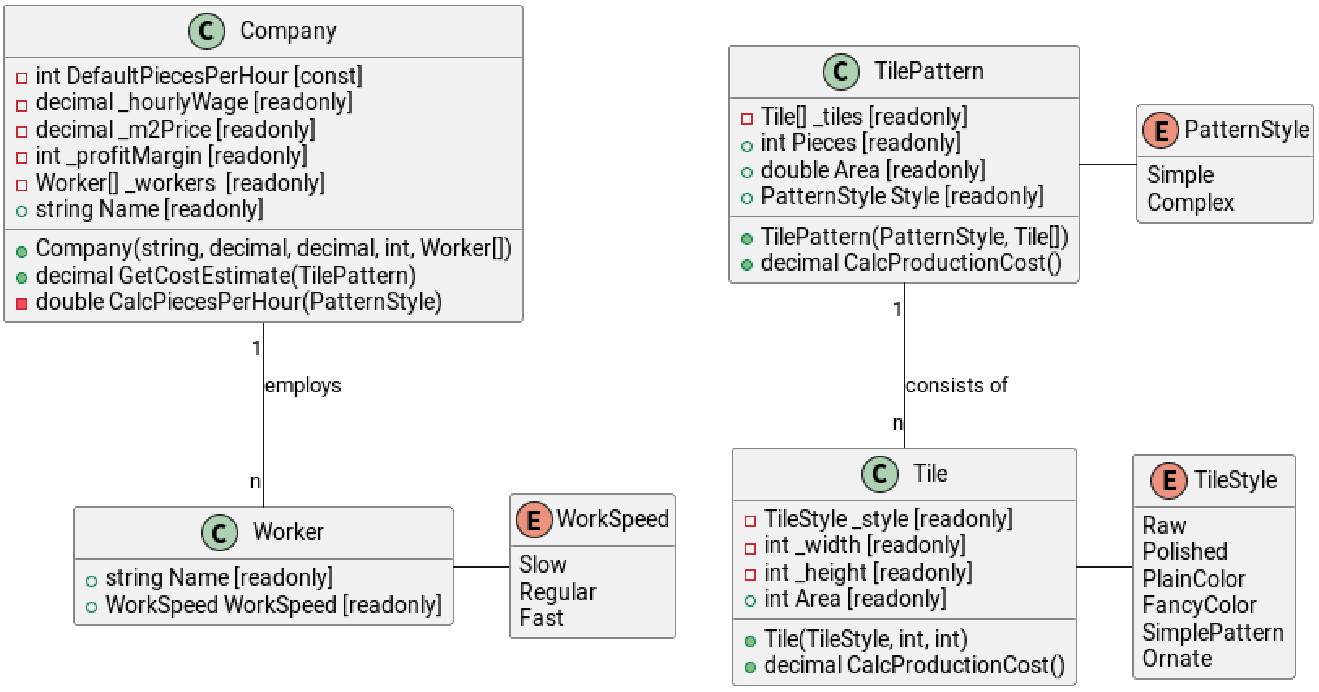
OO Relations — A Mosaic of Tiles

Table of Contents

In this exercise you will represent mosaic floor patterns as objects and have various companies calculate an offer for constructing them.

