COP3502 Module 06 Review

object – an instance of a **class**

Ex] Spongebob is an instance of a Sea Creature so Spongebob is an object

So, think back for a second to our Module 04A review… remember objects?
Well, classes and objects go hand in hand.
Class – a **design for an object** *contains **attributes (variables)** that an object might have, and **methods** that an object might use

A Sea Creature class for example, might look something like this:

```
public class SeaCreature
{
        private String name = "Fishy";
        private String species = "fish";
        private int numOfFriends = 0;

        public void swim(){…}
        public void goJellyFishing(){…}
}
```

This class gives a **blueprint (a design) for** a Sea Creature object. It says that all Sea Creatures will have a **name**, a **species**, and some **number of friends**. Additionally, it says that all Sea Creatures can **swim**, and can **go jelly Fishing**.

So how could we **initialize** a SeaCreature object? → A constructor!
Constructors are special kinds of methods that give us a way to build an object how we want.

A constructor for a SeaCreature object might look like any of the 4 examples below:

SeaCreature.java

```
1] public SeaCreature(){}

2] public SeaCreature(String _name)
   {
        name = _name;
   }

3] public SeaCreature(String _name, String _species)
   {
        name = _name;
        species = _species
   }

4] public SeaCreature(String _name, String _species, int _numOfFriends)
   {
        name = _name;
        species = _species;
        numOfFriends = _numOfFriends;
   }
```

The example on the left shows an overloaded constructor – 4 versions of a constructor that build a SeaCreature object in 4 different ways.
And remember, a constructor is just a special type of method that we use to build objects of a class… So, if we can overload constructors, that means we can overload other methods to!

Any time you have two or more versions of a method that have the SAME NAME, but DIFFERENT PARAMETERS, you can say that method is overloaded

```
public void increaseFriends()
{
    numOfFriends++;
}
```

```
public void increaseFriends(int num)
{
    numOfFriends+=num;
}
```

To initialize a specific instance of a SeaCreature object (like a Spongebob object) we could use any of the above constructors depending on our desired functionality

MyProgram.java

```
public static void main(String[] args)
{
    // Using constructor 1
    SeaCreature Spongebob = new SeaCreature ();

    // Using constructor 2
    SeaCreature Spongebob = new SeaCreature("Spongebob");

    // Using constructor 3
    SeaCreature Spongebob = new SeaCreature("Spongebob", "sponge");

    // Using constructor 4
    SeaCreature Spongebob = new SeaCreature("Spongebob", "sponge", 4);
}
```

```
/* creates a SeaCreature object with
   name = default name = "Fishy"
   species = default species = "fish"
   numOfFriends = default numOfFriends = 0 */
```

```
/* creates a SeaCreature object with
   name = "Spongebob"
   species = "sponge"
   numOfFriends = default numOfFriends = 0 */
```

COP3502 Module 06 Review

In addition to constructors, we have some other "special kinds" of methods that can belong to a class. Accessors("getters") and Mutators("setters") are special types of methods that we use in classes to help us access and change information about a particular object.

But before we discuss this, you need to know about access modifiers. Access modifiers tell us WHO has access to something.

public | **private** | **protected**

If something (like a variable) is **private** that means **it can ONLY be accessed inside of the class in which it resides**. So for example, if you have a private variable "species" inside of a SeaCreature class and you try to access that variable in your main method (outside of the SeaCreature class) by doing something like print (Spongebob.species), you will get an error.

In contrast, if something (like a method) is **public** that means **it can be used by "anyone" anywhere**. main() can use the method, the class can use the method, another class could use the method, etc.

More on **protected** later…

These functions (the getter and setter) will go inside of your SeaCreature class (in the SeaCreature.java file). Then in main, you can call these functions to access and change private variables like numOfFriends

```
// Getter == Accessor function:

public int getNumFriends()
{
    return numOfFriends;
}
```

```
// Setter == Mutator function

public void setNumFriends(int num)
{
    numOfFriends = num;
}
```

Good practice when making classes is to keep your variables (attributes) **private** and your methods **public**. But if variables are all **private** for a class, how can we access/change them for a particular object if we needed to?

For example, let's say we made "numOfFriends" a **private** variable of the SeaCreature class (in SeaCreature.java). Then let's say we initialize a Spongebob object in main() (in MyProgram.java). If Spongebob made a new friend, how could we increase his numOfFriends attribute if we cannot even access it (because it is a private variable existing in the SeaCreature class)? ANSWER: use getter (accessor) and setter (mutator functions) **Ex]**

Some quick notes:

- The "this" keyword in java is a variable that references the current object; So for instance in your SeaCreature constructor(s) if you wanted to have **this.name** = **_name** or **this.species** = **_species**, you can do that and it will set the **invoking object's name** and **species** attributes equal to the **_name** and **_species** variables being passed in. * THE INVOKING OBJECT IS THE OBJECT CALLING THE CONSTRUCTOR (in main()). IT IS THE OBJECT WE ARE TRYING TO CONSTRUCT/BUILD*
- Packages – fancy name for a group of classes

Passing Objects as Arguments:

So there's this **crazy** thing that happens when you create a **new** object using the **new** keyword… SeaCreature myObject = **new** SeaCreature(); The object that you create actually gets put in a part of your computer's memory called the **HEAP**. A *pointer to that object* (on the heap) is what is saved in the variable myObject. This pointer (myObject) is put in a part of your computer's memory called the **STACK**.

SeaCreature myObject = new SeaCreature("Squidward", "octopus");

The important take-away here is that **we cannot treat Objects like we do primitive data types** (int, float, boolean, char, etc.) We pass objects to methods differently than we pass primitive types, and we compare objects to other objects differently than we would compare primitive types to primitive types.

I am literally out of room on this review, (classes are a dense concept) but to learn more about this please read slides 12-30 in this ppt: https://ufl.instructure.com/courses/436424/files/61239187?wrap=1