Week 8 Lab Session

CS2030S AY21/22 Semester 2
Lab 14B

Yan Xiaozhi (David)
@david_eom
yan_xiaozhi@u.nus.edu

10 Mar 2022

Admin

- Contact tracing & QR code
- PE1
- Mock PE1 solution
- Lab 5 due 15 Mar (Tue)

What You’ll Need in Lab 5

- Nested wildcards
- Anonymous classes
- Nested classes
- Java packages

Nested Wildcards

-
  ```
  class Animal { }
  class Dog extends Animal { }
  class Box<T> { }
  ```

- Which ones compile?

  -
    ```
    class A {
      static <T> void foo(Box<List<T>> box) {
      }
    }
    ```

  -
    ```
    A.<Animal>foo(new Box<List<Animal>>());
    A.<Animal>foo(new Box<List<Dog>>());
    A.<Animal>foo(new Box<ArrayList<Animal>>());
    A.<Animal>foo(new Box<ArrayList<Dog>>());
    ```

Nested Wildcards

-
  ```
  class Animal { }
  class Dog extends Animal { }
  class Box<T> { }
  ```

- Which ones compile?

  -
    ```
    class A {
      static <T> void foo(Box<? extends List<T>> box) {
      }
    }
    ```

  -
    ```
    A.<Animal>foo(new Box<List<Animal>>());
    A.<Animal>foo(new Box<List<Dog>>());
    A.<Animal>foo(new Box<ArrayList<Animal>>());
    A.<Animal>foo(new Box<ArrayList<Dog>>());
    ```

Nested Wildcards

-
  ```
  class Animal { }
  class Dog extends Animal { }
  class Box<T> { }
  ```

- Which ones compile?

  -
    ```
    class A {
      static <T> void foo(Box<? extends List<? extends T>>) {
      }
    }
    ```

  -
    ```
    A.<Animal>foo(new Box<List<Animal>>());
    A.<Animal>foo(new Box<List<Dog>>());
    A.<Animal>foo(new Box<ArrayList<Animal>>());
    A.<Animal>foo(new Box<ArrayList<Dog>>());
    ```

Anonymous Class

- Suppose we use `AddK(3)` only once and never again, rewrite `AddK` as an anonymous class

-
  ```
  class AddK implements Transformer<Integer, Integer> {
    int k;
    AddK(int k) {
      this.k = k;
    }

    @Override
    public Integer transform(Integer t) {
      return t + k;
    }
  }
  ```

- `Box.of(4).map(new AddK(3));`

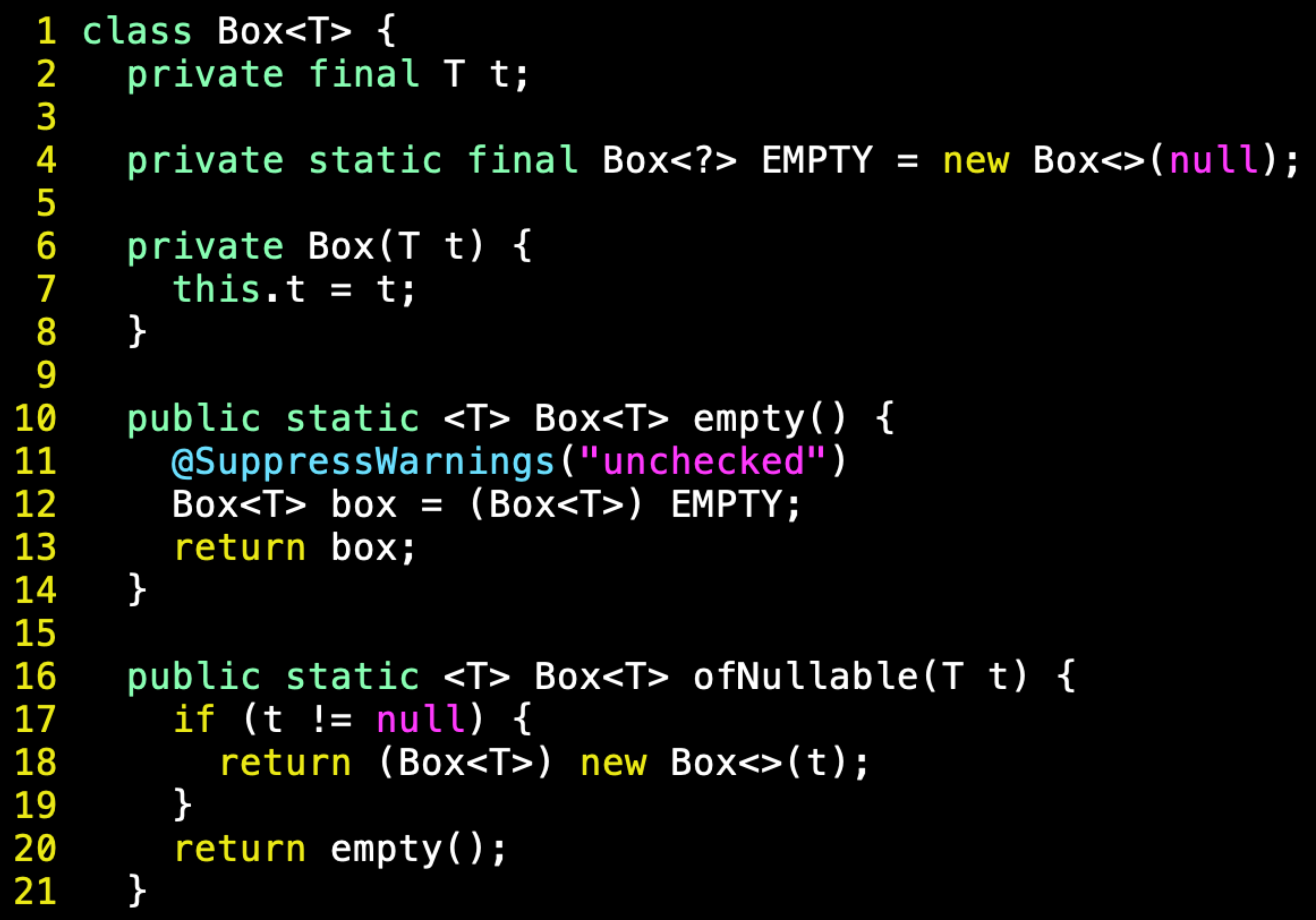
Nested Class

-
  ```
  if (this.t != null) {
    // do something to t
  } else {
    // handle null
  }
  ```

