Clean Code CH 2,3

POSTECH CSE Dongwon Kim

Meaningful name(1)

- Use intention revealing name – “change if you find better one”
  - negative example) Int d
  - Class를 따로 만들어 추상화 – method에 적절한 이름 부여.

```
public List<int[]> getFlaggedCells() {
  List<int[]> flaggedCells = new ArrayList<int[]>();
  for (int[] cell : gameBoard)
    if (cell[STATUS_VALUE] == FLAGGED)
      flaggedCells.add(cell);
  return flaggedCells;
}
```

```
public List<Cell> getFlaggedCells() {
  List<Cell> flaggedCells = new ArrayList<Cell>();
  for (Cell cell : gameBoard)
    if (cell.isFlagged())
      flaggedCells.add(cell);
  return flaggedCells;
}
```

Meaningful name(2)

- Avoid disinformation – Do not use word that have multiple meaning
  - Ex) names of system platform/variant
  - Ex) names of data structure – List/vector/array/set .. Etc
- Avoid Long & similar name – Modern editor has code completion
- Don’t use l or O(I vs l, 0 vs O) – because of this, I use special font for coding(Consolas or NanumGothicCoding)

Meaningful name(3)

- Avoid noise word – Is it really informative?
  - Ex) a1, a2, a3…
  - Ex) nameString – nameInt나 nameFloat도 있나..?
  -
```
getActiveAccount();
getActiveAccounts();
getActiveAccountInfo();
```

```
cyclomaticVisitor treeVisitor;
MethodDeclaration methodRepresent;
CompilationUnit classRepresent;
String classSource_;
char[] classSource;
```

Meaningful name(4)

- Pronounceable name – Working as a team

- Searchable name – scope에 비례해서 더 길고 상세한 이름
  - Use short/single letter name for noly short scope code

- Avoid encoding – type 같은 정보를 이름에 포함시키는..
  - IDE의 발전으로 인해 사실 이런 encoding은 거의 불필요하다 보임

Meaningful name(5)

- Avoid mental mapping - 모두가 이해할 수 있는 이름으로..
- Clarity is King
  - Use noun for class name / Don’t use processor, data, manager..(rule2)
  - Use verb for method name/ prefixed with is, get, set
  - 생성자 오버로딩의 경우.. 같은 기능을 하는 Static factory method를 만들고 오버로드 된 생성자는 private으로 두자.

```
Complex fulcrumPoint = Complex.FromRealNumber(23.0);
```

is generally better than

```
Complex fulcrumPoint = new Complex(23.0);
```

```
input wire [`WORD_SIZE-1:0] addr_I,
input wire [`WORD_SIZE-1:0] addr_D,
input wire read_I,
input wire read_D,
input wire write_D,

input wire update,

output reg read_MI,
output reg [`WORD_SIZE-1:0] addr_MI,
input reg [`WORD_SIZE-1:0] data_MI,

output reg read_MD,
output reg write_MD,
output reg [`WORD_SIZE-1:0] addr_MD,
inout wire [`WORD_SIZE-1:0] data_MD,

output reg read_MI,
output reg [`WORD_SIZE-1:0] addr_MI,
input wire [`WORD_SIZE-1:0] data_MI,

output wire [`WORD_SIZE-1:0] data_I,
inout wire [`WORD_SIZE-1:0] data_D,
output reg hit_I,
output reg hit_D,
```

Code from Architecture lab 6

Avoid mental mapping!

당시 팀메이트가 코드를 읽기 굉장히 힘들어했다.

Meaningful name(6)

- 이름으로 장난치지 말기

- 어떤 단어를 사용했다면, 그 개념을 위해 계속 사용하기
  - Controller vs manager vs driver – 주석 없이도 이해 가능해야 함

- Use solution/problem domain name –design pattern의 이름을 삽입한다던가..

```
cyclomaticVisitor treeVisitor;
```

```
MethodVisitor methodVisitor = new MethodVisitor();
```

Meaningful name(7)

- Add meaningful context

  Code from network lab6..

  This is code in main(){} ..

```
if(ip_hdr->ip_p == IPPROTO_TCP){
  tcp_num ++;
  //count app
  struct tcphdr *tcp_hdr = (struct tcphdr *)(pac_data +ip_hdr->ip_hl * 4);
  int dst_tcp = ntohs(tcp_hdr->dest);
  int src_tcp = ntohs(tcp_hdr->source);
  if(dst_tcp == 20 || dst_tcp == 21 || src_tcp == 20 || src_tcp ==21){
    ftp_num ++;
  }else if(dst_tcp == 22 || src_tcp == 22){
    ssh_num ++;
  }else if(dst_tcp == 53 || src_tcp == 53){
    dns_num ++;
  }else if(dst_tcp == 80 || src_tcp == 80){
    http_num ++;
  }
}
```

- But don’t add too much.. There is code completion of IDE !
  - Ex) 클래스 이름을 method 이름 앞에 일일이 붙인다던가..
  - Short name is better! (while it is clear)

