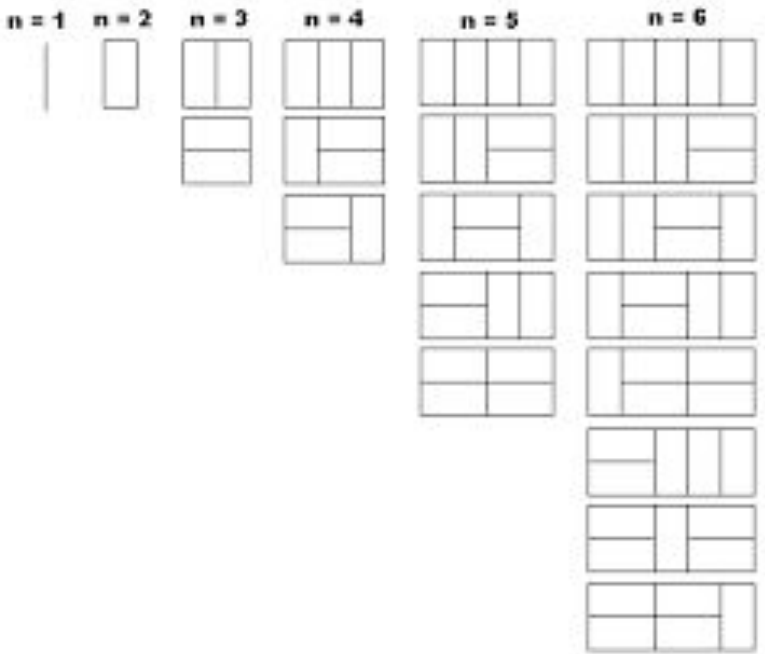
The “OO” Antipattern

Arthur O’Dwyer
2018-04-04

Counting unique domino tilings

```
// https://codereview.stackexchange.com/questions/188300
class DominoTilingCounter {
    // ...
};
```



Beginner code often looks like this

```
// https://codereview.stackexchange.com/questions/188300
class DominoTilingCounter {
    int height, width;
    bool done = false;
    int tilingCount;

    int countTilings(int h, int w, const std::string& prevRow, int rowIdx);

public:
    DominoTilingCounter(int h, int w) : height(h), width(w) {}

    int count() {
        if (!done) {
            tilingCount = countTilings(height, width, "", 0);
            done = true;
        }
        return tilingCount;
    }
};
```

Beginner code often looks like this

```
// https://codereview.stackexchange.com/questions/188300

class DominoTilingCounter {
    // ...
};

int main()
{
    DominoTilingCounter tc(4, 7);  // on a 4x7 grid

    std::cout << tc.count() << std::endl;
}
```

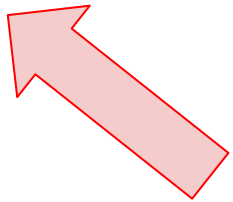
Beginner code often looks like this

```
// https://codereview.stackexchange.com/questions/188300

class DominoTilingCounter {
    // ...
};

int main()
{
    DominoTilingCounter tc(4, 7);  // on a 4x7 grid

    std::cout << tc.count() << std::endl;
}
```



Mutation!

Should I rewrite with C++11 gloss?

```
// https://codereview.stackexchange.com/questions/188300
class DominoTilingCounter {
    int height, width;
    mutable std::optional<int> tilingCount;

    int countTilings(int h, int w, const std::string& prevRow, int rowIdx);

public:
    DominoTilingCounter(int h, int w) : height(h), width(w) {}

    int count() const {
        if (!tilingCount.has_value()) {
            tilingCount = countTilings(height, width, "", 0);
        }
        return *tilingCount;
    }
};
```

Spoiler alert:
No.

How many times are you going to call `tc.count()`, anyway?

- If “frequently more than once per `tc` object,” then okay, good idea to memoize the answer in `tilingCount`.
- If “frequently zero times per `tc` object,” then I’d say you have a logic error in your program.
- The important thing is: when you construct a `DominoTilingCounter` object, it is *specifically for the purpose* of computing `tc.count()`, right?

How many times are you going to call `tc.count()`, anyway?

- If “frequently more than once per `tc` object,” then okay, good idea to memoize the answer in `tilingCount`.
- If “frequently zero times per `tc` object,” then I’d say you have a logic error in your program.
- The important thing is: when you **construct** a `DominoTilingCounter` object, it is *specifically for the purpose* of **computing** `tc.count()`, right?
- So put the **computation** in the **constructor**.
- Eliminate the “empty, uncomputed” state from your program.

Refactor = work better with const

```
// https://codereview.stackexchange.com/questions/188300
class DominoTilingCounter {
    int height, width;
    int tilingCount;

    int countTilings(int h, int w, const std::string& prevRow, int rowIdx);

public:
    DominoTilingCounter(int h, int w) : height(h), width(w) {
        tilingCount = countTilings(height, width, "", 0);
    }

    int count() const {
        return tilingCount;
    }
};
```

What else could we fix about this code?

Refactor = work better with const

```
// https://codereview.stackexchange.com/questions/188300
class DominoTilingCounter {
    int height, width;
    int tilingCount;

    int countTilings(int h, int w, const std::string& prevRow, int rowIdx);

public:
    DominoTilingCounter(int h, int w) : height(h), width(w) {
        tilingCount = countTilings(height, width, "", 0);
    }

    int count() const {
        return tilingCount;
    }
};
```

Our `height` and `width` member variables are unused!

Refactor = eliminate dead state

```
// https://codereview.stackexchange.com/questions/188300
class DominoTilingCounter {
    int tilingCount;

    static int countTilings(int h, int w,
                            const std::string& prevRow, int rowIdx);

public:
    DominoTilingCounter(int h, int w) {
        tilingCount = countTilings(h, w, "", 0);
    }

    int count() const {
        return tilingCount;
    }
};
```

What else could we fix about this code?

Avoid classes tantamount to `int`

```
// https://codereview.stackexchange.com/questions/188300
class DominoTilingCounter {
    int tilingCount;

    static int countTilings(int h, int w,
                            const std::string& prevRow, int rowIdx);

public:
    DominoTilingCounter(int h, int w) {
        tilingCount = countTilings(h, w, "", 0);
    }

    int count() const {
        return tilingCount;
    }
};
```

This class should probably be just a free function returning `int`.

Avoid classes tantamount to `int`

```
// https://codereview.stackexchange.com/questions/188300

static int countTilings(int h, int w, const std::string& prevRow, int rowIdx)
{
    // actual logic goes here
}

int countDominoTilings(int h, int w)
{
    return countTilings(h, w, "", 0);
}

int main()
{
    std::cout << countDominoTilings(7, 4) << std::endl;
}
```

Questions?