

Drupal 8 Plugins

Joe @eojthebrave Shindelar

"Modules, which provide plugins that extend Drupal's functionality, extend Drupal's functionality."

Joe Shindelar

@eojthebrave



drupalize 

This presentation starts out simple, and then gets pretty technical. Hold on tight!

<https://github.com/eojthebrave/icecream>

What Are Plugins?

"The D8 plugin system provides a set of guidelines and reusable code components to allow developers to expose pluggable components within their code and (as needed) support managing these components through the user interface."

- drupal.org Handbook

What Are Plugins?

"A discreet class that executes an operation within the context of a given scope, as a means to extend Drupal's functionality."

- @helior

What Are Plugins?

- Limited in scope (e.g. only act on images)
- Do only one thing (e.g. change image color)
- Configurable & reusable
(e.g. change to green, or change to pink)

Some Examples

- Blocks
- Field Types, Field Widgets, Field Formatters
- Actions
- Image effects
- CKEditor buttons

D8 Plugin = D7 Info Hook

Drupal 7

```
function example_block_info()  
function example_block_configure($delta = '')  
function example_block_view($delta = '')  
function example_block_save($delta = '', $edit = array())
```

Drupal 8

```
namespace Drupal\example\Plugin\Block;  
  
class MyBlock extends BlockBase {  
    // Plugin code goes here ...  
}
```


Why Plugins?

- All the code (def. & implementation) in one place
- Easy to re-use across projects
- Extensible, no need to re-write/copy - just extend
- Plugins are lazy loaded
- No more switch statements in `hook_block_view` implementations. 1 Block = 1 Plugin.

Before We Plugin

- You DO NOT need to learn Symfony
- PSR-4
- Annotations
- Dependency Injection
- The Drupal Service Container

PSR-4 Autoloading

VendorNamespace\SubNamespace\ClassName

Directory Structure

modules/peter/src/Plugin/Block/PeterBlock.php

Code

```
namespace Drupal\peter\Plugin\Block;
```

```
class PeterBlock extends BlockBase {}
```

<https://www.drupal.org/node/2156625>

Annotations

Regular comments
are ignored by
opcode caches

```
// This is a regular comment.
```

```
/* This is a normal mult-line  
comment. */
```

docblock style comments are
available for introspection

```
/**  
 * Provides a 'New forum topics' block.  
 *  
 * @Block(  
 *   id = "forum_new_block",  
 *   admin_label = @Translation("New forum topics"),  
 *   category = @Translation("Lists (Views)")  
 * )  
 */  
class NewTopicsBlock extends ForumBlockBase {
```

Dependency Injection

```
class MyClass {  
    public function __construct() {  
        $db = new MySQLDatabase();  
        // Do things with the $db ...  
    }  
}
```

```
$instance = new MyClass();
```

NOT INJECTED

```
class MyClass {  
    public function __construct(DatabaseInterface $db) {  
        // Do things with the $db ...  
    }  
}
```

```
$db = new MySQLDatabase();  
$instance = new MyClass($db);
```

INJECTED!

Service Containers

Automatically instantiate service-oriented classes with all their registered dependencies.

example.services.yml

services:

example.myService:

class: Drupal\example\namespace\ClassName

arguments: [*@db*]



```
$db = new Database();  
$instance = new ClassName($db);
```

```
$instance = \Drupal::service('my_module.my_service');
```

Recap

☒ PSR-4 Autoloading

☒ Annotations

☒ Dependency Injection

☒ Service Container

copy & paste

Block Plugin

modules/chad/src/Plugin/Block/ChadBlock.php

```
/**
 * @file
 * Contains \Drupal\chad\Plugin\Block\ChadBlock.
 */

namespace Drupal\chad\Plugin\Block;

use Drupal\block\BlockBase;

/**
 * Provides a simple hello world block plugin.
 *
 * @Block(
 *   id = "chad_block",
 *   admin_label = @Translation("Chad block"),
 *   category = @Translation("Tacos")
 * )
 */
class ChadBlock extends BlockBase {
  /**
   * {@inheritdoc}
   */
  public function build() {
    return array(
      '#type' => 'markup',
      '#markup' => 'Hello World',
    );
  }
}
```

Pro-Tip

Extend the BaseClass

In most cases when creating a plugin of a given type there is a base class that can be extended.

