was designed by James Gosling and first appeared in 1995.

**Java** source code is **compiled** into bytecode when we use the javac **compiler**. The bytecode gets saved on the disk with the file extension .class . When the program is to be run, the bytecode is converted, using the just-in-time (JIT) **compiler**. The result is machine code which is then fed to the memory and is executed.



# 

# Discuss the Basic Parts and Keywords of a Class

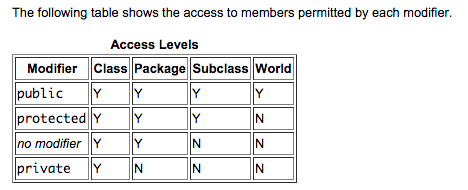
package

* Java 8 JavaDocs
* android javadocs

import statements

* date example

access modifiers:



https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

**Annotations**: When **Java** source code is compiled, **annotations** can be processed by compiler plug-ins called **annotation** processors. Processors can produce informational messages or create additional **Java** source files or resources, which in turn may be compiled and processed, and also modify the **annotated** code itself.

***Annotations are metadata.***

**Concepts**:

Extends keyword for inheritance

Implements keyword for implementing an interface

Difference between extending another class and implementing an interface?

logging: Log.d(a,a)

constants in java ( static final, ALL CAPS)

# Basic Java Structures

Java Naming Conventions

Java Primitives v Objects

A bit is the smallest piece of computer information. byte: Most computers use combinations of **eight bits**, called bytes, to represent one character of data. For example, the word "cat" has three characters, and it would be represented by three bytes. kilobyte (K or KB): A kilobyte is equal to **1,024 bytes**.

**Primitives::**

**boolean:** 1-bit. May take on the values true and false only.

true and false are defined constants of the language and are not the same as True and False, TRUE and FALSE, zero and nonzero, 1 and 0 or any other numeric value. Booleans may not be cast into any other type of variable nor may any other variable be cast into a boolean.

**byte:** 1 signed byte (two's complement). Covers values from -128 to 127.

**short:** 2 bytes, signed (two's complement), -32,768 to 32,767

**int:** 4 bytes, signed (two's complement). -2,147,483,648 to 2,147,483,647. Like all numeric types ints may be cast into other numeric types (byte, short, long, float, double). When *lossy* casts are done (e.g. int to byte) the conversion is done modulo the length of the smaller type.

**long:** 8 bytes signed (two's complement). Ranges from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807.

**float:** 4 bytes, IEEE 754. Covers a range from 1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative).

Like all numeric types floats may be cast into other numeric types (byte, short, long, int, double). When *lossy* casts to integer types are done (e.g. floatto short) the fractional part is truncated and the conversion is done modulo the length of the smaller type.

**double:** 8 bytes IEEE 754. Covers a range from 4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative).

**char:** 2 bytes, unsigned, Unicode, 0 to 65,535

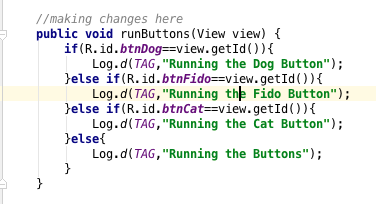
Chars are not the same as bytes, ints, shorts or Strings.

Java is Pass-By-Value:

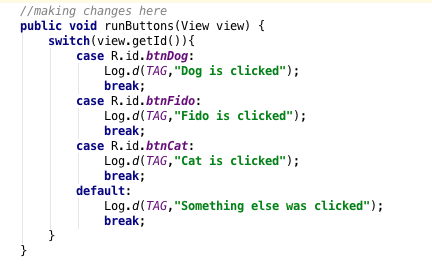
http://javadude.com/articles/passbyvalue.htm

**Control Structures:**

if/elseif/else

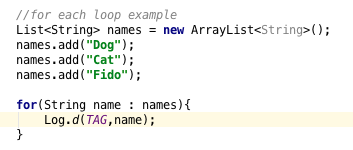


Switch:

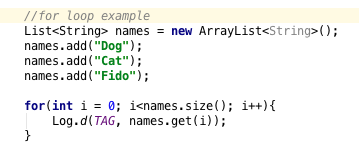


**Looping Structures:**

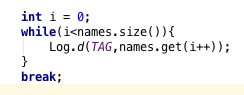
For Each:



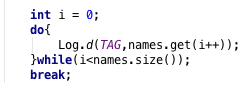
For loop:



while loop:



do while loop:



# Describe what object oriented programming means

When I say a language is object oriented, I mean can be described with:

* Abstraction
* Encapsulation
* Inheritance
* Composition
* Polymorphism

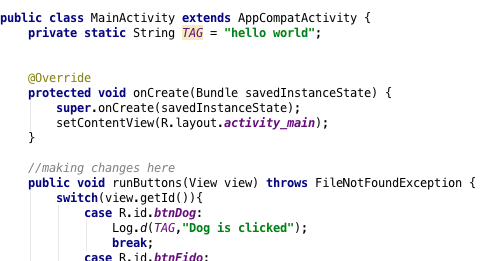
**Abstraction:**

Think about opening a file in Android:



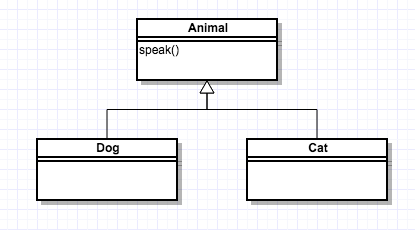
This is an abstraction of what really has to happen. We have api’s that abstract details from us. We work at high and high levels of abstraction.

