Effective Java

2 Chapter: Creating and destroying objects

1 Consider static factory methods instead of constructors

dont understand it much right now

3 Chapter: Methods common to all objects

4 Chapter

5 Chapter

6 Chapter

7 Chapter

42 Prefer lambdas to anonymous classes

This chapter shortly describes the lambdas and how it is a step up from the interfaces before. Now hey're called functional interfaces.

Lambdas works best when short. Ideal is 1 line, but 3 is max. Longer lambdas are clear sign that you should refactor your code.

Lambdas are prefered to anonymous classes, therefore more often used nowadays

43 Prefer method references to lambdas

Code like this:

```
map.merge(key, 1, Integer::sum);
```

is a great example of using method references, for the map if the key exists add 1 to the value if no then add key with value 1

Everything you can do with method reference you can do with lambdas
They're mainly used for shorter clearer code

Where method references are shorter and clearer, use them; where they aren't, stick with lambdas.

44 Favor the use of standard functional interfaces

Try to always use the standard methods from native java library or standard functional interfaces to make your code easier to understand.

Java lib java.util.Function consist of 43 interfaces. There are 6 basic of them:
Interface	Function Signature	Example
UnaryOperator	T apply(T t)	String::toLowerCase
Binary Operator	T apply(T t1, T t2)	BigInteger::add
Predicate	boolean test (T t)	

-
- UnaryOperator

8 Chapter: Methods

49 Item: Check parameters for viability

failure to validate parameters can result in violation of "failure atomicity". We must detect the errors from the parameters as soon as possible

it says that we should use the javadoc to describe possible errors thrown bny violating our api

for checkining for null we can use Objects.requireNonNull method

At non public methods, these that we are not exporting to the world we can use assertions in the code to ensure valid parameters

```
assert a != null;
```

they throw assertion errors

It is also super important to validate fields before construction of an object

Sometimes when methods throws an error it is different error then the one that caused the issue, in that case we can use 'exception translation idiom' 73 Item[##73 Item]

50 Make defensive copies when needed

you should always programm deffensively because someone even with good intensions may broke your methods.

It is GOOD PRACTICE to make defensive copies in constructor, to exclude any chances of mutating states of given parameters.

```
“public Period(Date start, Date end) {
    this.start = new Date(start.getTime());
    this.end   = new Date(end.getTime());”
}
```

in this example Date in java library is mutable so it would be possible to create Period object with start,end values and after that these values could have been changed

Also GOOD PRACTICE is to make defensive copies BEFORE validating parameters, this prevents from time-of-check/time-of-use TOCTOU attack. from another thread during the window of vulnerability

Also GOOD PRACTICE is to avoid using .clone() on parameter objects whose type can be subcassable by untrusted parties

Also GOOD PRACTICE is to modify accessors method to return defensive copies of mutable internal fields. like so

```
public Date start() {
    return new Date(start.getTime());
}
```

Try to use immutable objects whenever possible to not think about defensive copies

Defensive copies may be omitted when we trust the caller class, i.e when it in the same package

51 Design method signatures carefully

Choose method names carefully!!

GOOD PRACTICE is to keep signature with less than 4 parameters. to achieve that you can:

- break method into smaller pieces.
- create helpers methods (ussually as a staticmember class)
- adapt the BuilderPattern

GOOD PRACTICE is to use interfaces in signatures whenever possible

Prefer two-element enums to boolean parameters. Use bolleans only when it is 100 clear from the context to use it. Otherwise create enum type, its easier to read after, also you can add more options later

73 Item:

aasgasgasgasga

9 Chapter

10 Chapter

11 Chapter