CS2030S

Programming Methodology II

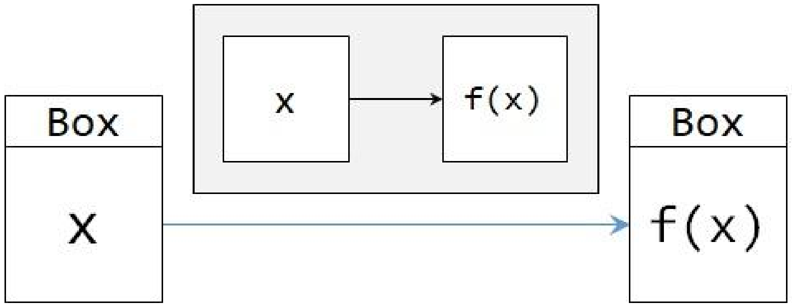
Recitation 06

Question 1

Question 1

Preliminary

map



Steps

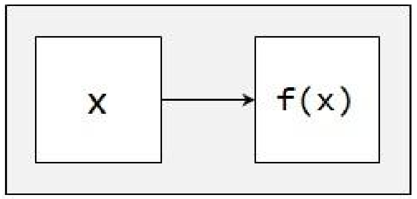
1. Open the box
2. Operate with function
3. Put into new box

Question 1

Preliminary
- map
- flatMap
Code

Preliminary

map



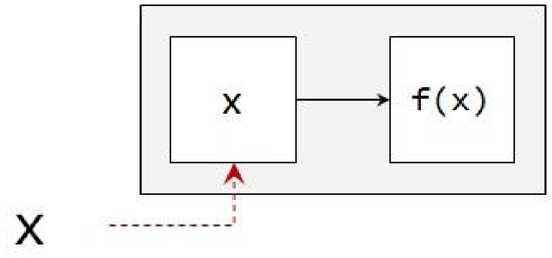
Steps

1. Open the box
2. Operate with function
3. Put into new box

Question 1

Preliminary

map



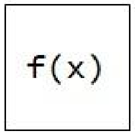
Steps

1. Open the box
2. Operate with function
3. Put into new box

Question 1

Preliminary

map



Steps

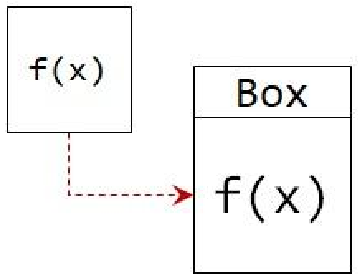
1. Open the box
2. Operate with function
3. Put into new box

Question 1

Preliminary
- map
- flatMap
Code

Preliminary

map



Steps

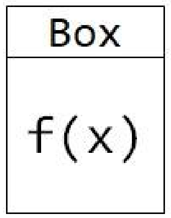
1. Open the box
2. Operate with function
3. Put into new box

Question 1

Preliminary
- map
- flatMap
Code

Preliminary

map



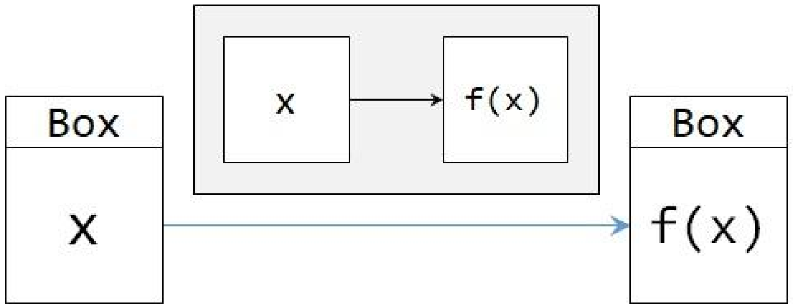
Steps

1. Open the box
2. Operate with function
3. Put into new box

Question 1

Preliminary

map



Steps

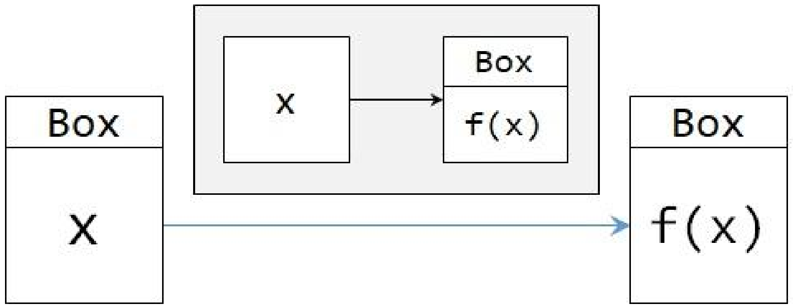
1. Open the box
2. Operate with function
3. Put into new box

