{

public function construct(Session $session, $key) { ... }

}

Right now we can create an instance of the cart manually like this:

$storage = new SessionStorage(Yii::$app->session, 'primary-cart');

$cart = new ShoppingCart($storage)

It allows us to create a lot of different implementations such as SessionStorage, CookieStorage, or DbStorage. And we can reuse the framework-independent ShoppingCart class with StorageInterface in different projects and different frameworks. We must only implement the storage class with the interface’s methods for needed framework.

But instead of manually creating an instance with all dependencies, we can use a dependency injection container.

By default the container parses the constructors of all classes and recursively creates all the required instances. For example, if we have four classes:

class A {

public function construct(B $b, C $c) { ... }

}

class B {

} " ' class C {

public function construct(D $d) { ... }

}

class D {

}

We can retrieve the instance of class A in two ways:

$a = Yii::$container->get('app\services\A')

// or

$a = Yii::createObject('app\services\A')

And the container automatically creates instances of the B, D, C, and A classes and injects them into each other.

In our case we mark the cart instance as a singleton: Yii::$container->setSingleton('app\cart\ShoppingCart');

This means that the container will return a single instance for every repeated call instead of creating the cart again and again.

Besides, our ShoppingCart has the StorageInterface type in its own constructor and the container does know what class it must instantiate for this type. We must manually bind the class to the interface like this: