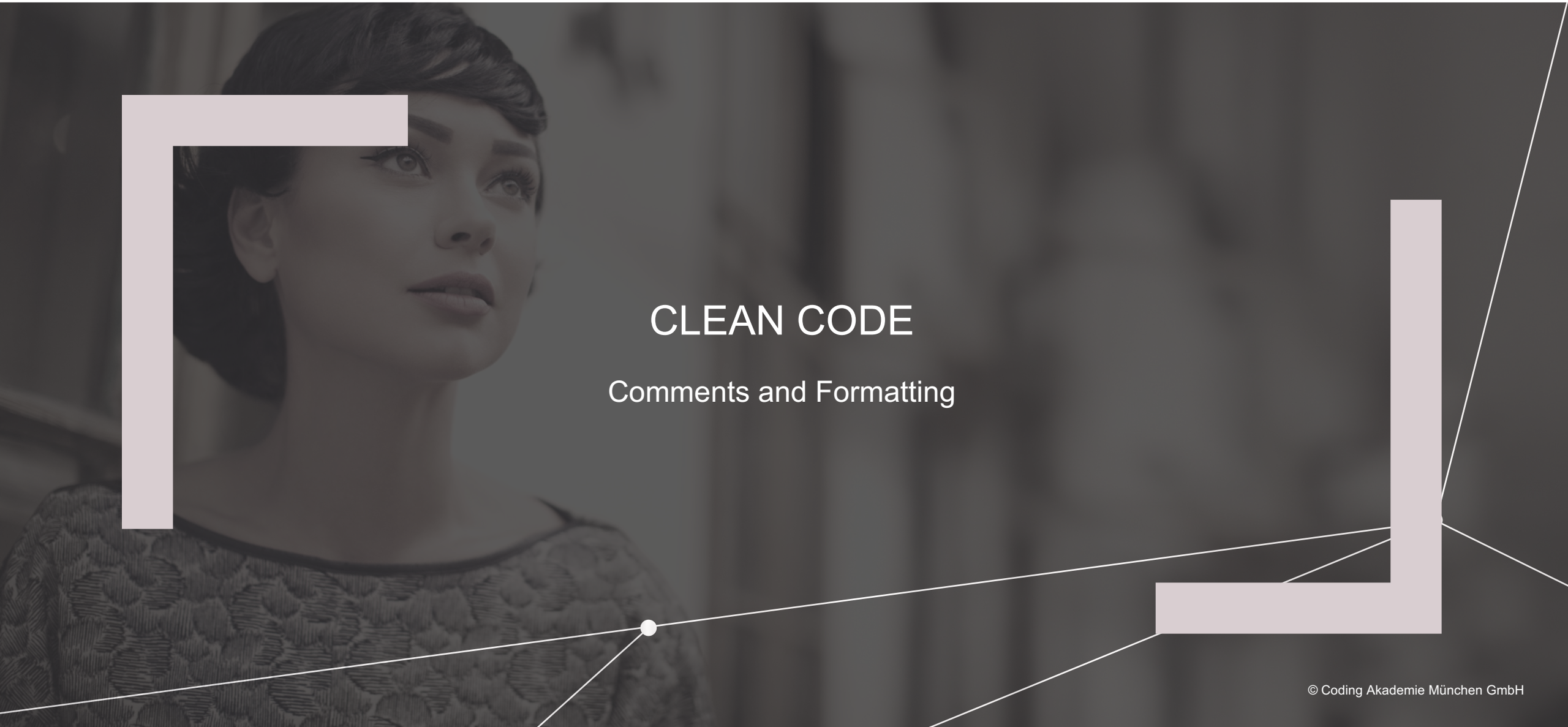
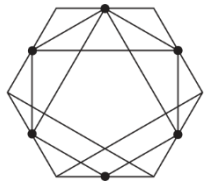
