Creating Modular Applications

obecto

Component-based Software Engineering
or
Modular Programming

obecto

Components

- multiple-use
- non-context specific
- composable with other components
- encapsulated
- unit of independent deployment and versioning

Modules

- improve decomposition and separation of concerns
- independent maintainability
- modules are incorporated in a program through interfaces
- the interfaces express the elements that are provided and required by the module

Both are written to a specification

obecto

Benefits of Using Modules

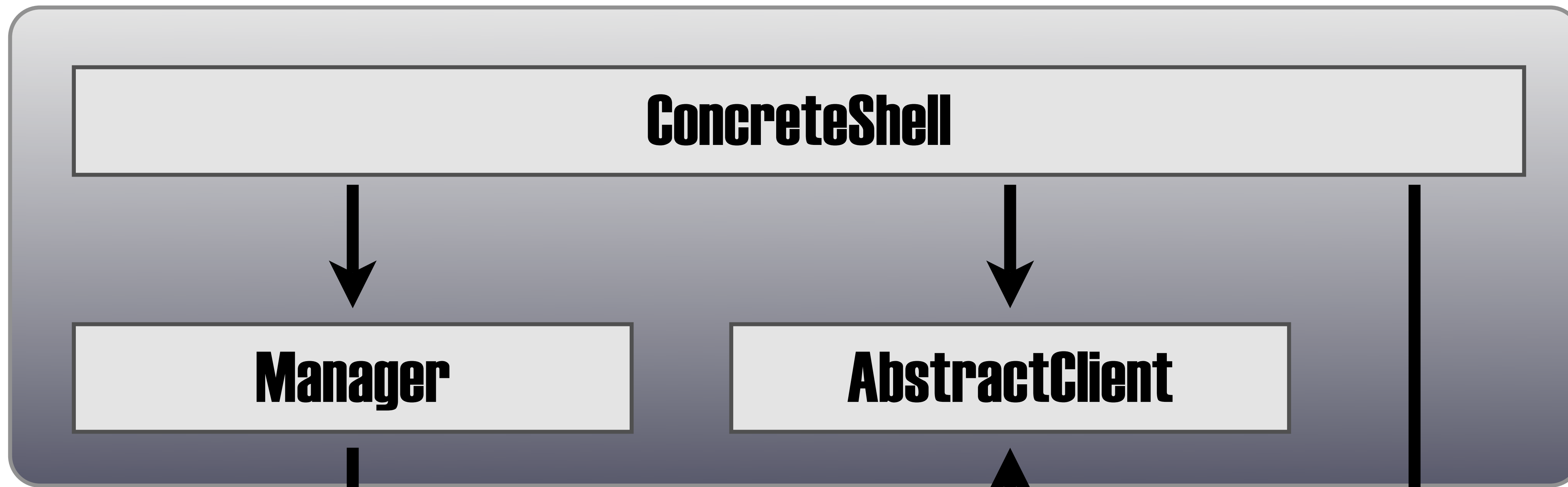
- allows an application to load code at run-time
- modules can be loaded and unloaded at run-time
- modules reduce application size and initial download times
- modules can be compiled without the application, thus reducing the time needed for application compilation

