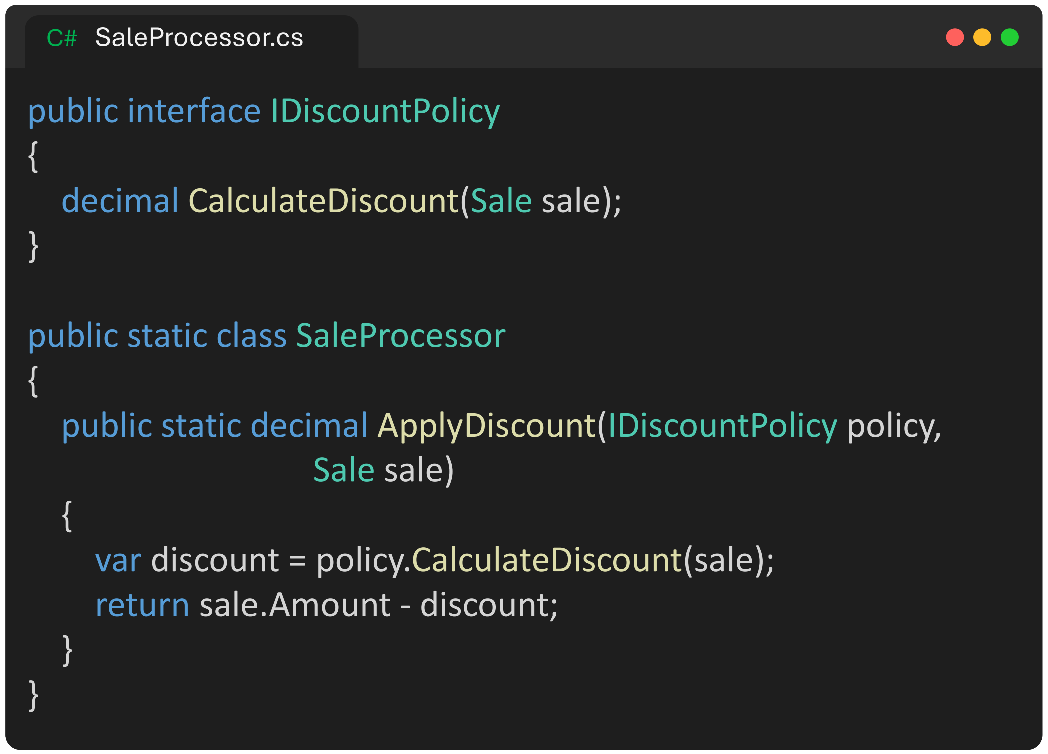
OBJECT ORIENTED PROGRAMMING

FUNCTIONAL PROGRAMMING

OOP Example

A Sale processor that calculates the price, while allowing you to supply custom discount policy

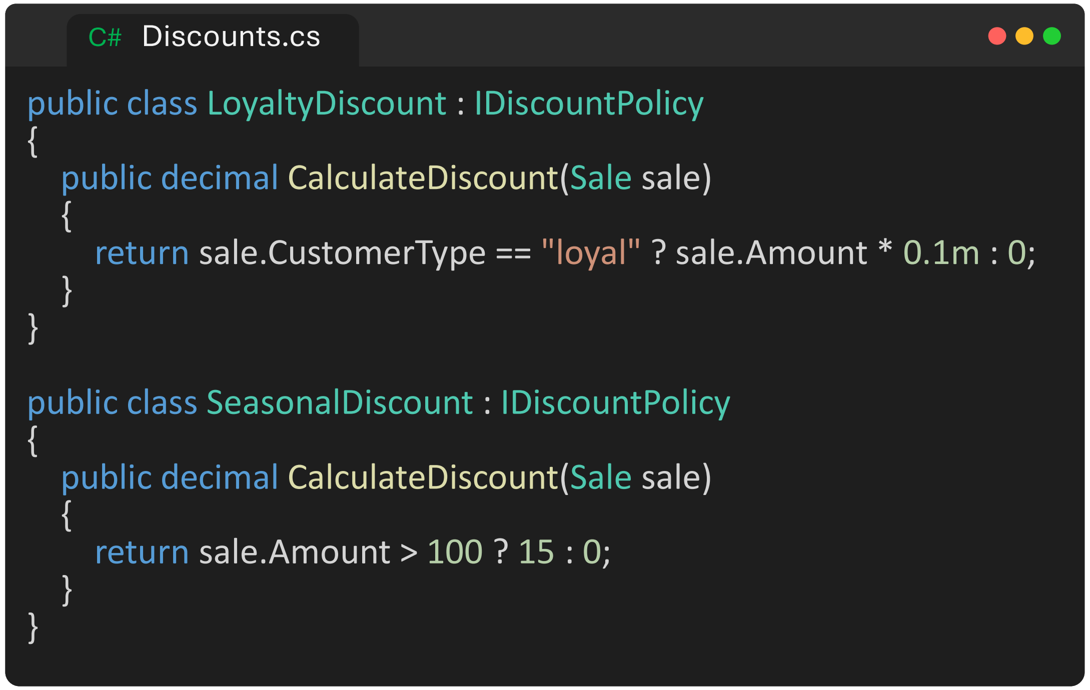


```csharp
public interface IDiscountPolicy
{
    decimal CalculateDiscount(Sale sale);
}

public static class SaleProcessor
{
    public static decimal ApplyDiscount(IDiscountPolicy policy,
                                        Sale sale)
    {
        var discount = policy.CalculateDiscount(sale);
        return sale.Amount - discount;
    }
}
```

