Dmitri Stanchevici
Unit 6
Module 2

Ex. 2.2

Here are the compile errors:

```
StaticExample2.java:22: error: non-static variable y cannot be referenced from a static context
        y = 9.67;
        ^
StaticExample2.java:23: error: non-static method printy() cannot be referenced from a static
context
        printy ();
        ^
2 errors
error: compilation failed
```

I wonder what makes the context static. Is it the presence of main()?

Ex. 2.6

I wonder if I have identified the disadvantage correctly: (1) always maintaining storage space on the heap and (2) the user has to remember to use the same pointer to evaluate different Strings.

If in a fully static class we store a static String, like this

```
static String classSentence;
```

then we can define and call the static methods of this class without parameters, like this

in SentenceToolExample's main()

```
SentenceTool2.classSentence = "Colorless green ideas sleep furiously.";
System.out.println (SentenceTool2.startsRight() + "  " + SentenceTool2.endsRight());
```

in SenteceTool

```
public static boolean startsRight ()
{
        return (Character.isUpperCase(classSentence.charAt(0)));
}
```

The disadvantage is that such a static variable is active in the global memory, and once it is assigned (it is made to point to) a String, this String object is maintained on the heap at all time

during the program's execution. The String itself (in our case, a sentence) may change, but it will be maintained by being pointed to by the static variable in the global memory.

The user has to remember to change the String (sentence they want to evaluate) by assigning this new String to the same pointer:

```
SentenceTool2.classSentence = "try not to ever split infinitives";
```

Ex. 2.10

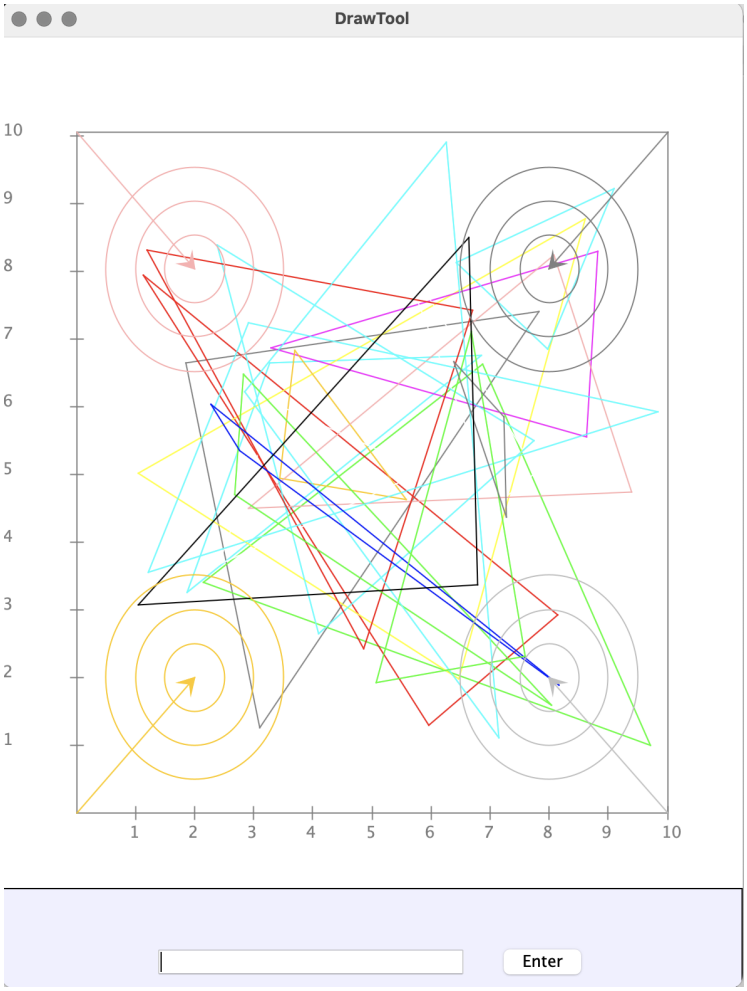
In DrawTool.java, most or even all **public** methods for drawing geometric objects are static. here is one example of such a method's signature:

```
public static void drawRectangle (double x1, double y1, double width, double height)
```

However, in the bottom quarter or third of the code, I see many methods for drawing geometric objects that are not modified with either **static** or **public**, such as

```
void drawPoint (Graphics g, DrawObject p)
```

Ex. 2.14



Ex. 2.15

- In a subclass, you can **override a parent's protected method.** -- Yes, if the overriding method is either protected or public (that is it provides greater visibility than the parent's method does), but no if the overriding method is default (no visibility keyword) or private.

- In a subclass, you can **override a parent's public method** and **make it protected in the subclass.** -- No. An error is thrown:

```
VisibilitySubclass.java:10: error: printPublic() in VisibilitySubclass cannot override
printPublic() in VisibilityParent
    protected static void printPublic ()
                          ^
  attempting to assign weaker access privileges; was public
1 error
```

- In a subclass, you can **override a parent's public method** and **make it private** in the subclass. -- No. The same error is thrown:

```
VisibilitySubclass.java:10: error: printPublic() in VisibilitySubclass cannot override
printPublic() in VisibilityParent
   private static void printPublic ()
                       ^
  attempting to assign weaker access privileges; was public
1 error
```