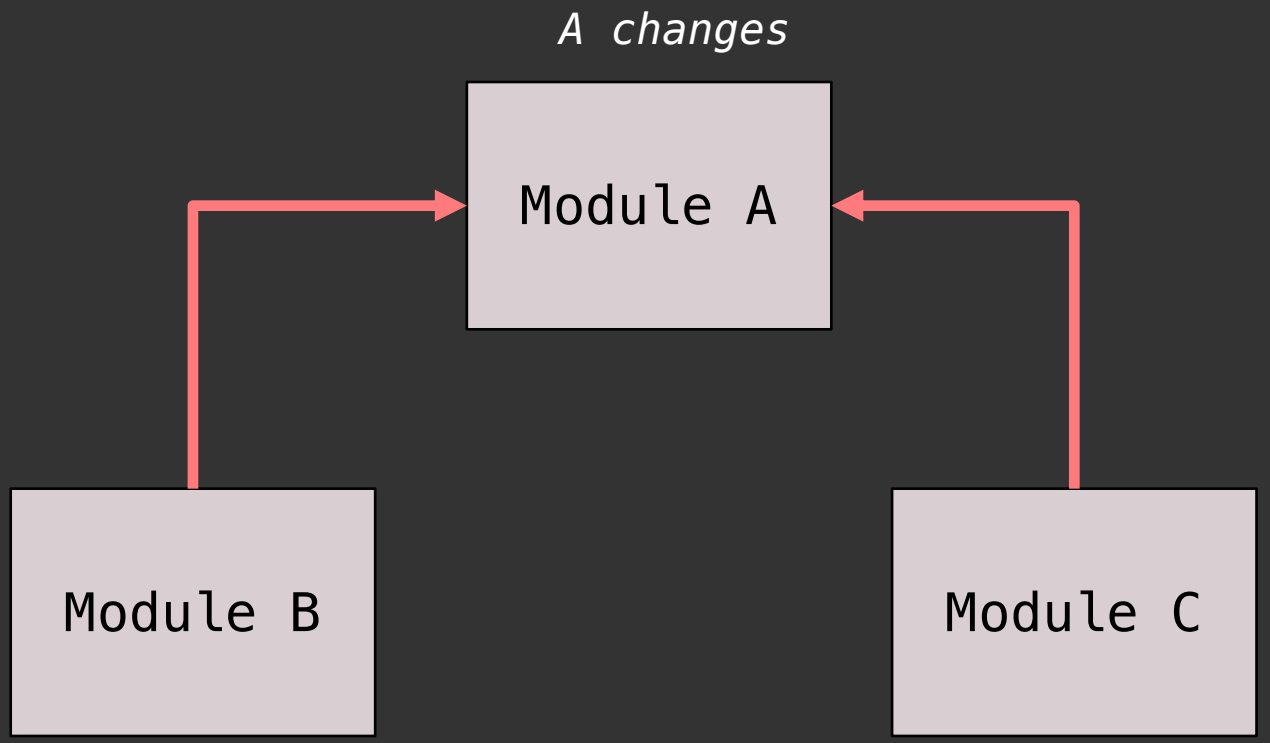


