## Polymorphism

### 1. Summary

The term “polymorphism” comes from the Greek words for “many shapes”. When used in programming, the term denotes the ability to treat different objects the same way. For example, cats and dogs are both animals. They do similar things like walk or sleep. If we programmed a class for cats and for dogs, we would have to make separate methods for each class, which seems redundant. By using polymorphism, we can create one method for walking or sleeping and both the cat and dog class can use them.

### 2. How it works

In order to use polymorphism, you will need to implement an interface or create a super class for your other classes to extend. Let’s go back to the cats and dogs. First, we need a super class such as Animal, which contains the methods for sleeping and walking. Next, we need to make a Cat class and a Dog class that extends the Animal class. The format would look like this:

public class Cat extends Animal {} and public class Dog extends Animal{}

The Cat and Dog classes inherit the same methods, variables, and constructors from the Animal class when they extend it. The Cat and Dog class can call the methods, such as walk, and can override them to make them unique to their classes. To call the methods, the notation is as follows:

```
public void walk{
     super.walk()
}

public void sleep{
     super.sleep()
}
```

The super makes the method refer back to the super class to call the method. Even though the Java virtual machine calls the super class for the method, it still is able to use the unique method of the child class. The JVM locates the correct method by first looking at the class of the actual object and then calling the method with the given name in that class. This technique for locating the appropriate method is called dynamic method lookup and it is what enables programmers to use polymorphism.

If Animal were an interface rather than a super class, the format would look like this:

public class Cat implements Animal{} or public class Dog implements Animal{}

When you implement an interface, the classes don’t inherit the methods, variables, and constructors like they did from the superclass. They inherit empty methods, variables, and

constructors and are free to  However, they do not use super to call them, rather they call them with the name of the class that is using them, like Dog.walk() or Cat.sleep().

```
/**
 * Example of polymorphism through inheritance
 *
 * @author James Savage
 *         Created Sep 12, 2010.
 */
public class HelloPoly {

	/**
	 * Runs the program
	 *
	 * @param args Unused
	 */
	public static void main(String[] args) {
		Room aRoom = new Room(20);
		aRoom.leaveRoom();
		aRoom.leaveRoom();
		aRoom.enterRoom();
		HelloPoly.printCount(aRoom);

		Bag aBag = new Bag();
		aBag.setNumberOfItems(12);
		HelloPoly.printCount(aBag);

		/*
		 * Countable c = new Countable();
		 *
		 * Exception in thread "main" java.lang.Error: Unresolved compilation problem:
		 *		Cannot instantiate the type Countable
		 *
		 *		at HelloPoly.main(HelloPoly.java:26)
		 */
	}

	/**
	 * Prints a description of any object implementing Countable
	 *
	 * @param obj An object which implements Countable
	 */
	public static void printCount(Countable obj) {
		System.out.println("There are " + obj.getCount() + " " + obj.getDescription());
	}
}
```

```
/**
 * Implemented by objects which hold a number of things
 *
 * @author James Savage
 *         Created Sep 12, 2010.
 */
public interface Countable {
        /**
         * Returns the number of items a given class has
         *
         * @return int representing the number of items a given class has
         */
        public int getCount();


        /**
         * Returns a description of the objects held by the class
         *
         * @return String description of the objects held by the class
         */
        public String getDescription();
}
```

```
/**
 * A bag can have any number of items, but it can not be individually added to
 * or subtracted from
 *
 * @author James Savage
 *         Created Sep 12, 2010.
 */
public class Bag implements Countable {
	private int numOfItems = 0;

	/**
	 * Set the number of items in the bag
	 *
	 * @param number Number of items in a bag
	 */
	public void setNumberOfItems(int number) {
		this.numOfItems = number;
	}

	@Override
	public int getCount() {
		return this.numOfItems;
	}

	@Override
	public String getDescription() {
		return (this.numOfItems == 1) ? "item" : "items" ;
	}
}
```

```
/**
 * Rooms can have an initial number of people, but their count can only be
 * changed by one at a time
 *
 * @author James Savage
 *         Created Sep 12, 2010.
 */
public class Room implements Countable {
	private int numOfPeople = 0;

	/**
	 * Create a new Room object with an initial number of people
	 *
	 * @param people The initial number of people in the room
	 */
	public Room(int people) {
		this.numOfPeople = people;
	}

	@Override
	public int getCount() {
		return this.numOfPeople;
	}

	@Override
	public String getDescription() {
		return "People";
	}

	/**
	 * Decrease the count of people in the room by one
	 */
	public void leaveRoom() {
		this.numOfPeople--;
	}

	/**
	 * Increase the count of people in the room by one
	 */
	public void enterRoom() {
		this.numOfPeople++;
	}
}
```