Java Programming for Beginners

Lab Exercise 6

Answer Key

1) Build a class to manage a simple "Car" object. Which components would a car be expected to have? Which of these components can be primitives? Which should be objects themselves?

2) Now, let's assume that we also have a "Person" class whose instances can interact (drive) "Car" objects. Based on this model, assign protection levels to the class components of the "Car". Which components must logically be **public** members, which can be **private**? (For example, if your "Car" class contains a SteeringWheel object, this object should probably be public, or have a **public** "get" method, so that the driver can interact with it.)

This is an interesting question when we begin to consider sensitive information like VIN and license plate numbers. Maybe only some viewers should be given access to certain information!

3) In Java, objects and primitives operate under different sets of rules. Declare separate variables of type **int** and **Integer** (a Java class that represents an int as an object) without initializing them, like this:

```
int intVar ;

Integer integerVar;
```

If we try to use these variables as values, Java will stop us because no values have been assigned to them yet. However, if we change the declaration of **integerVar** to read:

```
Integer integerVar = null;
```

Java will allow us to reference this value. But, our code will still break if we attempt to use it as a value. What does it mean to assign the value "null"? Why can we not assign the value "null" to a primitive?

In programming, the "null" pointer is an abstract concept referring to a location of the computer's memory that does not exist. In reality, this is often an actual memory location whose "job" is to be the null location. Whenever a program's control flow attempts to read from this location, an error will be thrown. But, in Java, programs can safely check if a pointer is pointing to this location (that's looking – but not reading).

Because primitives have fixed size memory locations there is no reason to ever point them to the null location. Rather, a memory location for the primitive is chosen and Java protects us from viewing this location until we've assigned a specific value to it. In other programming languages we would be able to utilize the "random" value found in the location before we've overwritten it (a common source of software bugs!)