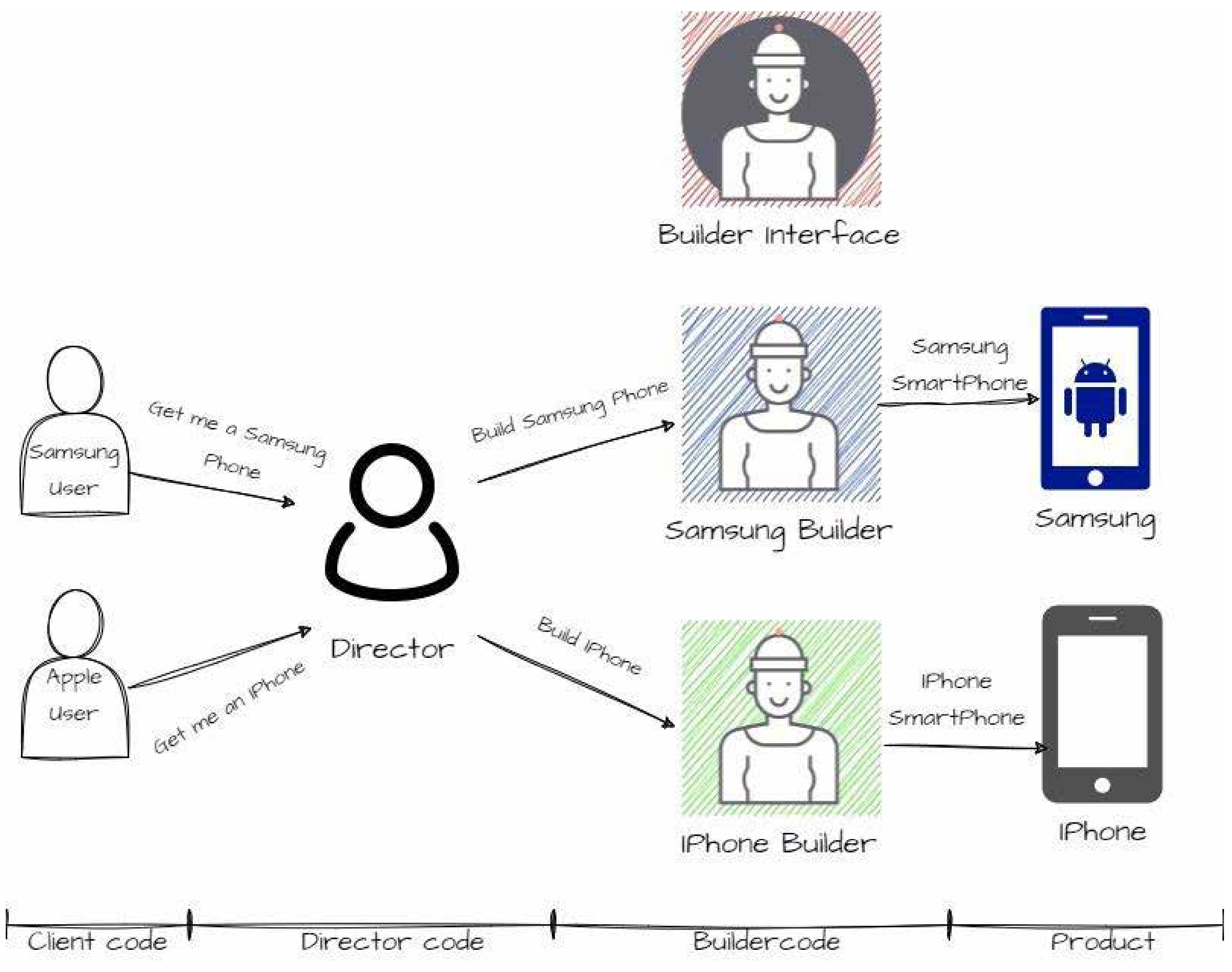
Builder Design Pattern

Understanding the Builder Design Pattern and Its Applications in Programming

In software development, the Builder Design Pattern is used to construct complex objects step by step in a flexible and readable manner. This pattern is especially useful for creating and managing queries, improving code readability and maintainability. In this article, we will explain the main components of the Builder pattern and provide a simple implementation example in PHP, specifically using Query Builder.



Ehsan Ghajar

Main Components of the Builder Design Pattern

The Builder pattern typically consists of the following components:

1. Builder:

This interface or abstract class defines the necessary methods for building different parts of an object.