CLEAN CODE

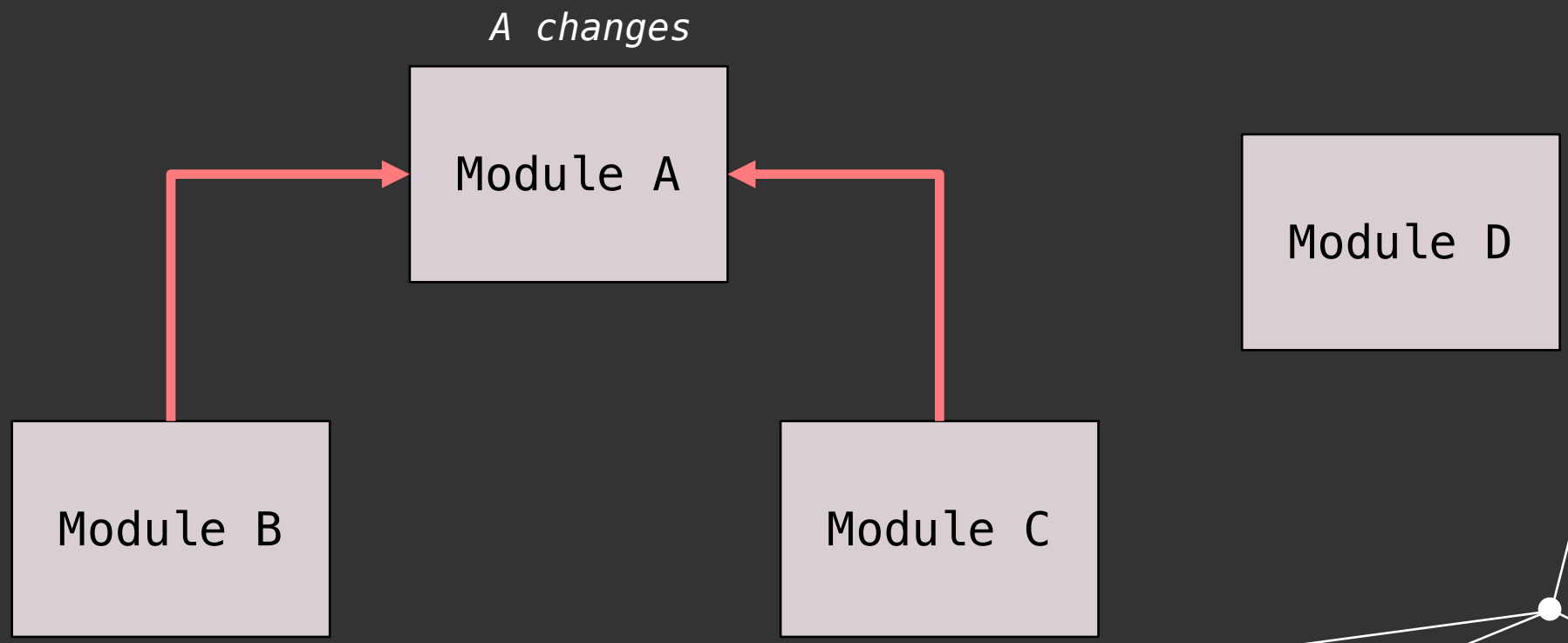
Comments and Formatting

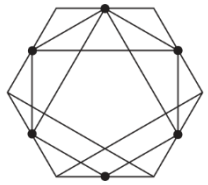


RIGIDITY



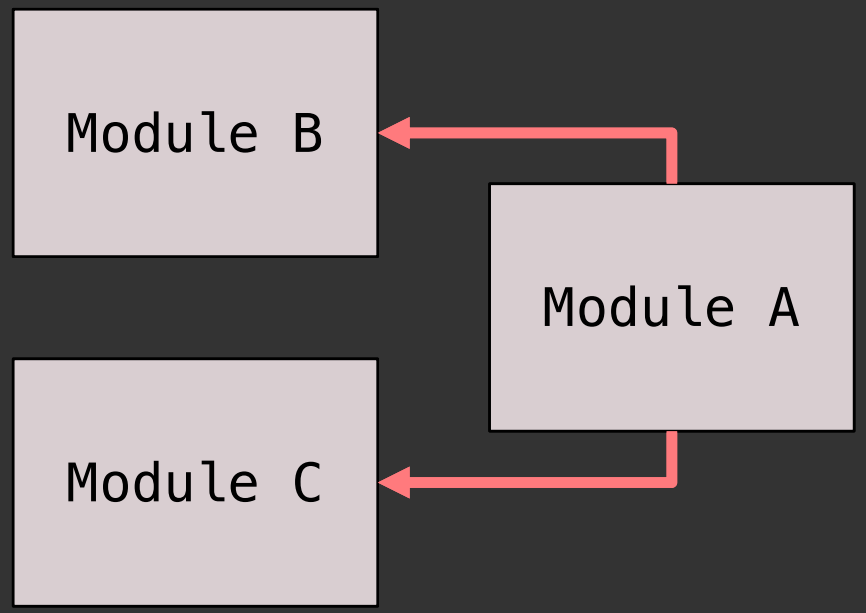
FRAGILITY



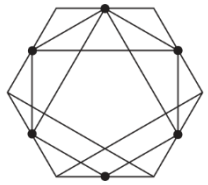


IMMOBILITY

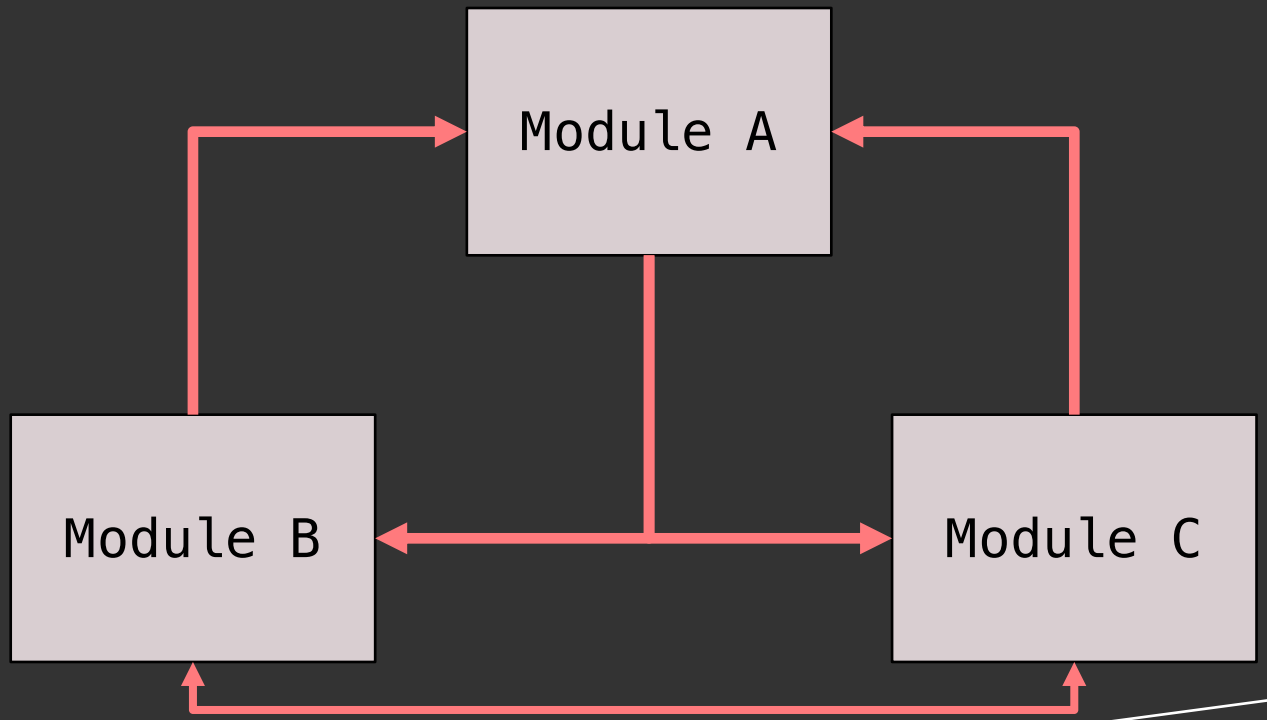
- PROJECT X

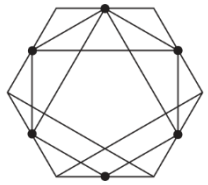


- PROJECT Y



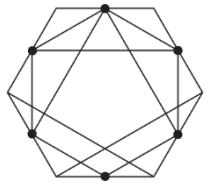
OPACITY





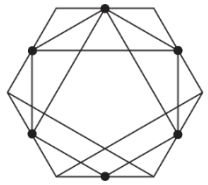
COMMENTS

- Comments compensate for our failure to express ourselves in code
- Whenever you want to write a comment, check that you cannot do it in code
- If possible, express yourself in code, not comments!



