Vocabulary of links (for easy reference if you forgot where to find the information; you can just start with Day 1 without reading any of these)

- abstract classes link
- arrays arrays
- char what you can do with a char (via the Character class) Character(java API)
- classes fields/attributes classes classes and objects
- colors in console link
- comments link
- constructors link
- data types link
- enums link1 link2 (methods/constructors in enums
- generics link
- if/else / choice link
- inheritance extends calling the superclass constructor
- input from user link
- interface link
- Map (dictionary) in Java link
- methods link1 link2
- operators link
- printing to screen link1 link2 nice representation of objects
- records link
- repetition/loops do-while-loop for-loop foreach loops recursion
- static and final variables link
- strings (equality) link
- switch (choice between more than 2 options) switch stamenents switch expressions
- variables link

Day 1: Challenge "Hello, World!"

You open your eyes and find yourself face down on the beach of a large island, the waves crashing on the shore not far off. A voice nearby calls out, “Hey, you! You’re finally awake!” You sit up and look around. Somehow, opening your IDE has pulled you into the Mysterious Realms of Java (not related to the (uncopyrighted) name of a certain island in Indonesia), a strange land where it appears that you can use Java programming to solve problems. The man comes closer, examining you. “Are you okay? Can you speak?” Creating and running a “Hello, World!” program seems like a good way to respond.

Objective:

- Create a program in your favorite Java IDE that prints "Hello World!" to the console, and run it.

To study if you find this difficult yet:

- W3 schools Java Tutorial Intro

Day 2: 1/3 - Challenge What Comes Next

The man seems surprised that you’ve produced a working “Hello, World!” program. “Been a while since I saw somebody program like that around here. Do you know what you’re

doing with that? Can you make it do something besides just say ‘hello’?”

Build on your original Hello World program with the following:

Objectives:

- Change your program to say something besides “Hello,World!”

Day 2: 2/3 - Challenge The Makings of a Programmer

The man, who tells you his name is Ritlin, asks you to follow him over to a few of his friends, fishing on the dock. “This one here has the makings of a Programmer!” Ritlin says. The group looks at you with eyes widening and mouths agape. Ritlin turns back to you and continues, “I haven’t seen nor heard tell of anybody who can wield that power in a million clock cycles of the CPU. Nobody has been able to do that since the Uncoded One showed up in these lands.” He describes the shadowy and mysterious Uncoded One, an evil power that rots programs and perhaps even the world itself. The Uncoded One’s presence has prevented anybody from wielding the power of programming, the only thing that might be able to stop it. Yet somehow, you have been able to grab hold of this power anyway. Ritlin’s companions suddenly seem doubtful. “Can you show them what you showed me? Use some of that Programming of yours to make a program? Maybe something with more than one statement in it?”

Objectives:

- Make a program with 5 `System.out.println` statements in it.
- Answer this question: How many statements do you think a program can contain?

To study if you find this difficult yet:

- W3 schools Java Output/Print

Day 2: 3/3 - Challenge Consolas and Telim

These lands have not seen Programming in a long time due to the blight of the Uncoded One. Even old programs are now crumbling to bits. Your skills with Programming are only fledgling now, but you can still make a difference in these people’s lives. Maybe someday soon, your skills will have grown strong enough to take on the Uncoded One directly. But for now, you decide to do what you can to help. In the nearby city of Consolas, food is running short.Telim has a magic oven that can produce bread from thin air. He is willing to share, but Telim is an Excelian, and Excelians love paperwork; they demand it for all transactions—no exceptions. Telim will share his bread with the city if you can build a program that lets him enter the names of those receiving it.

A sample run of this program looks like this:

```
Bread is ready.
Who is the bread for?
RB
Noted: RB got bread.
```

Objectives:

- Make a program that runs as shown above, including taking a name from the user

To study if you find this difficult yet:

- W3 schools Java Scanner

Day 3 Challenge: The Thing Namer 3000

As you walk through the city of Commenton, admiring its forward-slash-based architectural buildings, a young man approaches you in a panic. “I dropped my Thing Namer 3000 and broke it. I think it’s mostly working, but all my variable names got reset! I don’t understand what they do!” He shows you the following program:

```
System.out.println("What kind of thing are we talking about?");
Scanner input = new Scanner(System.in);
String a = input.next();
System.out.println("How would you describe it? Big? Azure? Tattered?");
String b = input.next();
String c = "of Doom";
String d = "3000";
System.out.println("The " + b + " " + a + " of " + c + " " + d + "!");
```

“You gotta help me figure it out!”