2. Concrete Builder:

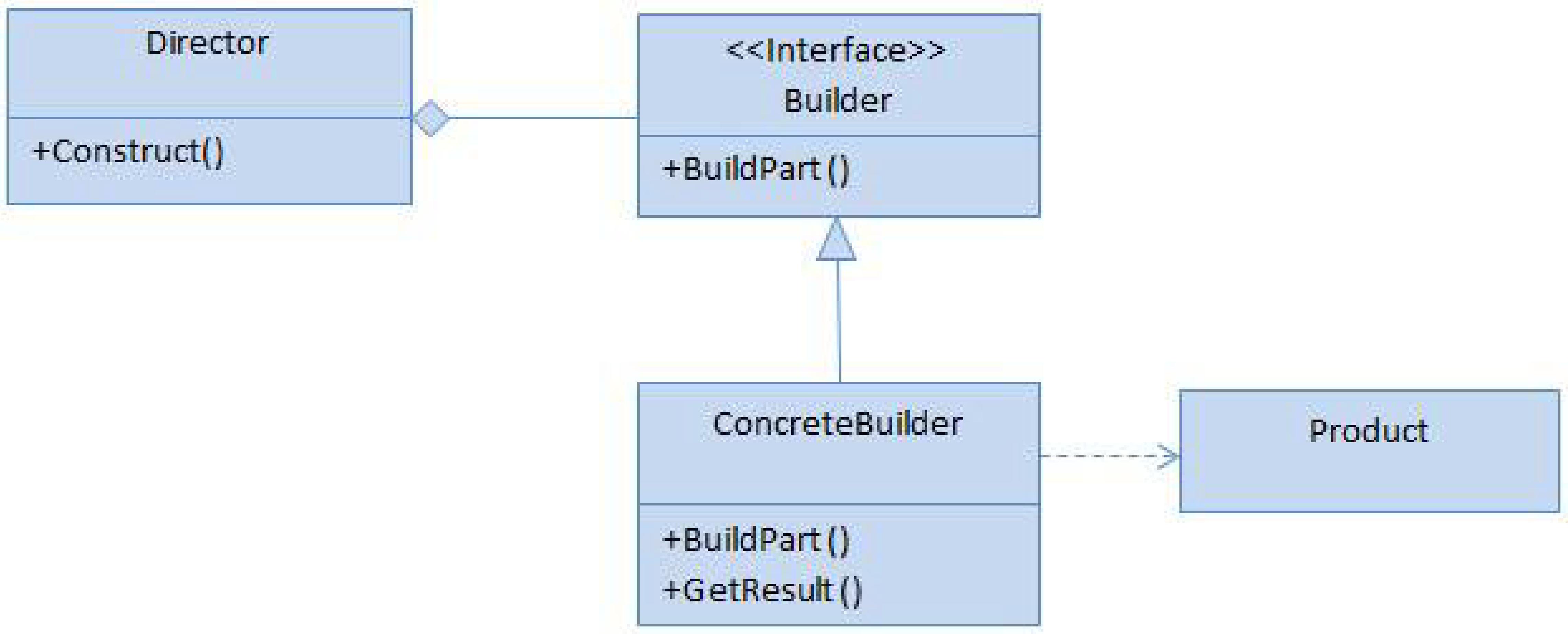
This class implements the Builder interface and creates the required components for constructing the final object.

3. Director:

This class is responsible for controlling and managing the object construction process.

4. Product:

The final object that is constructed, containing various information and components.