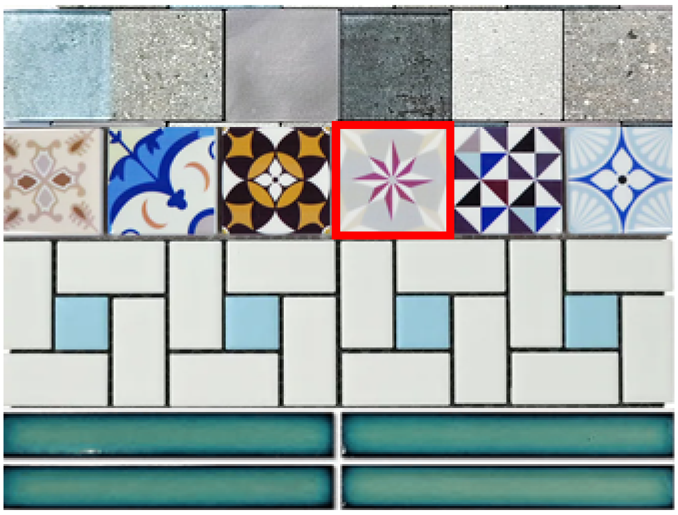

C Company
int DefaultPiecesPerHour [const]
decimal _hourlyWage [readonly]
decimal _m2Price [readonly]
int _profitMargin [readonly]
Worker[] _workers [readonly]
string Name [readonly]
Company(string, decimal, decimal, int, Worker[])
decimal GetCostEstimate(TilePattern)
double CalcPiecesPerHour(PatternStyle)
1
employs
n
C Worker
string Name [readonly]
WorkSpeed WorkSpeed [readonly]
E WorkSpeed
Slow
Regular
Fast
C TilePattern
Tile[] _tiles [readonly]
int Pieces [readonly]
double Area [readonly]
PatternStyle Style [readonly]
TilePattern(PatternStyle, Tile[])
decimal CalcProductionCost()
E PatternStyle
Simple
Complex
1
consists of
n
C Tile
TileStyle _style [readonly]
int _width [readonly]
int _height [readonly]
int Area [readonly]
Tile(TileStyle, int, int)
decimal CalcProductionCost()
E TileStyle
Raw
Polished
PlainColor
FancyColor
SimplePattern
Ornate

Some implementation as well as unit tests have been provided to you. Consider commenting some code to allow the application to compile until you have implemented all missing pieces. Also make sure to use the correct names and method signatures or the unit tests (and `Program`) won't work.

1. `Tile` class

A single mosaic tile in a pattern. It is defined by a specific style and a size (in mm). Based on this information it can calculate its own production cost.

Tiles can come in various styles with increasing production difficulty. For example a tile could have only a polished finish or a multicolored ornate pattern.



1.1. Production cost calculation

The following rules apply when calculating the production cost for a single tile.

- Each tile has a base price of 0.016€ per `cm^2`
- Size modification factors — very small or very large tiles are more expensive than a standard size
  - `a < 100 mm^2` ⇒ x1.5
  - `a < 400 mm^2` ⇒ x1.2
  - `400 \le a \le 2500` ⇒ x1
  - `a > 2500 mm^2` ⇒ x1.6
  - `a > 8100 mm^2` ⇒ x1.8
  - Only *one* of those modifiers is applied
    - So for size `50mm^2` it would be x1.5 and *not* x2.7
- Fancier tiles are more expensive to produce, so the production cost is increased by the following factors:
  - `Raw` ⇒ x0.8
  - `Polished` & `PlainColor` ⇒ x1
  - `FancyColor` ⇒ x1.1
  - `SimplePattern` ⇒ x1.25
  - `Ornate` ⇒ x2.3

Final calculation: `areaInCm^2 * pricePerCm^2 * sizeFactor * styleFactor`

2. `TilePattern` class

This class represents a combination of individual tiles in various shapes and colors to form a pattern. Cost estimates are always created by companies for a whole pattern as a single unit.

2.1. Implementation Hints

- The number of pieces equals the number of tiles contained in the pattern
- The area is the sum of the area of all tiles in `m^2`
    - We are ignoring the space the joints between the tiles would take
- The production cost is simply the sum of the production cost of all tiles
    - Pattern style does not change the production cost
        - It will make a difference when calculating the work cost later

It is possible to define numbers in scientific notation, e.g. `1E-6D`

3. `Company` class

Each instance of this class represents one company which specializes in laying mosaic floors.

- A company employees one or more `Worker`
- It demands a certain base price per `m^2` laid
- For *each* of the workers a certain hourly wage has to be paid
- All companies pay the same production cost for the tiles, but they put different amounts of a profit margin on top of that
    - e.g. 100€ production cost, 5% margin ⇒ 105€ total price for the material
    - We are ignoring costs for joints material etc.

4. `Worker` class

`Worker` is a simple, already provided class without any operations. It represents one employee of a Tile-laying company.

4.1. Pieces per hour (`Workspeed`)

Each worker can work either slowly, normal or fast:

- With regular speed 25 tiles can be laid down per hour.
- A fast worker is 20% faster
- A slow worker is 20% slower

It takes *twice* as long to work on a `Complex` pattern than on an easy one — the default speed assumes a `Simple` pattern.