obecto

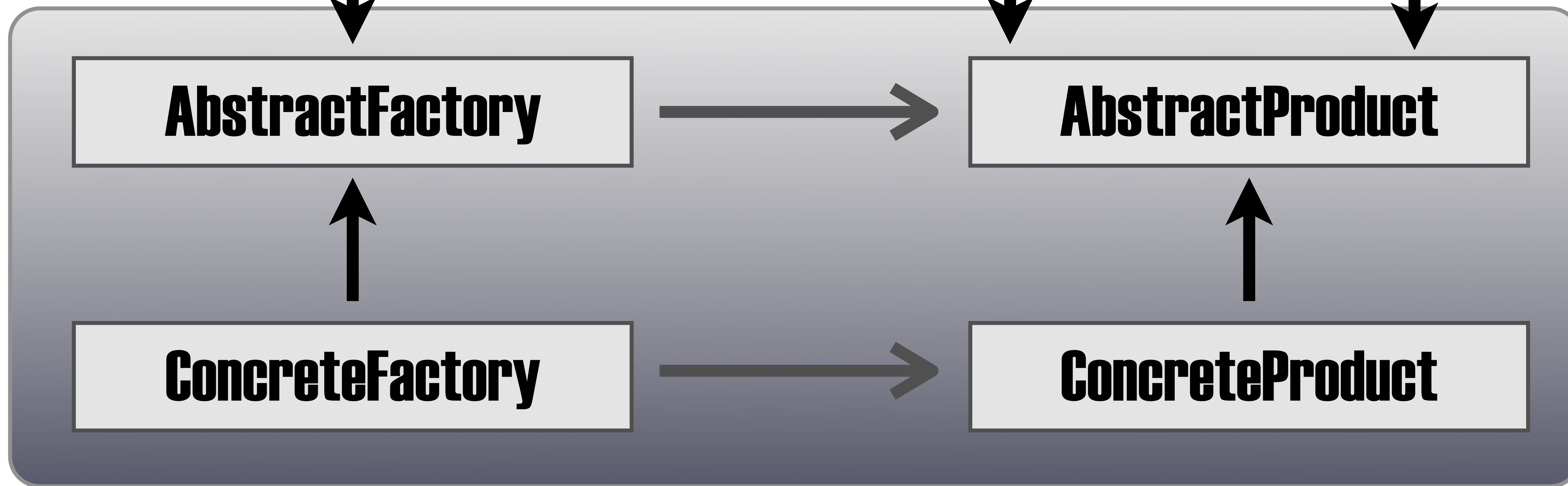
Relationship between the shell and the module's interface

