Benjamin Northrop

CS5004

4/10/24

Spring 2024

**Woodshop Game**

I found inspiration for this originally in Lab 3 – the automated house factory. I have a woodworking shop in my garage – it’s a combination of tools from my Dad and things I’ve picked up over time. It’s a hobby, however, due to family obligations and work, I don’t have much leftover time to spend in there. As a result I’ve fruitlessly searched for “woodworking games” that I could dabble in for a few minutes here and there.

This project takes inspiration from the idea of being able to use each tool as an object and work on other objects. It primarily works with the idea of your “workshop” being the model of the program – it holds all of the internal details, inventories, etc., held on the left side of the attached UML diagram. This workshop is composed of several collections – you have your tool collection, a jig1 collection, your wood collection, your built furniture collection, and your selection of blueprints to build from.

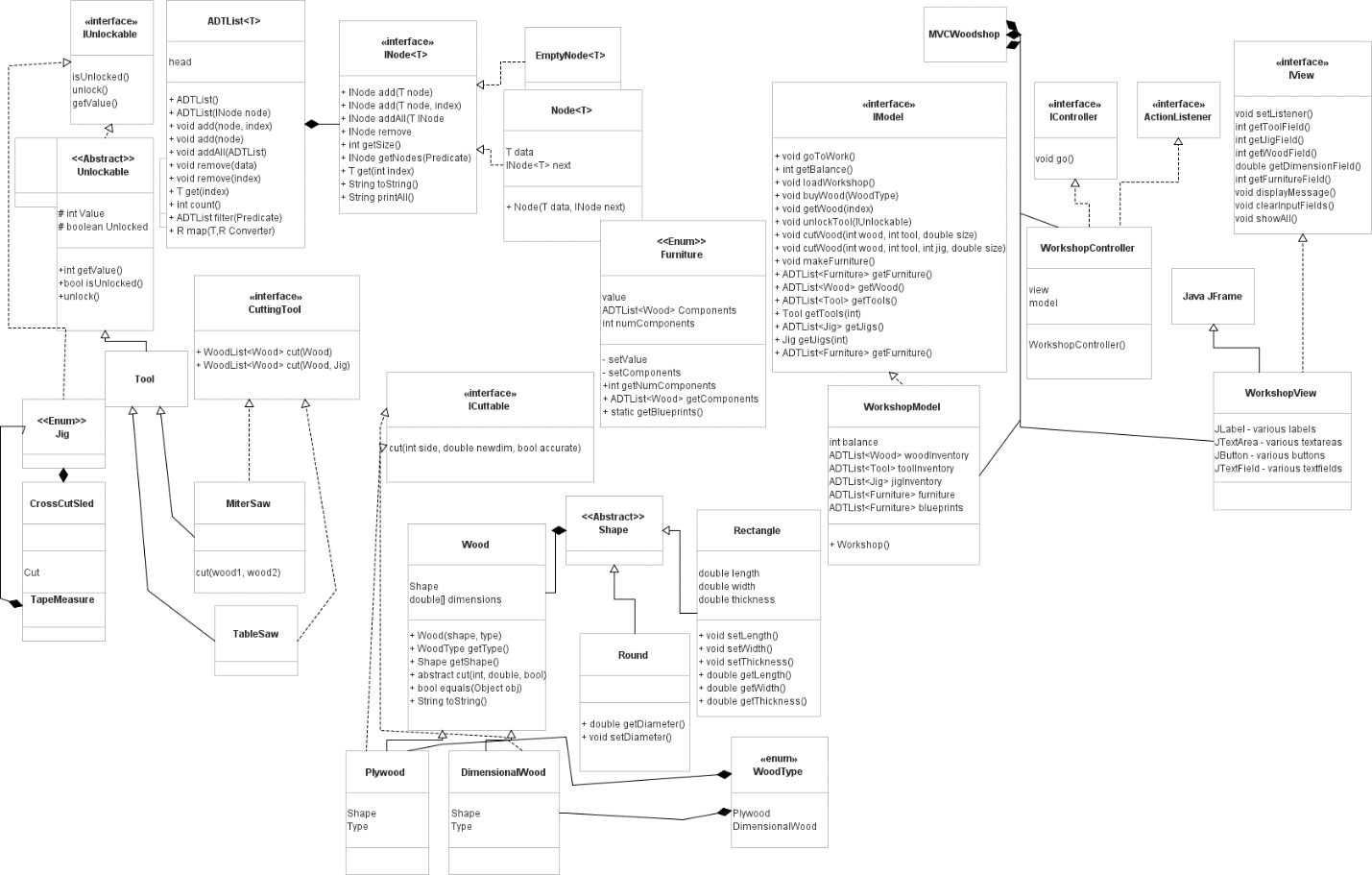
The premise is, you go to work to get enough money to “buy” or unlock items and make them available to you in your collection. You can buy wood, tools, or jigs. Wood comes in two forms – it is either plywood or dimensional wood2. Purchasing a tool will “unlock” it and allow you to use it to cut wood. Wood is considered a “cuttable object”

