More informal refactoring

Breaking bad habits

Programming

- Evolves, and advice is often wrong or out of date.

- Here is some cases of traditional advice which is now considered bad practice.

Evil -- **Comment Your Code**

```
for (Person person : people) {
// send cheque to persons old enough
// to qualify for pension payout
if (person.getAge() >= 65) {
sendCheque(person.getAddress());
}}
```

Communicating intent is essential – but this is not how we do it!

Comment as Method

```
for (Person person : people) {
if (oldEnoughForPensionPayout(person)) {
        sendCheque(person.getAddress());
}
}
...
boolean oldEnoughForPensionPayout(Person p) {
        return p.getAge() >= 65;
}
```

Comment as Local Variable

```
for (Person person : people) {
boolean oldEnoughForPensionPayout =
        person.getAge() >= 65;
if (oldEnoughForPensionPayout) {
        sendCheque(person.getAddress());
}
}
```

Comment as Constant

```
private static final int
        MIN_AGE_FOR_PENSION_PAYOUT = 65;
for (Person person : people) {
        if (person.getAge() >=
                MIN_AGE_FOR_PENSION_PAYOUT) {
        sendCheque(person.getAddress());
}
}
```

Evil -- **Explain APIs/Interfaces**

```
/**
* <p><b>Usage:</b></p>
*
* <pre>
* PersonRepository exampleRepo = makeRepository(bobJones, joelStevens,shirleySmith);
*
* List<Person> persons = exampleRepo.findPeople("jo");
*
* persons.get(0); // bobJones
* persons.get(1); // joelStevens
* </pre>
*
* @author Doug
*/

interface PersonRepository {
List<Person> findPeople(String text);
...
}
```

Explain APIs/Interfaces

```
@Test
public void testFindPeopleByFirstAndLastName() {
        Person bobJones = testPerson("Bob Jones");
        Person joelStevens = testPerson("Joel Stevens");
        Person shirleySmith = testPerson("Shirley Smith");
        PersonRepository exampleRepo = makeRepository(
        bobJones, joelStevens, shirleySmith);
        List<Person> persons = exampleRepo.findPeople("jo");
        assertEquals(bobJones, persons.get(0));
        assertEquals(joelStevens, persons.get(1);
}
```

Evil – **Don’t Waste Cycles**

```
void calcAverages2() {
        long ageSum = 0;
        long heightSum = 0;
        for (Person person : people) {
                ageSum += person.getAge();
                heightSum += person.getHeight();
        }
        double averageAge = ageSum / people.size();
        double averageHeight = heightSum / people.size();
...
}
```

Don’t Waste Cycles

```
void calcAverages1(List<Person> people) {
        double averageAge = averageAge(people);
        double averageHeight = averageHeight(people);
...
}
double averageAge(List<Person> people) {
        long ageSum = 0;
        for (Person person : people) {
                ageSum += person.getAge();
        }
        return (double) (ageSum / people.size());
}
double averageHeight(List<Person> people) { /* as above */ }
```

Don’t Waste Cycles

- Microbenchmark parameters:
  - 1,000,000 people
  - 1500 iteration warm-up
  - 10 runs (averaged)
- calcAverages1: 7237μs
- calcAverages2: 3640μs
- Roughly 100% slower (as expected)
- Only 3.6ms (imperceptible)

Don’t Waste Cycles

REMEMBER, in general ……

Designing for maintenance (solution 1)

Is preferred to

Designing for efficiency (solution 2)

Decide with Conditionals

- if, unless, else, ?:, etc.

- Implement conditional logic

- Handle errors

- Null checking

- switch, case, etc.

- exhaustively handle different cases

Conditionals make our code complex, complex code has errors. So we minimise complexity by minimising the volume of conditionals.

Avoiding conditionals

```
public List<String> parseNames(String names) {
        if (names == null) {
                return null;
        }
        return Arrays.asList(names.split(","));
}

void readNames() {
        String line = System.console().readLine();
        List<String> names = parseNames(line);
        if (names != null) {
                for (String name : names) {
                // ...
}
}
}
```

Null Object Pattern

```
public List<String> parseNames2(String names)
{
        if (names == null) {
                return Collections.emptyList();
        }
        return Arrays.asList(names.split(","));
}
```

Null Object Pattern

Again, simply – do not return null

```
public interface Animal {
        public void makeSound(); }
public class Dog implements Animal {
        public void makeSound() {
        System.out.println("woof!"); } }
public class NullAnimal implements Animal {
        public void makeSound() { } }
```

Worst Case option

Catch nulls as NullPointerExceptions

Unless null is part of the algorithm (from A=B)

```
@Override public int hashCode()  {
       int hash = 7;
       hash = 31 * hash ^ num;
       hash = 31 * hash ^ (null == data ? 0 : data.hashCode());
       return hash;  }
```

As seen before

Replace Conditional with Polymorphism

Decide with Conditionals

```
if (province == Province.MB
   && productType == ProductType.WIDGET) {
     tax = 0.05; // no PST on widgets in MB
  }  else if (province == Province.MB) {
      tax = 0.13;
  }  else if (province == Province.AB) {
      tax = 0.05;
  }  else if (...) {
             // and so on
  } else if (...) {
```

Replace Conditional with Map

```
static {
        taxByProvince.put(Province.MB, 0.13);
        taxByProvince.put(Province.AB, 0.05);
        // ... etc.
}

if (province == Province.MB
        && productType == ProductType.WIDGET) {
        tax = 0.05; // no PST on widgets in MB
        } else {
                tax = taxByProvince.get(province);
}
```

Avoid else

Can I remove part of the conditional?

Pseudo - Code

```
testFunc(expr) {
        if (expr) {
                ret = true
                        }
        else {
                ret = false
        }
 return ret
}
```

```
testFunc(expr) {
 ret = false
        if (expr) {
                ret = true
        }
        ~~else {~~
                ~~ret = false~~
        ~~}~~
 return ret
}
```

Pseudo - Code

```
testFunc(expr) {
 ret = false
        if (expr) {
                ret = true
        }
 return ret
}
```

```
testFunc(expr) {
        if (expr) {
                return true
        }
 return false
}
```

Basically

- Care some be taken when using
  - Comments
  - Assignment
  - Conditionals
  - Loops
  - ….
- But I was told programming was all about these items!