obecto

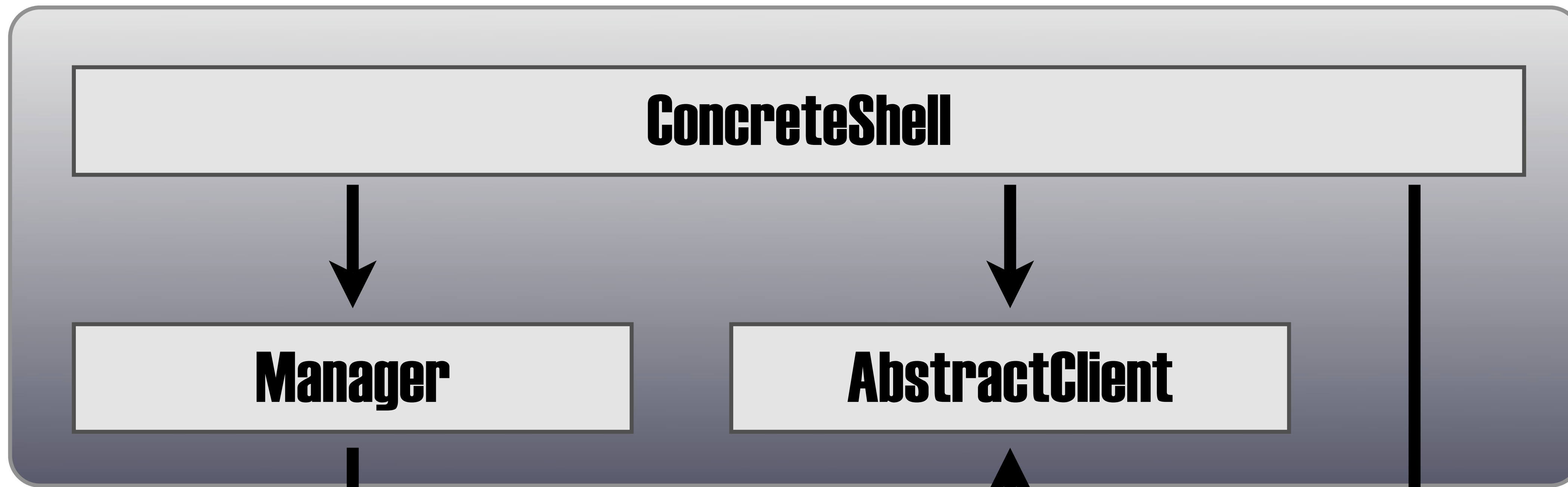
Application

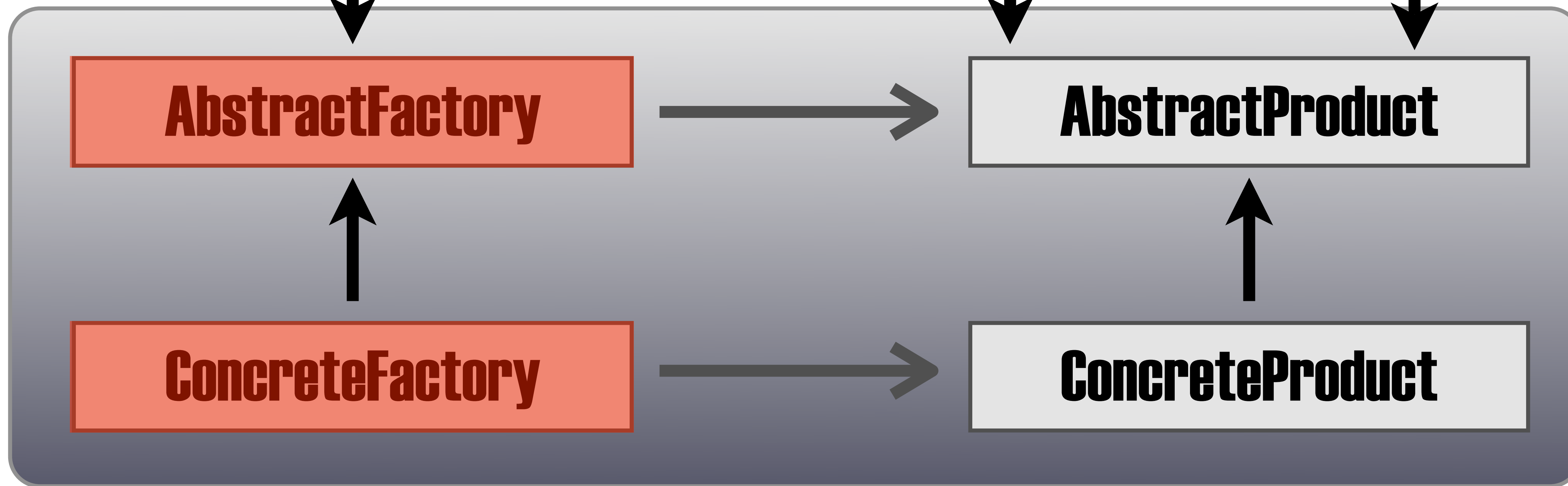


Module



obecto

