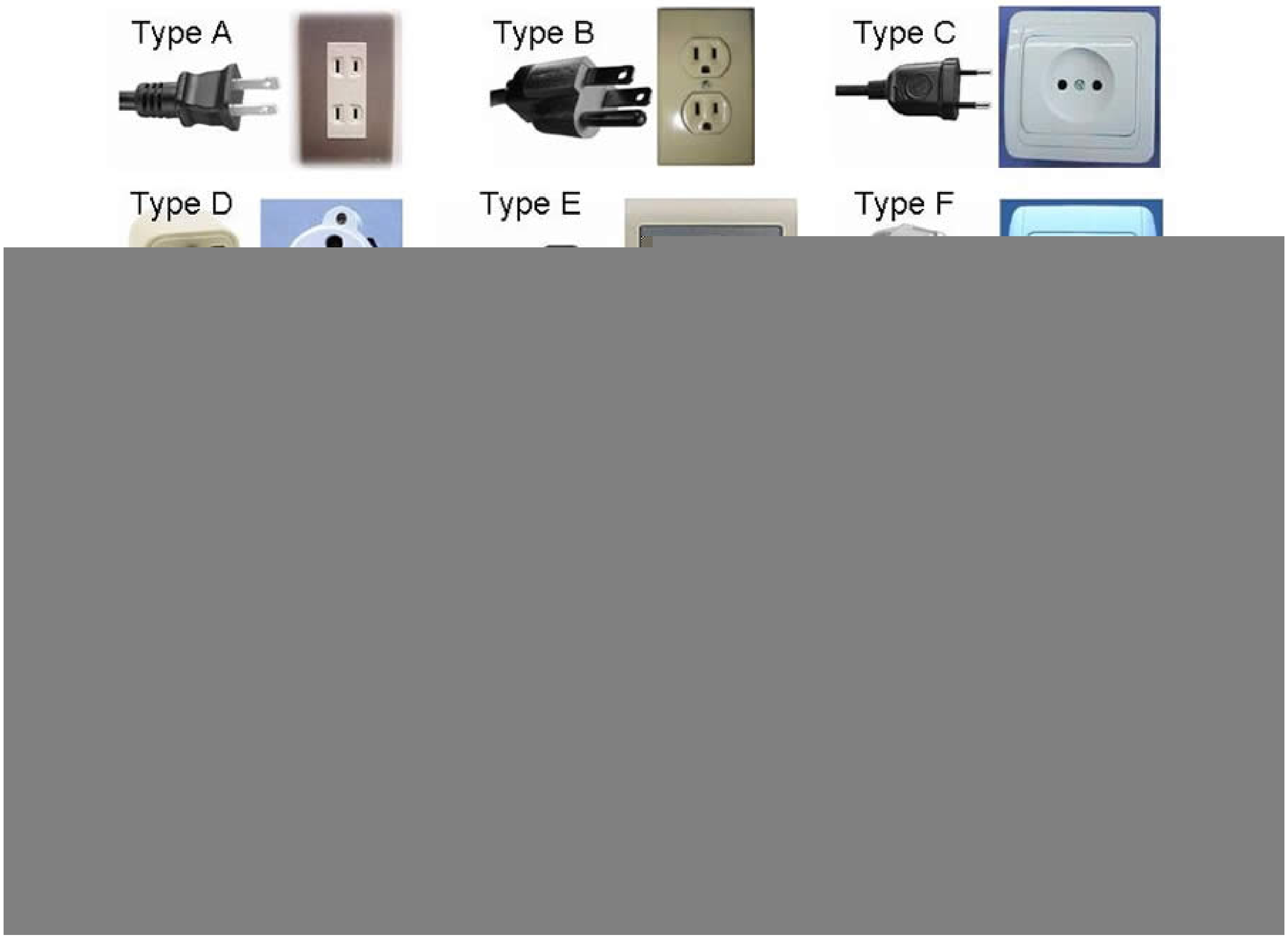
[0800] Working with Interfaces

At a theoretical level, Interfaces are specifications for how things can be joined or connected. This is true regardless of if you are talking about software interfaces or physical interfaces.

Physical Interfaces



All of the plugs above represent physical interfaces. There are well-documented standards for what "flavor" of electricity each hole in each socket supplies. As long as appliance manufacturers follow those standards, then different appliances can be plugged into the same type of plug.

Software Interfaces

In the software world, the word "interface" can describe a whole spectrum of concepts about how programs can communicate with other programs (or other parts of themselves). "Interfaces" can be high-level, open, general communication approaches (e.g., APIs) or they can be very, very specific, private things (e.g. a private method call). For the rest of this chapter, we will be talking about C#'s "interface" keyword and the things you can do with it.

Syntactically, C# interfaces are very similar to "pure abstract classes." They are "classes" that only contain method signatures without an implementation code. As such, they are really "API specifications" that can be enforced by the compiler and that don't require an inheritance relationship to work. That means they are very flexible.

Above I said that an Interface is a "class" but that's really not correct. It is mainly a "type" and can be used as such. Maybe even more accurately, you can think of an Interface as a "category" of classes that share a certain characteristic.

Interfaces can be used in a wide variety of ways. One of the most common ways interfaces are used is also one of the easiest ways to think about interfaces - as an "adverb" that can optionally be added to a class. Interfaces designed for this purpose usually have adverbs for names, especially adverbs ending in "-able." Real world examples include "Serializable", "Iterable", "Disposable", "Cloneable", etc.

(C# has a naming convention for interfaces that puts a capital-I at the front of all interface names. (I hate it but whatever.))

So, for example, if your class implements the "ICloneable" interface it means that your class supports a standard, well-defined way to create a clone of itself - specifically by implementing the Clone() method. Or, if your class implements the "ISerializable" interface, it means that your class has a way to save its internal state via serialization - specifically by adding the Serialize() method. Note that the relationship between the interface name (an adverb) and the method required by that interface (the verb version of the interface name) is common.

When you add an interface to a class, it is said that the class "implements" the interface. You add an interface to a class using the same colon syntax that inheritance uses. Just like with abstract classes, if your class implements an interface, it must contain concrete methods for every method in that interface.

Finally, it is important to remember that, unlike with inheritance, your class can implement more than one interface.