Question 1

Preliminary
- map
- flatMap
Code

Preliminary

flatMap



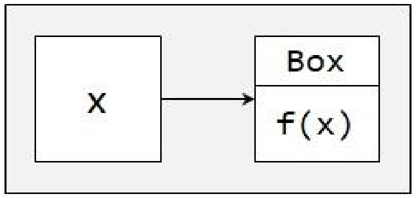
Steps

1. Open the box
2. Operate with function
3. Compose the two "context"

Question 1

Preliminary

flatMap



Steps

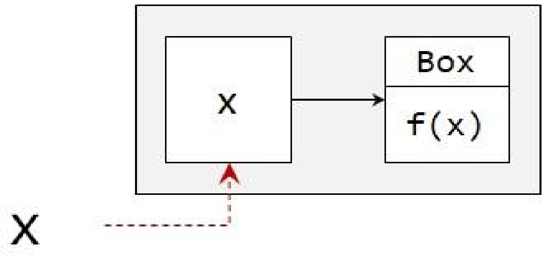
1. Open the box
2. Operate with function
3. Compose the two "context"

Question 1

Preliminary
- *map*
- *flatMap*
Code

Preliminary

flatMap



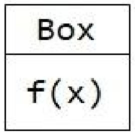
Steps

1. Open the box
2. Operate with function
3. Compose the two "context"

Question 1

Preliminary

flatMap



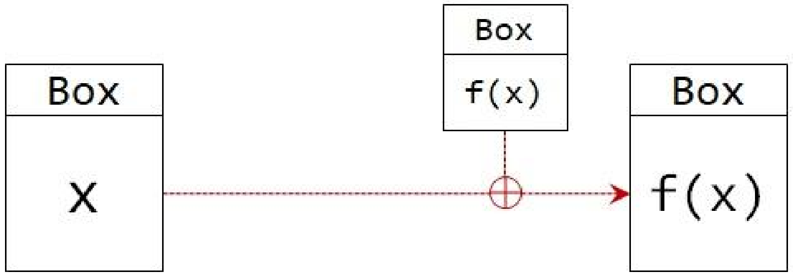
Steps

1. Open the box
2. Operate with function
3. Compose the two "context"

Question 1

Preliminary

flatMap



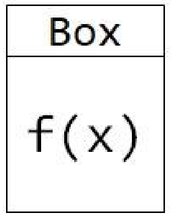
Steps

1. Open the box
2. Operate with function
3. Compose the two "context"

Question 1

Preliminary

flatMap



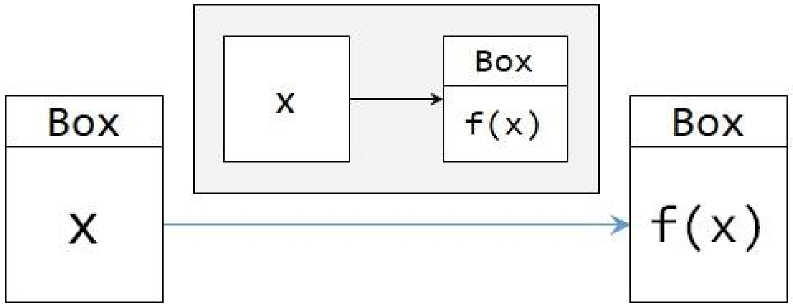
Steps

1. Open the box
2. Operate with function
3. Compose the two "context"

Question 1

Preliminary

flatMap



Steps

1. Open the box
2. Operate with function
3. Compose the two "context"

Question 1

Preliminary
Code
- Questions
- Transformed

Code

Original

```
Maybe<Internship> match(Resume r) {
  if (r == null) {
    return Maybe.none();
  }
  Maybe<List<String>> optList = r.getListOfLanguages();
  List<String> list;
  if (optList.equals(Maybe.none())) {
    list = List.of();
  } else {
    list = optList.get(); // cannot call
  }
  if (list.contains("Java")) {
    return Maybe.of(findInternship(list));
  } else {
    return Maybe.none();
  }
}
```

Questions

1. What is the type of getListOfLanguages()?
2. What is the type of contains("Java")?
3. What is the type of findInternship(list)?

