The Most Difficult Part of the Assignment

The most difficult part of the assignment was understanding and installing jUnit. The code was relatively simple to implement, but as a first time jUnit user, creating the all of the jUnit tests for each situation for eat(), move, and whatDidYouEat() was tedious. I create 30 tests to ensure each situation was tested. In addition, installing jUnit and hamcrest was a chore. I had to change the CLASSPATH environment variable several times to ensure the jUnit jar was pointing to the correct directory.

Status

Completed

Lines Of Code

Generic Thing class, has default and nondefault constructor, along with toString

Thing

```
public Thing(String na)
	{
		name=na;
	}
	public String getName()
	{
		return name;
	}
	public String setName(String nam)
	{
		name=nam;
		return name;
	}

	public String toString()
	{
		String className=getClass().getSimpleName();
```

```
                    String firstAndLastName=name + " " + className;
                    if(className.equals("Thing"))
                    {
                              return name;
                    }
                    else
                    {
                              return firstAndLastName;
                    }
          }
}
```

Generic Thing class, has default and nondefault constructor, along with toString, and abstract move, eat, and whatDidYouEat methods.

Creature

```
public abstract class Creature extends Thing
{


   boolean empty=true;
   Thing content;
          public Creature()
          {
                    super();
          }
          public Creature(String na)
          {
                    super(na);
          }
```

```
	public void eat(Thing aThing)
	{
		content=aThing;
		System.out.print(name + " " + "has just eaten" + " " + aThing + "\n");
		empty=false;
	}

	public void whatDidYouEat()
	{

		if(empty)
		{
			System.out.print(name + " " + this.getClass().getSimpleName() + " " +
"has had nothing to eat!\n");
		}
		else
		{
			System.out.print(name + " " + this.getClass().getSimpleName() + " " +
"has eaten" + " " + content + "\n");
		}
	}
}
```

Fly

Example of Fly Class, shows how methods are overridden.

```
public class Fly extends Creature implements Flyer
{
	public Fly()
```

```
        {
    super();
        }

        //Nondefault Constructor (inherits from Creature)
        public Fly(String na)
        {
          super(na);
        }

        public void eat(Thing aThing)
        {
                if(aThing instanceof Creature)
                {
                        System.out.print(name + " " + this.getClass().getSimpleName() + " " +
"won't eat" + " " + aThing + " " +"because it is a Creature" + "\n");
                        empty=true;
                }
                else
                {
                        System.out.print(name + " " + this.getClass().getSimpleName() + " " +
"has just eaten" + " " + aThing + "\n");
                        content=aThing;
                        empty=false;
                }
        }
        public void move()
        {
                fly();
        }
         public void fly()
         {
                System.out.print(name + " " + this.getClass().getSimpleName() + " " + "is buzzing
around in flight\n");
         }
}
```

Acceptance Test

Shows how creatures and things are created, filling and printing each instance in the array, and calling the eat(), whatDidYouEat(), and move ().

```
public class TestCreature extends Object
{

        public static final int THING_COUNT=10;
        public static final int CREATURE_COUNT=6;
        public static void main (String [] args)
        {
                Thing t[]=new Thing[THING_COUNT];
                Creature c[]=new Creature[CREATURE_COUNT];
                Thing thing1=new Thing("thing1");
                Thing thing2=new Thing("thing2");
                Thing thing3=new Thing("thing3");
                Thing thing4=new Thing("thing4");
                Thing thing5=new Thing("thing5");

                Tiger tiger1=new Tiger("tiger1");
                Tiger tiger2=new Tiger("tiger2");
                Tiger tiger3=new Tiger("tiger3");

                Tiger tiger4=new Tiger("tiger4");

                Tiger tiger5=new Tiger("tiger5");


                t[0]=thing1;

                t[1]=thing2;

                t[2]=thing3;

                t[3]=thing4;

                t[4]=thing5;


                t[5]=tiger1;

                t[6]=tiger2;

                t[7]=tiger3;

                t[8]=tiger4;

                t[9]=tiger5;


                //Fill Creature Array

                Ant ant1=new Ant("ant1");

                Ant ant2=new Ant("ant2");

                Bat bat1=new Bat("bat1");

                Bat bat2=new Bat("bat2");

                Fly fly1=new Fly("fly1");

                Fly fly2=new Fly("fly2");



                c[0]=ant1;

                c[1]=ant2;
```

```
c[2]=bat1;
c[3]=bat2;
c[4]=fly1;
c[5]=fly2;

System.out.println("Things:");
System.out.println(" ");

for(int k=0;k<THING_COUNT;k++)
{
        System.out.println(t[k]);
}



System.out.println(" ");



System.out.println("Creatures:");
System.out.println(" ");

                for(int j=0;j<CREATURE_COUNT;j++)
{
        System.out.println(c[j]);
}



System.out.println(" ");
```

```
}








          tiger1.eat(thing1);

          tiger1.move();

          tiger1.whatDidYouEat();

          tiger2.whatDidYouEat();


          ant1.eat(thing2);

          ant1.move();

          ant1.whatDidYouEat();

          ant2.whatDidYouEat();




          fly1.eat(thing3);

          fly2.eat(ant1);

          fly1.move()’

          fly1.whatDidYouEat();

          fly2.whatDidYouEat();


          bat1.eat(thing4);

          bat2.eat(fly2);

          bat1.move();

          bat2.whatDidYouEat();

          bat1.whatDidYouEat();

      }
```

```
}
```

JUnit Tests

Example of Unit Tests with Bat. Tests each situation with Bat eating, moving, and whatDidYouEat

```
@Test

public void batMoveTest()

   {

       Bat mBat=new Bat("mBat");

       mBat.move();

       assertEquals(mBat + " " +"is swooping through the dark\n",println.toString());

   }


@Test

   public void batThingEatTest()

   {

       Bat fullBat1=new Bat("fullBat1");

       fullBat1.eat(thing1);

       assertEquals(fullBat1 + " " +  "won't eat thing1" + " " +"because it is not a Creature\n",
println.toString());

   }


   @Test

   public void batAntEatTest()

   {

       Bat fullBat2=new Bat("fullBat2");

       fullBat2.eat(ant1);

       assertEquals(fullBat2 + " " + "ate ant1 Ant\n", println.toString());

   }
```

```
@Test
  public void batFullWhatDidYouEat()
  {
    //Tests full Bat whatDidYouEat()
    bat1.eat(ant1);
    println.reset();
    bat1.whatDidYouEat();
    assertEquals(bat1 + " " + "has eaten ant1 Ant\n", println.toString());
  }

  @Test
  public void batEmptyWhatDidYouEat()
  {
    Bat emptyBat=new Bat("emptyBat");
    emptyBat.whatDidYouEat();
    assertEquals(emptyBat + " " +  "has had nothing to eat!\n", println.toString());
  }
```

JCoverage

Covers all eat(), whatDidYouEat(), and move() methods. Missing JCoverage comes from Constructors, Accessors, and Mutators

C:\Users\djnig\Desktop\U5 Fall 2017\CS 445\cs445-fall-2017\HW1Project\.jacocoverage\report.html\index.html

Cyclomatic Complexity

E=7 (Creates Thing and Creature arrays, prints them, and calls eat, move, and whatDidYouEat for each Creature)

N=0 (No if statements)

P=1 (One Exit Point at the End of the Program)

M=E-N+2*P

=7-0+2*1

=5