Java Day 5

4 Pillars of Object-Oriented Programming

Abstraction

The process of hiding implementation and processes of an entity to reduce complexity or increase understanding of a system's properties.

* We do not need need to know *how* something works, only how to use it.
* In Java, we achieve this via Abstract Classes and Interfaces Why?
* **Abstract Classes** are structures that contain state and behaviors.
  + If a behavior is reliant on what something *is* -> abstract class.
  + Use an abstract class when we want to have a common "root" class, but don't want intstantiate it.
  + Adding more hierarchical structure to our code
* **Interfaces** better define behaviors only.
  + If the behavior can be described separately from the state -> interface.
  + Use an interface when we want to define behavior only.
  + Add functionality without affecting our parent class.
  + Java does not support multiple inheritance.
  + Interfaces use the keyword implements
    - and with interfaces we can implement as many as we would like.

Polymorphism

The ability for objects, classes, variables, and/or methods to alter functionality while maintaining structure.

* Poly -> many , morph -> form
* The ability for Java to take advantage of the difference between a *reference variable* and an *object in memory* (when the two are related by inheritance)
* An object's type determines the behaviors it has
* The reference variable type determins which behaviors can be accessed

**Method Overloading and Method Overriding**

* Method Overriding: changing the implementation of an inherited behavior
  + @Override
* Method Overloading: multiple implementations of a single behaviors, by changing the number or type of parameters

**Covariance (Covariant Typing)**

* Referencing an instance of a superclass using its subclass (or vice versa).
* Car c = new Subaru();

Inheritance

The ability for entities to adopt variables (fields) and/or methods (behaviors) from a parent (super) class.  
This allows for instantiating child objects from said parent class  
IS-A relationship.  
We use the extends keyword

* Essentially, Inheritance copies *visible* variables and methods from a parent class into a child class.
* It promotes code reusability, reduces duplication and redundancy.
* Enables polymorphism and code flexibility (more on this later)
* Structures classes into an understandable hierarchy.

All classes in Java implicitly inherit from the Object Class. This means that all objects inherit the Object Class' attributes and behaviors - which we can override to provide a unique implementation.

Encapsulation

The act of wrapping code into a single unit and then selectively exposing and restricting access to that code based on functionality or use within classes.

* Classes should allow minimum necessary access to their members.
* Access to class variables should be done through methods that can perform validation or are designed specifically to perform mutation or accessing functionality.
* These methods are commonly referred to as "Getters" and "Setters" (*Accessors and Mutators*)
* Least amount of access is standard practice
  + We want classes to be the only ones responsible for themselves.

Access Modifiers: How we achieve encapsulation in Java

* There are 4 access modifiers:
* **public**: accessible to all classes, everywhere
* **protected**: accessible to all classes in the same package **AND** all subclasses/children
* **default**: (package-private) accessible to all classes in the same package
* **private**: accessible only within the current class.
* default -> denoted by the absence of an Access Modifier

ACCESS MODIFIERS HAVE NOTHING TO DO WITH SCOPE

they are unrelated, but something that can easily be confused.

Access Modifiers => Accessibility Variable Scopes => Visibility

Encapsulation is not Protection/Security

* there are ways to change access modifiers at runtime (Java Reflections API)