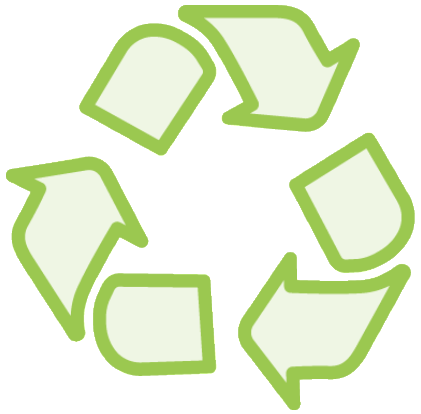
Working with Traits

Harit Himanshu
@harittweets

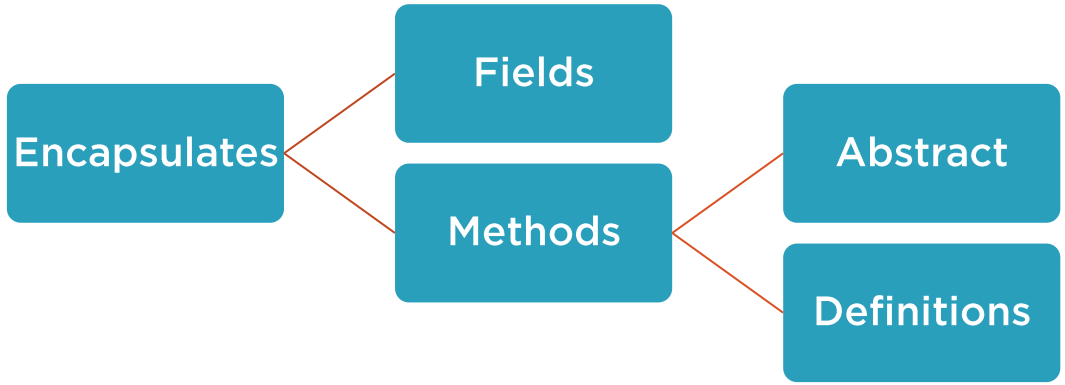
Overview

- Understanding Traits
- Developing Rich Interfaces with Traits
- Developing Stackable Modifications with Traits
- Understanding When to Use Traits or Not
- Project Demo

Traits



Code Reuse



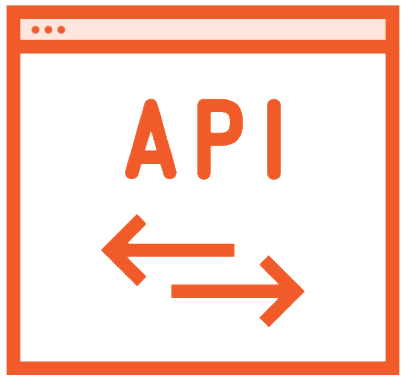
Trait Structure

```
trait Db {
 val type =  “MySQL”
 def get(id: Int)
}
```

Trait Mix-ins

```
// when extending class

class MyClass extends OtherClass with Trait1 with Trait2 with Trait3



// when just extending trait

class MyClass extends Trait1 with Trait2 with Trait3
```

