CS2030 Programming Methodology
Semester 1 2020/2021

30 September 2020
Problem Set #5

1. In Java, a `Set` is a `Collection` that does not contain duplicate elements (this is in contrast to a `List` which does allow duplicates). You are given the `Point` class below:

```
public class Point {
    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString() {
        return "(" + this.x + ", " + this.y + ")";
    }
}
```

   (a) What is the output of the following program fragment executed in jshell?

```
List<Point> points = new ArrayList<>()
points.add(new Point(1, 1))
points.add(new Point(1, 1))
points.indexOf(new Point(1, 1))
```

   (b) By defining an appropriate overriding `equals` method, demonstrate how the `indexOf` method can now give the correct behaviour.

   (c) What is the output of the following program fragment executed in jshell?

```
Point p = new Point(1, 1);
Point q = new Point(1, 1);
p.equals(q)
Set<Point> set = new HashSet<>()
set.add(p)
set.add(q)
set
```

   (d) Notice that although `p.equals(q)` returns `true`, the two points are considered distinct by `HashSet`. How do we ensure that only one point is maintained in the set?

   Hint: Refer to the definition of the `equals` *method in* `Object` *class*

2. The Java `Collection<E>` interface extends the `Iterable<E>` interface with the following abstract method declared.

```
Iterator<E> iterator();
```

   (a) Using the methods in the `Iterator` class, demonstrate how iteration is performed on a `List`, e.g.

```
List<Point> list = new ArrayList<>();
list.add(new Point(1, 1));
list.add(new Point(2, 2));
```

   (b) How is the use of an `Iterator` object, different from the following

```
for (Point p : list) {
    System.out.println(p);
}
```

3. What is the output of the following program fragment? Explain.

```
class A {
    static void f() throws Exception {
        try {
            throw new Exception();
        } finally {
            System.out.print("1");
        }
    }

    static void g() throws Exception {
        System.out.print("2");
        f();
        System.out.print("3");
    }

    public static void main(String[] args) {
        try {
            g();
        } catch (Exception e) {
            System.out.print("4");
        }
    }
}
```

4. 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).