What Are Plugins Types?

Describe how plugins of this "type" will be located, instantiated, and generally what they'll do.

There's a pretty good list at:

<https://api.drupal.org/api/drupal/core!modules!system!system.api.php/group/annotation/8>

Behind The Scenes

- Plugin Manager
- Plugin Discovery
- Plugin Factory
- Mapper
- PluginManagerInterface

Plugin Discovery

AnnotatedClassDiscovery

```
new AnnotatedClassDiscovery('Plugin/Block', $namespaces, 'Drupal\block\Annotation\Block');
```

HookDiscovery

```
new HookDiscovery($this->moduleHandler, 'block_info');
```

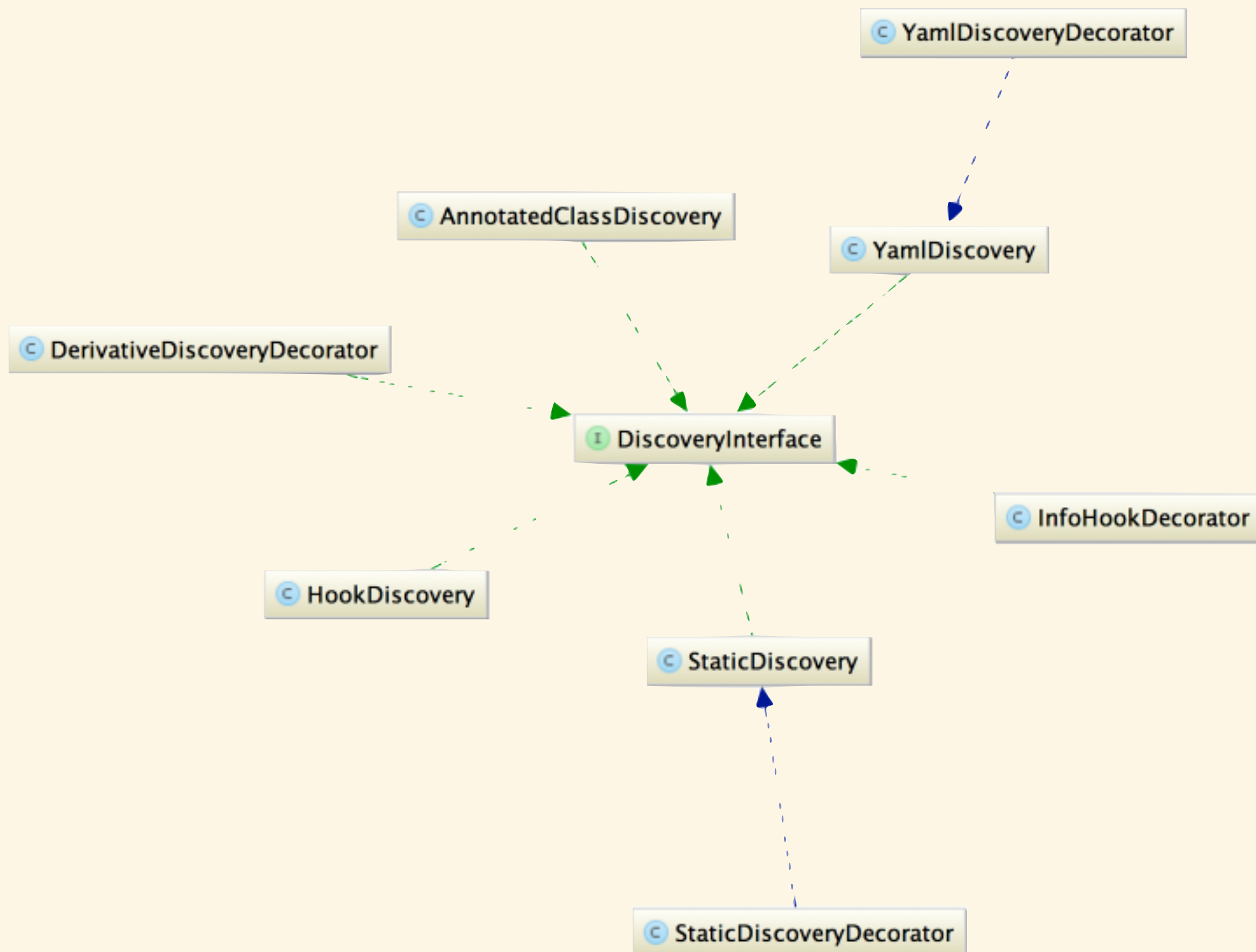
YamlDiscovery

```
new YamlDiscovery('blocks', $module_handler->getModuleDirectories());
```