Tool have certain properties that determine their purpose. Table saws are meant for rip3 cuts, and as such it will always cut “parallel” to the longer dimension of the piece of wood we are cutting. (When speaking of dimensions, we’re only going to refer to the length and width for now, as thickness is going to be a set dimension for this application). Miter saws are meant for “cross cuts”4 and can only cut parallel to the shorter dimension, and they have a maximum reach of 12”. Therefore, we cannot cut across a piece of plywood (48” wide) until it’s been cut down into smaller strips on the tablesaw.

Jigs allow us to do some change. A cross-cut sled used with a tablesaw allows us to do cross-cuts on it, therefore we can cut either dimension of a piece of wood. The application will default to the crosscut when a jig is used, and a rip cut when no jig is used.

*Note: A feature that has not yet been incorporated, but the groundwork has been laid out, is the miter saw has an accuracy factor. It does not have a built in measuring system like the tablesaw, so a miter saw would introduce a small variation to the desired dimension (due to eyeballing the distance) until it is used in conjunction with a tape measure, which would provide accurate cuts.*

After cutting wood to the correct dimensions, we can built furniture. We choose the furniture we want to build, and the application will compare our inventory to the components required in the furniture item chosen, and only create the furniture if all components are there. This saves it in our inventory and removes the wood from our inventory.



