Polymorphism is a programming language feature that allows values of different data types to be handled using a uniform interface.

Let we have superclass Animal and these subclasses: Dog, Cat, Wolf, Hippo and Lion.

First of all, with polymorphism, the reference and the object can be different: Animal myCat= new Cat();. The reference variable type is declared as Animal, but the object is created as new Cat().

With polymorphism, the reference type can be a superclass of the actual object type. This lets to do things like make <u>polymorphic arrays</u>.

Animal[] animals = new Animal[5];	Declare an array of type Animal. In other words, an array that will hold objects of type Animal.

Animals[0] = new Dog();
Animals[1] = new Cat();
Animals[2] = new Wolf();
Animals[3] = new Hippo();
Animals[4] = new Lion();
	We can put <u>any</u> subclass of Animal in *the Animal array*.

for (int i=0; i<animals.length; i++) {
animals[i].eat();
animals[i].roam(); }
	We get to loop through the array and call one of the Animal-class methods, and every object does the right thing.

We can have polymorphic <u>arguments</u> and <u>return</u> types.

Class Vet {
public void giveShot (Animal a) {
a.makeNoise(); }}
	The Animal parameter can take <u>any</u> Animal type as the argument. And when the Vet is done giving the shot, it tells the Animal to makeNoise(), and whatever Animal is really out there on the heap, that's whose makeNoise() method will run.

class PetOwner {
public void start() {
Vet v = new Vet();
Dog d = new Dog();
Hippo h = new Hippo();
	The Vet's giveShot() method can take any Animal we give it.

v.giveShot(d);	Dog's makeNoise() runs.
v.giveShot(h); }}	Hippo's makeNoise() runs.

So **with polymorphism, we can write code that doesn't have to change when we introduce new subclass types into the program.**