IT and Me Works
Lesson Plan for Programming Strand

Topic: Object-Oriented Programming
Teacher: April Bellafiore
Subject: IT & Me Works
Grade: 9th
Time: 2-4 hours (based on length of course)
Objectives: At the end of this lesson students will:

- Understand general object-oriented programming concepts
  - object, class, instance, method, inheritance

Standards:

- Students will demonstrate an increasing ability to recognize parts of any object or system, and understand how the parts interrelate in the operation of that object or system.

Setting: Computer Lab

Materials: Internet Connection

Teacher's Role:

- Discuss Object-Oriented Programming

Evaluation/Assessment:

Object oriented programming is an extremely difficult concept for most students (and adults) to understand. The goal of this module is to introduce object concepts and to have students consider the types of objects around them (desk, chair, door, etc..) and what attributes and methods each object may have. A good project would be to have the students rework the peanut butter and jelly example from the structured programming module as an object oriented program. If this is not feasible then see if a local programmer can visit the class and explain how the code might be written using object oriented coding processes.

School to Career Connection:

Have an object-oriented programmer visit the class and explain why OO is different from procedural programming.

Sun Microsystems is located just across the border in MA and it may be a good resource of speakers/information/etc...

Employability: Analyzing and organizing information

Object-Oriented Programming Overview

Dogs, cats, fish and spiders all are animals. Animals all have certain characteristics in common – they eat and they have some mechanism of movement – legs, fins, wings, etc... Some animals are also considered pets – so they have a name and play with their owners.

In object-oriented programming we would say that all of these animals are **objects**, and that they share certain **attributes** like number of legs (fins, etc...) and **methods** like eating and playing.

Objects are just that – objects. When you explain what a cat is, do you really need to have a cat in the room? The answer is no – you can describe in general terms what a cat is without it actually being there. If a cat walked into the room while you were talking about it you could say that this particular cat (a tabby with a red collar and whose name is George) is an **instance** of the cat object. You could have other instances of the cat object – each cat that walks into the room is a separate instance. They all share the same characteristics of the cat object – each cat has 4 legs, eats and plays – but every cat is different.

Attributes describe an object. For example, the statement "a dog has 4 legs" means that the dog object has a leg attribute and the value of that attribute is 4.

Attributes are usually defined as being of a particular type. If we say that a "dog has 4 legs" the attribute type is an integer – the number 4. However, if we say that the dog has a name of "Bagel" then the attribute type is a string (meaning a series of characters and letters). Most object-oriented languages require that you define what type of attribute you are working with. It wouldn't make sense to say that "a dog has Bagel legs" or that "my dog's name is 4." So, as a programmer you need to define what types of answers are appropriate, and this where **data typing** comes in. Don't worry too much about this now – just understand that how you define an attribute will influence what kind of information can be processed about that attribute.

Methods are things that objects can do. So, the statement "a dog can play" means that the dog object has a play method. And, the statement "a cat can eat" means that the cat object has an eat method.

Since all animals share certain traits it makes sense to create a parent object that all children objects (dogs, cats, fish, etc...) can get these characteristics from - just like you get your hair and eye color from your parents. Traits are defined once and **inherited** by the children objects. So, the attributes and methods (both are sometimes referred to as a **behavior**) of an animal will automatically be found in a dog, fish, spider, and cat.

Take a look at the code on the next page. This is an example of Java programming – one of the many object-oriented languages that you can code in. The code has some comments (indicated by //) that should help you understand the concepts of how object-oriented programming works. For a comparison, try using structured programming to define each animal – you'll see that in each case you need to define how each one walks, eats and plays – there is no way in structured programming to inherit characteristics from a parent class to a child. The ability to have inherited behaviors is why object-oriented programming is very popular in today's businesses environment – it really cuts down on the time required to program!

Object-Oriented Programming Example – Java Syntax

Code courtesy of Alan Jordan, New Hampshire Community Technical College

```
// The Animal class is the parent class of all other specific animals in this
// example.  Any behavior defined in a parent class will be found in any
// child class.  So, the behavior of an Animal will automatically be found
// in a dog, fish, spider, and cat

public abstract class Animal {
  //number of legs
  protected int legs;

  // default constructor – a constructor is used to create an object
  protected Animal() {
    legs = 0;
  }

  // Another constructor which creates an animal with a given number of legs
  protected Animal(int numberOfLegs) {
    legs = numberOfLegs;
  }

  // a method which must be defined in any child class of Animal
  public abstract void eat();

  // walk method will be inherited by all children of Animal
  public void walk() {
    System.out.println("I am walking on " + legs + " legs.");
  }
}

// Pet interface defines a number of methods that need to be implemented
// when any Animal is a pet

public interface Pet {
  String getName();
  void setName(String name);
  void play();
}

// A spider is a child of the animal class.
// It will have a number of legs and also will eat.

public class Spider extends Animal {

  //Spider constructor calls the animal constructor with the
  //proper number of legs
  public Spider() {
    super(8);
  }

  //implement the eat method from the parent Animal class
  public void eat() {
    System.out.println("Yummy!  I am eating flies!  Delicious!");
  }
}
```

```
// A fish is also a child of the animal class. Fish will eat but
// they will not have any legs (so the code will pass other information)

public class Fish extends Animal{
  //constructor passes no legs because fish don’t have legs
  public Fish() {
  }

  //override the walk method from the parent class because fish don't walk
  public void walk() {
    System.out.println("Fish don't walk you fool!  We swim!  The lazy ones
just float.");
  }

  //implement the eat method found in the parent Animal class
  public void eat() {
    System.out.println("Gulp Gulp");
  }
}

// The Cat class is an Animal but it is also a Pet

public class Cat extends Animal implements Pet {
  private String name;

//The Cat constructor accepts a name and passes the correct number of legs
  public Cat(String name) {
    super(4);
    this.name = name;
  }

  //This Cat constructor is for orphaned cats (who have no name)
  public Cat() {
    super(4);
    name = "";
  }

  // Implement setName method found in the Pet interface
  // When a cat is adopted, its owners can run this method to give it a name
  public void setName(String name) {
    this.name = name;
  }

  // Implement getName method defined in the Pet interface
  public String getName() {
    return name;
  }

  // implement the eat method found in the Animal class
  public void eat() {
    System.out.println("Get me some fresh Salmon!");
  }

  // Implement the play method found in the Pet interface
  public void play() {
    System.out.println("I am chasing a ball of yarn.  Ho hum.");
  }
}
```

```
// The Dog class is very similar to the Cat class, except that some of its
// behavior is a little different

public class Dog extends Animal implements Pet {
  private String name;

  public Dog(String name) {
    super(4);
    this.name = name;
  }

  public Dog() {
    super(4);
    name = "";
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }

  public void eat() {
    System.out.println("I am sitting under the table, throw me some
scraps!");
  }

  public void play() {
    System.out.println("drool... BARK BARK BARK!!!  pant pant...");
  }
}
```