Interfaces

An interface is a **contract**: the guy writing the interface says, "*hey, I accept things looking that way*", and the guy using the interface says "*Ok, the class I write looks that way*".

**An interface is an empty shell**, there are only the signatures of the methods, which implies that the methods do not have a body. The interface can't do anything. It's just a pattern.

E.G (pseudo code):

// I say all motor vehicles should look like this:

interface MotorVehicle

{

void run();

int getFuel();

}

// my team mate complies and writes vehicle looking that way

class Car implements MotorVehicle

{

int fuel;

void run()

{

print("Wrroooooooom");

}

int getFuel()

{

return this.fuel;

}

}

Implementing an interface consumes very little CPU, because it's not a class, just a bunch of names, and therefore there is no expensive look-up to do. It's great when it matters such as in embedded devices.

Abstract classes

Abstract classes, unlike interfaces, are classes. They are more expensive to use because there is a look-up to do when you inherit from them.

***Abstract class are incomplete class because their instance cannot be created and it must be inherited by some child class.***

***Abstract method are incomplete method they are only declared in abstract class and get defined in child class.***

Abstract classes look a lot like interfaces, but they have something more : you can define a behavior for them. It's more about a guy saying, "these classes should look like that, and they have that in common, so fill in the blanks!".

e.g:

// I say all motor vehicles should look like this :

abstract class MotorVehicle

{

int fuel;

// they ALL have fuel, so why not let others implement this?

// let's make it for everybody

int getFuel()

{

return this.fuel;

}

// that can be very different, force them to provide their

// implementation

abstract void run();

}

// my team mate complies and writes vehicle looking that way

class Car extends MotorVehicle

{

void run()

{

print("Wrroooooooom");

}

}

Implementation

While abstract classes and interfaces are supposed to be different concepts, the implementations make that statement sometimes untrue. Sometimes, they are not even what you think they are.

In Java, this rule is strongly enforced, while in PHP, interfaces are abstract classes with no method declared.

In Python, abstract classes are more a programming trick you can get from the ABC module and is actually using metaclasses, and therefore classes. And interfaces are more related to duck typing in this language and it's a mix between conventions and special methods that call descriptors (the \_\_method\_\_ methods).

As usual with programming, there is theory, practice, and practice in another language :-)

=====================================================================================

**The key technical differences between an**[**abstract class**](http://us2.php.net/abstract)**and an**[**interface**](http://us2.php.net/interface)**are**:

* Abstract classes can have constants, members, method stubs (methods without a body) and defined methods, whereas interfaces can only have constants and methods stubs.
* Methods and members of an abstract class can be defined with any visibility, whereas all methods of an interface must be defined as public (they are defined public by default).
* When inheriting an abstract class, a concrete child class must define the abstract methods, whereas as an abstract class can extend another abstract class and abstract methods from the parent class don't have to be defined.
* Similarly, an interface extending another interface is not responsible for implementing methods from the parent interface. This is because interfaces cannot define any implementation.
* A child class can only extend a single class (abstract or concrete), whereas an interface can extend or a class can implement multiple other interfaces.
* A child class can define abstract methods with the same or less restrictive visibility, whereas a class implementing an interface must define the methods with the exact same visibility (public).

**What is the benefit of using abstract class?**

**In abstract class we can implement functions as well as only-define them. We have option of doing both.**