Exercises module 20 – Interfaces

20.1 – Interfaces

1. In this exercise, we will continue on exercise 19.1

2. Create a new project and copy the code from exercise 19.1

3. Replace the use of the abstract class with an interface instead.

   As you notice we can't include the fields as part of the interface, why?

   See this question
   https://stackoverflow.com/questions/9446893

4. Interfaces should only focus on the behaviour not the private or public fields on a class that's why we can only include the **Describe()** method as part of the interface contract.

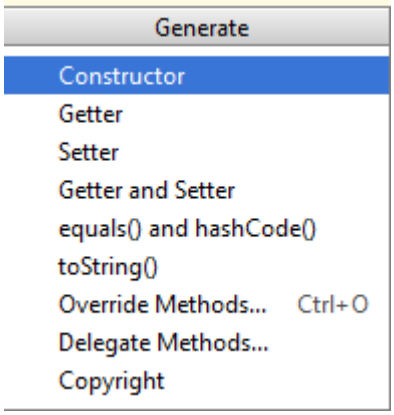
   The **id** and **productName** fields needs to be copied into each class instead, do this operation so that the code will run and print out the same result as in the previous exercise.

5. Initialization of the objects using the code below works fine, but often leads to bugs because you might forget to set important fields.

```
Processor cpu1 = new Processor();
Processor cpu2 = new Processor();
Keyboard kbd = new Keyboard();
Monitor monitor = new Monitor();

cpu1.productName = "Intel Core I7";
cpu1.id = 1;
cpu1.frequency = 3000;
cpu2.productName = "Amd threadripper";
cpu2.id = 2;
cpu2.frequency = 3500;
kbd.productName = "Microsoft Natural keyboard";
kbd.id = 3;
monitor.productName = "Samsung T191T";
monitor.id = 4;
```

6. Use the built-in refactoring tool in IntelliJ to code-generate a constructor by pressing **alt+insert** (inside a class)



7. Can you define the constructor in the Product interface?

   see "Constructor in an Interface?"
   https://stackoverflow.com/questions/2804041

8. Make the fields private and fix the code initialization to use the constructor instead.

20.2 – Improving the shapes using shapes

In this exercise, we are going to replace the abstract class with an interface.

1. In the previous module we made a simple drawing application using an abstract Shape base class.

2. Create a new project and copy the code from the preview module.

3. Rewrite the abstract Shape class and use an interface instead.

   Regarding naming of interfaces, read:
   - **Java Interfaces/Implementation naming convention**
     http://stackoverflow.com/questions/2814805
   - **Interface naming in Java**
     http://stackoverflow.com/questions/541912
   - **Should interface names begin with an "I" prefix?**
     http://programmers.stackexchange.com/questions/117348

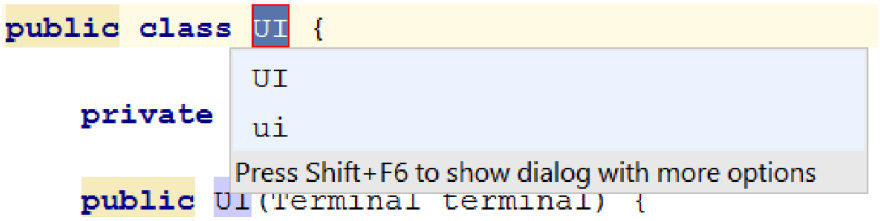
4. Fix the remaining code so that it works as before, run it!

5. As you see the difference here is small between an abstract class and an interface. Read more about the differences here:

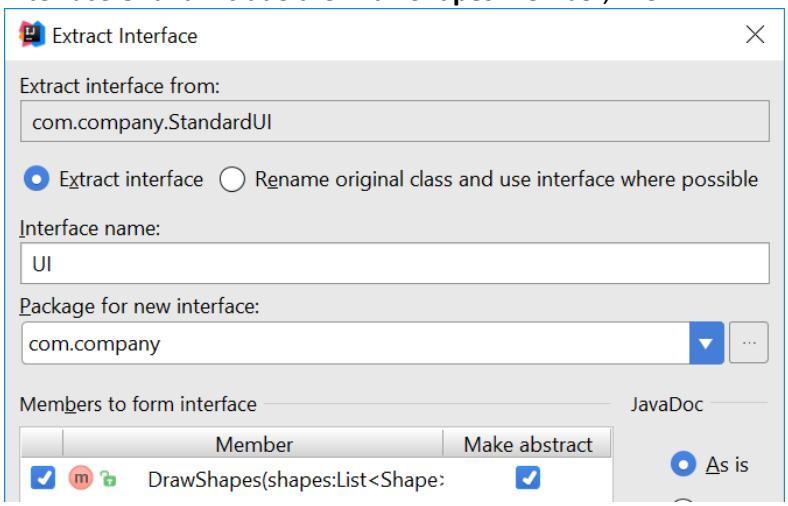
   Abstract class vs Interface in Java
   http://stackoverflow.com/questions/10040069

6. We have a UI class in our application that we need to create an interface for.
   a. First rename the **UI** class to **StandardUI,** to do this we use the built in rename feature by right clicking on the UI name and selecting **Refactor -> Rename**

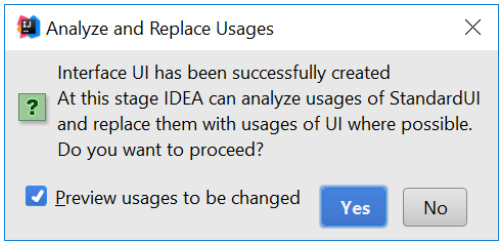
      If you see this on the screen then you can just start typing **StandardUI** and the code will automatically be renamed on the fly.



7. Then right-click inside the **StandardUI** class and select **Refactor -> Extract -> Interface.** Name the interface **UI** and include the **DrawShapes** member, like:



If this appears, then select **no**.



8. Review the newly created UI interface and make sure it looks good.

9. Create a new class called ZoomedUI that implements the UI interface. Copy the code from the StandardUI.

10. In the drawShapes method (in the ZoomedUI class) we will change how we display the shapes by simulating a zoom effect, use this code to plot a point on the screen:

```
int x = point.x*2;
int y = point.y*2;
terminal.moveCursor(x, y);
terminal.putCharacter('O');
terminal.moveCursor(x+1, y);
terminal.putCharacter('O');
terminal.moveCursor(x, y+1);
terminal.putCharacter('O');
terminal.moveCursor(x+1, y+1);
terminal.putCharacter('O');
```

Basically, we make each point a 2x2 block on the screen.

Try the new ZoomedUI in your main method by running this code:

```
UI gui = new ZoomedUI(terminal);
gui.drawShapes(shapes);
```

(Do notice the type of the gui variable)

11. The end result should look something like

The main benefit here is that we can write different rendering implementations and in a flexible way change how we display the shapes on the screen.

In modern development, we will usually use interfaces more often than inheritance.

Further improvements

- We could use the UI interface for something more useful than we do right now
- Add more shapes to the system
- Support dynamic zoom levels
- Use colors

Questions and concepts to study further on your own:

- Abstract vs Interfaces
- Can an interface have a constructor?
- Default methods in Interfaces and is it a good idea to use?
  https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html
- What is the difference between loose coupling and tight coupling in the object-oriented paradigm?
  https://stackoverflow.com/questions/2832017
- Interface vs Base class
  https://stackoverflow.com/questions/56867