CS2030S

Programming Methodology II

Lab 05

Lab 04

Lab 04

Act

Action< T >

```
void call(T t);
```

Actionable< T >

```
void act(Action<? ___ T> f);
```

> Fill in the rest
>
> - Fill in the blank for `___`
> - Keep in mind PECS
> - We will use `f` as `f.call(T)`
>
>   *(i.e., the input argument is of type **T**)*

Lab 04

Act

Probably< T >

```
public void act(Immutator<? ___ T> f) {
  :
}
```

Possible Implementation

```
  if (this.value == null) {
    return;
  }
  f.call(this.value); // can you rearrange this to be better?
```

Lab 04

Act
Immutate

Immutate

Immutator< R,P >

```
R invoke(P p);
```

Immutatorable< R,P >

```
<R> Immutatorable<R> transform(
  Immutator<? ___ R, ? ___ T> f
);
```

> Fill in the rest
>
> - Fill in the blank for `___`
> - Keep in mind PECS
> - We will use `f` as `R r = f.invoke(T)`
>
>   *(i.e., the input argument is of type **T** and the return type is **R**)*

Lab 04

Act
Immutate
- Probably

Immutate

Probably< T >

```
public <R> Immutatorable<R> transform(Immutator<? ___ R, ? ___ T> f) {
  :
}
```

> Do we need to return `Immutatorable<R>` or can we return something else?
>
> - Keep in mind the concept of overriding
> - Don't forget to add `@Override`

Lab 04

Act
Immutate
- Probably

Immutate

Probably< T >

```
public <R> Immutatorable<R> transform(Immutator<? ___ R, ? ___ T> f) {
  :
}
```

Possible Implementation

```
  if (this.value == null) {
    return Probably.none();
  }
  return Probably.<R>just(f.invoke(this.value));
```

Lab 04

Act
Immutate
Apply

Apply

Applicable< T>

```
<R> Probably<R> apply(Probably<___> p);
```

> Make it as flexible as possible. Consider `C <: B <: A`:
>
> - The following should be accepted
>
>   ```
>   Probably.<B>just(new  B())
>       .<B>apply(Probably<Immutator<C,A>>)
>   ```
>
> - We expect `Probably<Immutator<B,B>>` and we should accept `Probably<Immutator<C,A>>`

Lab 04

Act
Immutate
Apply
Sample
- Immutator

Sample

Immutator< R,P >

Increment (Integer -> Integer)

```
class Incr implements<Integer,
                      Integer> {
 public Integer invoke(Integer t) {
  return t + 1;
 }
}
```

Length (String -> Integer)

```
class Length implements<Integer,
                        String> {
 public Integer invoke(String t) {
  return t.length();
 }
}
```

Lab 04

Act
Immutate
Apply
Sample
- Immutator

Sample

Immutator< R,P >

Definition

> Consider `C <: B <: A`

```
class CfromA implements<C,A> {
 public C invoke(A t) {
  return new C();
 }
}
```

Usage

```
Probably<B> obj
  = Probably.<B>just(new B());
B res = obj.<B>transform(CfromA);
// Need to make sure that
//   Immutator<C,A>
//   <:
//   Immutator<? ___ B, ? ___ B>
```

Lab 04

Act
Immutate
Apply
Sample
- *Immutator*
- *Applicative*

Sample

Applicative

Given

```
class Incr implements<Integer,Integer> { .. }
class Length implements<Integer,String> { .. }
```

Probably< Immutator< R,P > >

```
Probably<Immutator<Integer,Integer>> maybeIncr = Probably.just(Incr);
Probably<Immutator<Integer,String>> maybeLen = Probably.just(Length);
```

Lab 05

Lab 05

Access

Access Modifiers

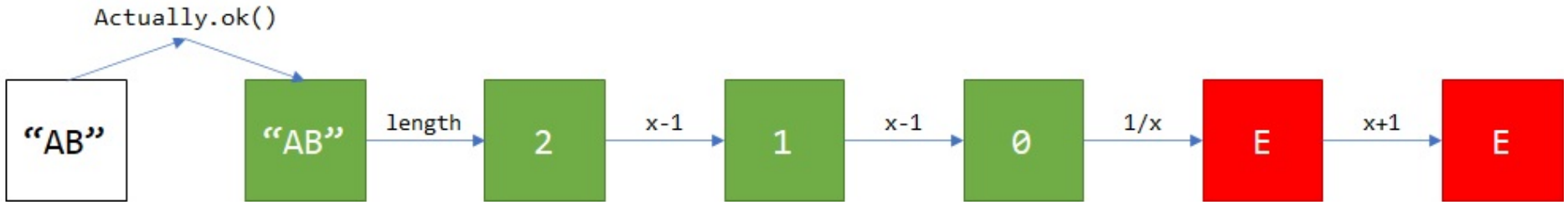
Modifier	Self	Package	Subclass	Others
`private`	✅	❌	❌	❌
default	✅	✅	✅ *(same package)* ❌ *(different package)*	❌
`protected`	✅	✅	✅	❌
`public`	✅	✅	✅	✅