On each job all workers are working together and at the same time. So the total amount of pieces laid per hour is based on the stats of all workers of the company, adjusted by their individual speed and modified if the pattern is complex.

4.2. Cost calculation

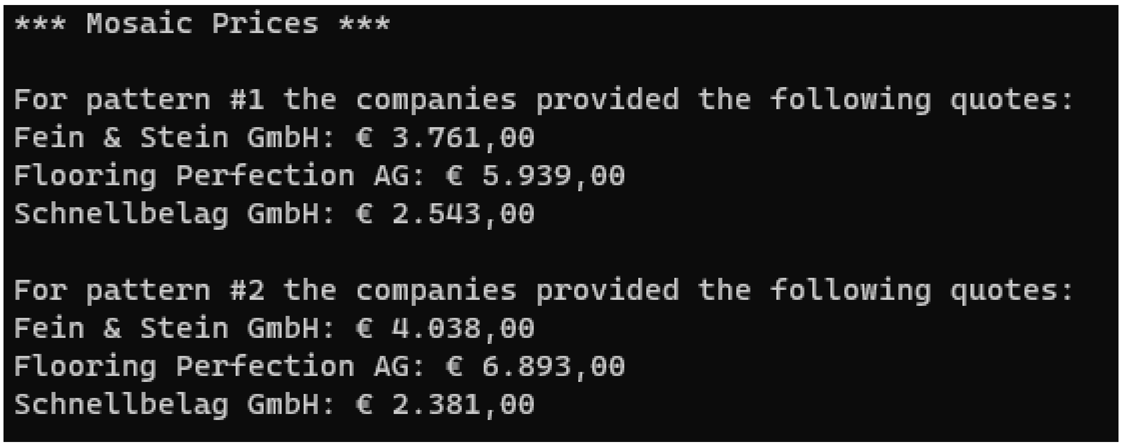
The total quote a company calculates for laying a mosaic is based on the following parts:

1. The base price which has to be paid for the total amount of `m^2` the mosaic covers
2. The work time
   - Time needed to construct the pattern, multiplied by hourly wage
3. The production cost of the tiles
   - Plus profit margin on top

The final amount is rounded *up* to the next whole number.

5. `Program` & Sample Run

The `Program` class is already complete. It will create some sample tile patterns and companies from which it requests cost estimates for those.



```
*** Mosaic Prices ***

For pattern #1 the companies provided the following quotes:
Fein & Stein GmbH: € 3.761,00
Flooring Perfection AG: € 5.939,00
Schnellbelag GmbH: € 2.543,00

For pattern #2 the companies provided the following quotes:
Fein & Stein GmbH: € 4.038,00
Flooring Perfection AG: € 6.893,00
Schnellbelag GmbH: € 2.381,00
```

6. Additional Feature: CSV-Import

If you have time and motivation left, add the following feature:

- Represent the data of `Worker`, `Tile`, `TilePattern` and `Company` in CSV-Format.
- Write a public static `Import` method for each of these classes.
  - Start with the basic building blocks: `Tile.Import(string[] csvLines)` and `Worker.Import(string[] csvLines)`
  - Then, implement the Import of `Company` and `TilePattern`. . Write Unit Tests for all four import methods! . You may (and should!) use AI to generate sample data for the CSV-Files. Ask the teacher for access to the internet, if necessary. . You should not use AI to generate the import code itself. It still has to be written by hand.

7. Grading

- Only TileTests ok: Range of Genügend
- in addition, TilePatternTests ok: Range of Befriedigend
- in addition, CompanyTests ok: Range of Gut
- in addition, Import implemented and tested: Range of Sehr gut

Deductions:

- Hand-in of obj or bin folders: One grade worse (Use .gitignore or `CleanSolutionDir.ps1` or delete them manually.)
- Gross violations of coding standard: might go down to genügend.
- Build failure, no working unit tests, mangled solution or project files: 49 (nicht genügend; verify your hand-in by extracting the ZIP file to a new folder and opening and building the solution)
- Empty hand-in or no visible work: 40 (nicht einmal nicht-genügend)