**Abstract Classes versus Interfaces**

We have learned how the concepts of class abstraction and object interfaces work nicely as added dimensions of inheritance to deliver common behaviors and standards for derived objects. There are frequent debates about when to use abstract classes and when to use interfaces. Although we have gone through the practical use cases of both via our exercises, the topic still needs discussion.

Abstract classes are meant to deliver common behaviors or actions via methods to extended objects while keeping vital room for common methods to be implemented differently by derived objects. In contrast, interfaces are for setting standard ways of interacting with objects. An abstract class must have at least one abstract method, whereas all the methods in an interface are abstract. Remember, this is not a concept of having one or more abstract methods versus all abstract methods. Both have their own use cases when it comes to inheritance: the abstract class delivers the common functionalities and allows us to implement our own functionalities, whereas the interface is not about sharing functionalities at all; rather, the interface is all about setting standards for certain actions.

Simple abstract classes can have implemented methods and attributes, whereas interfaces cannot as they contain constants and method signatures without bodies. Therefore, it is not possible to share code via interfaces.

In previous exercises, the abstract class provided us with common engine functionalities and allowed us to deal with specific features of the engine in our own way. The interface showed us the standards to drive the car and we followed the guidelines accordingly to achieve our own goal of driving actions.

**Class Type Hinting Plays a Role in Dependency Injection**

Type hinting allows us to define the type of data to be passed as arguments into a function. PHP supports class type hints, which means that, in function arguments, you can mention which class type the passed argument object belongs to. For example, a **User** class might want to use a **Mailer** service to deliver email. The **Mailer** object can be passed to the **User** class and the **User** needs to make sure that nothing except a **Mailer** object is passed to it.

Check out the following example where the function argument was expected to be an instance of a specific class:

function myMethod($object)

{

if(!($obj instanceof ClassName))

{

throw new Exception

('Only Objects of ClassName can be sent to this function.');

}

}

If the object is not an instance of the expected class, then an exception is thrown with the message **'Only Objects of ClassName can be sent to this function.**'.

**Note:** An exception is a class that is throwable and catchable with an error message so that the catch block can catch the exception and work accordingly. *Chapter 8, Error Handling,* discusses exceptions in detail.

The preceding snippet is equivalent to the following class type hinting syntax:

function myMethod(ClassName $object)

{

}

So, with class type hinting, we can enforce the function or method caller to pass the appropriate type of object. When class type hinting is applied, PHP automatically performs **instanceof** checking and produces an error if the object doesn't satisfy the class relationship.

**Dependency injection** is the technique of supplying an object to another object that is dependent on the first object. For example, a **user** object might need to send out emails and perform certain database operations; therefore, the user is dependent on the **mailer** object and the **database** object. We could supply such **mailer** and **database** objects to the **user** object as follows:

<?php

class User

{

public $name;

private $mailer;

private $database;

function \_\_construct(string $name, Mailer $mailer, Database $db)

{

$this->name = $name;

$this->mailer = $mailer;

$this->database = $db;

}

}

class Mailer

{

//class body

}

class Database

{

//class body

}

$mailer = new Mailer();

$database = new Database();

$user = new User('John Doe', $mailer, $database);

Here, while instantiating the **User** class, we have passed the name of the user, a **mailer** object, and a **database** object as arguments. The **Mailer** **$mailer** class type hint ensures that the only instance of the **Mailer** class can be supplied and the other class type hint at **Database $database** ensures that the only instance of the **Database** class can be supplied. We have added these two object dependencies in the user's constructor so that the object gets loaded with certain dependencies and any failure upon injecting dependencies will prevent object creation.

The preceding technique is called **constructor injection**. You can inject dependencies using a setter method or you can use a dependency injection container. You could search for books or online resources to extend your learning about dependency injection even further.