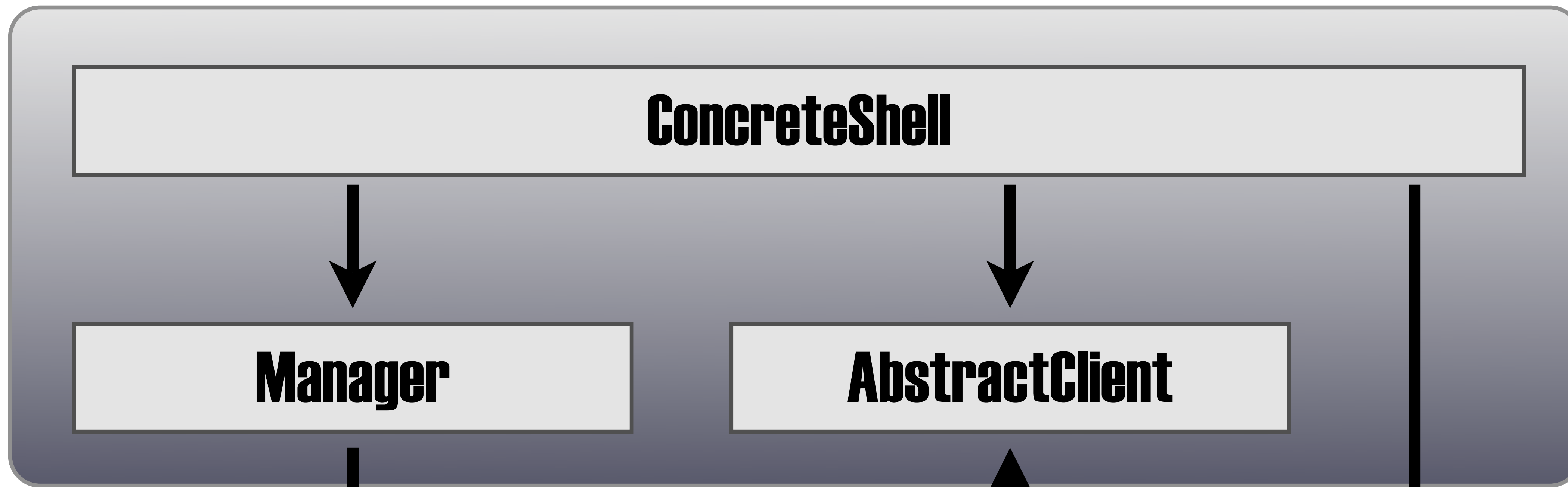




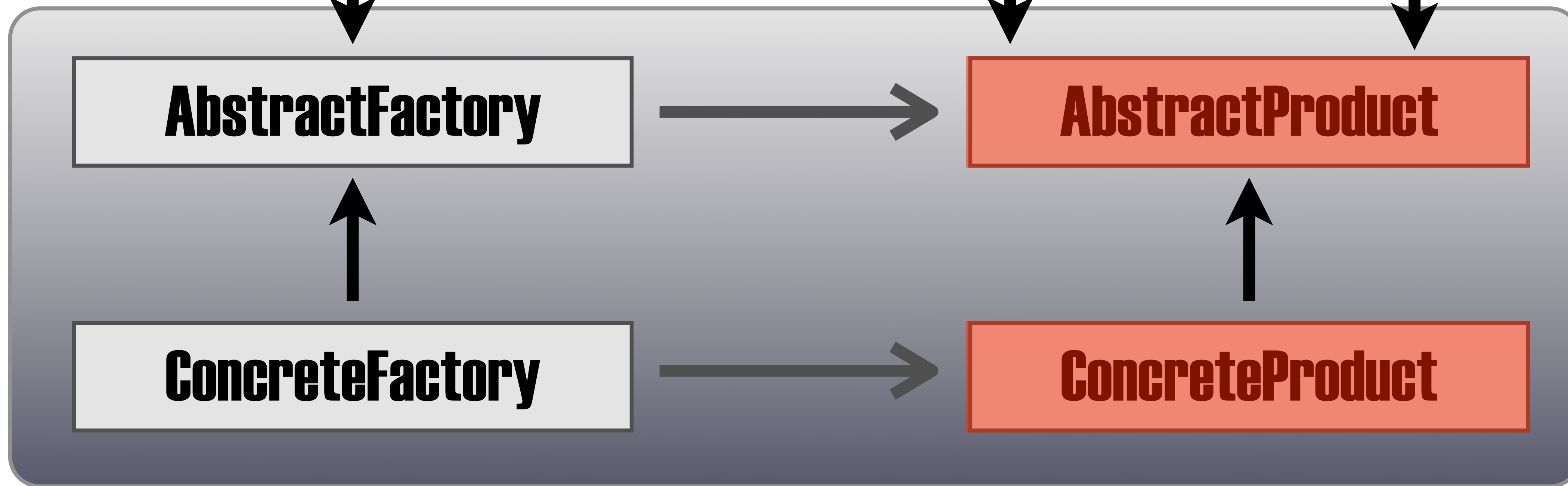
modules implement a class factory with a standard interface

