

Terminal

Setup

```
~/vimrc

set hlsearch
set incsearch
set number

set background=dark
color gruvbox
```

BASH

```
# delete swap files
$ find . -name "*.swp" -type f
$ find . -name "*.swp" -type f -delete
```

Testing

```
# always remember (to remove prior OUT):
$ rm -rf OUT
```

```
# Run with inputs and redirect output
$ java Ex2 < inputs/Ex2.1.in > OUT
```

```
# Check for differences
$ vim -d OUT outputs/Ex2.1.out
```

VIM

(Selected commands only)
· - repeat last command

Undo/Redo
u - undo — Ctrl + r - redo
Ctrl + u - Delete all newly added characters in current line (INSERT mode)

Navigation
^ - front of line — \$ - back of line

Insert
e - basically x then i
i - before cursor
I - beginning of line — A - end of line
o - new line below — O - new line above
Can also exit using Ctrl + c

Search (and Replace)
n - next occurrence
Replace: :%s/<search-phrase>/<replace-phrase>/options
Select words, replace all with confirmation: :%s/\<search\>/replace/gc

Registers, Deleting, Yanking
:reg - show registers (only lines are saved in the history)

Note that you can also do yw to yank a word
use above with :xp to paste contents of register x, eg :3p.
p - paste after cursor — P - paste before cursor

Code Samples

equals()

```
@Override
public boolean equals(Object o) {
    if (o == this) { return true; }
    if (o == null) { return false; }

    if (o instanceof Lazy<?>) {
        // IMPORTANT
        Lazy<?> o = (Lazy<?>) o;

        if (o.get() == null) {
            return this.get() == null;
        }

        return this.get().equals(o.get());
    }

    return false;
}
```

Monad example: Try

```
public abstract class Try<T> {
    // Constructors
    public static <T> Try<T> failure(Throwable a) {
        @SuppressWarnings("unchecked")
        Try<T> t = (Try<T>) new Failure(a);
        return t;
    }

    public static <T> Try<T> success(T value) {
        return new Success<T>(value);
    }

    public static <T> Try<T> of(Producer<? extends T> producer) {
        try {
            return success(producer.produce());
        } catch (Throwable e) {
            return failure(e);
        }
    }

    // Nested classes
    private static class Success<T> extends Try<T> { ... }
    private static class Failure<T> extends Try<T> { ... }
}
```

flatMap()

```
@Override
public <R> Try<R> flatMap(Transformer<? super T,
    ? extends Try<? extends R>> mapper) {
    @SuppressWarnings("unchecked")
    Try<R> t = (Try<R>) mapper.transform(value);
    return t;
}
```

Anonymous Class Example

```
Transformer <Integer, Integer> addThree =
    new Transformer<Integer, Integer>() {
        @Override
        public Integer transform(Integer t) { return t + 3; }
    };
```

Misc

```
// Most General Way:
private static final Maybe<?> NONE = new None();
```

Misc

Exceptions

Only important checked exception may be `java.util.NoSuchElementException`.
Otherwise, `java.lang.`, eg `java.lang.NullPointerException`.

File Start Order

```
/** ... */
package ...;

import ...;

public class MyClass<T> { ... }
```

Class Order

1. Fields
2. Constructors
3. **Factory** Methods
4. **Abstract** Methods
5. Methods
6. **Inner** Classes

Other Stuff

```
.map(String::valueOf)
Primitives
```

- byte <: short <: int <: long <: float <: double
- char <: int

Streams

API

Stream API (truncated)

Create Stream

```
<T> Stream<T> of(T ... values)

<T> Stream<T> iterate(T seed, UnaryOperator<T> f)

<T> Stream<T> iterate(T seed, Predicate<? super T> hasNext,
    UnaryOperator<T> f)
// Terminates when hasNext fails.

<T> Stream<T> generate(Supplier<T> s)
```

Modify Stream

```
Stream<T> filter(Predicate<? super T> predicate)

Stream<T> map(Function<? super T,? extends R> mapper)

<R> Stream<R> flatMap(Function<? super T,
    ? extends Stream<? extends R>> mapper)

void forEach(Consumer<? super T> action)

Stream<T> distinct()
// Based on Object.equals(Object o)

Stream<T> sorted()

Stream<T> sorted(Comparator<? super T> comparator)

Stream<T> peek(Consumer<? super T> action)
// runs action on elements and returns same stream

Stream<T> limit(long maxSize)

Stream<T> takeWhile(Predicate<? super T> predicate)

Stream<T> dropWhile(Predicate<? super T> predicate)
```

Terminal Operations

```
T reduce(T identity, BinaryOperator<T> accumulator)

<U> U reduce(U identity, BiFunction<U,? super T,U> accumulator,
    BinaryOperator<U> combiner)
// for all u, combiner(identity, u) is equal to u
// combiner.apply(u, accumulator.apply(identity, t))
// == accumulator.apply(u, t)

long count()

boolean allMatch(Predicate<? super T> predicate)

boolean anyMatch(Predicate<? super T> predicate)

boolean noneMatch(Predicate<? super T> predicate)
```

Excerpts

```
// _ to Stream
list.stream();
Stream.of(array);

// Stream to List
List<> ... = stream.collect(Collectors.toList());

// Print all
stream.forEach(System.out::println);

Default Functional Interfaces

Consumer<T>.accept()

Function<T,R>.apply()

Predicate<T>.test()

Supplier<T>.get()

UnaryOperator<T>.identity()
// operand and result same type

Comparator<T>.compare(T o1, T o2)
// o1 < o2 - negative
// o1 = o2 - 0
// o1 > o2 - positive
// (if o1 and o2 are related to numbers), return o1 - o2
```