- Can we tidy up our code?
  - Separate two cases into different classes
  - Let dynamic binding take care of the conditional statements

Nested Class

- Copy files by running `cp -r ~cs2030s/lab-week8 ~/<location>`
  - Simplified version of `Box<T>` from lab 4
  - Run `jshell < test.jsh` to test `Box`



```java
class Box<T> {
  private final T t;

  private static final Box<?> EMPTY = new Box<>(null);

  private Box(T t) {
    this.t = t;
  }

  public static <T> Box<T> empty() {
    @SuppressWarnings("unchecked")
    Box<T> box = (Box<T>) EMPTY;
    return box;
  }

  public static <T> Box<T> ofNullable(T t) {
    if (t != null) {
      return (Box<T>) new Box<>(t);
    }
    return empty();
  }
```

Nested Class

- Make `Box<T>` an `abstract` class
- Create `private static` nested classes `Empty` and `NonEmpty<T>`
- Put fields/methods related to empty box into `Empty`
- Put fields/methods related to non-empty box into `NonEmpty<T>`
- Box dictates the API to be implemented in `Empty` and `NonEmpty<T>`

Java Packages

- Encapsulation that groups relevant classes together
- Advantages:
  - Provide additional abstraction barrier
  - Namespace management
- Every package has a name using hierarchical dot notation
  - `java.util`
  - `com.google.common.math`
- Every class we’ve written belongs to the `default` package

Java Packages

- Ability to control accessibility outside a package
- Without access modifier, field/method accessible within the package only

Access Modifier	Class	Package	Subclass (same package)	Subclass (diff package)	World
`public`	✅	✅	✅	✅	✅
`protected`	✅	✅	✅	✅	
no modifier	✅	✅	✅		
`private`	✅				

Java Packages

- Package: `cs2030s.fp`
  - Make directories `cs2030s/fp`: `mkdir -p cs2030s/fp`
  - Move to package: `mv BooleanCondition.java cs2030s/fp`
  - Tell Java that it is part of a package: `package cs2030s.fp;` as first line
  - Make class/interface accessible outside the package: `public interface`
- We can now use `cs2030s.fp.BooleanCondition` in `Box<T>`
- To avoid typing its full name, at the top of Box.java:
  - `import cs2030s.fp.BooleanCondition;`

Lab 5 Overview

Lab 5: Maybe

- Encapsulate a value that may be null
- Common abstraction in programming languages
  - `Nullable<T>` in C#
  - `Option<T>` in Rust
  - `Optional<T>` in Swift

Lab 5: Maybe

- Why is the wrapper needed?
-
  ```
  public int calculateTotalCost(
      Item item1, Item item2, Item item3) {
    int cost = 0;
    if (item1 != null) { cost += item1.getCost(); }
    if (item2 != null) { cost += item2.getCost(); }
    if (item3 != null) { cost += item3.getCost(); }
    return cost;
  }
  ```
- Disadvantages:
  - Multiple checks needed
  - No explicit way to show that a variable can be `null`

Lab 5: Maybe

-
  ```
  public int calculateTotalCost(
      Optional<Item> item1,
      Optional<Item> item2,
      Optional<Item> item3) {
    return item1.orElse(emptyItem).getCost() +
           item2.orElse(emptyItem).getCost() +
           item3.orElse(emptyItem).getCost();
  }
  ```
- Using `Maybe<T>`
  - Eliminates the use of `null` to indicate “not there”
  - Prevents `null` checks and `NullPointerException`

Lab 5: Maybe

- Implement `Maybe` class and its methods
  - Inner classes and factory methods
  - `filter`
  - `map`
  - `flatMap`
  - `orElse`
- Modify the `getGrade()` method to use `Maybe`

Tips

- 12 marks correctness + 2 marks style, 3% of final grade
- Make full & proper use of wildcards
- Apply PECS in your method signature
  - Especially for `flatMap`!
- Lab 6 and Lab 7 contingent upon Lab 5 completion
- Submission related issues

Happy coding! 👨‍💻