Comments

Chapter - 4

Mohamed Abdul

CS-5800

Disclaimer

- Presentation uses or contains materials including verbatim sentences and examples taken from the Clean Code textbook.

Agenda

- Comments are always good?
- Do we need to avoid comments all the time?
- Good Comments
  - Example of Informative Comments
  - Example of Explanation of Intent
- Bad Comments
  - Example of Redundant Comments
  - Example of Noise Comments
  - Example of Closing Brace Comments
  - Example of Attributes and Bylines
- Refactoring the code
- Summary

Comments are always good?

- Not all the time, in fact comments are needed because the code is unable to express the intent (Common reason).
- If we can express the intent through the code, we do not need comments.
- Older comments can be misleading, because the code evolves due to constant changes in the business requirements or security upgrades.
- The comments with unimportant information's could clutter up the code base.

“are not ‘pure gold’”

Do we need to avoid comments all the time?

- Not all the time, but we should try to minimize the usage of the comments.
- When the urge of comments are raised, instead try your best to refactor the code to express themselves.
- Sometimes comments are needed or helpful because of corporate standards or its useful to describe the process
- But keep in my, its still better to minimize the comment usage.

“Clear and expressive code with few comments is far superior to cluttered and complex code with lots of comments.”

Good Comments

- Comments are required or beneficial sometimes, the following reason why we might want to use comments.
  - **Legal Comments**
    - To comply with corporate standards
  - **Informative Comments**
    - To provide basic information about code
  - **Explanation of Intent**
    - To explain an intent behind a decision (ex. return value)
  - **Clarification**
    - It is useful to translate some return value into a readable text
  - **Warning of Consequence**
    - Sometimes its useful to warn programmers about piece of code.

Good Comments

```
// Returns minimum eligible age for membership
public int getMinEligibleAge(){

       return 18;
}
```

Informative Comments

Good Comments

```
public int getProcessCodeByType(String type){

         if (PROCESS_TYPE_A == type){
                 return CODE_A;
         }

        // invalid type was passed so returning invalid code
        return -1;
}
```

Explanation of Intent

Good Comments

```
// this test modifies the person age
@Test
public void testInvalidPersonAge(){
                    …
          this.person.setAge(61);
                    ….
          assertFalse(MemberValidator.validate(person));
}
```

Warning of consequence

Bad Comments

- Unfortunately, most of the comments are bad.
  - **Redundant Comments**
    - If code explains better, then why should we comment it?
  - **Noise Comments**
    - The comments which do not provide new information.
  - **Closing Brace Comments**
    - Putting comments in closing braces in deeply nested code
  - **Commented-Out-Code**
    - Commented out code misleads other coders that the comments are there for reasons.
  - **Attributes and Bylines**
    - Adding comment about who added a certain code. Version control systems are good at tracking the changes made by anyone, so we can avoid this.

Bad Comments

```
// the following code checks if the memory has exceeded, if its
// then returns true otherwise returns false indicating that
// memory has not exceeded
public boolean isMemoryExceedAllocatedMemory() {

        if(this.usedMemory > ALLOCATED_MEMORY){
                return true;
        }

        return false;
}
```

Redundant Comments

Bad Comments

```
class Person{

          /** name of person **/
          private String name;

          /** age of person **/
          private int age;

          /** Default constructor**/
          public Person(){..}

          /** Parametrized constructor **/
          public Person(String name, int age){..}

          /** returns the age of a person **/
          public int getAge(){..}

          /** returns the name of a person **/
          public String getName(){..}
}
```

Noise Comments

Bad Comments

```
while(memory.isAvailable()){

            int processId = createProcess();

            if (isValidProcessId(processId){

                        int subProcessId = createSubProcess(processId);

                        if(IsValidProcessId(subProcessId){
                                    System.out.print(“Successful”);
                        } // if
                        else
                        {
                                    throw new Exception(“Failed …”)
                        }// else

            } // if
            else
            {
                        throw new Exception(“Failed …”)
            } // else
} // while
```

Closing Brace Comments

Bad Comments

```
class Member {
        …
        /** Added by Mohamed as part of new policy upgrade **/
        private String creditCardInfo;
}
```

Attributes and Bylines

Refactoring

```
Person person = new Person(“John”, 25);


// in order for a person to be eligible for
// membership, they must be older than 18 and
// less than 60 and join date has to
if (person.getAge() > 18 && person.getAge() < 60) {

       // business logic
 }
```

Refactoring the code

```
Person person = new Person(“John”, 25);


// in order for a person to be eligible for
// membership, they must be older than 18 and
// less than 60
if (person.getAge() > 18 && person.getAge() < 60)
{

         // business logic
}
```

```
Person person = new Person(“John”, 25);


if (person.isEligibleForMembership()) {

     // business logic …
 }
```

Summary

- We should always try to minimize the usage of comments
- If code is good enough of to express the intent, avoid comments
- Try to refactor the code to express themselves instead of comments
- Use comments unless it's required by corporate standards or its beneficial to the reader.

Thank you!