CECS 277 – Lecture 3 – Polymorphism

Polymorphism –

Polymorphism is the ability to take on multiple forms, in programming, it is the idea that a variable of the data type of the superclass can reference objects that are of the subclass. Here is an example where Animal is the superclass, and Dog is a subclass of Animal.

```
Animal a = new Dog( x, y, w, h );
```

A Dog object is constructed and the address to the location of that object is stored in the reference variable a of type Animal. This did not cause any compiler errors since Dog is-an Animal.

Unfortunately though, by creating your subclass object using a superclass variable, you no longer have access to the methods that are specific to your subclass. Though you do have access to any methods that were overridden from the superclass itself.

There are two remedies for this problem:

> **Downcasting** – Temporarily typecasting the object back as a variable of the subclass. That way you have access to the methods that are only available to the subclass. (This method is usually frowned upon for breaking encapsulation).
>
> ```
> Dog d = (Dog) a;
> d.bark();
> ```
>
> **Dynamic Binding** – Create the needed methods in the superclass, and then override those methods in any subclasses that require them. The method call will always use the object's version of the method. Dynamic Binding should be used whenever possible (when it makes sense), otherwise use downcasting.
>
> ```
> a.speak();
> ```

When using downcasting, there may be times when you may not know what type of object is stored in a variable.

```
Animal [] animals = new Animal [10];
```
- or -
```
public void printName(Animal a){ ...
```

In these cases we can ask the object itself what class it was made from.

```
if( animals[0] instanceof Dog ){ ...
```
- or -
```
if( Dog.class.isInstance(a)){ ...
```

Which possibly means that you may need to check each different subclass type to find out which one it is. Be careful not to ask it if it is of the type of the superclass, because if the subclass was inherited from the superclass, then it will return true.

Abstract –

Abstract Methods – An abstract method is defined, but not created (ie. does not have a body). All subclasses created from its class must override this method. If a class has an abstract method in it, then objects cannot be made from it.

```
public abstract void printName(String n);
```

Abstract Classes – An abstract class can be fully defined and then extended, but objects cannot be created from it. Classes that are created as abstract are usually used as a generic superclass from which subclasses are made.

```
public abstract class Animal{
     ...
}
```

Interfaces – An interface specifies a behavior for a class. They are similar to an abstract class that contains all abstract methods. By default, all methods created in an interface are automatically public and abstract. Variables can be created in an interface but they are treated as final and static.

```
public interface Trainable{
     void speak( );
}
```

Using an interface is similar to extending a class, except that it uses the keyword implements. Only one class may be extended, but several interfaces can be implemented, this allows us to do multiple-inheritance. Note: It is also possible to create polymorphic objects using interfaces by creating a reference variable that is of the interface's type.

```
public class Dog extends Animal
                         implements Trainable{
     public void speak( ){
          ...
     }
}
```

Final – We have seen the keyword final used to create constant values, but final can also be applied to classes and methods.

Final Methods – A final method cannot be overridden.

```
public final void printName( );
```

