6. Object Oriented Concepts and Techniques

Lesson 2:
Encapsulation – Part 2

6.7. Encapsulation
6.7.3. static Modifier

- The static modifier specifies that a variable or method is the same for all objects of a particular class.
- When a variable is declared as being static, it is only allocated once regardless of how many objects are instantiated (Typically new variables are allocated for each instance of a class).
- All instantiated objects share the same instances of the static variable.
- When you declare a method static, it is not instantiated with each object, but is part of the entire class. Therefore you invoke the method by preceding it with the class name.

```
static int refCount;
static int getRefCount()
{
    return refCount
}
```

6.7. Encapsulation
6.7.4. abstract Modifier

- There are situations where you need to define a class which declares the structure of a given abstraction without providing a complete implementation of every method.

- You can declare that certain methods are required to be overridden by subclasses using the abstract type modifier.

- Any class which contains any methods declared abstract must also be declared abstract.

6.7. Encapsulation
6.7.4. abstract Modifier Cont…

- To declare a class abstract, you simply use the abstract keyword in front of the class keyword at the beginning of
- the class declaration.

- Such classes cannot be directly instantiated with the new operator since their complete implementation is undefined. You cannot declare abstract constructors, or abstract static methods.

- Any subclass of an abstract class must either implement all of the abstract methods in the super class, or be
- itself declared abstract.

```
abstract class Enemy
{
        abstract void move();
        abstract void move(int x, int y);
}
```

6.7. Encapsulation
6.7.5. final modifier

- All methods and instance variables may be overridden by default.

- If you wish to declare that you want to no longer allow subclasses to override your variables or methods, you can declare them final.

```
final int numDollars = 25;
```

6.7. Encapsulation
6.7.6. synchronised modifier

- The synchronised modifier is used to specify that a method is thread safe.
- Only one path of execution is allowed into a synchronised method at a time.
- In a multithreaded environment like Java, it is possible to have many different paths of execution running through the same code
- The synchronized modifier changes this rule by only allowing a single thread access to a method at once. This will force the others to wait their turn. Threads and Multithreading covered in detail later

6.7. Encapsulation
6.7.7. volatile modifier

- The volatile modifier explicitly declares a field as being potentially changed by multiple threads asynchronously.

- May be modified by asynchronous threads

6.7. Encapsulation
6.7.8.native modifier

- Identify methods that have native implementations. The native modifier informs the Java compiler that the methods implementation is in an external C file.

- native method declarations look different from other Java methods. They have no body.

```
native int calctotal();
```

- Subclasses of any class containing your native methods can still override them.