

Nested and Immutable

Preliminary

The following questions are not going to be discussed. Answers is given without explanation. You are encouraged to discuss them on Ed.

1. Consider the following nested class[1].

Nested Class

```
interface I {
  void bar(int x);
}
class A {
  int x = 1;                    /* (1) */
  class B {
    private int x = 2;          /* (2) */
    public int getX () {
      return this.x;
    }
  }
  I foo(int x) {                /* (3) */
    class C extends B implements I {
      private int x = 4;        /* (4) */
      public int getX () {
        return this.x;
      }
      public void bar(int x) { /* (5) */
        System.out.println("__Blank__");
      }
    }
    return new C();
  }
}
```

We have 5 different versions of `x` and we want to print each version at line 19, replacing `"__Blank__"`.
`System.out.println("__Blank__");`

For each of the version, write an expression that can be used in place of `"__Blank__"` to print that version of `x` or answer "**impossible**" if it is not possible.

(i) `/* (1) */`
(ii) `/* (2) */`

(iii) `/* (3) */`
(iv) `/* (4) */`
(v) `/* (5) */`

Discussion

The following questions are problem sets to be discussed during recitation. You are expected to attempt them before coming to the recitation. You may be randomly called to present your answer during recitation. You are encouraged to discuss them on Ed.

1. For each of the code snippets below, revise the code so that we use *qualified name* (i.e., the dot (`.`) notation) to access the fields[1]. For example, use `this.a` and `X.b` instead of just `a` and `b`.

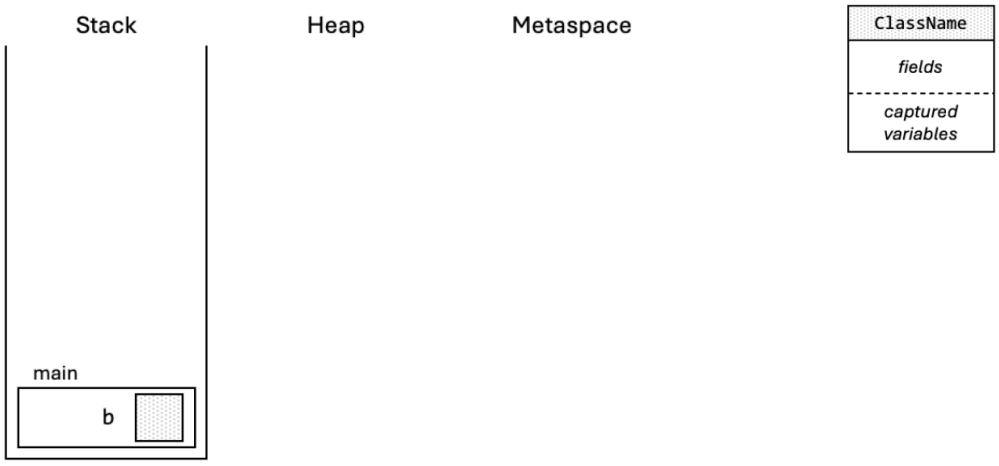
   Now, suppose that the following code snippet is invoked.

   Code Snippet

   ```
   B b = new B();
   b.f();
   ```

   Sketch the content of the stack and heap at `// Line A`. Java runtime stores static fields in a region of memory called metaspace[3]. Items in metaspace looks similar to elements in the heap except that we only store static fields.

   Use the following template for the stack and heap diagram (left) as well as captured variables (right).



   (a) **Code 1**

   ```
   class B {
     static int x = 0;
     void f() {
       A a = new A();
       // Line A
     }
     static class A {
       int y = 0;
       A() {
         y = x + 1;
       }
     }
   }
   ```

[1] This is similar to canonical name[2] but shorter without the class name.