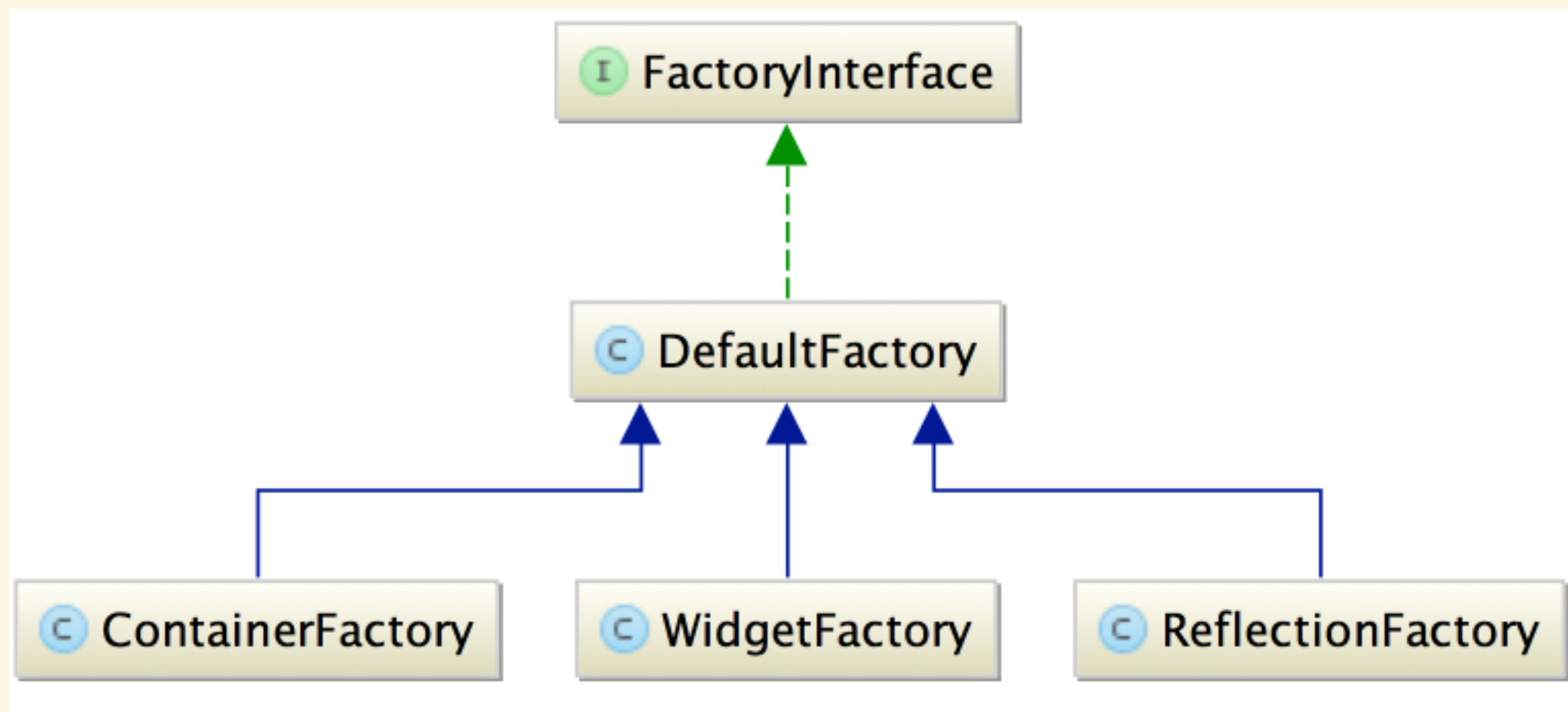
StaticDiscovery

```
new StaticDiscovery();
```