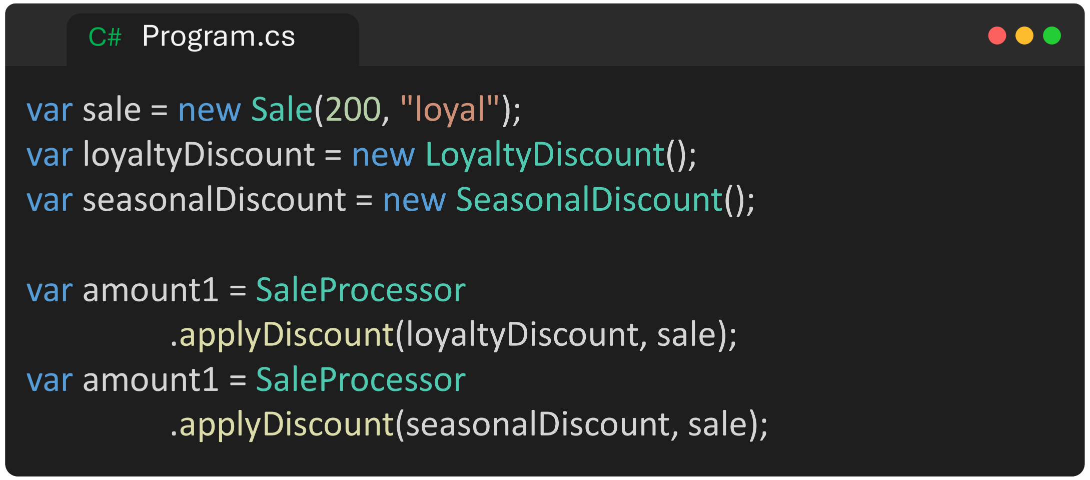
OOP Example

A Sale processor that calculates the price, while allowing you to supply custom discount policy



```csharp
public class LoyaltyDiscount : IDiscountPolicy
{
   public decimal CalculateDiscount(Sale sale)
   {
      return sale.CustomerType == "loyal" ? sale.Amount * 0.1m : 0;
   }
}

public class SeasonalDiscount : IDiscountPolicy
{
   public decimal CalculateDiscount(Sale sale)
   {
      return sale.Amount > 100 ? 15 : 0;
   }
}
```

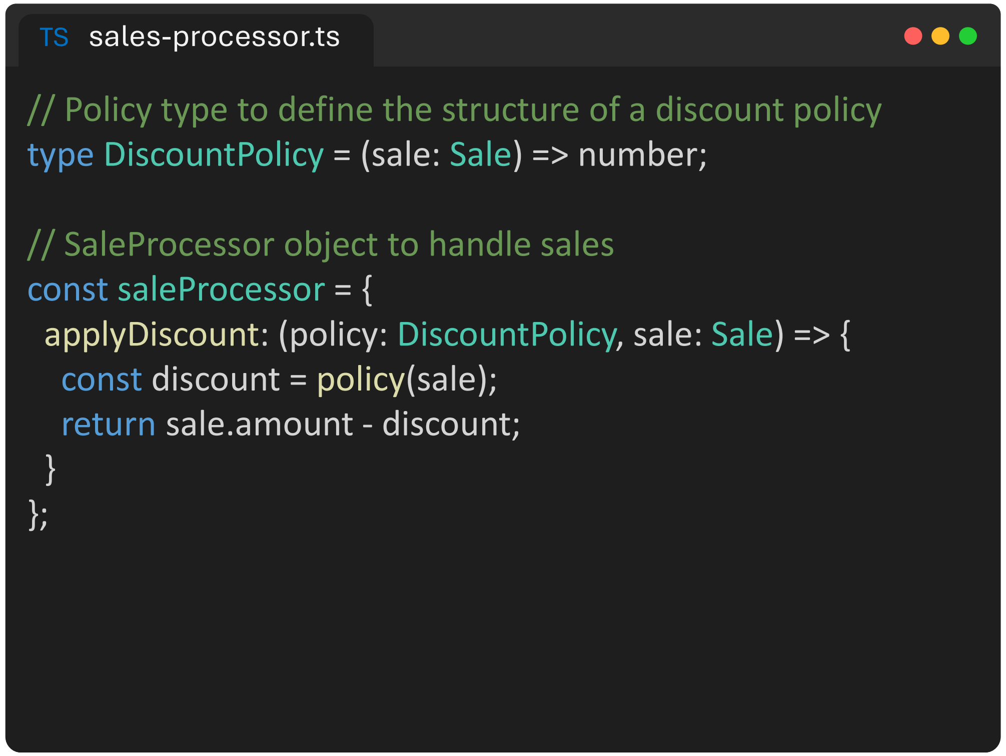


```csharp
var sale = new Sale(200, "loyal");
var loyaltyDiscount = new LoyaltyDiscount();
var seasonalDiscount = new SeasonalDiscount();

var amount1 = SaleProcessor
            .applyDiscount(loyaltyDiscount, sale);
var amount1 = SaleProcessor
            .applyDiscount(seasonalDiscount, sale);
```

