Joe Skimmons, jws2191

Assignment #4

1.1 Rod Johnson states that checked exceptions can fit into three categories:

1. Minor exceptions that are “basically secondary return codes”, in which checked exceptions should be used.

2. Major exceptions, where something “‘went horribly wrong’”, in which unchecked exceptions should be used.

3. The“middle ground”, where only a small number of times the exception result in the program being saved by the caller, in which unchecked exceptions should be used.

Johnson decides to treat the second and third cases as almost the same, using unchecked exceptions because the majority of callers will not be able to save the program.

The first two categories are well defined because they are black and white situations. But the middle ground scenario is a grey area and may not always look the same, requiring discretion from the programmer. ("Java Theory and Practice: The Exceptions Debate.")

Exceptions in general, including unchecked exceptions, can be made more transparent through user being given control over if an exception is checked or unchecked. This will end the debate over checked exceptions because those who do not agree can simply “uncheck” the offending exceptions. Eclipse can have an extension that allows the user to decide which Exceptions should be checked. This would even allow unchecked exceptions to be checked, giving the developer more control over his program. Also Eclipse could return the snippet of code that is causing the error, checked or unchecked, in order to make debugging more

transparent and seamless. Finally, increased documentation could make exceptions more transparent in general.

1.2 Possible missing constructors are:

Rectangle (Point P, int width, int height)

Rectangle (int x, int y, Dimension D)

Rectangle (Point p1, Point p2, Point p3)

Rectangle (Point p1, Point p2, Point p3, Point p4)

Rectangle (int x, int y)

Rectangle (int i) (This one does not exist because it is too vague, there are four possible int values for i to be, so it would not make an effective constructor)

Rectangle (x1, y1, x2, y2)

Some of these constructors are missing because they are pointless, or counter-intuitive. For example, four points are redundant when three points could be used, or one point and a dimension. Other constructors, like Rectangle (int x, int y, Dimension D) or Rectangle (Point P, int width, int height) are also redundant because given an x and y value where is simple to create a point object, or a dimension object from a width and height. Those constructors are simplified, "zipf-ed", and preferred enough that the API will only accept them.

1.3 The protected modifier should be used when a class is being written for inheritance, meaning that you will have multiple subclasses written that need to access that variable or method. This will allow the "children" to use the information, but will not allow classes outside the package to view them. An example of a language that does not use these modifiers is python.

1.4 A "null object" would be allocated space for an object that is not yet filled by that object. The getWeight() and toString() methods would just return null because there is nothing there. In the real world, this could be represented as literally space set aside for that object, such as space to put a box or a lamp. Moving this object would be making space for this object in another location, to be filled later by that box or object.

1.5 The values() method is implicit and returns all of the declared values in the Enum. For example, if you were trying to list all of the sizes of a pizza for a customer.

The valuesOf() method returns a single Enum value. For example, if you wanted to print a single pizza size you would call this method.

The ordinal() method returns the index in the enum that a value has. For example, a small pizza would have an ordinal of 0, a medium would be 1, etc. The name method returns the name of the enum constant as it was declared in its enum declaration. For example, it can be used to return the name as a string for use in a pizza menu.

Values() is not in the API because it is an "implicit" method, carried out in the background by the compiler. It is used to carry out the valuesOf() method and in its description in the API. It is similar to the "for each" concept because it is implicit and used in the

background of another method for “fake” iteration, similar to how ArrayLists and Arrays are interated through without a “true” iterator, like the one used by a LinkedList.

1.6 In submission folder

1.7

```
public interface Expression(){
        public *DataType* evaluate();
}
```

See UML diagram in theory folder of submission

NOTE: the *dataType* represents floats, longs, ints, doubles, shorts, etc. This is because it would be redundant and “overkill” (as stated by Professor Kender on Pizza) to list every possible combination.

Works Cited

"Design Patterns and Refactoring." *Design Patterns & Refactoring*. N.p., n.d. Web. 29 Nov. 2016.

"Enum (Java Platform SE 7)." *Enum (Java Platform SE 7)*. N.p., n.d. Web. 29 Nov. 2016.

"Java Theory and Practice: The Exceptions Debate." *Java Theory and Practice: The Exceptions Debate*. N.p., n.d. Web. 29 Nov. 2016.

"Rectangle (Java Platform SE 7)." *Rectangle (Java Platform SE 7)*. N.p., n.d. Web. 29 Nov. 2016.