Decorator Classes



Plugin Factories



```
$plugin = $this->createInstance($plugin_id, $configuration);
```

Plugin Factories

what class?

```
$instance = new ClassName($arg1, $arg2, $arg3);
```

what arguments?

Plugin Factories

DefaultFactory

```
new $plugin_class($config, $plugin_id, $plugin_def);
```

ContainerFactory

```
$plugin_class::create(\Drupal::getContainer(), $config, $plugin_id, $plugin_def);
```

WidgetFactory

```
$plugin_class($plugin_id, $plugin_def, $config['field_definition'], $config['settings']);
```

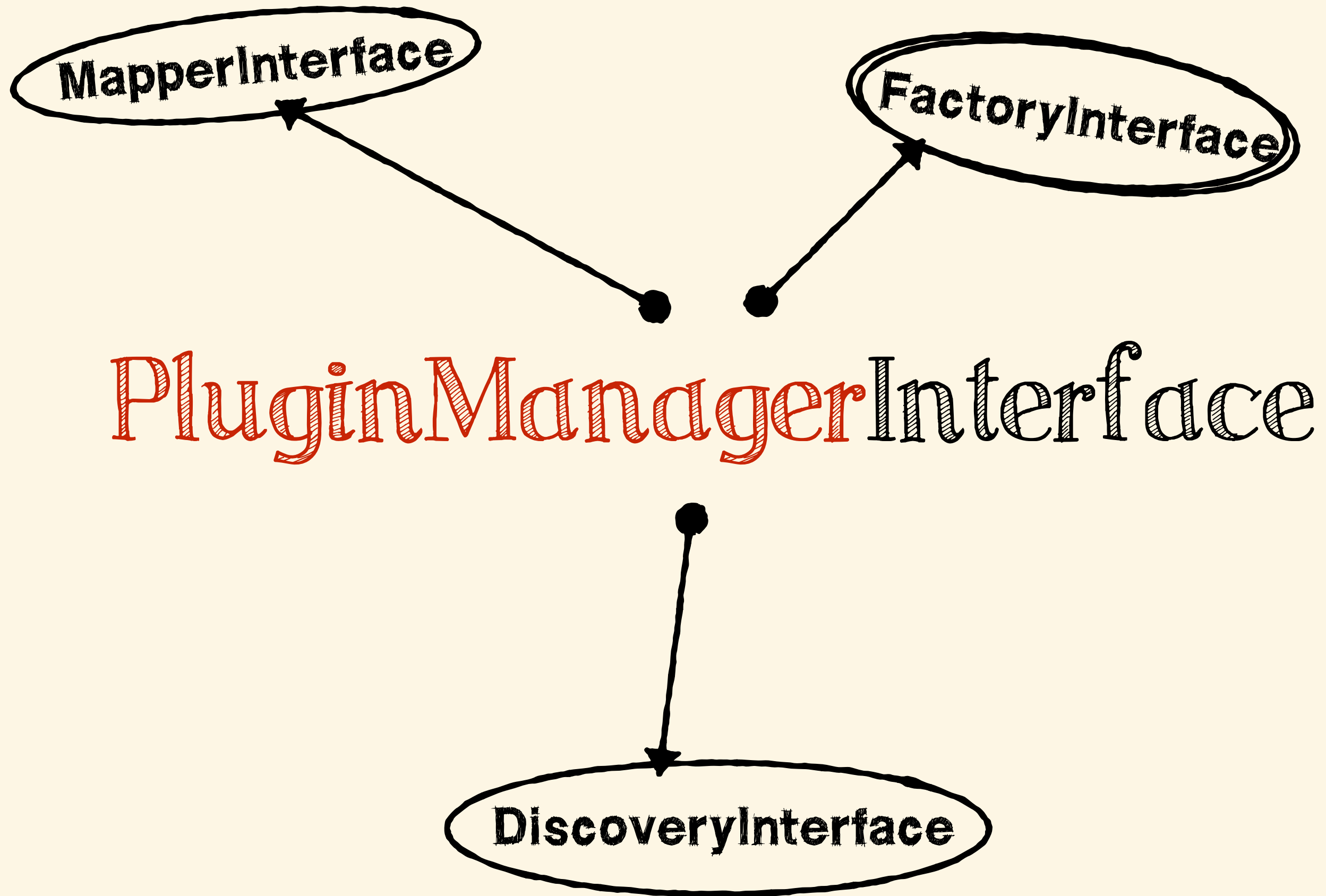
ReflectionFactory

```
$ref = new \ReflectionClass($plugin_class);  
$arguments = $this->getInstanceArguments($ref, $plugin_id, $plugin_def, $config);  
$instance = $reflector->newInstanceArgs($arguments);
```