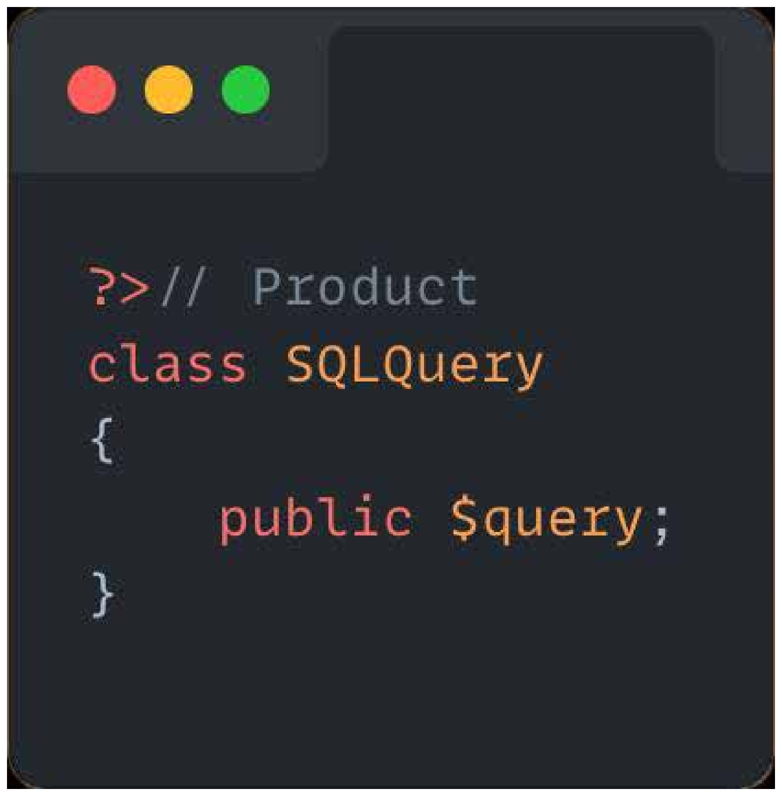
Builder Pattern

Ehsan Ghaja

Simple Implementation of the Builder Pattern in PHP

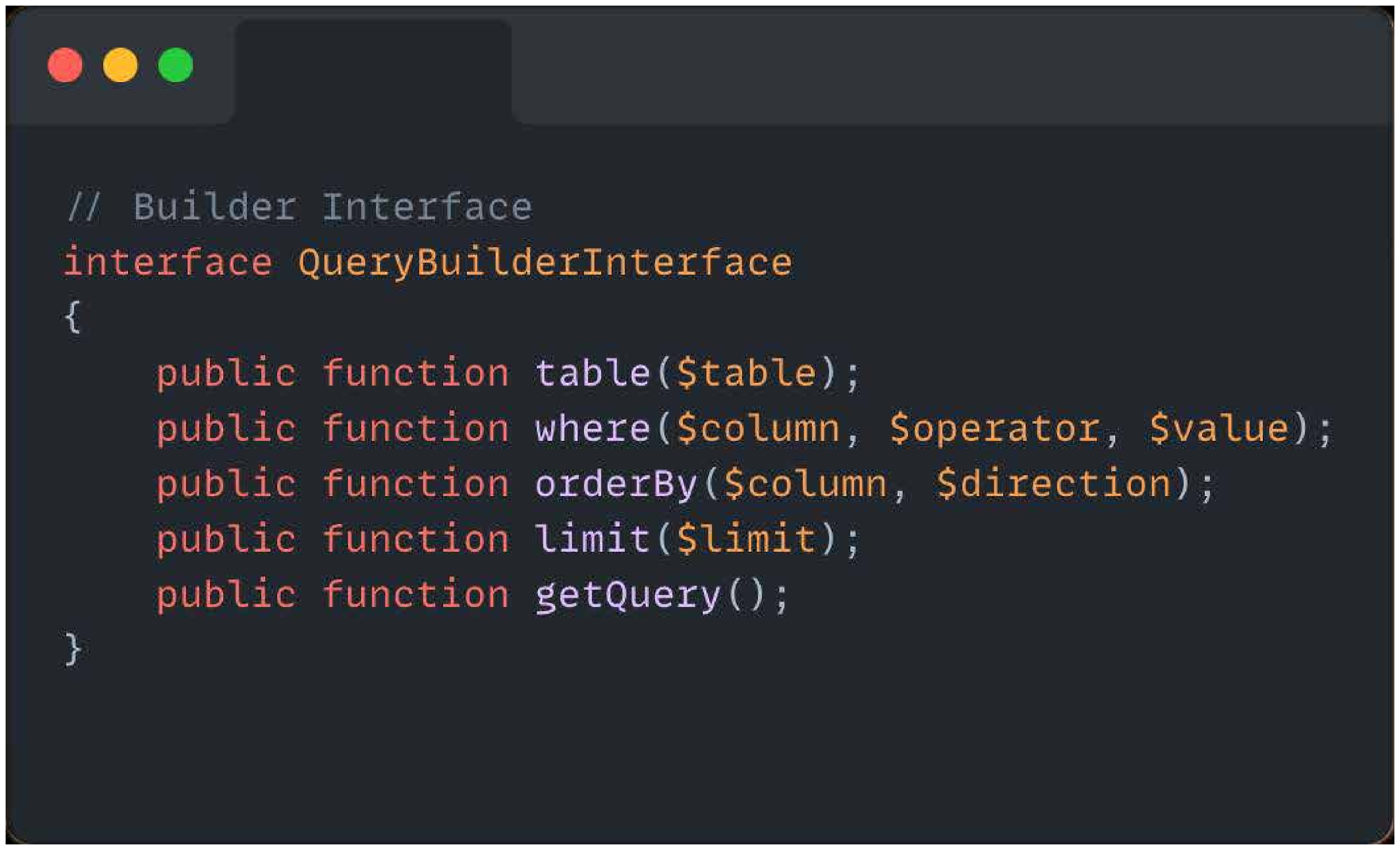
In this section, we present each component of the Builder pattern separately.