Functions(1)

- Use method extraction! – 코드 가독성 대폭 증가.

```
174     * mm_realloc - Implemented simply in terms of mm_malloc and mm_free
175     */
176    void *mm_realloc(void *ptr, size_t size)
177    {…
333    }
```

Code form CS_APP malloc assignment
160 line
Filled with so many duplicated code

- Function should be SHORT (shorter is better)

Function(2)

- If/else/while 등의 block : should be one line long, 하나의 function call로 구성
- Function 의 indent level도 2를 넘어서는 안된다.
- Function은 단 하나의 동작만을 수행한다.
  - Method extraction등을 통해 하나의 동작으로 추상화하자.
- Function마다 하나의 abstraction level을 유지.
  - High abstracted function과 low abstraction function을 섞어쓰지 말자

```
295         treeV.addDoubleClickListener(new IDoubleClickListener() {
296             @Override
297             public void doubleClick(DoubleClickEvent event) {
298                 // TODO Auto-generated method stub
299                 IStructuredSelection target = (IStructuredSelection) event.getSelection();
300                 componentModel targetObject = (componentModel) target.getFirstElement();
301                 if(targetObject == null)
302                     return;
303                 if(targetObject instanceof fieldComp ||
304                    targetObject instanceof methodComp) {
305                     ICompilationUnit icu = ((sourceComp)targetObject.parent).classFile;
306                     ISourceRange range = null;
307                     if(targetObject instanceof fieldComp) {
308                         try {
309                             range = ((fieldComp)targetObject).field.getSourceRange();
310                         } catch (JavaModelException e1) {
311                             // TODO Auto-generated catch block
312                             e1.printStackTrace();
313                         }
```

Code from SD homework – eclipse plugin

Function(3)

- Top to Bottom: stepdown rule
  - 코드를 계층적인 설명으로 표현하고 이 단계에 따라 추상화 레벨 결정
- Switch statement : avoid long switch statement -> use factory pattern

Listing 3-5
Employee and Factory

```
public abstract class Employee {
  public abstract boolean isPayday();
  public abstract Money calculatePay();
  public abstract void deliverPay(Money pay);
}
-----------------
public interface EmployeeFactory {
  public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType;
}
-----------------
public class EmployeeFactoryImpl implements EmployeeFactory {
  public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType {
    switch (r.type) {
      case COMMISSIONED:
        return new CommissionedEmployee(r) ;
      case HOURLY:
        return new HourlyEmployee(r);
      case SALARIED:
        return new SalariedEmploye(r);
      default:
        throw new InvalidEmployeeType(r.type);
    }
  }
```

이후 새로운 value를 계산하고 싶다거나.. 타입이 추가되는 경우에도 괜찮다..

확장성 증가

Function(4)

- Descriptive names. Don’t afraid long name.
  - 이름만 보고 동작을 예상할 수 있도록 해라
  - SetupTeardownIncluder.render > testableHtml
- Function argument – argument의 수를 2아래로 해라.
- Monadic form
  - Argument를 변형한다면 그것을 return값으로 해라. – pointer 받아서 return 값 없이 수정하고.. -> bad
  - Event – 이름을 신중하게 정할 것
  - Argument를 output으로 사용하지 말자.

Function(5)

- Flag argument를 절대로 사용하지 말 것. – 함수는 하나의 일만을 해야 한다. – 나눌 것

- Dyadic function – arg 사이에 natural ordering이 있는 경우에만 사용하는 것이 권장
  - 아닌 경우에는 Monadic으로 바꾸자 – class를 변형하여 가능

- Triad function – think very carefully before creating it

Function(6)

- Argument가 많을 경우에는 class로 wrappin하거나/list를 사용하도록 하자.

- Argument name이 function name과 대응되도록 하자.
  - Write(Name) vs writeField(name)
  - assertExpectedEqualsActual(expected, actual)

- Side effect가 없도록 하자
  - Function은 하나만 해야함.
  - Temporal coupling이 생긴다. – side effect에 의한 coupling

Function(7)

- Split query and command
  - Function은 대답하거나/동작 수행 둘 중 하나만을 해야 함.
  - public boolean set(String attribute, String value); -> bad
  - attributeExists("username“)과 setAttribute("username", "unclebob"); 로 split.
- Error code보다는 exception handling을 사용하자.
  - 여러 error를 flow control 없이 한번에 handle가능.
  - Try와 catch의 body를 따로 metho로 extrac하는 것이 좋다. – 1func, 1 thing
  - Error code는 dependency를 만든다.

```
try {
  deletePage(page);
  registry.deleteReference(page.name);
  configKeys.deleteKey(page.name.makeKey());
}
catch (Exception e) {
  logger.log(e.getMessage());
}
```

Function(8)

- 반복하지 말 것.
- 처음에 작성한 함수를 분석하여 나누고, 정돈할 것.
- The art of programming is the art of language design.