Pr301 Advanced Programming assignment 2

Bad Smell 1 – Duplicate code in TurtleDrawer class.

This Bad Smell is of code duplication – the program performs the same two actions in multiple places.  
lines 47&48, 57&58, and 77&78 are two identical calls with slightly changed parameters

Refactoring strategies – Extract method seems most appropriate here

Bad Smell 2 – Long method in TigrParser class – Refactor cycle 1.

This bad smell is of the Long Method type – the method parse() within Tigrparser is ~50 lines long, well over the recommended 10ish. It also contains multiple sections where comment explanation is required. Lines 22-71

This method is long and complex – is likely to require several refactoring strategies. (Or possibly several rounds of refactoring.

Decompose Conditional and Extract method seem the most appropriate here.

Decompose conditional and extact method are somewhat hard because of the requirement to track line\_number throughout the method (for error returning).

Bad Smell 3 – Temporary Fields in Parse method in TigrParser

This bad smell is of the temporary field type – the method creates three temporary fields – self.source, self.data, and self.command. These three variables can easily be tracked as local variables within the function.

Most effective solution to this seems to be Replace temp with Query – which will also assist in correcting bad smell #2

After correcting the bad smell one of the unit tests failed. The test\_source\_valid test is however a white-box test because it is validating an internal field within the class – one of the temporary fields that has been removed by this refactoring process. The failure of this test is therefore deemed reasonable

Shotgun surgery – Error handling – Refactor Cycle 2

This bad smell is of the Shotgun surgery type. The error handling behavior is scattered throughout two classes.

The most effective solution to this seems to be Extract class – centralizing this behavior within an error handling class, allowing it to conform to the single responsibility principle