(b) **Code 2**

```
class B {
  void f() {
    int x = 0;
    class A {
      int y = 0;
      A() {
        y = x + 1;
      }
    }
    A a = new A();
    // Line A
  }
}
```

(c) **Code 3**

```
class B {
  int x = 1;
  void f() {
    int y = 2;
    class A {
      void g() {
        x = y;
      }
    }
    A a = new A();
    // Line A
    a.g();
  }
}
```

2. Consider the following implementation of a stack.

Mutable Stack

```
public class Stack<T> {
  private T head;
  private Stack<T> tail;
  private static final Stack<?> EMPTY_STACK = new Stack<>(null, null);

  private Stack(T head, Stack<T> tail){
    this.head = head;
    this.tail = tail;
  }

  public void push(T t){
    this.tail = new Stack<T>(this.head, this.tail);
    this.head = t;
  }

  public void pop(){
    if (this.head == null) {
      throw new IllegalStateException("Stack is empty");
    }
    this.head = this.tail.head;
    this.tail = this.tail.tail;
```

```
  }

  public T head(){
    if (this.head == null) {
      throw new IllegalStateException("Stack is empty");
    }
    return head;
  }

  public boolean isEmpty(){
    if (this.head == null) {
      return true;
    } else {
      return false;
    }
  }

  public static <T> Stack<T> createNew(){
    @SuppressWarnings("unchecked")
    Stack<T> emptyStack = (Stack<T>) EMPTY_STACK;
    return emptyStack;
  }
}
```

The `push` and `pop` operations on the stack change (or mutate) the stack. Here is an example of how the `Stack` class can be used. The example below pushes 1, 2, and 3 into the stack and pops them out in reverse order.

Sample Run

```
jshell> /open Stack.java
jshell> Stack<Integer> s = Stack.createNew();
jshell> s.push(1);
jshell> s.push(2);
jshell> s.push(3);
jshell> s.head();
$.. ==> 3
jshell> s.pop();
jshell> s.head();
$.. ==> 2
jshell> s.pop();
jshell> s.head();
$.. ==> 1
jshell> s.pop();
jshell>
```

Change the implementation of `Stack` to make it immutable up to the generic type `T`[2]. Create a new class `ImmutableStack`. Show how it can be used by pushing 1, 2, and 3 into the immutable stack and by popping 3, 2, and 1 out from the stack.

[2] In other words, our stack is immutable only if `T` is immutable.

Exercise

The following questions are to be discussed on Ed as no answer will be given. ***Use only concepts that you have learnt****. You are expected to attempt them before the next lecture as the concepts may be used and/or mentioned in the lecture. The concept may also be tested for assessment.*

1. In this question, we want to be able to perform an arbitrary operations. We are going to build on the `Int` class from previous recitation, reproduced below.

   Int.java

   ```
   public class Int {
     private int val;

     public Int(int val) {
       this.val = val;
     }

     public static Int add(Int n, int m) {
       return new Int(n.val + m);  // this + m
     }
     public static Int sub(Int n, int m) {
       return new Int(n.val - m);  // this - m
     }
     public static Int mul(Int n, int m) {
       return new Int(n.val * m);  // this * m
     }
   }
   ```

   Currently, it has 3 different operations: (i) addition, (ii) subtraction, and (iii) multiplication.

   (a) Does the class `Int` allow the integer division operation to be performed? If not, how would you extend the capability of `Int` to allow this?

   Note: By default, division `x / y` when both operands (i.e., `x` and `y`) are `int` will result in integer division. Ignore potential division by zero.

   (b) Does the class `Int` allow the XOR operation to be performed? If not, how would you extend the capability of `Int` to allow this?

   Note: XOR is a bitwise operation. It acts on each bit of an integer according to the rule represented in the table below. It can be performed using `x ^ y`.

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

(c) How would you extend this capability to an **arbitrary** integer operation?

Hint: While you cannot pass a function as argument, you can pass an object. If this object can operate on integer, you have simulated a function. But how would you make this general?

Bibliography

[1] Oracle, “Nested Classes.” [Online]. Available: https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

[2] Oracle, “Fully Qualified Names and Canonical Names.” [Online]. Available: https://docs.oracle.com/javase/specs/jls/se25/html/jls-6.html#jls-6.7

[3] OpenJDK, “Metaspace.” [Online]. Available: https://wiki.openjdk.org/display/HotSpot/Metaspace