Encapsulation and APIs Summary

By making only certain parts of a class public and keeping the remaining parts "hidden" to outside code, we can:

– restrict what can be done with objects of our type
– change the way we do things behind the scenes without fear of breaking other code as long as the behavior of our class via public methods and variables does not change
A private method can be accessed from a method in a different class.

1. True
2. False

A private method can be accessed from a method in the same class.

1. True
2. False
Libraries

In general, the idea of a code library is a collection of useful functions that can be used by many different programs.

These are things that would need to be "re-invented" (and re-tested) over and over either by multiple coding teams in a company or by many times that around the world.

Useful-Sounding Tools

Some examples of useful-sounding generic functions:

– compute the absolute value of something
– sort a list of numbers in ascending order
– convert all letters in a string to uppercase
Object Oriented Programming

Under the object oriented programming paradigm, things live inside of classes.

When building something for the code library, you really need to build a class which contains the useful thing(s) you want to build.

Organizational Issues

The three examples on the previous slide have no logical connection, so would not belong in the same class philosophically.

There might be several classes worth of useful library methods that you would like to somehow group as a family.
The **String and Integer Classes**

- We've used several library classes that are available to us in Java so far this semester.
- One of those is the **String** class.
- Another is the **Integer** class.

- These are examples of things that are so common and useful that it makes sense to have a version available so that people don't have to write their own.
  
  - **NOTE:** There are times you still might WANT to write your own version. We'll see one soon…

---

**Packages**

In Java, we use packages to group useful classes full of useful methods and variables and constants together.

Consider the line at the top of some of our `.java` files:

```java
import java.util.Scanner;
```

This line says to look in the package `java.util` for a class named `Scanner` and make it available for use by the code in the class we are creating.

If we were going to use several classes from this package, we would likely use the following import instead:

```java
import java.util.*;
```
java.lang

The String and Integer classes are actually part of a package named java.lang which is automatically made accessible when writing code in Java.

If you are curious to see what else is in this package and available for use, you could visit a javadocs page for the package.
http://download.oracle.com/javase/1.7/docs/api/java/lang/package-summary.html

The classes it contains are listed. There are also other things listed that we will begin to explore soon (exceptions) and some things we will talk about much later (interfaces).

Our own packages

We can create our own "local" packages within a project to help organize things.

• In Project 3 (for example) we do this.
• There is a package called p3_student where the PhotoTools class lives.
• At the top of that file there is an indication that it is part of the p3_student package.
  package p3_student;
• There is a package called photo where things like the Photograph and Pixel classes live.
Using our own packages

We can import from our own packages just as we import from places like java.util.

In the PhotoTools Java file, we tell it to:

```java
import photo.Photograph;
import photo.Pixel;
```

This tells Java that there is a package called photo which contains the Photograph and Pixel class we want to access.