Final Classes – A final class cannot be inherited. If a class is declared as final, all of its methods are implicitly final as well.

```
public final class Dog{
     ...
}
```

```java
/**
 *  Animal is an abstract representation of an animal
 *  @author Shannon Foss
 *  @version Date: Jan 29, 2012
 */
public abstract class Animal {
     /** Name of the animal. */
     private String name;
     /** Type of the animal. */
     private String type;
     /** Color of the animal. */
     private String color;

     /** Initializes the animal using input parameters
      *    @param n  The name.
      *    @param t  The type.
      *    @param c  The color.
      */
     public Animal(String n, String t, String c){
          name = n;
          type = t;
          color = c;
     }
     /** Accessor for the animal's name.
      *  @return name
      */
     public String getName(){
          return name;
     }
     /** Accessor for the animal's type.
      *  @return type
      */
     public String getType(){
          return type;
     }
     /** Accessor for the animal's color.
      *  @return color
      */
     public String getColor(){
          return color;
     }
}
```

```java
/**
 *  Trainable is an interface for trainable behaviors of Animals
 *  @author Shannon Foss
 *  @version Date: Jan 29, 2012
 */
public interface Trainable {
	/** Method to make the animal speak */
	void speak();
	/** Method to make the animal sit */
	void sit();
	/** Method to make the animal fetch */
	void fetch();
}
```

```java
/**
 *  Cat is a representation of a cat, subclass of Animal
 *  @author Shannon Foss
 *  @version Date: Jan 29, 2012
 */
public class Cat extends Animal {
	/** Initializes the cat using input parameters
	 *    @param n  The name.
	 */
	public Cat(String n) {
		super(n, "cat", "gray");
		// TODO Auto-generated constructor stub
	}
	/** Makes the cat sleep -- the only thing the cat does */
	public void sleeps(){
		System.out.println(getName()+" happily sleeps in the
warm sunbeam.");
	}
}
```

```java
/**
 *  Lion is a representation of a lion, subclass of Animal
 *  @author Shannon Foss
 *  @version Date: Jan 29, 2012
 */
public class Lion extends Animal implements Trainable{
	/** Initializes the lion using input parameters
	 *    @param n  The name.
	 */
	public Lion(String n) {
		super(n, "lion", "golden");
		// TODO Auto-generated constructor stub
	}

	@Override
	public void speak() {
		// TODO Auto-generated method stub
		System.out.println("\"Speak "+getName()+", speak.\"");
		System.out.println("The "+getColor()+" "+getType()+"
lets out a great ROAR!");
	}

	@Override
	public void sit() {
		// TODO Auto-generated method stub
		System.out.println("\"Sit "+getName()+", sit.\"");
		System.out.println("The "+getColor()+" "+getType()+"
sits upon the ground.");
	}

	@Override
	public void fetch() {
		// TODO Auto-generated method stub
		System.out.println("You throw a raggedy toy.");
		System.out.println("\"Go get it "+getName()+".\"");
		System.out.println("The "+getColor()+" "+getType()+"
happily trots over to the doll and begins chewing its head
off.");
	}
}
```

```java
/**
 *  Dog is a representation of a dog, subclass of Animal
 *  @author Shannon Foss
 *  @version Date: Jan 29, 2012
 */
public class Dog extends Animal implements Trainable{
	/** Initializes the dog using input parameters
	 *   @param n  The name.
	 */
	public Dog(String n) {
		super(n, "dog", "brown");
		// TODO Auto-generated constructor stub
	}

	@Override
	public void speak() {
		// TODO Auto-generated method stub
		System.out.println("\"Speak "+getName()+", speak.\"");
		System.out.println("The "+getColor()+" "+getType()+"
lets out a loud bark!");
	}

	@Override
	public void sit() {
		// TODO Auto-generated method stub
		System.out.println("\"Sit "+getName()+", sit.\"");
		System.out.println("The "+getColor()+" "+getType()+"
sits upon the ground.");
	}

	@Override
	public void fetch() {
		// TODO Auto-generated method stub
		System.out.println("You throw a stick.");
		System.out.println("\"Go get it "+getName()+".\"");
		System.out.println("The "+getColor()+" "+getType()+"
furiously runs over to the stick, picks it up, runs back to you,
and then growls as you reach for it.");
	}
}
```

```java
/* Author: Shannon Foss
 * Date: January 29, 2012
 * Program: AnimalTraining.java
 * Desc: Program to test the Animal superclass and its
subclasses.
 */
import java.util.Scanner;
public class AnimalTraining {
	static Scanner in = new Scanner (System.in);
	public static void main(String[] args) {
		//create 3 animals
		Animal [] animals = new Animal [3];
		animals [0] = new Lion("Simba");
		animals [1] = new Cat("Fluffy");
		animals [2] = new Dog("Spot");
		int animal = animalMenu();
		//cats don't do tricks
		if(animals[animal-1] instanceof Cat){
			Cat c = (Cat) animals[animal-1];
			c.sleeps();
		}else{
			int trick = trickMenu();
			Trainable t = (Trainable) animals[animal-1];
			switch(trick){
				case 1: t.speak();
					break;
				case 2: t.sit();
					break;
				case 3: t.fetch();
					break;
			}
		}
	}
	//method to display Animal menu
	public static int animalMenu(){
		System.out.println("Animal Menu");
		System.out.println("1. Lion");
		System.out.println("2. Cat");
		System.out.println("3. Dog");
		return getValidInt();
	}
	//method to display trick menu
	public static int trickMenu(){
		System.out.println("Trick Menu");
```

```
            System.out.println("1. Speak");
            System.out.println("2. Sit");
            System.out.println("3. Fetch");
            return getValidInt();
      }
      //method to check user input
      public static int getValidInt(){
            boolean rep = true;
            int value = 0;
            while(rep){
                  if(in.hasNextInt()){
                        value = in.nextInt();
                        rep = false;
                  }else{
                        System.out.println("Invalid input");
                  }
            }
            return value;
      }

}
```