Plugin Mapper

Mapper Interface

Dynamically figure out which plugin to load based on application state and context.



MyManager extends DefaultPluginManager

Use the default, it does
almost everything you need!

it's a test

Create a system that:

- ☐ Provides a new plugin type for defining ice-cream flavors
- ☐ Defines what info an ice-cream flavor plugin should contain
- ☐ Provides a base class that can be extended to ease new flavor creation
- ☐ Provides 2 sample ice-cream flavors

src/Plugin/IcecreamManager.php

```
namespace Drupal\icecream;

use Drupal\Core\Plugin\DefaultPluginManager;
use Drupal\Core\Cache\CacheBackendInterface;
use Drupal\Core\Extension\ModuleHandlerInterface;

/**
 * Icecream plugin manager.
 */
class IcecreamManager extends DefaultPluginManager {

    /**
     * Constructs an IcecreamManager object.
     */
    public function __construct(\Traversable $namespaces, CacheBackendInterface $cache_backend,
                               ModuleHandlerInterface $module_handler) {
        parent::__construct('Plugin/Flavor', $namespaces, $module_handler,
                            'Drupal\icecream\Annotation\Flavor');

        $this->alterInfo('icecream_flavors_info');
        $this->setCacheBackend($cache_backend, 'icecream_flavors');
    }
}
```

src/Plugin/IcecreamManager.php

```
namespace Drupal\icecream;

use Drupal\Core\Plugin\DefaultPluginManager;
use Drupal\Core\Cache\CacheBackendInterface;
use Drupal\Core\Extension\ModuleHandlerInterface;

/**
 * Icecream plugin manager.
 */
class IcecreamManager extends DefaultPluginManager {

    /**
     * Constructs an IcecreamManager object.
     */
    public function __construct(\Traversable $namespaces, CacheBackendInterface $cache_backend,
                                ModuleHandlerInterface $module_handler) {
        parent::__construct('Plugin/Flavor', $namespaces, $module_handler,
                            'Drupal\icecream\Annotation\Flavor');

        $this->alterInfo('icecream_flavors_info');
        $this->setCacheBackend($cache_backend, 'icecream_flavors');
    }
}
```

src/FlavorInterface.php

```
namespace Drupal\icecream;

/**
 * Defines an interface for ice cream flavor plugins.
 */
interface FlavorInterface {

    /**
     * Return the name of the ice cream flavor.
     *
     * @return string
     */
    public function getName();

    /**
     * Return the price per scoop of the ice cream flavor.
     *
     * @return float
     */
    public function getPrice();

    /**
     * A slogan for the ice cream flavor.
     *
     * @return string
     */
    public function slogan();
}
```


