Java Workbook

01. Java and algorithms basics

~by Andrzej "Andret2344" Chmiel

Exercise 1: Quick implementation

Create a class that implements both `A` and `B` interfaces so the code compiles.

```
interface A {
    A test();
}

interface B {
    B test();
}
```

Exercise 2: Useless class

You are an author of the following class and can modify it as you wish:

```
class UselessClass {
    public static boolean check;

    public static void hello() {
        System.out.println("hELLO wORLD!!1");
    }
}
```

There are two classes that you cannot modify:

```
public class UsefulClass {
    public static boolean check;

    public static void hello() {
        System.out.println("Hello World!");
    }
}

class MainUseful {
    private static final String str = "Hello!";

    public static void main(final String[] args) {
        UselessClass.check = false;
        UsefulClass.check = true;
        System.out.println(str);
        if (UselessClass.check) {
            UselessClass.hello();
        }
        if (UsefulClass.check) {
            UsefulClass.hello();
        }
    }
}
```

Modify only the `UselessClass` class so the code prints out this:

```
Good bye!
hELLO wORLD!!1
```

Exercise 3: Object initialization

Fill in the following class:

```
class Initializers {
    private final String field1;
    private final String field2;
    private final static String staticField1;
    private final static String staticField2;

    public Initializers() {
        field2 = staticField2;
    }

    public String getField1() {
        return field1;
    }

    public String getField2() {
        return field2;
    }

    public static String getStaticField1() {
        return staticField1;
    }

    public static String getStaticField2() {
        return staticField2;
    }
}
```

So running the following program:

```
class MainInitializers {
    public static void main(final String[] args) {
        System.out.println(Initailizers.getStaticField1());
        System.out.println(Initailizers.getStaticField2());
        final Initializers initializers = new Initializers();
        System.out.println(initializers.getField1());
        System.out.println(initializers.getField2());
    }
}
```

prints out this:

```
aaa
bbb
aaa
bbb
```

Do not change any existing line of code or method, you can only add new code in the `Initializers` class.

Exercise 4: Quadratic equations' solver

Fill in the `solve` method, so it solves the square trinomial equation $ax^2 + bx + c = 0$ for given `a`, `b`, and `c`.

```
interface Printable {
    void print();
}

record Pair<F, S>(F first, S second) {
}

record Solver(int a, int b, int c) {
    public Pair<Printable, Printable> solve() {
        return null;
    }
}
```

So running the following code:

```
public class MainSquareTrinomials {
    public static void main(final String [] args) {
        final Solver solver1 = new Solver(-2.0, 3.0, -1.0);
        final Pair<Printable, Printable> pair1 = solver1.solve();
        pair1.first().print();
        pair1.second().print();
        final Solver solver2 = new Solver(1.0, 2.0, 4.0);
        final Pair<Printable, Printable> pair2 = solver2.solve();
        pair2.first().print();
        pair2.second().print();
        final Solver solver3 = new Solver(4.0, 4.0, 1.0);
        final Pair<Printable, Printable> pair3 = solver3.solve();
        pair3.first().print();
        pair3.second().print();
    }
}
```

prints out this:

```
First real solution: 1,000.
Second real solution: 0,500.
No real solutions.
No real solutions.
First real solution: -0,500.
Second same real solution -0,500.
```

Watch out for creating too many classes!

Exercise 5: Enumerators

Given the following code:

```
enum Operation {
    ADD, SUB, MUL, DIV
}

class MainMathematicalOperators {
    public static void main(final String[] args) {
        final double a = 12;
        final double b = 3;
        System.out.println(Operation.ADD.get(a, b));
        System.out.println(Operation.SUB.get(a, b));
        System.out.println(Operation.MUL.get(a, b));
        System.out.println(Operation.DIV.get(a, b));
    }
}
```

Fix the enum so the code compiles and prints out this:

```
15.0
9.0
36.0
4.0
```

Exercise 6: `Math` package

Given the following code:

```
class MainCalculations {
    public static void main(final String[] args) {
        System.out.println(f(2, 4));
    }

    public static double f(final double a, final double b) {
        return -1;
    }
}
```

Fill in the `f` method, so it calculates the result of the expression $\left|\frac{a^b}{\sqrt{a}\sqrt{b}}\right|$. You can use the word `Math` only once.

