Effective Java

Item 1: Consider static factory methods instead of constructors

- Use this:

```
public Integer valueOf(int value)
```

- Instead of this:

```
public Integer(int value)
```

Item 2: Consider a builder when faced with many constructor parameters

```
Request = Request.builder()
    .setMethod("GET")
    .setUri(Uri.create("/v1/person/foo"))
    .addHeader("Content-Type", "application/json")
    .setBodyGenerator(bodyGenerator)
    .build();
```

Item 4: Enforce noninstantiability with a private constructor

Example of getInstance() vs constructor

Item 7: Avoid finalizers *

* Never use finalizers!

Item 8: Obey the general contract when overriding equals

- Reflexive, symmetric, transitive, consistent
- Use `EquivalenceTester` from Platform

Item 9: Always override hashCode when you override equals

- `equal` objects must have equal hash codes
- Note: equal hash codes doesn't guarantee equal objects (hash collision is not an error)

Item 11: Override clone judiciously *

* Never implement clone!

Item 15: Minimize mutability

- Make all fields `final`
- Use atomic wrappers: `AtomicInteger`, `AtomicReference`, etc.

Item 16: Favor composition over inheritance

- This is the decorator pattern
- The Java I/O library uses this pattern
- Guava's forwarding classes are useful: `ForwardingList`, `ForwardingMap`, etc.

Item 21: Use function objects to represent strategies

- This is the strategy pattern
- The `Comparator` interface is an example
- Guava’s `Function` is a useful interface

Item 22: Favor static member classes over non-static

- Non-static member classes maintain a reference to the parent: this is almost never what you want!

Item 25: Prefer lists to arrays

Example

Item 30: Use enums instead of int constants

Example

Item 31: Use instance fields instead of ordinals

Example

Item 32: Use EnumSet instead of bit fields

Example

Item 36: Consistently use the Override annotation

Example

Item 38: Check parameters for validity

- **Use Guava** `Preconditions`: `checkNotNull`, `checkArgument`, `checkState`

Item 39: Make defensive copies when needed

- Copy collections using Guava’s immutable collections: `ImmutableList`, `ImmutableMap`, etc.

Item 43: Return empty arrays or collections, not nulls

Example

Item 45: Minimize the scope of local variables

Example

Item 46: Prefer for-each loops to traditional for loops

Example

Item 47: Know and use the libraries

- Always look in the JDK, Guava and Platform

Item 48: Avoid float and double if exact answers are required

- Never use floating point for money!
- Use a fixed-point representation
- Use `BigDecimal`

Item 49: Prefer primitive types to boxed primitives

- Unboxing a null boxed type produces `NullPointerException`

Item 50: Avoid strings where other types are more appropriate

Example?

Item 52: Refer to objects by their interfaces

- Use `List` or `Map` rather than `ArrayList` or `HashMap`

Item 55: Optimize judiciously

It is easier
To make a good program fast
Than it is
To make a fast program good.

Item 56: Adhere to generally accepted naming conventions

- Follow conventions from Guava and the JDK (especially Collections and other modern APIs)

Item 57: Use exceptions only for exceptional conditions

Example

Item 59: Avoid unnecessary use of checked exceptions

Example

Item 60: Favor the use of standard exceptions

Example

Item 63: Include failure-capture information in detail messages

- Exceptions should contain enough information to determine what went wrong
- You will thank yourself when you are reading log files at 3am

Item 65: Don't ignore exceptions

- Log ignored exceptions
- Log or re-throw, but never both!
- Use Guava's `Throwables.propagate`

Item 68: Prefer executors and tasks to threads *

* Never use `Thread` directly!

- Learn java.util.concurrent
- Brian Goetz: Java Concurrency in Practice

Item 69: Prefer concurrency utilities to wait and notify *

* Never use `wait` and `notify`!

Item 74: Implement Serializable judiciously *

* Never use Java serialization!