obecto

Application



Module

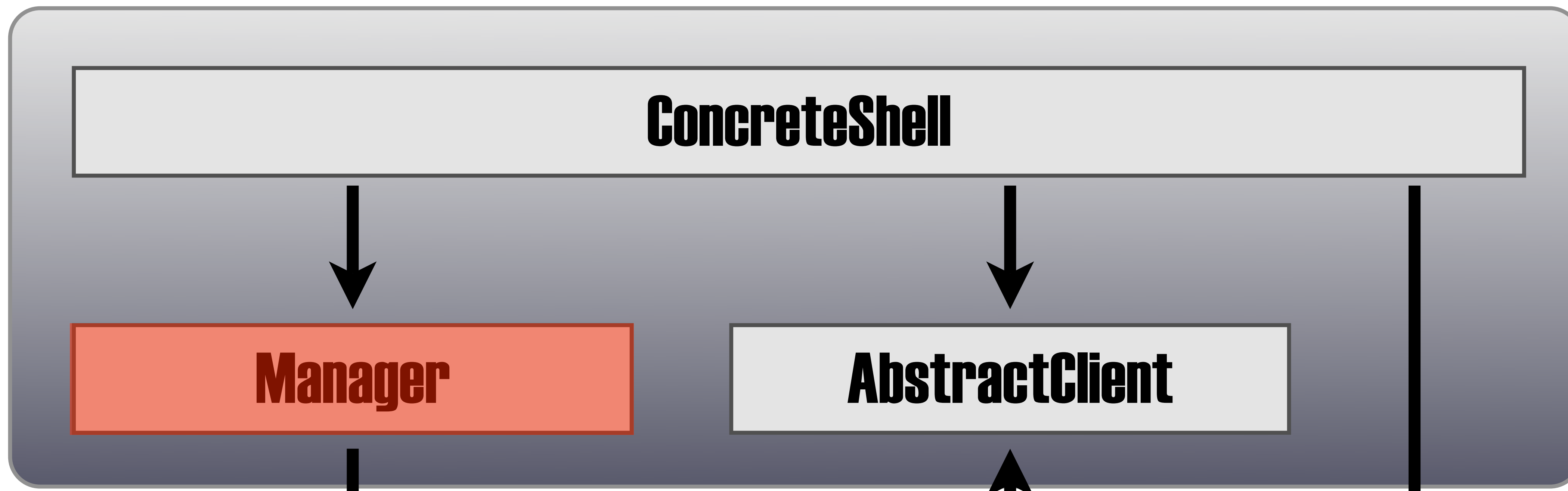


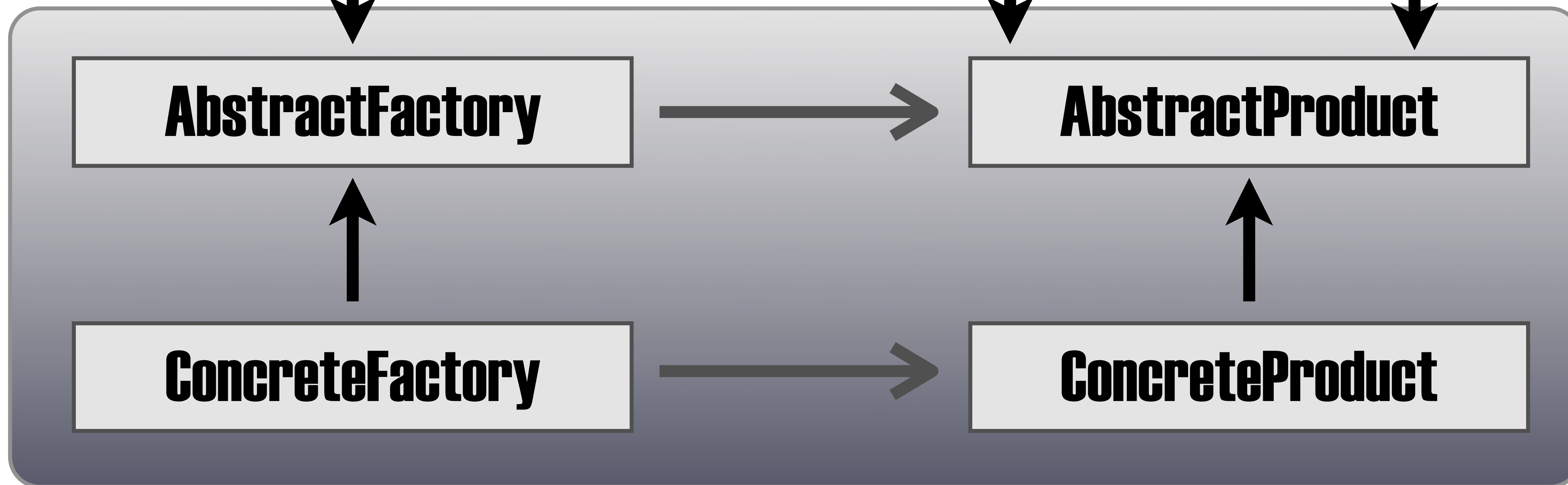
the product of the factory implements an interface known to the shell

or

the shell implements an interface known to the modules

obecto





- the *ModuleManager* manages the loaded modules
- map of singletons indexed by module URL
- modules are only loaded once
- on every module load completion *ModuleEvent.READY* is dispatched
- you can also use the *ModuleLoader*

obecto

MXML-based modules

```
<?xml version=”1.0”?>
<mx:Module
    xmlns:mx=”http://www.adobe...”
    width=”100%” height=”100%”>

    <!-- Module code -->

</mx:Module>
```

- to create a module with MXML use *mx.core.Module* as a root tag
- *mx.core.Module* inherits *LayoutContainer*

obecto

ActionScript-based modules

```
package {
    import mx.modules.ModuleBase;

    class SimpleModule
        extends ModuleBase {

        // module ActionScript

    }
}
```

- again you can extend the *mx.core.Module*
- a better approach is to extend the *mx.modules.ModuleBase*
- the *ModuleBase* is just an EventDispatcher

obecto

Compiling Modules

Using modules in a single project

- modules are automatically recompiled with the application
- you can compile modules individually
- the application and the modules share the same compiler settings

- the application and the modules must be in the same source directory
- you can't use the *load-externs* to remove overlapping dependencies

obecto

Using multiple projects

- create separate projects for each module
- module project can be placed anywhere
- every project can have different compiler settings
- module projects can use the load-externs compiler option

Reducing module size

- by default a module includes all framework code that it's components depend on
- you can instruct the module to externalize classes that are included by the application
- first you need to generate a linker report and then pass it to the *load-externs* compiler option

```
mxmlc -link-report=report.xml MyApplication.mxml
mxmlc -load-externs=report.xml MyModule.mxml
```

obecto

Communication between parts

- modules can access the application from the *parentApplication* property
- the application can access the module from the *child* property of the ModuleLoader and the ModuleManager's *factory*
- query string parameters added to the module's URL - you can parse this string in the module by accessing *this.loaderInfo.url*

obecto

Summary

- comparison between components and modules
- the benefits of using modules
- the relationship between the application and the modules
- MXML-based modules
- ActionScript-based modules
- module compilation and project setup
- reducing module size
- different ways to communicate between the application and the modules and among modules

obecto