Exercise 7: Icons on the screen

Given the following interface:

```
interface IScreen {
    /**
     * Adds a new icon to the screen.
     *
     * @param x The {@code x} coordinate.
     * @param y The {@code y} coordinate.
     * @param name The icon's name.
     */
    void addIcon(int x, int y, String name);

    /**
     * Returns the information of icons on the screen.
     *
     * @return A {@code String} containing icons names separated by {@code "\n"}.
     */
    String getInfo();
}
```

Create the `Screen` class that implements this interface. Any new classes can be created only inside the class `Screen`. So running the following code:

```
class MainScreenAndMouse {
    public static void main(final String[] args) {
        final IScreen screen = new Screen();
        screen.addIcon(0, 0, "Internet browser");
        screen.addIcon(1, 0, "Java IDE");
        screen.addIcon(2, 0, "Music player");
        System.out.println(screen.getInfo());
    }
}
```

prints out the following text:

```
Internet browser
Java IDE
Music player
```

Exercise 8: A mouse on the screen

Given the following interface:

```
interface IMouse {
    /**
     * Moves the mouse by given values.
     *
     * @param x The length of the mouse {@code x} movement.
     * @param y The length of the mouse {@code y} movement.
     */
    void slideCursor(int x, int y);

    /**
     * Read what the mouse points to.
     *
     * @return The name of the icon the mouse points to, or {@code null} if the mouse
     */
    String getIconName();
}
```

Not adding any additional methods to the `Screen` class, create a class that implements the `IMouse` interface so running the following code

```
public class MainScreenAndMouse {
    public static void main(String[] args) {
        final IScreen screen = new Screen();
        screen.addIcon(0, 0, "Internet browser");
        screen.addIcon(1, 0, "Java IDE");
        screen.addIcon(2, 0, "Music player");
        System.out.println(screen.getInfo());
        /* Initialize the mouse object correctly */
        IMouse mouse = null;
        System.out.println(mouse.getIconName());
        mouse.slideCursor(1, 0);
        System.out.println(mouse.getIconName());
        mouse.slideCursor(0, 1);
        System.out.println(mouse.getIconName());
    }
}
```

prints out this:

```
Internet browser
Java IDE
null
```

Exercise 9: Triangulation

Given the following code:

```
interface Triangle {
    double getArea();

    double getPerimeter();

    String getSides();
}

class MainTriangulation {
    public static void main(final String[] args) {
        final List<Triangle> triangles = triangulate(1, 2, 3, 4, 5, 6, 99);
        for (final Triangle triangle : triangles) {
            System.out.printf(
                    "%s %d %d%n",
                    triangle.getSides(),
                    triangle.getArea(),
                    triangle.getPerimeter());
        }
    }

    private static List<Triangle> triangulate(final int... sides) {
        return Collections.emptyList();
    }
}
```

Fill in the `triangulate` method, so it returns a list of a possible to create triangles of the given sides. You cannot use the keyword `class` for this purpose. So, running the above code prints this:

```
(2-3-4) 2.9047375096555625 9.0
(2-4-5) 3.799671038392666 11.0
(2-5-6) 4.683748498798798 13.0
(3-4-5) 6.0 12.0
(3-4-6) 5.332682251925386 13.0
(3-5-6) 7.483314773547883 14.0
(4-5-6) 9.921567416492215 15.0
```

Remember that one side cannot be used twice!

Exercise 10: Unobvious object creation

Not changing any method in the `SimpleObject` class and not creating any additional constructors, fill the class, so it is possible to create its objects. Don't use the reflection mechanism.

```
class SimpleObject {
    private final String name;
    private final int number;

    private SimpleObject(final String name, final int number) {
        this.name = name;
        this.number = number;
    }

    @Override
    public String toString() {
        return "SimpleObject{" +
                "name='" + name + '\'' +
                ", number=" + number +
                '}';
    }

    // HERE
}

class MainSimpleObjectCreation {
    public static void main(final String[] args) {
        // Fix the following line
        final SimpleObject o = null;
        System.out.println(o);
    }
}
```

So running the code above prints out:

```
SimpleObject{name='John', number=100}
```