CS2030 Programming Methodology
Semester 1 2018/2019

14 September 2018
Tutorial 3 Suggested Guidance
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.

   Yes. The "desirable property" here is that `toggleUnderline()` toggles the `isUnderlined` flag. Since URL— changes the behavior of `toggleUnderline()`, this property no longer holds for subclass `URL`. Places in a program where the super-class (i.e. `FormattedText`) is used cannot be simply replaced by the sub-class (i.e `URL`).

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.

There is no compilatin in the given program fragment as any existing code that invokes `A`'s `method` prior to being inherited would still work if the code invokes `B`'s `method` instead after `B` inherits `A`. Saying that LSP is not violated is not exactly right, as Java does not check LSP violations during compilation (notice that the questions asks whether the program compiles, and not whether the program violates LSP).

When we switch the method definitions, `A`'s `method` now returns a reference to a `B` object, but overriding it with a method that returns a reference-type `A` does not gaurantee that the object is a `B` object. So the overriding is not allowed and results in a compilation error.

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");
        }
    }
}
```

```
214
```

- Since `main()` calls `g()`, `2` will be printed first.
- `f()` is then executed, which leads to an exception being thrown.
- Before `f()` returns, it executes the `finally` block, leading to `1` being printed.
- After returning to `g()`, since an exception was thrown, `3` will NOT be printed, with control returning to `main()`, which catches the exception.
- The `catch` block is executed and `4` is then printed.

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).

Here are some design issues that needs to be addressed:

- Abstract out common code in `TFQ` and `MCQ` to a common place `QA`
- `MCQ` and `TFQ` inherits from `QA`
- `QA`'s abstract method `getAnswer` throws a more general exception than `MCQ` and `TFQ`
- Create the general exception class `InvalidQuestionException`

```
import java.util.*;

abstract class QA {
    String question;
    char answer;

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

    abstract void getAnswer() throws InvalidQuestionException;

    char askQuestion() {
        Scanner sc = new Scanner(System.in);
        System.out.print(question + " ");
        return sc.next().charAt(0);
    }
}

class MCQ extends QA {
    public MCQ(String question) {
        super(question);
    }

    void getAnswer() {
        answer = askQuestion();
        if (answer < 'A' || answer > 'E') {
            throw new InvalidMCQException("Invalid MCQ answer");
        }
    }
}
```

```
class TFQ extends QA {
    public TFQ(String question) {
        super(question);
    }

    void getAnswer() {
        answer = askQuestion();
        if (answer != 'T' && answer != 'F') {
            throw new InvalidTFQException("Invalid TFQ answer");
        }
    }
}

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

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

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

Sample client class:

```
class Main {
    public static void main(String[] args) {
        try {
            QA mcq = new MCQ("What is the answer to this MCQ?");
            QA tfq = new TFQ("What is the answer to this TFQ?");

            mcq.getAnswer();
            tfq.getAnswer();
        } catch (InvalidQuestionException ex) {
            System.err.println(ex);
        }
    }
}
```

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");
        }
    }
}
```