Objectives:

- Rebuild the program above on your computer.
- Add comments near each of the four variables that describe what they store. You must use at least one of each comment type (// and /* */).
- Find the bug in the text displayed and fix it.
- Answer this question: Aside from comments, what else could you do to make this code more understandable?

To study if you find this difficult yet:

- W3 schools Java Comments

Day 4- Challenge: The Variable Shop

You see an old shopkeeper struggling to stack up variables in a window display. “Hoo-wee! All these variable types sure are exciting but setting them all up to show them off to excited new programmers like yourself is a lot of work for these aching bones,” she says. “You wouldn’t mind helping me set up this program with one variable of every type, would you?”

Objectives:

- Build a program with a variable of all eight primitive Java types (as well as the 'text' type).
- Assign each of them a value using a literal of the correct type.
- Use System.out.println to display the contents of each variable.

To study if you find this difficult yet:

- W3 schools Java Data Types

Day 5 1/2-Challenge - The Variable Shop Returns

“Hey! Programmer!” It’s the shopkeeper from the Variable Shop who hobbles over to you. “Thanks to your help, variables are selling like RAM cakes! But these people just

aren’t any good at programming. They keep asking how to modify the values of the variables they’re buying, and… well… frankly, I have no clue. But you’re a programmer, right? Maybe you could show me so I can show my customers?”

Objectives:

- Modify your Variable Shop program to assign a new, different literal value to each of the 9 original variables. Do not declare any additional variables.
- Use System.out.println to display the updated contents of each variable.

To study if you find this difficult yet:

- W3 schools Java Variables

Day 5: 2/2-Challenge - The Triangle Farmer

As you pass through the fields near Arithmetica City, you encounter P-Thag, a triangle farmer, scratching numbers in the dirt. “What is all of that writing for?” you ask. “I’m just trying to calculate the area of all of my triangles. They sell by their size. The bigger they are, the more they are worth! But I have many triangles on my farm, and the math gets tricky, and I sometimes make mistakes. Taking a tiny triangle to town thinking you’re going to get 100 gold, only to be told it’s only worth three, is not fun! If only I had a program that could help me….” Suddenly, P-Thag looks at you with fresh eyes. “Wait just a moment. You have the look of a Programmer about you. Can you help me write a program that will compute the areas for me, so I can quit worrying about math mistakes and get back to tending to my triangles? The equation I’m using is this one here,” he says, pointing to the formula, etched in a stone beside him:

Area = base x height / 2

Objectives:

- Write a program that lets you input the triangle’s base size and height.
- Compute the area of a triangle by turning the above equation into code.
- Write the result of the computation.

To study if you find this difficult yet:

- W3 schools Java Operators

Day 6: Challenge - The Four Sisters and the Duckbear

Four sisters own a chocolate farm and collect chocolate eggs laid by chocolate chickens every day. But more often than not, the number of eggs is not evenly divisible among the sisters, and everybody knows you cannot split a chocolate egg into pieces without ruining it. The arguments have grown more heated over time. The town is tired of hearing the four sisters complain, and Chandra, the town’s Arbiter, has finally come up with a plan everybody can agree to. All four sisters get an equal number of chocolate eggs every day, and the remainder is fed to their pet duckbear. All that is left is to have some skilled Programmer build a program to tell them how much each sister and the duckbear should get.

Objectives:

- Create a program that lets the user enter the number of chocolate eggs gathered that day.

- Using / and %, compute how many eggs each sister should get and how many are left over for the duckbear.
- Answer this question: What are the total egg counts where the duckbear gets more than each sister does? You can use the program you created to help you find the answer.

To study if you find this difficult yet:

- Take another look at W3 schools Java Operators

Day 7: Challenge The Dominion of Kings

Three kings, Melik, Casik, and Balik, are sitting around a table, debating who has the greatest kingdom among them. Each king rules an assortment of provinces, duchies, and estates. Collectively, they agree to a point system that helps them judge whose kingdom is greatest: Every estate is worth 1 point, every duchy is worth 3 points, and every province is worth 6 points. They just need a program that will allow them to enter their current holdings and compute a point total.

