The OrderRecord Class

Now, we'll examine the class which converts CSV lines to records.

The main class that is used to compute results is `OrderRecord`. It's a direct representation of a line from a CSV file.

```
class OrderRecord
{
  public:
  // constructors...

  double CalcRecordPrice() const noexcept;
  bool CheckDate(const Date& start, const Date& end) const noexcept;

  private:
  Date mDate;
  std::string mCouponCode;
  double mUnitPrice{ 0.0 };
  double mDiscount{ 0.0 }; // 0... 1.0
  unsigned int mQuantity{ 0 };
};
```

The conversion

Once we have lines we can convert them one by one into objects:

```
[[nodiscard]] std::vector<OrderRecord>
LinesToRecords(const std::vector<std::string_view>& lines)
{
  std::vector<OrderRecord> outRecords;
  std::transform(lines.begin(), lines.end(),
                 std::back_inserter(outRecords), LineToRecord);

  return outRecords;
}
```

The code above is just a transformation, it uses `LineToRecord` to do the hard

The code above is just a transformation, it uses `LineToRecord` to do the hard work:

```
[[nodiscard]] OrderRecord LineToRecord(std::string_view sv)
{
  const auto cols = SplitString(sv, CSV_DELIM);
  if (cols.size() == static_cast<size_t>(OrderRecord::ENUM_LENGTH))
  {
    const auto unitPrice = TryConvert<double>(cols[OrderRecord::UNIT_PRICE]);
    const auto discount = TryConvert<double>(cols[OrderRecord::DISCOUNT]);
    const auto quantity = TryConvert<unsigned int>(cols[OrderRecord::QUANTITY]);

    if (unitPrice && discount && quantity)
    {
      return { Date(cols[OrderRecord::DATE]),
               std::string(cols[OrderRecord::COUPON]),
               *unitPrice,
               *discount,
               *quantity };
    }
  }
  throw std::runtime_error("Cannot convert Record from " + std::string(sv));
}
```

Firstly, the line is split into columns, and then we can process each column.

If all elements are converted, then we can build a record.

For conversions of the elements we're using a small utility based on `std::from_chars`:

```
template<typename T>
[[nodiscard]] std::optional<T> TryConvert(std::string_view sv) noexcept
{
  T value{ };
  const auto last = sv.data() + sv.size();
  const auto res = std::from_chars(sv.data(), last, value);
  if (res.ec == std::errc{} && res.ptr == last)
    return value;

  return std::nullopt;
}
```

`TryConvert` uses `std::from_chars` and returns a converted value if there are no errors. As you remember, to guarantee that all characters were parsed, we also have to check `res.ptr == last`. Otherwise, the conversion might return success for input like "123xxx".

All that's left is to compute the actual sum.