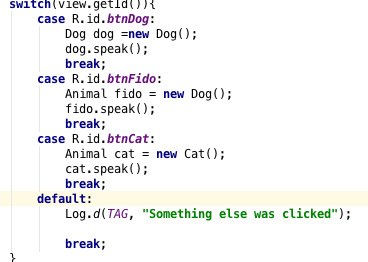
**Encapsulation**:



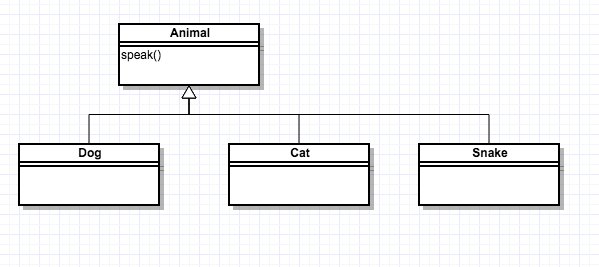
**Inheritance**:

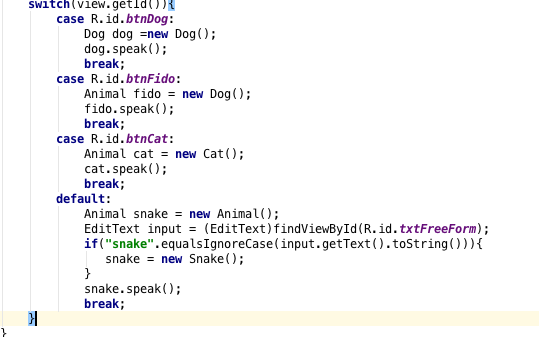
Use animal example. Start by creating a dog, then cat, then animal for shared behavior. Use parent and override local class. Then add snake to it for a different behavior. That is not shared. Discuss interfaces.



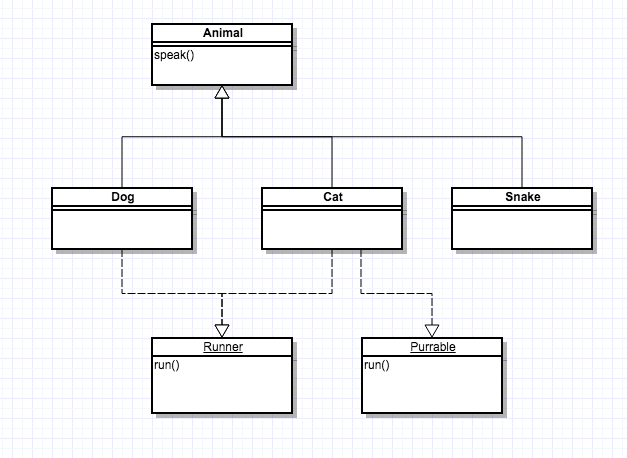


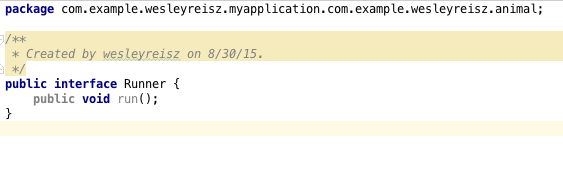
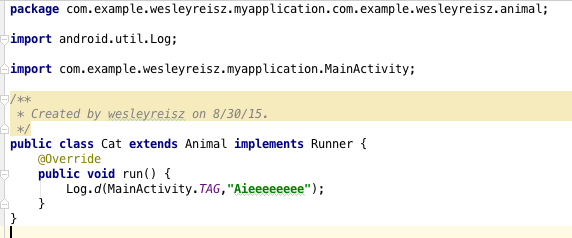
What if we start adding to it… we add snake.



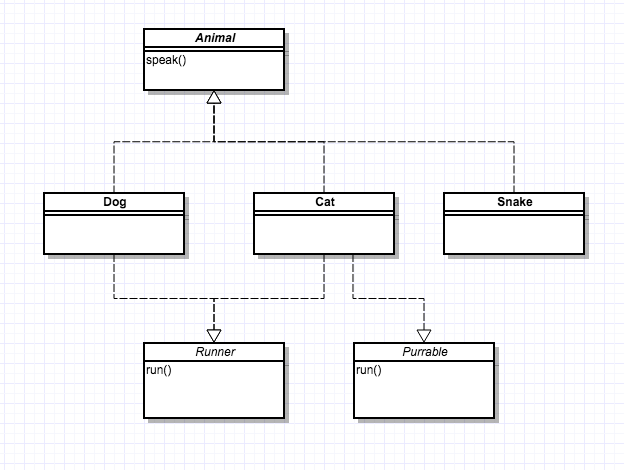


What if we need multiple “parents”? What happens if we need something common to dog/cat but not snake?

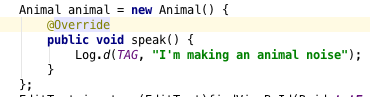




Interfaces can be used to abstract behaviour. What if we make animal an interface?

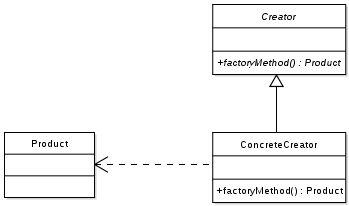


Show Anonymous inner classes



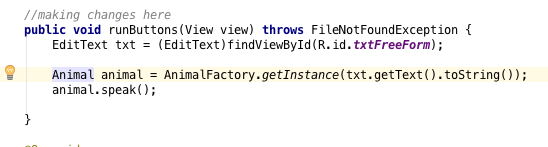
**Design pattern:** In software engineering, a **design pattern** is a general reusable solution to a commonly occurring problem within a given context in software **design**. A **design pattern** is not a finished **design** that can be transformed directly into source or machine code.

**Simple Factory Design Pattern:**



Designed to create objects.





This is not hte greatest, but it’s an example of abstraction and polymorphic behavior.