1. Product



```php
?>// Product
class SQLQuery
{
    public $query;
}
```

2. Builder Interface

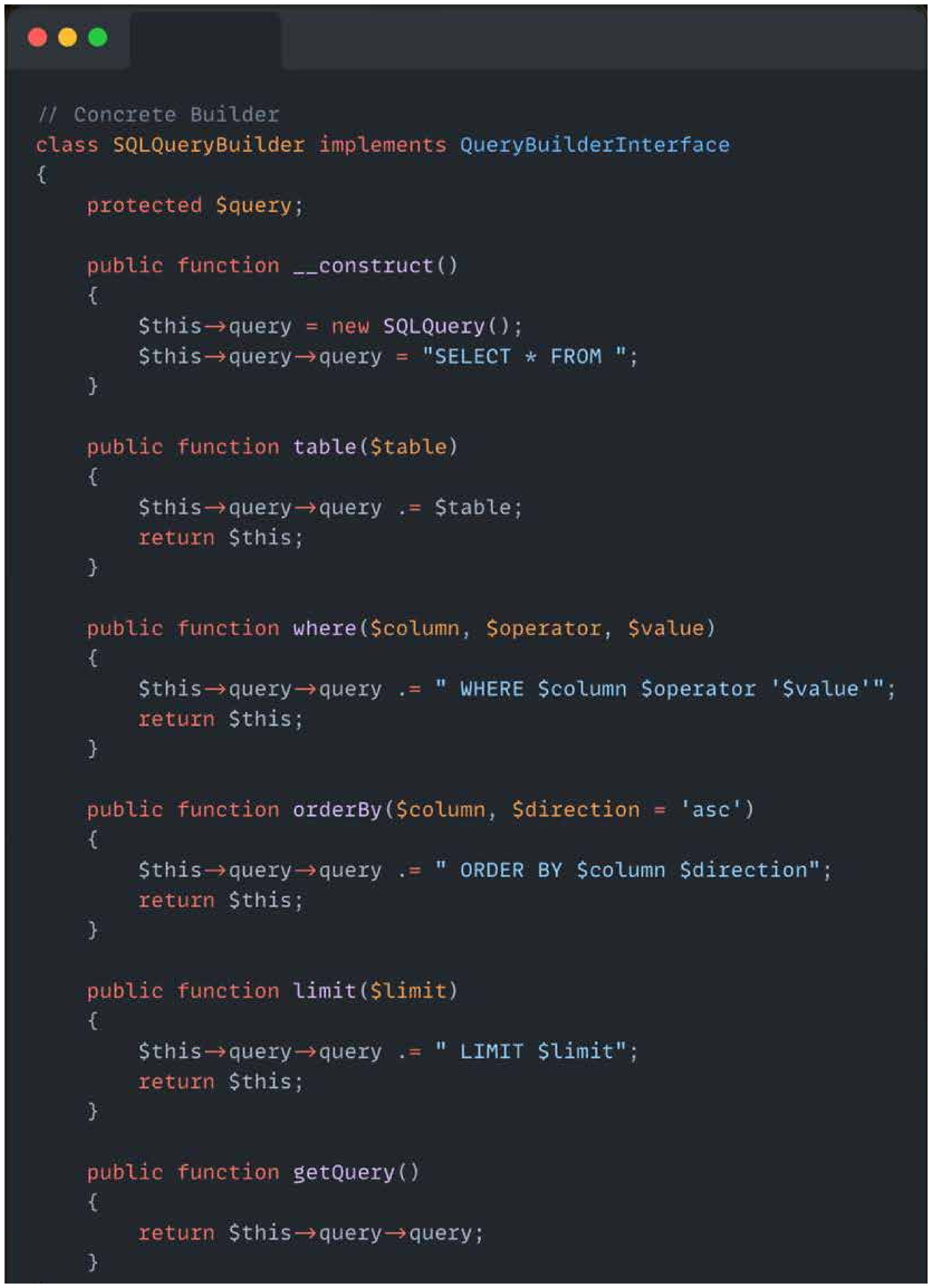


```php
// Builder Interface
interface QueryBuilderInterface
{
    public function table($table);
    public function where($column, $operator, $value);
    public function orderBy($column, $direction);
    public function limit($limit);
    public function getQuery();
}
```