**Concept Map**

|  |  |
| --- | --- |
| **Concept 1: Recursion in Practice** | Recursion is used in each of the ADTList instances inside the model. The primary one that gets manipulated is the wood inventory list. We primarily use this to cut wood, initiated by the model by with the method cutWood() (**WorkshopModel.java Line 146).** This method takes the inputs from the controller, searches for the appropriate tool, jig, and wood in its respective ADTList, then passes those parameters to the tool for the correct procedure. That tool then passes it along to the wood, which actually does the transformation. The code for the actual get recursion is in **ADTList.java Line 76, Node.java Line 29, EmptyNode.Java Line 134.**  It is also used in the makeFurniture (**WorkshopModel.java line 187)** process, in that once we receive the required information from the controller, the model will determine what piece is being made, verify the required pieces are in the inventory, then iterate through the blueprints’ components **(Furniture.java)** to systematically remove the wood pieces from the inventory. This removal combines the recursive get method with an equality check to ensure it only removes the correct piece. |
| **Concept 2: Logical Structure/Design using Abstract Classes and Interfaces** | This design heavily incorporates interfaces and abstract classes to allow for expansion and different variations in the tools. Tools are broken down into its abstract class for its instance variables and interfaces for what it can do. A generic tool is abstract, (**Tool.java)** because it’s just the collective term, but all tools in this game have a value and a locked status. Tools have different purposes – for example, saws can cut (**ICuttingTool.java),** but a table saw and a miter saw cut differently. Therefore, we have different implementations for the two of them.  Each portion of the MVC is abstracted out to its interface, **(IModel, IView, IController**), so each piece can be replaced by another MVC component with minimal issues.  Wood, in particular, uses abstraction and interfaces as well, as wood itself is an abstract class, since there are different types of wood **(Wood.java)**, we have them broken out to plywood and dimensional wood. This lets us expand down the road, to greater varieties of wood.  The methods of these two are dictated by the **ICuttable.java** , which has just one method – cut – which takes an index for which dimension to change, the new measurement, and a Boolean on the accuracy of the cut. This accuracy portion will be later incorporated, as explain below in the *Areas for Improvement/Expansion*. |
| **Concept 3: Useful/Logical Abstractions using Generics and Lambda Expressions** | The primary means of abstraction and interaction comes from the ADTList being used by 4 different types of objects – Wood, Tools, Jigs, and Furniture (**WorkshopModel.java line 17-21)**. The reason for this is because the idea is the game would have a number of items available to unlock and then use, so we store each of these inventories in its own separate list so that it can be accessed quickly and easily. The primary class that dictates this is ADTList and it implements the **IADT.java** interface, allowing us to swap out our ADTList with any other datatype we want. This gets used by the **WorkshopController.java line 37** as it sends the data to **WorkshopView.java line 280** for the user to see.  Lambda expressions are used in **WorkshopModel.java line 195** as a means to verify the wood inventory has the sufficient materials in order to build the furniture that is being requested. It goes through the different component arrays that make up each piece, compare it to the woodInventory, and then gets summarized. If it passes this check, it will build the furniture and remove the materials from our wood inventory. |
| **Concept 4: Higher Order Functions** | The primary higher order function used in this program is a **fold** method that is used under the method name **count(Predicate)** contained in **IADT.java line 20, ADTList.java line 131, Node.java line 149.**  **WorkshopModel.java line 192**  This method, when called, iterates through each piece of wood in the wood inventory, compares the dimensions of the wood to the dimensions in the constants file, and adds one to the count if it finds a match. After going through the entire inventory it returns the number of pieces that matched the required dimensions. |
| **Concept 5: Hierarchical Data Representation or Linked List ADT** | The various inventories are held within Linked List ADTs, which are contained in the **ADTList.java** class and implement the **IADT.java** interface. When we first initialize the model **WorkshopModel.java Line 28**, it creates two instances for the tool and jig inventories, and fills them in with all the tools we have available **WorkshopModel.java line 83**. That is because all tools and jigs are visible, but we must *unlock* them to actually use them.  Buying wood, adds wood to this inventory **Line 103**, which then calls the ADTList’s add() method, which in turn pushes it down to the root node. Node will add this item to the back of the list via recursion.  When cutting wood, we don’t just throw the cutoff away, it actually becomes in theory, two pieces of wood. To handle this, cutting a piece of wood will create a new ADTList, add a new Node containing the new piece of wood, and return that list back in **Plywood.java Line 44**. The model handles this by having a means of appending the returned list to the original using addAll **Node.java line 91.** This will add the head of the new list to the back of the original list, affectively appending it to the back. |
| **Concept 6: MVC Design Pattern** | This design incorporates an MVC design, with the files **WorkshopModel.java, WorkshopController.java, and WorkshopView.java.** Each component implements its own respective interface. The Model is the program, and contains the instance variables to hold the various lists. It also provides the user interface for the controller to send commands to the model. The view will show all of the data, as provided from the model via the controller, and it implements a GUI to interact with the user and notify the controller of click events.  At the beginning, the MVCWorkshop driver instantiates the three components, and calls the controller’s Go method. This go drives the controller to tell the model to load the workshop (for the initial configuration), and then show the current status. It then waits for an action by the view.  **WorkshopView.java** The view is a single JFrame window broken down into 12 panels. These panels are arranged in a 3x4 rectangle and should be maximized for maximum visibility. It gives the user the overlay of what they have in the workshop, as if you were standing in your garage looking at all your tools, jigs, and wood.  Pressing a button such as GoToWork sends an action event from the View to the Controller, which then decides what to do and attempts to do it. The entire process is wrapped in a try/catch statement, so that any errors can be displayed to the user without crashing the program. **WorkshopController.java line 54.**  The Controller sends the command to the model for computing, then calls for a new set of state information from the model and displays it again to the user. This program continues running until the user selects the Exit button or X’s out of the window. |
| **Concept 7: SOLID Design Principles** | **Single Responsibility – Unlockable.java, ADTList.java, Node.java** The Unlockable class is an example of single responsibility because it’s only purpose is to determine/control the unlocked/locked status of an unlockable item. The Tool abstract class extends it, and the jig enum implements it.  **Open/Closed – Tool.java** The tool method stays very generic – in that it doesn’t care if it’s a cutting tool, smoothing tool, etc. Therefore, it pools together all of the required information (value, unlocked status) but then allows subclasses to extend this and do what it wants. We use interfaces like **ICuttingTool.java** to allow us to change the behaviors of subclasses.  **Liskov Substitution – TableSaw.java Line 36** Cut method() only requires a Wood object, so it can take in either a plywood or dimensional wood object. It doesn’t matter, and will handle both of them equally. The saw only determines the side to cut, but the actual “cutting” method gets sent to the specific wood object. Therefore, either the dimensional or the plywood object could be calling the Cut method from the same tablesaw call. Because they both implement the same ICuttable interface, then both can handle this call appropriately.  **Interface Segregation – Tool.java / TableSaw.java / MiterSaw.java** Both of these tools fall under the Tool abstract class, but they are both cutting tools, so they use the cutting tool interface. For expansion, we can include another interface for smoothing surfaces – like sanding machines or a router. This would still be a tool, but can employ different methods to meet the sanding interface.  **Dependency Inversion** – **Wood.java line 12 / Plywood.java line 16 / Shape.java line** The wood can be of multiple shapes, right now the program has support for Rectangle or Round shapes, both of which extend the Shape class. Wood takes a Shape in its constructor as a composition, but when don’t get specific until we talk about a Plywood constructor by itself. This is because we’ve created a standard Plywood shape (which are rectangular sheets). Plywood line 32 also shows that it can dynamically choose the shape right now by determining the number of dimensions provided – if a constructor only provided 2 dimensions, it assumes it’s a diameter and thickness. If it has 3 dimensions, it assumes length, width, and thickness. This would allow future expansion to have a tool cut circles out of rectangles, and we can then change the shape from Rectangle to Round. |
| **Concept 2 Extension** | In order to ensure only certain items were selectable, the program has extensive use of enums (**Furniture.java, Jig.java, WoodType.java)**. The point of this was not just to dictate the names of these objects, but to also give them some use. Furniture and Jig both are used as classes and have methods within them, as well as instance variables. In the case of a Jig, it didn’t make a lot of sense to create an object class out of a Jig, because it was only ever going to be used as a parameter of a saw’s cut method. Therefore, we encapsulated the jigs in an enum so there were only 2 to choose from. (Line 12). As a result, a saw could take any Jig class as an argument, but the saw itself would determine whether or not the jig was compatible with it.  To note, the Jig class implements IUnlockable (line 11) and is the driving factor for having unlockable not just be an abstract class. I learned that Enums cannot extend other classes, so the only way to extend the lock/unlock capabilities was to bind it to the IUnlockable interface. This actually helped in having the model unlock via the interface rather than just the class.  Future expansion of the program could allow undesirable things to happen – such as breaking the jig, causing it to lock again so you buy a new one. Or destroying the wood that you were trying to cut.  **Furniture** blueprints were an even greater challenge. Trying to figure out how to verify that I had 1) the correct wood pieces to build a piece of furniture and 2) didn’t count duplicates of the same piece when verifying it, I decided to encapsulate it all into the Furniture class. This acts as a kind of “Constants” file for the composition of each piece of furniture. **Furniture.java line 38** They have a constructor that adds a “sample” of each type.  Unfortunately, this likely violates the Open/Closed principle. To refactor this, it would probably be easiest to turn furniture into an abstract class, then create individual classes of each type of furniture that extends the furniture class. This would allow us to load the workshop with one instances of each object as the “blueprints” and not worry about this code specifically. |