Objectives:

- Create a program that allows users to enter how many provinces, duchies, and estates they have.
- Add up the user's total score, giving 1 point per estate, 3 per duchy, and 6 per province.
- Display the point total to the user.

Day 8: Challenge - The Defense of Consolas

The Uncoded One has begun an assault on the city of Consolas; the situation is dire. From a moving airship called the Manticore, massive fireballs capable of destroying city blocks are being catapulted into the city. The city is arranged in blocks, numbered like a chessboard. You can assume its defenders are smart, so they can also move outside the city to say row -1 if needed. The city's only defense is a movable magical barrier, operated by a squad of four that can protect a single city block by putting themselves in each of the target's four adjacent blocks, as shown in the picture to the right. For example, to protect the city block at (Row 6, Column 5), the crew deploys themselves to (Row 6, Column 4), (Row 5, Column 5), (Row 6, Column 6), and (Row 7, Column 5). (you may want to make a sketch of the situation on a piece of paper)

The good news is that if we can compute the deployment locations fast enough, the crew can be deployed around the target in time to prevent catastrophic damage to the city for as long as the siege lasts. The City of Consolas needs a program to tell the squad where to deploy, given the anticipated target. Something like this:

```
Target Row? 6
Target Column? 5
Deploy to:
(6, 4)
(5, 5)
(6, 6)
(7, 5)
```

Objectives:

- Ask the user for the target row and column.
- Compute the neighboring rows and columns of where to deploy the squad.
- Display the deployment instructions in a different color of your choosing.
- Play a sound with `Toolkit.getDefaultToolkit().beep();` when the results have been computed and displayed.

To study if you find this difficult yet:

- How to produce colors in a console with Java

Day 9: Challenge - Repairing the Clocktower

The recent attacks damaged the great Clocktower of Consolas. The citizens of Consolas have repaired most of it, except one piece that requires the steady hand of a Programmer. It is the part that makes the clock tick and tock. Numbers flow into the clock to make it go, and if the number is even, the clock's pendulum should tick to the left; if the number is odd, the pendulum should tock to the right. Only a programmer can recreate this critical clock element to make it work again.

Objectives:

- Take a number as input from the console.
- Display the word "Tick" if the number is even. Display the word "Tock" if the number is odd.
- Hint: Remember that you can use the remainder operator to determine if a number is even or odd.

To study if you find this difficult yet:

- W3 schools if-else

Day 10: Challenge: Watchtower

There are watchtowers in the region around Consolas that can alert you when an enemy is spotted. With some help from you, they can tell you which direction the enemy is from the watchtower.

Objectives:

- Ask the user for an x value and a y value. These are coordinates of the enemy relative to the watchtower's location. Positive x-values are east, positive y-values are north (as they would be on a normal graph)
- Using if statements and relational operators, display a message about what direction the enemy is coming from. For example, "The enemy is to the northwest!" or "The enemy is here!

Day 11 Challenge 1 of 2: Buying Inventory

It is time to resupply. A nearby outfitter shop has the supplies you need but is so disorganized that they cannot sell things to you. "Can't sell if I can't find the price list," Tortuga, the owner, tells you as he turns over and attempts to go back to sleep in his reclining chair in the corner. There's a simple solution: use your programming powers to build something to report the prices of various pieces of equipment, based on the user's selection: The following items are available:

1. Rope
2. Torches

3. Climbing Equipment
4. Clean Water
5. Machete
6. Canoe
7. Food Supplies

What number do you want to see the price of? 2 Torches cost 15 gold. You search around the shop and find ledgers that show the following prices for these items: Rope: 10 gold, Torches: 15 gold, Climbing Equipment: 25 gold, Clean Water: 1 gold, Machete: 20 gold, Canoe: 200 gold, Food Supplies: 1 gold.

Objectives:

- Build a program that will show the menu illustrated above.
- Ask the user to enter a number from the menu.
- Using the information above, use a switch (either type) to show the item's cost.

To study if you find this difficult yet:

- W3 schools switch stamenents
- Switch expressions

Day 11 Challenge 2 of 2: Discounted Inventory

After sorting through Tortuga's outfitter shop and making it viable again, Tortuga realizes you've put him back in business. He wants to repay the favor by giving you a 50% discount on anything you buy from him, and he wants you to modify your program to reflect that. After asking the user for a number, the program should also ask for their name. If the name supplied is your name, cut the price in half before reporting it to the user.