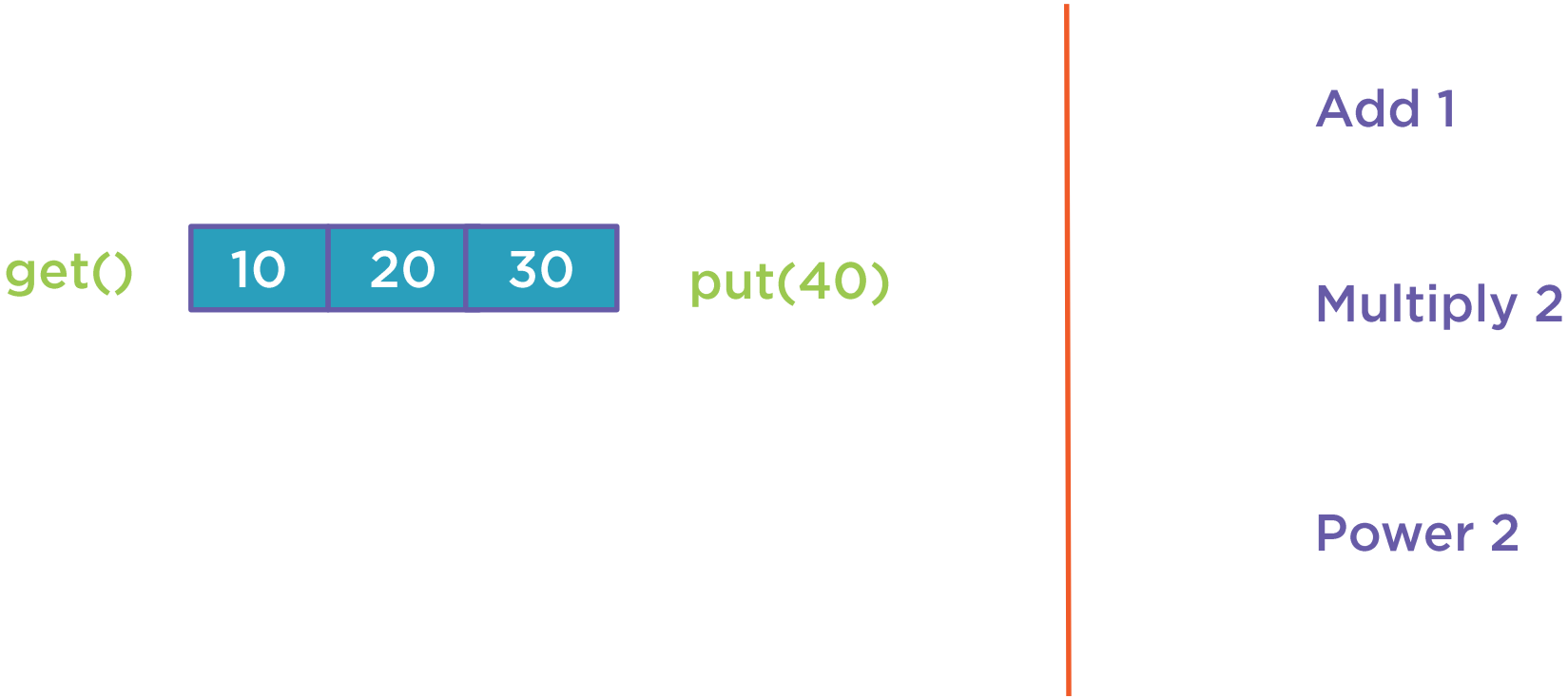
Trait Use Cases



Rich Interfaces

Stackable Modifications

Stackable Modification Example

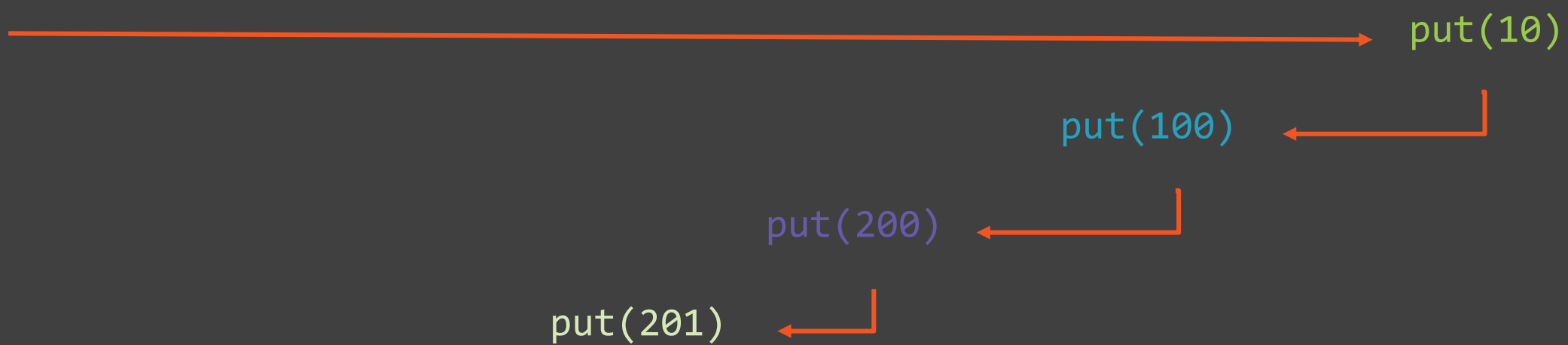


Order of Traits Example

```
class Add1Q extends Q with Add1
```

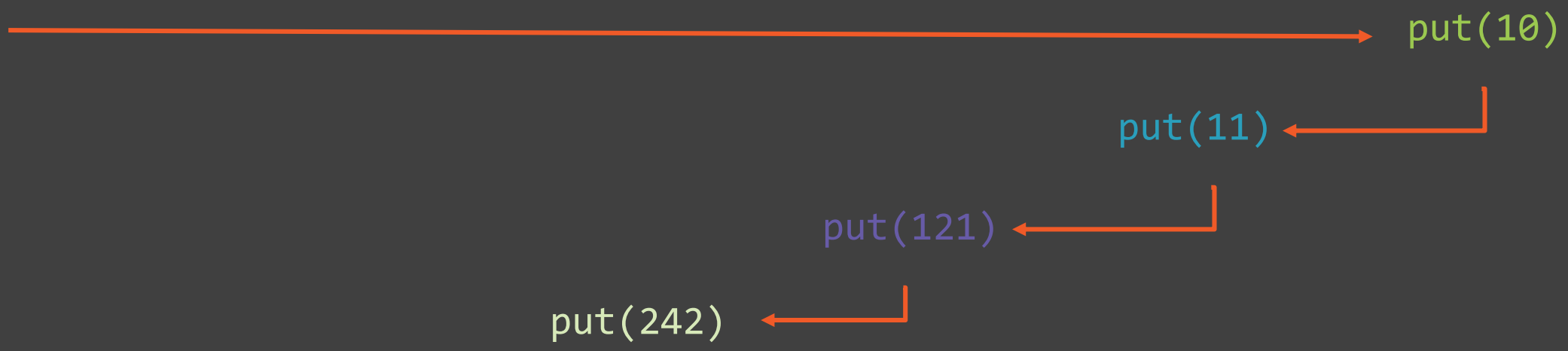
Order of Traits Example

```
class Add1Mul2Pow2 extends Q with Add1 with Multiply2 with Power2
```

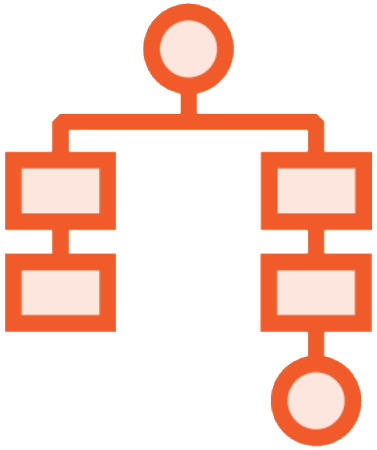


Order of Traits Example

```
class Mul2Pow2Add1 extends Q with Multiply2 with Power2 with Add1
```



Abstract Class vs. Traits



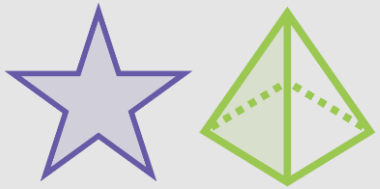
Abstract Classes

Traits

Guidelines for Traits vs. Classes

No Reuse

Concrete Class



Unrelated Uses

Traits

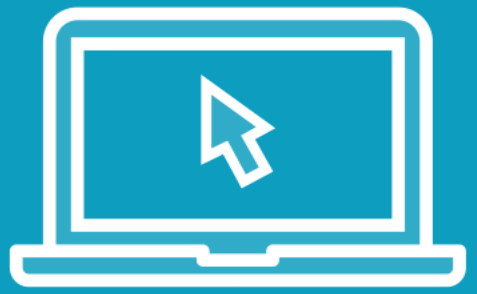
Related Uses

Abstract Class

Not Sure

Traits

Demo



Creating Database as Traits

Creating Services as Traits

Extending `Dollar` Class by mix-in

Summary

- Understanding Traits
- Developing Rich Interfaces with Traits
- Developing Stackable Modifications with Traits
- Understanding When to Use Traits or Not
- Project Demo