1.1 API Familiarity

Constructors: In both Java and Eclipse, the constructors for the List class create a scrolling list where the developer can put objects selectable by the user. In the java.awt.List API, the List constructor has three forms, one default, one taking only an integer, and a third taking in an integer and a boolean. The default constructor creates a list with no multi-line selection and no visible lines in the list. The second constructor takes in a single integer determining the number of visible lines, and the third constructor takes in a boolean in addition to the integer, which determines if multi-line selection is enabled.
The Eclipse API, in comparison, only has one constructor that takes in a Composite parent object and an integer. The parent is a control that can be chosen by the developer, and the integer represents a style constant from the SWI class that styles the list.

Methods:
public void add(String item)
public void deselect(int index)
public void select(int index)

These methods appear to do the same thing because the APIs and usage of the List class in both Java and Eclipse are very similar. Both List objects fulfill similar purposes and are written in the same language, so it would make sense that they have very similar methods in them.

Field:
The fields from the Eclipse List class and the Java List class were both inherited from parent classes. Eclipse inherited them the Control class, while Java inherited them from the Component and ImageObserver classes.

Deprecated:
Many methods from each class are deprecated. They were updated with a new method that could override them, and became obsolete and were deprecated. For example, addItem(), allowsMultipleSelections(), clear(), were all deprecated in the Java class in favor of updated methods.

1.2 Hello World Again

```
public class HelloUser(
        public static void main (String[] args){
                NameMaker myNameMaker = new NameMaker(“World”);
                String myName  = myNameMaker.whatToSay();
                System.out.println(“Hello” + myName);
}
}

public class NameMaker{

private String name;
private String title;
private String local;

        public NameMaker(String in){
                name = in;
        }

        //default “natural” constructor
        public NameMaker(){
                name = “World”;
        }

        //constructor that allows for a title
        public NameMaker(String t, String in){
                title = t;
                local = in;
                name = title + “ “ + local;

        public String whatToSay(){
                Return name;
        }
```

1.3 Arrays, ArrayLists, LinkedLists

1. The post office use is similar to an array to sort the mail because they have a fixed number of containers, much like an array is instantiated with a fixed size.
2. The train use is similar to a LinkedList because the train needs to be able to insert boxcars and take them out from any position depending on where they are going. This is similar to how objects and data types can be inserted into any position on a LinkedList.
3. The stapler is similar to an ArrayList because objects are always added to one end of the list in both cases, and as they are used or deleted, every other object shifts down one spot.

A laundromat is similar to an Array because each one only has a fixed number of machines to wash and dry clothes in.
The luggage security check in an airport is like an ArrayList because there is not a fixed number of bags, and they are added to one end and removed from another.
The baggage claim at an airport is similar to a LinkedList because there is not a fixed number of bags, and they can be removed from the line at any time without affected the other bags.

1.4 Conflicting Design Requirements

1. The rent (cost) of the apartment, which is like the ease of learning the syntax.
2. Size of the apartment is like the heap size of java or the memory footprint.
3. Security of the apartment is like the Security mechanisms active in java.
4. Design of the apartment, which is like the user interface of different IDEs.
5. Location is like the portability of Java to different systems.
6. Furniture (Appliances) are like the ADIs or class libraries.

1.5 Bad User Requirements

Can you clarify what you mean by “robot pizza thing”? When a customer enters, do you want them to prompt the machine to order, or should the machine recognize that the customer is there and prompt them? In what order should it ask for the pizza, toppings, and drinks? All at once, or at different times? Should the machine accept multiple orders at once, or should it take an order, cook the food, and then take another order? You mentioned that you want the pizzas cooked 3 or 4 at once, would you like to pick one of the numbers? In the use case, it is said that the machine should fold the boxes and “do” the drinks at the same time, is this hyperbole or should every process take place at once? When the machine accepts the money, is there a limit to what type of money should be accepted (change, $100 bills, etc.)

Works Cited

By Default, There Are Four Visible Lines and Multiple Selections Are. "List (Java Platform SE 7)." *List (Java Platform SE 7)*. N.p., n.d. Web. 27 Sept. 2016.

"Help - Eclipse Platform." *Help - Eclipse Platform*. N.p., n.d. Web. 27 Sept. 2016.

"Stack Overflow." *Stack Overflow*. N.p., n.d. Web. 27 Sept. 2016.