CS2030S Lab 12B 🚀

9 March 2023 (Week 8)

What you need for Lab 5

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

Nested Wildcards

Exercise: Nested Wildcards

Launch JShell and follow along:

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

Which one compiles (PART 1)?

```java
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>>());
```

Which one compiles (PART 2)?

```java
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>>());
```

Which one compiles (PART 3)?

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

When you have nested generics, remember to apply PECS at all levels.

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

Anonymous Class

Suppose we use `AddK` only once and never again. Rewrite `AddK` as an anonymous class.

```java
 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));
```

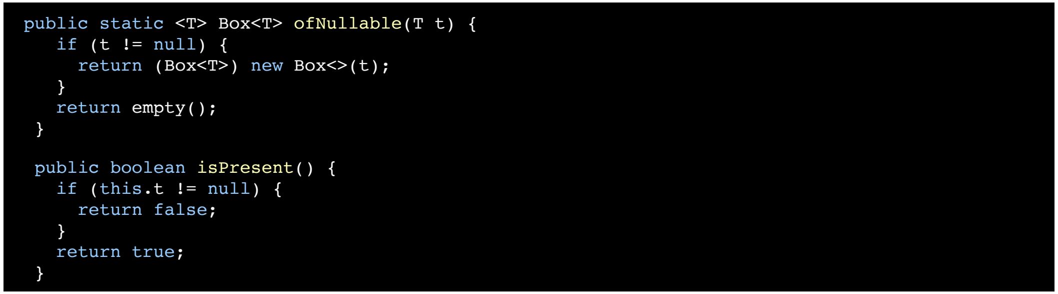
Nested Class

Exercise: Nested Class

- Copy files from `~cs2030s/lab-week8` with

```
cp -r ~cs2030s/lab-week8 ~/
```

- This is a simplified version of `Box<T>` from Lab 4
- Look at `Box.java`.
- Run `jshell < test.jsh` to test Box.



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

 public boolean isPresent() {
   if (this.t != null) {
     return false;
   }
   return true;
 }
```

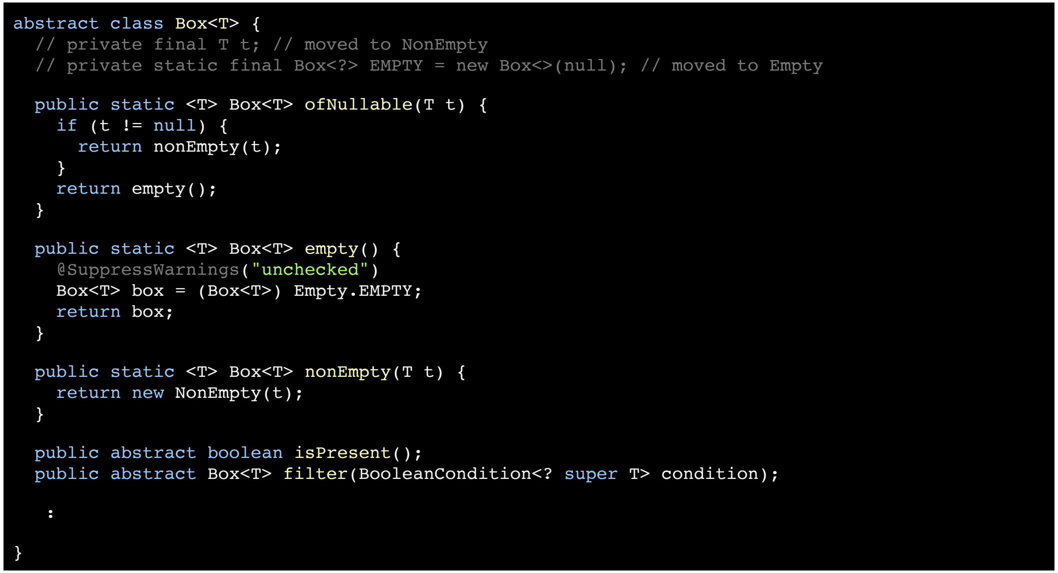
```java
public Box<T> filter(BooleanCondition<? super T> condition) {
  if (this.t != null) {
    if (condition.test(this.t) == false) {
      return empty();
    }
    return (Box<T>) this;
  }
  return empty();
}

