CS2030 Programming Methodology
Semester 1 2018/2019

14 September 2018
Tutorial 3
OOP Design Principles and Exceptions

1. Consider the following classes: `FormattedText` that adds formatting information to the text. We call `toggleUnderline()` to add or remove underlines from the text. A `URL` is a `FormattedText` that is always underlined.

```
class FormattedText {
    public String text;
    public boolean isUnderlined;
    public void toggleUnderline() {
        isUnderlined = (!isUnderlined);
    }
}

class URL extends FormattedText {
    public URL() {
        isUnderlined = true;
    }

    public void toggleUnderline() {
        return;
    }
}
```

   Does the above violate Liskov Substitution Principle? Explain.

2. Consider the following program fragment.

```
class A {
    int x;
    A(int x) {
        this.x = x;
    }
    public A method() {
        return new A(x);
    }
}
```

```
class B extends A {
    B(int x) {
        super(x);
    }
    @Override
    public B method() {
        return new B(x);
    }
}
```

Does it compile? What happens if we switch the method definitions between class `A` and class `B` instead? Give reasons for your observations.

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. 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");
        }
    }
}
```

(i)
```
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");
        }
    }
}
```