Objectives:

- Modify your program from before to also ask the user for their name.
- If their name equals your name, divide the cost in half.

To study if you find this difficult yet:

- W3 schools String equality

Day 12 Challenge: The Prototype

Mylara, the captain of the Guard of Consolas, has approached you with the beginnings of a plan to hunt down The Uncoded One's airship. "If we're going to be able to track this thing down," she says, "we need you to make us a program that can help us home in on a location." She lays out a plan for a program to help with the hunt. One user, representing the airship pilot, picks a number between 0 and 100. Another user, the hunter, will then attempt to guess the number. The program will tell the hunter that they guessed correctly or that the number was too high or low. The program repeats until the hunter guesses the number correctly. Mylara claims, "If we can build this program, we can use what we learn to build a better version to hunt down the Uncoded One's airship."

Sample Program:

```
User 1, enter a number between 0 and 100: 27
After entering this number, the program should clear the screen and continue like
this:
User 2, guess the number.
What is your next guess? 50
50 is too high.
What is your next guess? 25
25 is too low.
What is your next guess? 27
You guessed the number!
```

Objectives:

- Build a program that will allow a user, the pilot, to enter a number.
- If the number is above 100 or less than 0, keep asking.
- Clear the screen once the program has collected a good number (you can just write 50 blank lines to clear the screen :)).
- Ask a second user, the hunter,to guess numbers.
- Indicate whether the user guessed too high, too low, or guessed right.
- Loop until they get it right, then end the program.

To study if you find this difficult yet:

- W3 schools do-while-loop
- W3 schools for-loop

Day 13 Challenge: The Magic Cannon

Skorin, a member of Consolas's wall guard, has constructed a magic cannon that draws power from two gems: a fire gem and an electric gem. Every third turn of a crank, the fire gem activates, and the cannon produces a fire blast. The electric gem activates every fifth turn of the crank, and the cannon makes an electric blast. When the two line up, it generates a potent combined blast. Skorin would like your help to produce a program that can warn the crew about which turns of the crank will produce the different blasts before they do it.

A partial output of the desired program looks like this:

```
1: Normal
2: Normal
3: Fire
4: Normal
5: Electric
6: Fire
7: Normal
```

Objectives:

- Write a program that will loop through the values between 1 and 100 and display what kind of blast the crew should expect. (The % operator may be of use.)
- Change the color of the output based on the type of blast. (For example, red for Fire, yellow for Electric, and blue for Electric and Fire).

Day 14 Challenge: Challenge The Replicator of D'To

While searching an abandoned storage building containing strange code artifacts, you uncover the ancient Replicator of D'To. This can replicate the contents of any int array into another array. But it appears broken and needs a Programmer to reforge the magic that allows it to replicate once again.

Objectives:

- Make a program that creates an array of length 5.
- Ask the user for five numbers and put them in the array.
- Make a second array of length 5.
- Use a loop to copy the values out of the original array and into the new one.
- Display the contents of both arrays one at a time to illustrate that the Replicator of D'To works again.

To study if you find this difficult yet:

- W3 schools arrays

Day 15 Challenge The Laws of Freach

The town of Freach recently had an awful looping disaster. The lead investigator found that it was a faulty ++ operator in an old for loop, but the town council has chosen to ban all loops but the foreach loop. Yet Freach uses the code presented earlier in this level to compute the minimum and the average value in an int array. They have hired you to rework their existing for-based code to use foreach loops instead.

Objectives:

- Start with the code for computing an array's minimum and average values (see below)

```
int[] array = { 4, 51, -7, 13, -99, 15, -8, 45, 90 };
int currentSmallest = Integer.MAX_VALUE; // Start higher than anything in the
array.
for (int index = 0; index < array.length; index++)
{
  if (array[index] < currentSmallest)
      currentSmallest = array[index];
}
System.out.println(currentSmallest);
```