FP Example

A Sale processor that calculates the price, while allowing you to supply custom discount policy


TS sales-processor.ts

```typescript
// Policy type to define the structure of a discount policy
type DiscountPolicy = (sale: Sale) => number;

// SaleProcessor object to handle sales
const saleProcessor = {
  applyDiscount: (policy: DiscountPolicy, sale: Sale) => {
    const discount = policy(sale);
    return sale.amount - discount;
  }
};
```

FP Example

A Sale processor that calculates the price, while allowing you to supply custom discount policy

TS discounts.ts

```ts
const loyaltyDiscount = (sale: Sale) =>
   sale.customerType === "loyal"
      ? sale.amount * 0.1
      : 0;
const seasonalDiscount = (sale: Sale) =>
   sale.amount > 100
      ? 15
      : 0;
```

TS program.ts

```ts
const sale:Sale = { amount: 200, customerType: 'loyal' };
const amount1 = saleProcessor.applyDiscount(
   sale, loyaltyDiscount);
const amount2 = saleProcessor.applyDiscount(
   sale, seasonalDiscount);
```

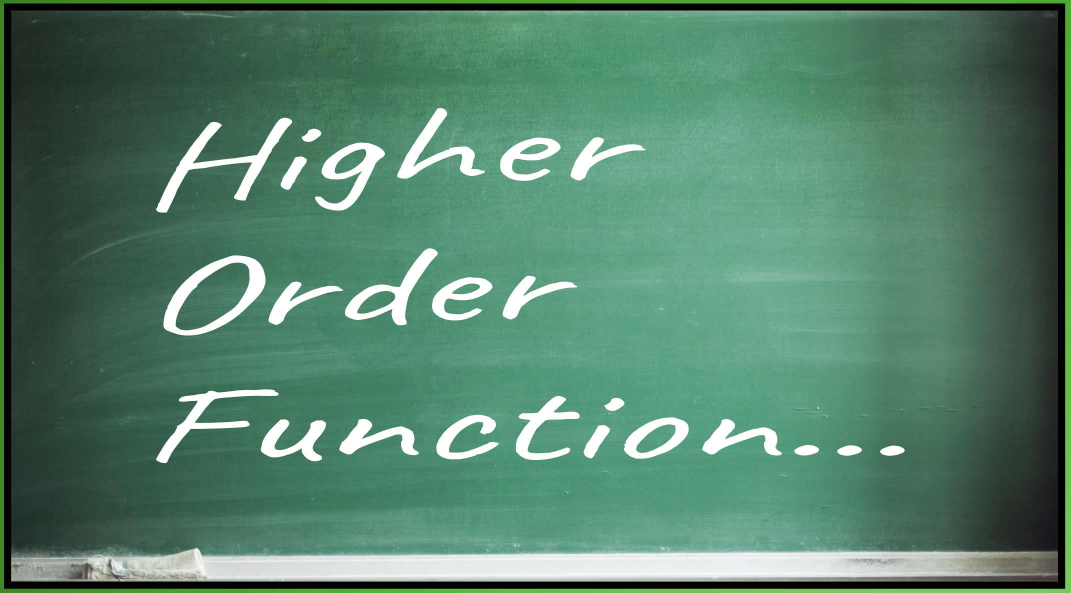
Object Oriented Programming

- Everything is an `object`
- Including Numbers and Booleans.
- `Objects` are Instantiated from `classes`
- …That combine state and `methods`
- `Methods` mutate the state
- `Methods` always belong to classes
- Variables hold `objects`
- `Methods` receive and return `objects`

Functional Programming

- `Functions` are first class values
- Variables can hold `functions`
- `Functions` can receive `functions` as parameters
- …which is called “Higher order `functions`”
- `Functions` can return `functions`
- It’s about Pure `functions`
- And `closures`
- And `Immutability`

Ah HA!



Functions that either <u>receive</u> or <u>return</u> another function