Ehsan Ghajar

Simple Implementation of the Builder Pattern in PHP

3. Concrete Builder

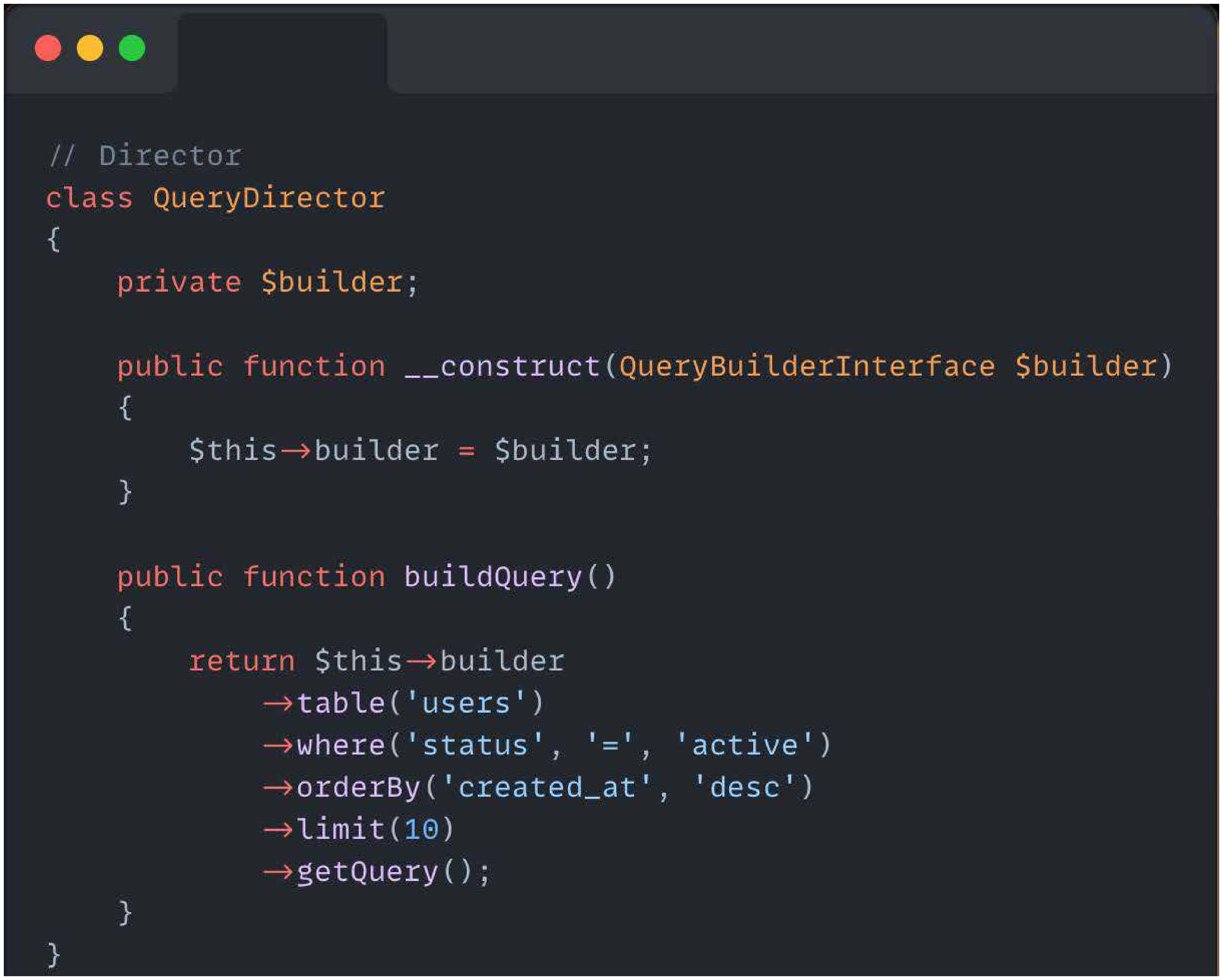


```php
// Concrete Builder
class SQLQueryBuilder implements QueryBuilderInterface
{
    protected $query;

    public function __construct()
    {
        $this->query = new SQLQuery();
        $this->query->query = "SELECT * FROM ";
    }

    public function table($table)
    {
        $this->query->query .= $table;
        return $this;
    }

    public function where($column, $operator, $value)
    {
        $this->query->query .= " WHERE $column $operator '$value'";
        return $this;
    }

    public function orderBy($column, $direction = 'asc')
    {
        $this->query->query .= " ORDER BY $column $direction";
        return $this;
    }

    public function limit($limit)
    {
        $this->query->query .= " LIMIT $limit";
        return $this;
    }

    public function getQuery()
    {
        return $this->query->query;
    }
```