@Override
public String toString() {
  if (this.t != null) {
    return "[" + t + "]";
  }
  return "[]";
}
```

- Observe the pattern:

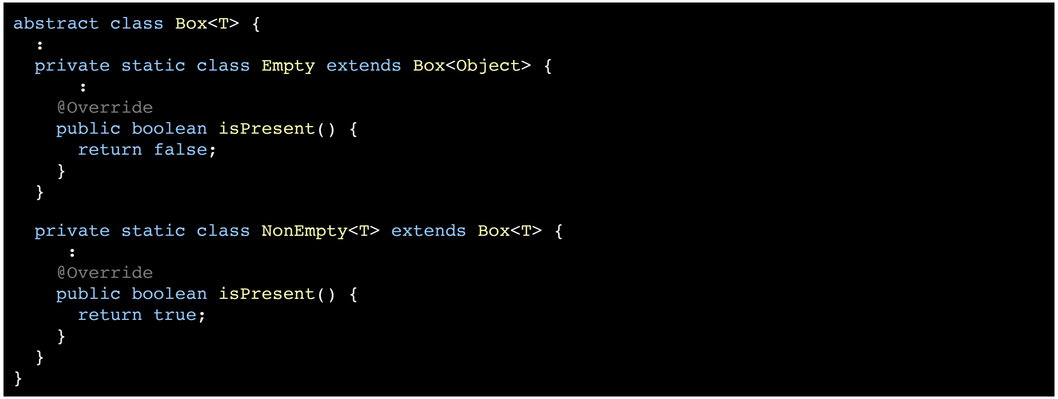
```
if (this.t != null) {
  // do something to t
} else {
  // handle case where t is null
}
```

- Can we tidy up our code, separate these two cases into different classes?
- Let dynamic binding take care of the conditional statements for us.

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



```java
abstract class Box<T> {
  // private final T t; // moved to NonEmpty
  // private static final Box<?> EMPTY = new Box<>(null); // moved to Empty

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

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

  public static <T> Box<T> nonEmpty(T t) {
    return new NonEmpty(t);
  }

  public abstract boolean isPresent();
  public abstract Box<T> filter(BooleanCondition<? super T> condition);

    :

}
```



```java
abstract class Box<T> {
  :
  private static class Empty extends Box<Object> {
      :
    @Override
    public boolean isPresent() {
      return false;
    }
  }

  private static class NonEmpty<T> extends Box<T> {
     :
    @Override
    public boolean isPresent() {
      return true;
    }
  }
}
```

Java packages

- We can group related classes into a *package* in Java to provide an additional abstraction barrier and to manage the namespace.
- Every package has a name using hierarchical dot notation (e.g., `com.google.common.math`, `java.io`)
- So far, every class that we write belongs to the same, *default*, package.

- We can control whether a field/method/class is accessible outside a package
- Without any access modifier, a field/method is accessible by any class within the package only
- With `protected` modifier, a field/method is accessible by any class within the package and outside the package through inheritance.

Creating a package

- We name our package `cs2030s.fp`
- Make directories `cs2030s/fp`

```
mkdir -p cs2030s/fp
```

- Move `BooleanCondition.java` to `cs2030s/fp`:

```
mv BooleanCondition.java cs2030s/fp
```

- Tell Java that `BooleanCondition` is part of a package. Add the line

  ```java
  package cs2030s.fp;
  ```

  as the first line of `BooleanCondition.java`
- Make a class/interface accessible from outside the package. Add the access modifier `public` to the declaration:

  ```java
  public interface BooleanCondition<T> { }
  ```

- We can now use `cs2030s.fp.BooleanCondition` in our `Box<T>`
- To avoid typing its full name, import it at the top of `Box.java`;

```java
import cs2030s.fp.BooleanCondition;
```

Lab 5

Due next Tuesday night

3%

`Maybe<T>`

- Encapsulate a value that may be `null`
- Common abstraction in programming languages
- E.g.,
  - `Nullable<T>` in C#,
  - `T | None` in Python,
  - `Option<T>` in Rust,
  - `Optional<T>` in Swift, etc.

Using `Maybe<T>` properly eliminates the use of `null` to indicate "not there", and thus, null checks and `NullPointerException`.

```
void find(Map<Int, String> map) {
  this.add(map.get(0).trim());  // may crash with NullPointerException
}
```

```
void find(Map<Int, String> map) {
  if (map.get(0) != null) {    // littered with null checks
    this.add(map.get(0).trim());
  }
}
```

Goals of Lab 5

- Create `cs2030s.fp` with useful types
- Implement `Maybe<T>`
- See how `Maybe<T>` can be used to eliminate `null`s

(`Maybe<T>` is an important component for Lab 6 and 7)

Lab 5

- Run `~cs2030s/get-lab5` on PE hosts.
- Solve and submit before Tuesday night

Version: v1.0

Last Updated: Tue Mar 7 09:18:03 +08 2023