int total = 0; for (int index = 0; index < array.length; index++) total += array[index]; double average = (double)total / array.length; System.out.println(average);

```
- Modify the code to use foreach loops instead of for loops

**To study if you find this difficult yet:**
- [W3 schools foreach loops](https://www.w3schools.com/java/java_foreach_loop.asp)


## Day 16 Challenge: Taking a Number
Many previous tasks have required getting a number from a user. To save time writing
this code
repeatedly, you have decided to make a method to do this common task.
```

```
**Objectives:**

- Make a method with the signature int askForNumber(String text). Display the text
parameter in the console window, get a response from the user, convert it to an int,
and return it.
This might look like this: int result = askForNumber("What is the airspeed velocity
of an unladen swallow?");.
- Make a method with the signature `int askForNumberInRange(String text, int min, int
max)`. Only return if the entered number is between the min and max values. Otherwise,
ask again.
- Place these methods in at least one of your previous programs to improve it.

**To study if you find this difficult yet:**
- [W3 schools methods](https://www.w3schools.com/java/java_methods.asp)
- [W3 schools methods (continued)]
(https://www.w3schools.com/java/java_methods_param.asp)

## Day 17 Challenge: Challenge Countdown

_Hint: try read a bit about 'recursion'_

The Council of Freach has summoned you. New writing has appeared on the Tomb of Algol
the Wise, the
last True Programmer to wander this land. The writing strikes fear and awe into the
hearts of the looploving people of Freach: “The next True Programmer shall be able to
write any looping code with a
method call instead.” The Senior Counselor, scared of a world without loops, asks you
to put your skill to
the test and rewrite the following code, which counts down from 10 to 1, with no
loops:
```

```
for (int x = 10; x > 0; x--) System.out.println(x);
```

```
**To study if you find this difficult yet:**
- [W3 schools recursion](https://www.w3schools.com/java/java_recursion.asp)

## Day 18 Challenge: Boss Battle Hunting the Manticore

The Uncoded One’s airship, the Manticore, has begun an all-out attack on the city of
Consolas. It must be
destroyed, or the city will fall. Only by combining Mylara’s prototype, Skorin’s
cannon, and your
programming skills will you have a chance to win this fight. You must build a program
that allows one
user—the pilot of the Manticore—to enter the airship’s range from the city and a
second user—the city’s
defenses—to attempt to find what distance the airship is at and destroy it before it
can lay waste to the
town.
```

```
The first user begins by secretly establishing how far the Manticore is from the city,
in the range 0 to 100.
The program then allows a second player to repeatedly attempt to destroy the airship
by picking the
range to target until either the city of Consolas or the Manticore is destroyed. In
each attempt, the player
is told if they overshot (too far), fell short (not far enough), or hit the Manticore.
The damage dealt to the
Manticore depends on the turn number. For most turns, 1 point of damage is dealt. But
if the turn number
is a multiple of 3, a fire blast deals 3 points of damage; a multiple of 5, an
electric blast deals 3 points of
damage, and if it is a multiple of both 3 and 5, a mighty fire-electric blast deals 10
points of damage. The
Manticore is destroyed after taking 10 points of damage.

However, if the Manticore survives a turn, it will deal a guaranteed 1 point of damage
to the city of
Consolas. The city can only take 15 points of damage before being annihilated.

Before a round begins, the user should see the current status: the current round
number, the city’s health,
and the Manticore’s health.

A sample run of the program is shown below. The first player gets a chance to place
the Manticore:
```

Player 1, how far away from the city do you want to station the Manticore? 32

```
At this point, the display is cleared, and the second player gets their chance:
```

Player 2, it is your turn.

STATUS: Round: 1 City: 15/15 Manticore: 10/10 The cannon is expected to deal 1 damage this round. Enter desired cannon range: 50 That round OVERSHOT the target.

STATUS: Round: 2 City: 14/15 Manticore: 10/10 The cannon is expected to deal 1 damage this round. Enter desired cannon range: 25 That round FELL SHORT of the target.

STATUS: Round: 3 City: 13/15 Manticore: 10/10 The cannon is expected to deal 3 damage this round. Enter desired cannon range: 32 That round was a DIRECT HIT!

STATUS: Round: 4 City: 12/15 Manticore: 7/10 The cannon is expected to deal 1 damage this round. Enter desired cannon range: 32 That round was a DIRECT HIT!

STATUS: Round: 5 City: 11/15 Manticore: 6/10 The cannon is expected to deal 3 damage this round. Enter desired cannon range: 32 That round was a DIRECT HIT!