Lab 05

Access
Actually

Actually< T >

> Representation of *either* a result *(i.e., non-error)* with value of type T or an error *(i.e., exception thrown)*



- If error occurs, we `transform` from `Success<T>` to `Failure`
- After error occurs, `transform` does nothing

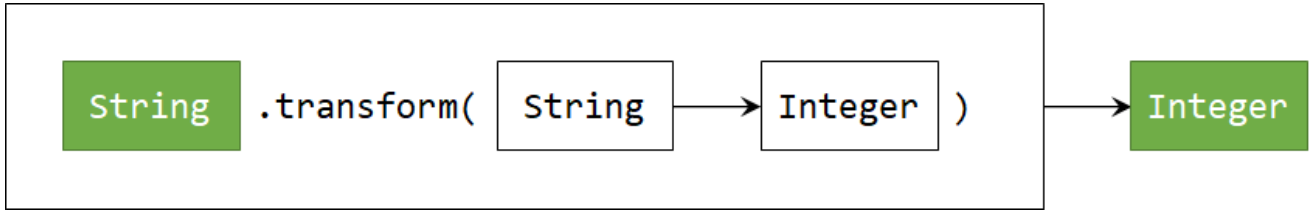
#Let green box represents result and red box represents error

Lab 05

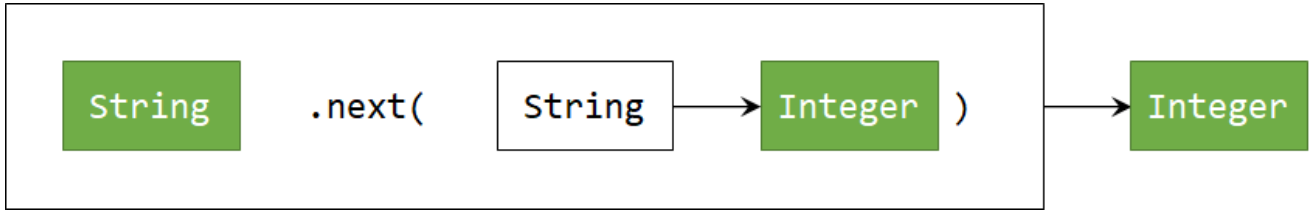
Access
Actually
Transform vs
Next

Transform vs Next

Transform



Next

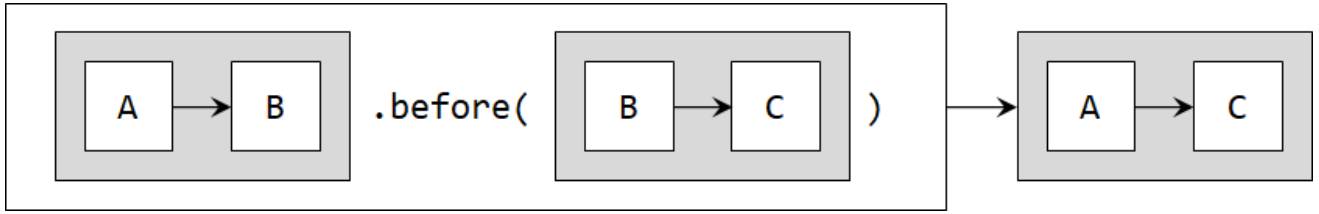


Lab 05

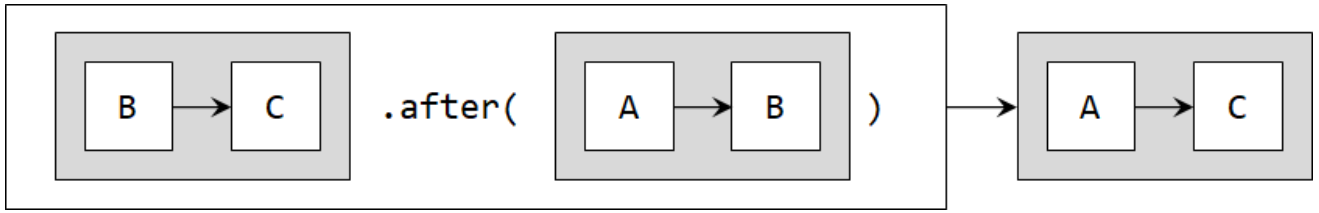
Access
Actually
Transform vs
Next
Composition

Transformer Composition

Before



After




```
(A -> B) ∘ (B -> C) = (A -> B -> C) = (A -> C)
```

PE 1

Lab 05

Preparation

This Week

- Start Panopto to record your screen
- Familiarise yourself with screen recording
- Record into canvas
- Try to record at least an hour to make sure no issue
- Check periodically to make sure recording works

Note

- You may use other software if you prefer
- Stricter rule for PE1 about screen recording

Lab 05

Preparation

Next Week

- Mock PE1
- Familiarise yourself with Unix environment
- Try to solve the practice PE1

```
jshell> /exit
|    Goodbye
```