# CS2030S
## Problem Set 5

AY22/23 S2 — 8-9 March 2023

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

   Now suppose that the following is invoked:

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

   Sketch the content of the stack and heap at Line A. Java run-time stores static fields in a region of memory called *metaspace*. Sketch that as well where appropriate.

   (a)
   ```
   class B {
     static int x = 0;

     void f() {
       A a = new A();
       // Line A
     }

     static class A {
       int y = 0;

       A() {
         y = x + 1;
       }
     }
   }
   ```

   (b)
   ```
   class B {
     void f() {
       int x = 0;

       class A {
         int y = 0;
         A() {
           y = x + 1;
         }
       }

       A a = new A();
       // Line A
     }
   }
   ```

   (c)
   ```
   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.

```
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.

```
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();
$6 ==> 3
jshell> s.pop();
jshell> s.head();
$8 ==> 2
jshell> s.pop();
jshell> s.head();
$10 ==> 1
jshell> s.pop();
jshell>
```

Change the implementation of `Stack` to make it immutable. 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.