STATUS: Round: 6 City: 10/15 Manticore: 3/10 The cannon is expected to deal 3 damage this round. Enter desired cannon range: 32 That round was a DIRECT HIT! The Manticore has been destroyed! The city of Consolas has been saved!

```
**Objectives**:

- Establish the game’s starting state: the Manticore begins with 10 health points and
the city with 15.
The game starts at round 1.
- Ask the first player to choose the Manticore’s distance from the city (0 to 100).
Clear the screen
afterward.
- Run the game in a loop until either the Manticore’s or city’s health reaches 0.
- Before the second player’s turn, display the round number, the city’s health, and
the Manticore’s
health.
- Compute how much damage the cannon will deal this round: 10 points if the round
number is a
multiple of both 3 and 5, 3 if it is a multiple of 3 or 5 (but not both), and 1
otherwise. Display this to
the player.
- Get a target range from the second player, and resolve its effect. Tell the user if
they overshot (too
far), fell short, or hit the Manticore. If it was a hit, reduce the Manticore’s health
by the expected
amount.
- If the Manticore is still alive, reduce the city’s health by 1.
- Advance to the next round.
- When the Manticore or the city’s health reaches 0, end the game and display the
outcome.
- Use different colors for different types of messages.
- Note: This is the largest program you have made so far. Expect it to take some time!
- Note: Use methods to focus on solving one problem at a time.
- Note: This version requires two players, but in the future, we will modify it to
allow the computer
to randomly place the Manticore so that it can be a single-player game.

**To study if you find this difficult yet:**
- You may possibly want to use fields/instance variables/attributes to make it easier
to pass data like the Manticore's current health [W3 schools fields]
(https://www.w3schools.com/java/java_class_attributes.asp)
- This program can grow quite big; possibly you want to create a separate class (say
ManticoreHunting) in its own java file and call it from your main program, like
```

```
// Program.java class Program { public static void main(String[] args) {
ManticoreHunting hunting = new ManticoreHunting(); hunting.run(); } }

...

// ManticoreHunting.java

class ManticoreHunting { }
```

```
## Day 19 Challenge: Challenge Simula’s Test
As you move through the village of Enumerant, you notice a short, cloaked figure
```

```
following you. Not
being one to enjoy a mysterious figure tailing you, you seize a moment to confront the
figure. “Don’t be
alarmed!” she says. “I am Simula. They are saying you’re a Programmer. Is this true?”
You answer in the
affirmative, and Simula’s eyes widen. “If you are truly a Programmer, you will be able
to help me. Follow
me.” She leads you to a backstreet and into a dimly lit hovel. She hands you a small,
locked chest. “We
haven’t seen any Programmers in these lands in a long time. And especially not ones
that can craft types.
If you are a True Programmer, you will want what is in that chest. And if you are a
True Programmer, I
will gladly give it to you to aid you in your quest.”
The chest is a small box you can hold in your hand. The lid can be open, closed (but
unlocked), or locked.
You’d normally be able to go between these states, opening, closing, locking, and
unlocking the box, but
the box is broken.You need to create a program with an enumeration to recreate this
locking mechanism.

The box can be in three states:
if the box is in the OPEN state, you can close it, and it becomes CLOSED
if the box is in the CLOSED state, you can open it, and it becomes OPEN
if the box is in the CLOSED state, you can lock it, and it becomes LOCKED
if the box is in the LOCKED state, you can unlock it, and it becomes CLOSED

Nothing happens if you attempt an impossible action in the current state, like opening
a locked box.
The program below shows what using this might look like:
```

The chest is locked. What do you want to do? unlock The chest is unlocked. What do you want to do? open The chest is open. What do you want to do? close The chest is unlocked. What do you want to do?

```
**Objectives:**

- Define an enumeration for the state of the chest.
- Make a variable whose type is this new enumeration.
- Write code to allow you to manipulate the chest with the lock, unlock, open, and
close
commands, but ensure that you don’t transition between states that don’t support it.
- Loop forever, asking for the next command.

**To study if you find this difficult yet:**
- [W3-schools enums](https://www.w3schools.com/java/java_enums.asp)
- Enums are surprisingly versatile in Java-they can have their own constructors and
methods! [How to do in Java-enums](https://howtodoinjava.com/java/enum/enum-tutorial/)

## Day 20 Challenge: Challenge Simula’s Soup
```