Ehsan Ghajar

Simple Implementation of the Builder Pattern in PHP

4. Director

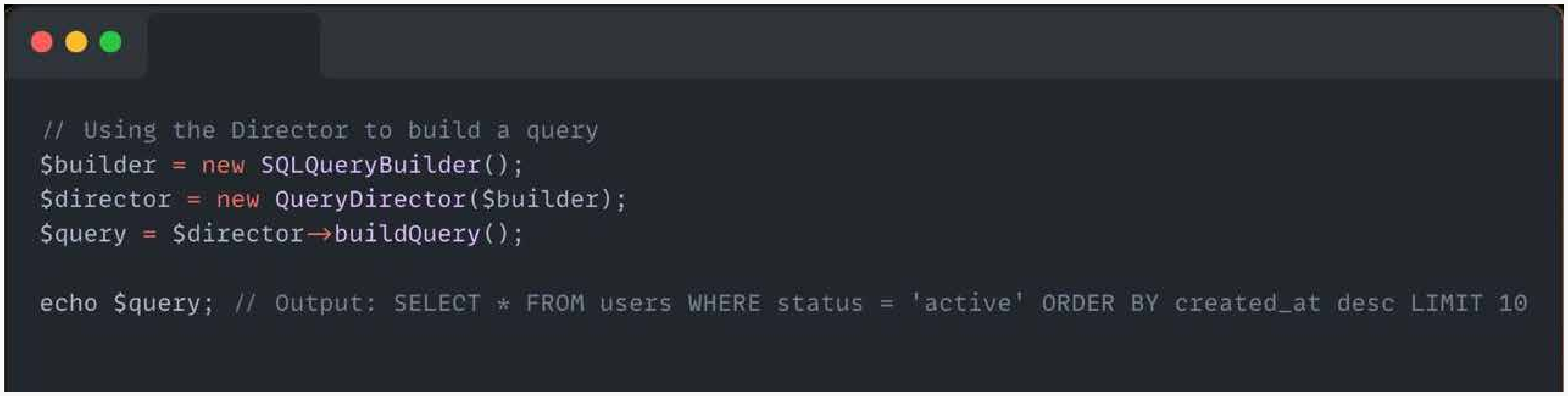


```php
// Director
class QueryDirector
{
    private $builder;

    public function __construct(QueryBuilderInterface $builder)
    {
        $this->builder = $builder;
    }

    public function buildQuery()
    {
        return $this->builder
            ->table('users')
            ->where('status', '=', 'active')
            ->orderBy('created_at', 'desc')
            ->limit(10)
            ->getQuery();
    }
}
```