Areas for improvement/Expansion

There are a few areas that I wanted to note that have been identified as deficient, or incomplete in this model. These are known limitations that can be addressed in a future time.

* Currently, nothing cuts or creates a Round. Originally, there was a router tool that could cut round shapes given a template. However, due to time constraints, this tool was removed from the model for the time being to be readded later.
* The miter saw currently does not have the accuracy factor built in. This currently means the MiterSaw class isn’t 100% compliant with the Open/Closed principle, as it’s an incomplete implementation. We’ll need to add the last bit to adjust the dimension by a small factor to account for an “inaccurate cut.”
* Not everything has been completely abstracted out. There are a few methods that still use ADTList<type> specifically rather than the IADT interface as its provided or returned list. This will be fixed in a future implementation.
* Map and Filter were not included. There was an opportunity to use filter for making furniture, but I opted to do this fold/count method instead. Map also did not have a prime opportunity for implementation. Ideas for use of the map would be to include a total value of all of the tools we have unlocked. It would look like.

Glossary:

1jig – an item that works in harmony with a tool to either allow it to do something it wasn’t originally designed for, or to improve the performance of the tool.

2Plywood / Dimensional – Plywood are thin sheets of wood glued together in alternating grain patterns. It provides strength, rigidity, and stability at low cost. Dimensional wood is a single piece of solid wood, typically with a dimensional measurement like 2x4. It looks better and is easier to work with in some manners.

3rip cuts are where you cut parallel to the grain direction, so you can make long, narrow strips. Typically this coincides with the long dimension of a piece of wood.

4cross cuts are when you cut perpendicular to the grain. This typically cuts along the shorter dimension of wood.

I understand that my learning is dependent on individual effort and struggle, and I acknowledge that this assignment is a 100% original work and that I received no other assistance other than what is listed here.

Acknowledgements and assistance received:

I did not use generative AI in any form to create this content and the final content was not adapted from generative AI created content.

I did not view content from any one else’s submission including submissions from previous semesters nor am I submitting someone else’s previous work in part or in whole.

I am the only creator for this content. All sections are my work and no one else’s with the exception being any starter content provided by the instructor. If asked to explain any part of this content, I will be able to.

***By putting your name and date here you acknowledge that all of the above is true and you acknowledge that lying on this form is a violation of academic integrity and will result in no credit on this assignment and possible further repercussions as determined by the Khoury Academic Integrity Committee.***

|  |  |
| --- | --- |
| Name: Benjamin Northrop | Date: 4/11/24 |