No one has really explained in plain terms how interfaces are useful, so I'm going to give it a shot (and steal an idea from Shamim's answer a bit).

Lets take the idea of a pizza ordering service. You can have multiple types of pizzas and a common action for each pizza is preparing the order in the system. Each pizza **has to be prepared** but each pizza **is prepared differently**. For example, when a stuffed crust pizza is ordered the system probably has to verify certain ingredients are available at the restaurant and set those aside that aren't needed for deep dish pizzas.

When writing this in code, technically you could just do

public class Pizza()

{

public void Prepare(PizzaType tp)

{

switch (tp)

{

case PizzaType.StuffedCrust:

// prepare stuffed crust ingredients in system

break;

case PizzaType.DeepDish:

// prepare deep dish ingredients in system

break;

//.... etc.

}

}

}

However, deep dish pizzas (in C# terms) may require different properties to be set in the Prepare()method than stuffed crust, and thus you end up with a lot of optional properties, and the class doesn't scale well (what if you add new pizza types).

The proper way to solve this is to use interface. The interface declares that all Pizzas can be prepared, but each pizza can be prepared differently. So if you have the following interfaces:

public interface IPizza

{

void Prepare();

}

public class StuffedCrustPizza : IPizza

{

public void Prepare()

{

// Set settings in system for stuffed crust preparations

}

}

public class DeepDishPizza : IPizza

{

public void Prepare()

{

// Set settings in system for deep dish preparations

}

}

Now your order handling code does not need to know exactly what types of pizzas were ordered in order to handle the ingredients. It just has:

public PreparePizzas(IList<IPizza> pizzas)

{

foreach (IPizza pizza in pizzas)

pizza.Prepare();

}

Even though each type of pizza is prepared differently, this part of the code doesn't have to care what type of pizza we are dealing with, it just knows that it's being called for pizzas and therefore each call to Prepare will automatically prepare each pizza correctly based on its type, even if the collection has multiple types of pizzas.