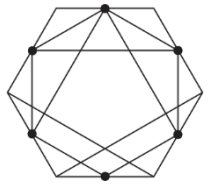
HOW COMMENTS FAIL

- Comments are difficult to maintain
- Therefore they often lie
  - Don’t get changed when code gets updated
  - Don’t get moved when code gets moved

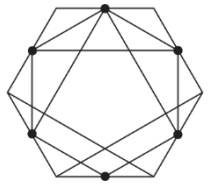


```
// Check to see if the employee is eligible for full benefits
if ((employee.flags & HOURLY_FLAG) &&
    (employee.age > 65))

    …
```

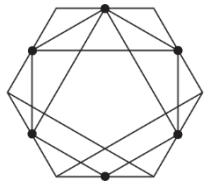


```
if (employee.isEligibleForFullBenefits())
```



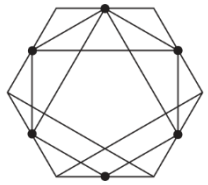
GOOD COMMENTS

- Required legal comments
- Sometimes to explain concepts that cannot be expressed in code
- Comments explaining the intent of the code
- Comments explaining code that you cannot clean up (e.g., a published interface)
- Comments documenting published interfaces (e.g, doxygen)
- TODO comments (if used sparingly)
- Comments for amplification (“This is very important, because…”)



BAD COMMENTS 1

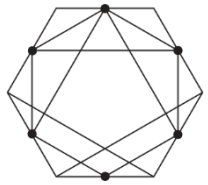
- Unclear comments (mumbling)
- Redundant comments (takes longer to read than the code without being clearer)
- Misleading comments
- Mandated comments
- Journal comments (history of the file)
- Noise comments
- Position comments



MISLEADING COMMENTS

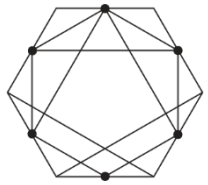
```
// Method returns when this.closed becomes true. (Java)

public synchronized void waitForClose(final long timeoutMillis) throws Exception
{
    if(!closed)
    {
        wait(timeoutMillis);
        if(!closed)
            throw new Exception("MockResponseSender could not be closed");
     }
}
```



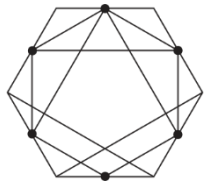
BAD COMMENTS 1

- Attributions and bylines
- Commented-out code
- HTML comments
- Nonlocal information
- Too much information (e.g., a history of the algorithm)
- Inobvious connection
- Function headers



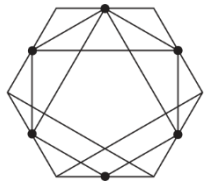
COMMENTED-OUT CODE

- Has a tendency to never get deleted
- Unclear why it is there: was it meant to be deleted or commented back in?
- Rely on the source control system and delete the code



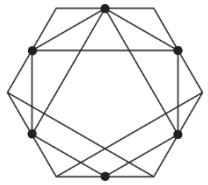
EXPLANATORY VARIABLES

```
// Multiply the seconds in a day times the days of work.
int duration_in_seconds{60 * 60 * 24 * days_of_work};
```



EXPLANATORY VARIABLES

```
int seconds_per_day{ 60 * 60 * 24 };
int duration_in_seconds{seconds_per_day * days_of_work};
```



CODE FORMATTING

- Use vertical distance to express conceptual affinity
- Keep the conventional order for placing instance variables
- Keep related functions close together
- Be consistent
- Use an automated code formatter