Ehsan Ghajar

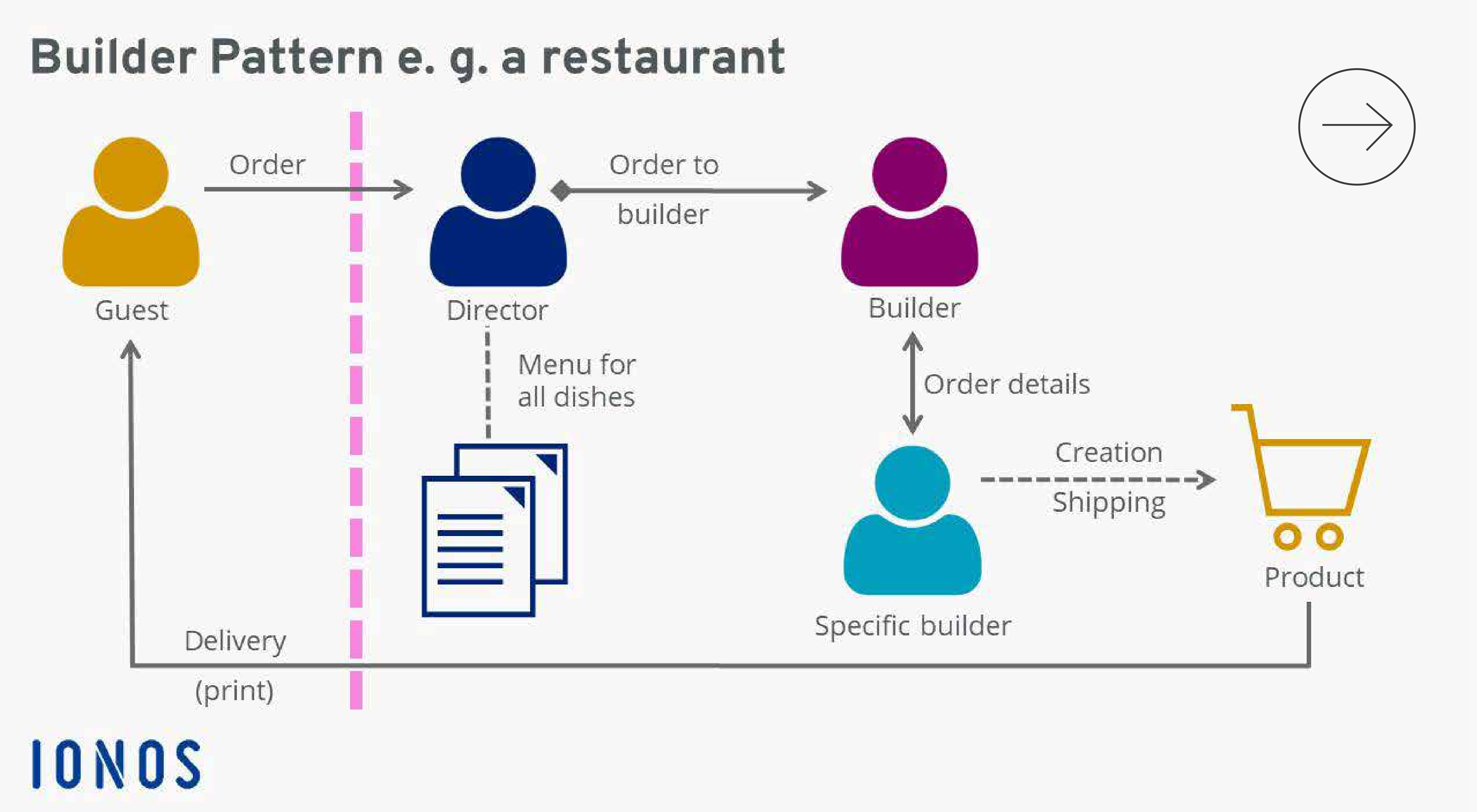
Simple Implementation of the Builder Pattern in PHP

5. Using the Director to Build a Query



```php
// Using the Director to build a query
$builder = new SQLQueryBuilder();
$director = new QueryDirector($builder);
$query = $director->buildQuery();

echo $query; // Output: SELECT * FROM users WHERE status = 'active' ORDER BY created_at desc LIMIT 10
```

This implementation has a simple structure but effectively demonstrates how the Builder pattern allows you to create complex queries step by step in a readable manner.



Other Applications of the Builder Design Pattern

1. Creating Complex Forms:

The Builder pattern can help create complex forms that include various types of inputs and settings.

2. Configuration Settings:

It can be used to construct a configuration object in applications that require loading and managing complex settings.

3. Creating Objects in Games:

In game development, the Builder pattern can be used to create various objects such as characters, weapons, and environmental items.

4. Document and Report Generation:

For producing complex documents or reports in various formats (PDF, HTML, etc.), the Builder pattern can manage the construction of these documents.

