CS2030 Programming Methodology
Semester 1 2019/2020

20 September 2019
Problem Set #4
Java Collections and Exceptions

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 double x;
    private final double y;

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

    @Override
    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }
        if (obj instanceof Point) {
            Point point = (Point) obj;
            return this.x == point.x && this.y == point.y;
        } else {
            return false;
        }
    }

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

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

```
Point p = new Point(1.0, 1.0);
Point q = new Point(1.0, 1.0);
p.equals(q)
Set<Point> set = new HashSet<>()
set.add(p)
set.add(q)
set
```

   (b) Notice that despite the `p.equals(q)` returns `true`, they are considered as unique elements in `set`. How do we ensure that only one point is maintained in `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(1.0, 1.0));
list.add(new Point(2.0, 2.0));
```

   (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).

5. For each of the questions 5a and 5b 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.

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

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

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

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

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

    static class A {
        int y = 0;

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

6. In each of the following program fragments, will it compile? If so, what will be printed?

(a)
```
class Main {
    static void f() throws IllegalArgumentException {
        try {
            System.out.println("Before throw");
            throw new IllegalArgumentException();
            System.out.println("After throw");
        } catch (IllegalArgumentException e) {
            System.out.println("Caught in f");
        }
    }

    public static void main(String[] args) {
        try {
            System.out.println("Before f");
            f();
            System.out.println("After f");
        } catch (Exception e) {
            System.out.println("Caught in main");
        }
    }
}
```

(b)
```
class Main {
    static void f() throws IllegalArgumentException {
        try {
            throw new IllegalArgumentException();
        } catch (IllegalArgumentException e) {
            System.out.println("Caught in f");
        }
    }

    public static void main(String[] args) {
        try {
            System.out.println("Before f");
            f();
            System.out.println("After f");
        } catch (Exception e) {
            System.out.println("Caught in main");
        }
    }
}
```

(c)
```
class Main {
    static void f() throws IllegalArgumentException {
        try {
            throw new Exception();
        } catch (IllegalArgumentException e) {
            System.out.println("Caught in f");
        }
    }

    public static void main(String[] args) {
        try {
            System.out.println("Before f");
            f();
            System.out.println("After f");
        } catch (Exception e) {
            System.out.println("Caught in main");
        }
    }
}
```

(d)
```
class Main {
    static void f() throws Exception {
        try {
            throw new IllegalArgumentException();
        } catch (Exception e) {
            System.out.println("Caught in f");
        }
    }

    public static void main(String[] args) {
        try {
            System.out.println("Before f");
            f();
            System.out.println("After f");
        } catch (Exception e) {
            System.out.println("Caught in main");
        }
    }
}
```

(e)
```
class Main {
    static void f() throws Exception {
        try {
            throw new ArrayIndexOutOfBoundsException();
        } catch (IllegalArgumentException e) {
            System.out.println("Caught in f");
        }
    }

    public static void main(String[] args) {
        try {
            System.out.println("Before f");
            f();
            System.out.println("After f");
        } catch (Exception e) {
            System.out.println("Caught in main");
        }
    }
}
```

(f)
```
class Main {
    static void f() throws Exception {
        try {
            throw new ArrayIndexOutOfBoundsException();
        } catch (IllegalArgumentException e) {
            System.out.println("Caught IA exception in f");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Caught AIOOB exception in f");
        }
    }

    public static void main(String[] args) {
        try {
            System.out.println("Before f");
            f();
            System.out.println("After f");
        } catch (Exception e) {
            System.out.println("Caught in main");
        }
    }
}
```

(g)
```
class Main {
    static void f() throws Exception {
        try {
            throw new ArrayIndexOutOfBoundsException();
        } catch (Exception e) {
            System.out.println("Caught exception in f");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Caught AIOOB exception in f");
        }
    }

    public static void main(String[] args) {
        try {
            System.out.println("Before f");
            f();
            System.out.println("After f");
        } catch (Exception e) {
            System.out.println("Caught in main");
        }
    }
}
```

(h)
```
class Main {
    static void f() throws Exception {
        try {
            throw new ArrayIndexOutOfBoundsException();
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Caught AIOOB exception in f");
        } catch (Exception e) {
            System.out.println("Caught exception in f");
        }
    }

    public static void main(String[] args) {
        try {
            System.out.println("Before f");
            f();
            System.out.println("After f");
        } catch (Exception e) {
            System.out.println("Caught in main");
        }
    }
}
```