Week 10 – Polymorphism

Polymorphism allows you to create a “common interface” that can work with many underlying structures.

In short it allows you to use a parent class as the data type for its children

Polymorphism and Inheritance

The best way to understand polymorphism is to see it in action

We are going to create a parent class representing an animal. This system will assume all animals have a name and an ability to speak.

Polymorphism takes advantage of the similarities that each child has as a result of inheriting from a parent class

Here is the animal class

```
public abstract class Animal
{
	protected String name;

	public Animal(String nameIn)
	{
		name = nameIn;
	}

	public String getName()
	{
		return name;
	}

	public abstract String speak();

}
```

From here we are also going to create child classes:

- Dog
- Cat
- Horse

These can be found on the next page

Author: Darren Cunningham
Version: 1.0

Week 10 – Polymorphism

Dog.java

```
public class Dog extends Animal
{
	public Dog(String nameIn)
	{
		super(nameIn);
	}

	public String speak()
	{
		return "Bark! Bark! Bark! Grrrrr!";
	}

}
```

Cat.java

```
public class Cat extends Animal
{
	public Cat(String nameIn)
	{
		super(nameIn);
	}

	public String speak()
	{
		return "Meow!";
	}

}
```

Horse.java

```
public class Horse extends Animal
{
	public Horse(String nameIn)
	{
		super(nameIn);
	}

	public String speak()
	{
		return "Neigh!";
	}

}
```

Author: Darren Cunningham
Version: 1.0

Week 10 – Polymorphism

Polymorphism

So now we have an abstract parent class and 3 child classes, but where does polymorphism fit in to this?

I will use an array to demonstrate this, although it can be used with other things such as variables.

Let's say I created a load of animals:

```
public static void main(String[] args)
{
	Dog d1 = new Dog("Fido");
	Dog d2 = new Dog("Lucky");

	Cat c1 = new Cat("Wiskers");
	Cat c2 = new Cat("Socks");

	Horse h1 = new Horse("Trigger");
	Horse h2 = new Horse("Daisy");
}
```

Now I want to store all of them in an array… But how can I do this?
If I make the array type Dog then it can only store dogs and will break if I try to put a cat or a horse in there.
Making it of type Cat or Horse will have a similar effect – it will only allow the array to store one type of animal

This is where polymorphism comes in:

Instead of worrying about the differences between these classes we can instead look at what they have in common:

They are all animals (they inherit from the animal class)

Thanks to polymorphism, we can make the datatype of the array "Animal" and it will then allow us to store all types of animal.

Author: Darren Cunningham
Version: 1.0

Week 10 – Polymorphism

Coded Example

```
public static void main(String[] args)
{
	Dog d1 = new Dog("Fido");
	Dog d2 = new Dog("Lucky");

	Cat c1 = new Cat("Wiskers");
	Cat c2 = new Cat("Socks");

	Horse h1 = new Horse("Trigger");
	Horse h2 = new Horse("Daisy");

	// Create an array
	Animal[] animalList = new Animal[6];

	// Adding the dog, cat and horse objects
	animalList[0] = d1;
	animalList[1] = c1;
	animalList[2] = h1;
	animalList[3] = d2;
	animalList[4] = c2;
	animalList[5] = h2;
}
```

In addition to this, because the animal class has set up some common features amongst all its children, we can make use of them here. Even better is that polymorphism will call the appropriate method depending on what object it is dealing with.
i.e. if it is a dog object then the dog classes implementation of the methods will be called:

```
for (int i = 0; i < animalList.length; i++ )
{
	System.out.println(animalList[i].getName() + " says " +
				animalList[i].speak());
}
```

The output for this would be:

```
Fido says Bark! Bark! Bark! Grrrrr!
Wiskers says Meow!
Trigger says Neigh!
Lucky says Bark! Bark! Bark! Grrrrr!
Socks says Meow!
Daisy says Neigh!
```

Author: Darren Cunningham
Version: 1.0