CS2030 Programming Methodology

Semester 2 2019/2020

5 March 2020

Problem Set #6

1. You are given two classes `MCQ` and `TFQ` that implements a question-answer system:

   - MCQ: multiple-choice questions comprising answers: `A B C D E`
   - TFQ: true/false questions comprising answers: `T F`

```
class MCQ {
    String question;
    char answer;

    public MCQ(String question) {
        this.question = question;
    }

    void getAnswer() {
        System.out.print(question + " ");
        answer = (new Scanner(System.in)).next().charAt(0);
        if (answer < 'A' || answer > 'E') {
            throw new InvalidMCQException("Invalid MCQ answer");
        }
    }
}

class TFQ {
    String question;
    char answer;

    public TFQ(String question) {
        this.question = question;
    }

    void getAnswer() {
        System.out.print(question + " ");
        answer = (new Scanner(System.in)).next().charAt(0);
        if (answer != 'T' && answer != 'F') {
            throw new InvalidTFQException("Invalid TFQ answer");
        }
    }
}
```

   In particular, an invalid answer to any of the questions will cause an exception (either `InvalidMCQException` or `InvalidTFQException`) to be thrown.

```
class InvalidMCQException extends IllegalArgumentException {
    public InvalidMCQException(String mesg) {
        super(mesg);
    }
}

class InvalidTFQException extends IllegalArgumentException {
    public InvalidTFQException(String mesg) {
        super(mesg);
    }
}
```

By employing the various object-oriented design principles, design a *more general* question-answer class `QA` that can take the place of both MCQ and TFQ types of questions (and possibly more in future, each with their own type of exceptions).

2. For each of the questions below, suppose the following is invoked:

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

Sketch the content of the stack, heap and metaspace *immediately after* the line

```
A a = new A();
```

is executed. Label the values and variables/fields clearly. You can assume `b` is already on the heap and you can ignore all other content of the stack and the heap before `b.f()` is called.

static valuables are stored in metaspace

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

    void f() {
        A a = new 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();
    }
}
```

(c)
```
class B {
    int x = 1;

    void f() {
        int y = 2;

        class A {
            void g() {
                x = y;
            }
        }

        A a = new A();
        a.g();
    }
}
```

a nested class object store a copy of the valuables that it use in order to do an operation. For ex, in part c, it stores a copy of y, but not x. The copied value must be final or effectively final
a nested class object also store the reference to its bound class using qualifier "this". For example, in part c, x is actually B.this.x

3. Java implements lambda expressions as anonymous classes. Suppose we have the following lambda expression `Function<String,Integer>`:

```
Function<String,Integer> findFirstSpace = str -> str.indexOf(' ');
```

Write the equivalent anonymous class for the expression above.

4. Suppose we have a class `A` that implements the following methods:

```
class A {
    int x;
    boolean isPositive;

    static A of(int x) {
        A a = new A();
        a.x = x;
        a.isPositive = (x > 0);
        return a;
    }

    A foo(Function<Integer, A> map) {
        return map.apply(this.x);
    }

    A bar(Function<Integer, A> map) {
        if (this.isPositive) {
            return map.apply(this.x);
        } else {
            return A.of(this.x);
        }
    }
}
```

Which of the following conditions hold for `A` for all values of `x`? `f` and `g` are both variables of type `Function<Integer,A>`; `a` is an object of type `A`.

(a) `A.of(x).foo(f)` always returns `f.apply(x)` T

(b) `a.foo(f).bar(g)` equals to `a.foo(x -> f.apply(x).bar(g))` T

(c) `a.bar(f).bar(g)` equals to `a.bar(x -> f.apply(x).bar(g))`

False for x > 0 and isPositive = false. Since we can still access the attribute of A.
If we add additional condition that A can only be initialize through A.of(x), and the attributes of A become private, then c holds for all x.