**Problem**

Remember our Olympic Medals? Could we store those in an array and process later?

Part of our code that read from a file, put into a partial method:

public processFile(File inputFile,

{

while (inputFile.hasNext())

{

//Declare Variables

int gold = 0;

int silver = 0;

int bronze = 0;

//Read the line from the file, add each value to a sum

String country = inputFile.next();

for(int i=1; i<=4;i++)

{

//Sum of gold, silver and bronze medals

gold += inputFile.nextInt();

silver += inputFile.nextInt();

bronze += inputFile.nextInt();

}

}

**Grouping Data Together in a Class**

Classes are comprised of two parts:

A class that has data stored in it (that aren’t constants) is accessed through an

JDK example:

How should we treat our data members?

* keep them **private:**
* Constants are initialized when they are declared
* Non-constants are initialized in the *constructor*
* Declare them at the beginning of the class, before all methods, and outside methods

Let’s start our OlympicCountry class (see handout page) by declaring our data members.

**Field / Instance variables /Data members:** When do I need them, how do I figure it out??

An instance variable should meet the following requirements:

1. Information used for a particular instance of a class
2. Defines the object’s *state*
3. Is a property that every object from that class will need to know
4. Information that needs to be remembered during more than a single calculation or a single task

If they are private, what is their scope?

To set their values when we first create an OlympicCountry object, we need a

Purpose:

Parameters:

Syntax:

**Using Objects in Java**

When you have a class with data members, and methods that interact with those data members, they should NOT be static.

Now you need to use objects to interact with those classes:

1. Make sure all of your java files are in the **same directory (folder)**

In a DIFFERENT class (such as the one with main in it):

1. **Declare** an object reference (aka reference variable) that can refer to an object of that

1. **Construct** an object by calling a constructor by using *new*
2. **Call** a method—so the object “does” something