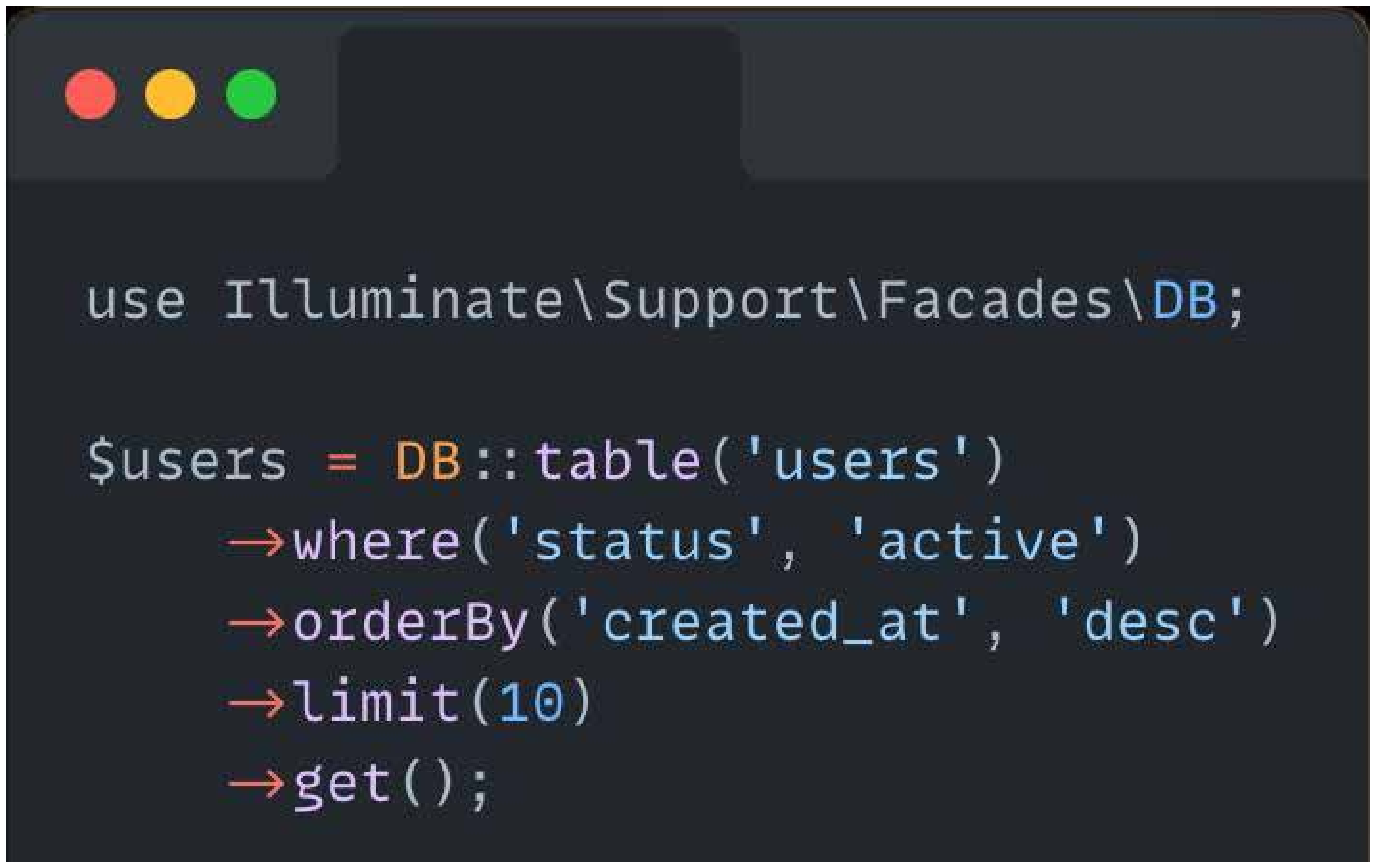
5. Managing HTTP Requests:

In API design, the Builder pattern can construct complex HTTP requests with various parameters and headers.

Ehsan Ghajar

Practical Use in Frameworks

The **Query Builder** in many frameworks, including **Laravel, Django** in Python, and **.NET**, utilizes the Builder pattern to create complex query structures, often using method chaining. For instance, in Laravel, you can easily create queries like the following:



```php
use Illuminate\Support\Facades\DB;

$users = DB::table('users')
    →where('status', 'active')
    →orderBy('created_at', 'desc')
    →limit(10)
    →get();
```

Ehsan Ghajar

Benefits of the Builder Pattern

1. Improved Code Readability:

By breaking down the construction process into distinct methods, the code becomes easier to read and understand.

2. Encapsulation of Construction Logic:

The Builder pattern encapsulates the logic required to construct an object, making the main code cleaner and easier to maintain.

3. Flexibility:

The pattern allows for different representations of a product, as you can create multiple builders for different types of objects.

4. Ease of Maintenance:

When changes are needed, the modifications can typically be made within the builder classes without affecting the client code that uses them.

5. Method Chaining:

The pattern often supports method chaining, allowing for more concise and expressive code.

Ehsan Ghajar

Conclusion

The Builder Design Pattern is a powerful and flexible pattern that allows for the step-by-step and chainable construction of complex objects. By utilizing this pattern, developers can write cleaner, more maintainable, and extensible code while benefiting from improved readability and flexibility in object construction.

Ehsan Ghajar

Ehsan Ghajar

“The Builder Design Pattern is one of my favorite design patterns. Those who have worked with Eloquent and Query Builder in Laravel can truly appreciate how this pattern simplifies the process of constructing complex queries. With the Builder pattern, writing clean and readable code becomes effortless, allowing developers to focus on functionality rather than dealing with cumbersome syntax. It’s a perfect example of how design patterns can enhance our coding experience and lead to more maintainable applications.”