Question 1

Code

Original

```
Maybe<Internship> match(Resume r) {
  if (r == null) {
    return Maybe.none();
  }
  Maybe<List<String>> optList = r.getListOfLanguages();
  List<String> list;
  if (optList.equals(Maybe.none())) {
    list = List.of();
  } else {
    list = optList.get(); // cannot call
  }
  if (list.contains("Java")) {
    return Maybe.of(findInternship(list));
  } else {
    return Maybe.none();
  }
}
```

Transformed

Question 2

Question 2

Question

Question

Code

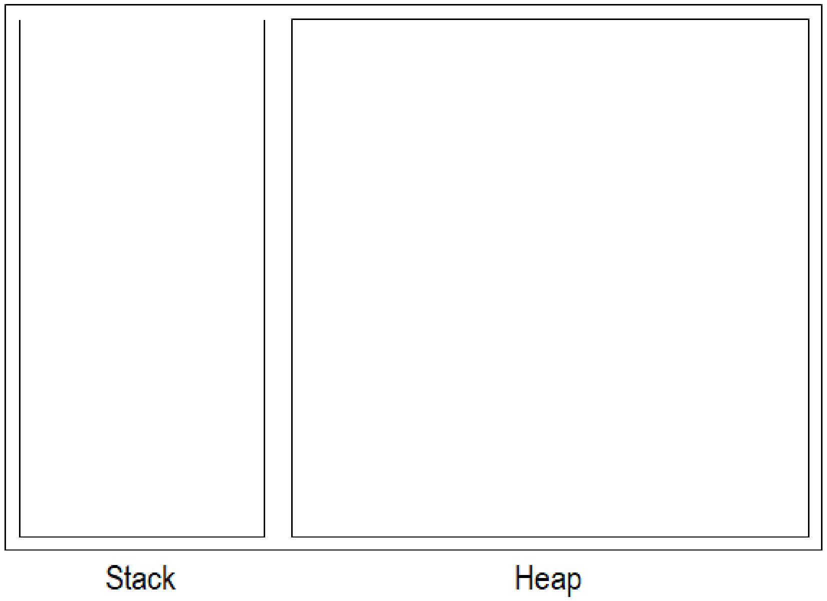
```
class A {
  private int x;

  public A(int x) {
    this.x = x;
  }

  public int get() {
    // Line A
    return this.x;
  }
}
```

```
A a = new A(5);
Producer<Integer> p = () -> a.get();
p.produce();
```

Stack/Heap Diagram



Question 3

Question 3

Preliminary
- *Code*
- *Design*
Compute

Preliminary

Original Code

```
static long sum(long n, long result) {
  if (n == 0) {
    return result;
  } else {
    return sum(n - 1, n + result);
  }
}
```

Rewritten Code

```
static Compute<Long> sum(long n, long s) {
  if (n == 0) {
    return new Base<>(() -> s);
  } else {
    return new Recursive<>(() -> sum(n - 1, n + s));
  }
}
```

Usage

```
static long summer(long n) {
  Compute<Long> result = sum(n, 0);
  while (result.isRecursive()) {
    result = result.recurse();
  }
  return result.evaluate();
}
```

Question 3

Preliminary
- Code
- Design
Compute

Preliminary

Design

```
static Compute<Long> sum(long n, long s) {
  if (n == 0) {
    return new Base<>(() -> s);
  } else {
    return new Recursive<>(() -> sum(n - 1, n + s));
  }
}
```

```
static long summer(long n) {
  Compute<Long> result = sum(n, 0);
  while (result.isRecursive()) {
    result = result.recurse();
  }
  return result.evaluate();
}
```

Class Diagram

Question 3

Preliminary
Compute

Compute

Usage

```java
static Compute<Long> sum(long n, long s) {
  if (n == 0) {
    return new Base<>(() -> s);
  } else {
    return new Recursive<>(() -> sum(n - 1, n + s));
  }
}
```

```java
static long summer(long n) {
  Compute<Long> result = sum(n, 0);
  while (result.isRecursive()) {
    result = result.recurse();
  }
  return result.evaluate();
}
```

Classes

jshell> /exit
| Goodbye