src/Annotation/Flavor.php

```
namespace Drupal\icecream\Annotation;

use Drupal\Component\Annotation\Plugin;

/**
 * Defines a flavor item annotation object.
 *
 * @Annotation
 */
class Flavor extends Plugin {

    /**
     * The plugin ID.
     *
     * @var string
     */
    public $id;

    /**
     * The name of the flavor.
     *
     * @var \Drupal\Core\Annotation\Translation
     */
    public $name;

    /**
     * The price of one scoop of the flavor in dollars.
     *
     * @var float
     */
    public $price;
}
```

src/FlavorBase.php

```
/**
 * @file
 * Provides Drupal\icecream\FlavorBase.
 */

namespace Drupal\icecream;

use Drupal\Component\Plugin\PluginBase;

class FlavorBase extends PluginBase implements FlavorInterface {

    public function getName() {
        return $this->pluginDefinition['name'];
    }

    public function getPrice() {
        return $this->pluginDefinition['price'];
    }

    public function slogan() {
        return t('Best flavor ever.');
```

src/Plugin/Flavor/Vanilla.php

```
/**
 * @file
 * Contains \Drupal\icecream\Plugin\Flavor\Vanilla.
 */

namespace Drupal\icecream\Plugin\Flavor;

use Drupal\icecream\FlavorBase;

/**
 * Provides a 'vanilla' flavor.
 *
 * @Flavor(
 *   id = "vanilla",
 *   name = @Translation("Vanilla"),
 *   price = 1.75
 * )
 */
class Vanilla extends FlavorBase {}
```

src/Plugin/Flavor/Chocolate.php

```
/**
 * @file
 * Contains \Drupal\icecream\Plugin\Flavor\Chocolate.
 */

namespace Drupal\icecream\Plugin\Flavor;

use Drupal\icecream\FlavorBase;

/**
 * Provides a 'chocolate' flavor.
 *
 * @Flavor(
 *   id = "chocolate",
 *   name = @Translation("Chocolate"),
 *   price = 1.75
 * )
 */
class Chocolate extends FlavorBase {
  public function slogan() {
    return t('The other best flavor.');
  }
}
```

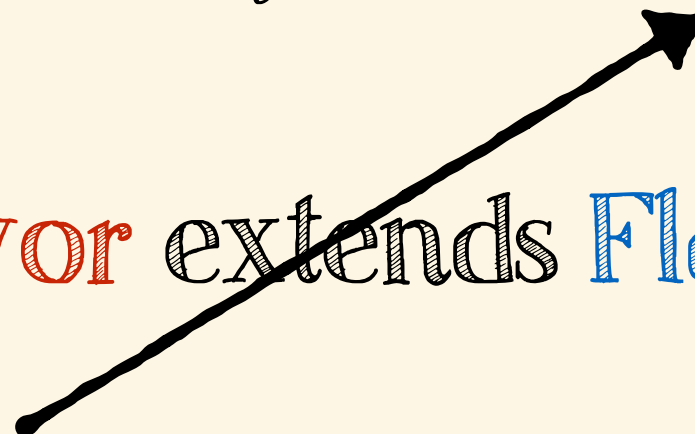
1. Follow the PSR-4 standard



mymodule/src/Plugin/Flavor/MyFlavor.php

namespace Drupal\mymodule\Plugin\Flavor

class MyFlavor extends FlavorBase {}



2. Use the "Plugin/Flavor" sub-namespace



3. Extend the base class.

icecream.services.yml

```
services:
```

```
  plugin.manager.icecream:
```

```
    class: Drupal\icecream\IcecreamManager
```

```
    parent: default_plugin_manager
```

Use your IcecreamManager Service

```
$manager = \Drupal::service('plugin.manager.icecream');  
$plugins = $manager->getDefinitions();
```

// OR

```
$vanilla = $manager->createInstance('vanilla');  
print $vanilla->getPrice();
```

❑ Plugins are reusable bits of functionality that are configurable, re-usable, and do exactly one thing.

❑ Plugins are PHP classes that implement a defined interface.

❑ Creating new plugins requires knowledge of PSR-4, Annotations, and sometimes Dependency Injection and Service Containers.

❑ Plugins types are defined and managed by a plugin manager.

TaDa!

Drupal 8 Plugins

Joe @eojthebrave Shindelar