Simula is impressed with how you reconstructed the box with an enumeration. When the box opened,
you saw a glowing emerald gem inside. You don’t know what it is, but it seems important. Also in the box
were three vials of powder labeled HOT, SALTY, and SWEET.
“Finally! I can make soup again!” Simula says. She casually tosses the small glowing gem to you but is
wholly focused on the powders. “You stick around and help me make soup with your programming skills,
and I’ll tell you what that gem does.”
She pulls out a cookpot, knocks the clutter off the table with a quick sweep of her arm, and begins
cooking. She says, “I’m the best soup maker in town, and you’re in for a treat. I’ve got recipes for soup,
stew, and gumbo. I’ve got mushrooms, chicken, carrots, and potatoes for ingredients. And thanks to you
getting that box open, I’ve got seasonings again! Spicy, salty, and sweet seasoning. Pick a recipe, an
ingredient, and a seasoning, and I’ll make it. Use your programming skills to help us track what we make.”

Objectives:

- Define enumerations for the three variations on food: type (soup, stew, gumbo), main ingredient
(mushrooms, chicken, carrots, potatoes), and seasoning (spicy, salty, sweet).
- Make a class to represent a soup composed of the three above enumeration types.
- Let the user pick a type, main ingredient, and seasoning from the allowed choices and fill the "soup object"
with the results. Hint: You could give the user a menu to pick from or simply compare the user’s
text input against specific strings to determine which enumeration value represents their choice.
- When done, display the contents of the soup object in a format like “Sweet Chicken Gumbo.”
Hint: You don’t need to convert the enumeration value back to a string. Simply displaying an
enumeration value with System.out.print or System.out.printlin will display the name of the enumeration value.)

To study if you find this difficult yet:
- [W3-schools classes](https://www.w3schools.com/java/java_classes.asp)
- [Programiz classes/objects](https://www.programiz.com/java-programming/class-objects)

Narrative 1: The Fountain of Objects

As you eat soup with Simula, she explains that she is the Caretaker of the Heart of Object-Oriented
Programming—the glowing green gem in the box. For thousands of clock cycles, she has held onto it,
hoping that someday, a Programmer who understood object-oriented programming would

appear to
restore the Fountain of Objects, destroyed by The Uncoded One, back to what it once was: the lifeblood
of the entire island.
She tells you that to do this, you must gather the five keys of Object-Oriented Programming and make
your way to the Fountain of Objects, whose location is secret. She tells you you can discover its location
if you visit the Catacombs of the Class and marks that location on your map.
You leave Simula’s hovel behind and begin the quest to restore the Fountain of Objects to what it once
was. Your next destination: the Catacombs of the Class!

Day 21 Challenge 1/2: Vin Fletcher’s Arrows
Vin Fletcher is a skilled arrow maker. He asks for your help building a new class to represent arrows and
determine how much he should sell them for.“A tiny fragment of my soul goes into each arrow; I care
not for the money; I just need to be able to recoup my costs and get food on the table,” he says.
Each arrow has three parts: the arrowhead (steel, wood, or obsidian),the shaft (a length between 60 and
100 cm long), and the fletching (plastic, turkey feathers, or goose feathers).
His costs are as follows: For arrowheads, steel costs 10 gold, wood costs 3 gold, and obsidian costs 5 gold.
For fletching, plastic costs 10 gold, turkey feathers cost 5 gold, and goose feathers cost 3 gold. For the
shaft, the price depends on the length: 0.05 gold per centimeter.

Objectives:

- Define a new Arrow class with fields for arrowhead type, fletching type, and length. (Hint: arrowhead types and fletching types might be good enumerations.)
- Allow a user to pick the arrowhead, fletching type, and length and then create a new Arrow instance.
- Add a getCost method that returns its cost as a float based on the numbers above, and use this to display the arrow’s cost.

Day 21 Challenge 2/2: Vin’s Trouble
“Master Programmer!” Vin Fletcher shouts at you as he races to catch up to you. “I have a problem. I
created an arrow for a young man who took it and changed its length to be half as long as I had designed.
It no longer fit in his bow correctly and misfired. It sliced his hand pretty bad. He’ll survive, but is there
any way we can make sure somebody doesn’t change an arrow’s length when they walk away from my
shop? I don’t want to be the cause of such self-inflicted pain.”With your knowledge of information hiding,
you know you can help.

Objectives:

- Modify your Arrow class to have private instead of public fields.
- Add in getter methods for each of the fields that you have.

Day 22 Challenge: Arrow Factories
Vin Fletcher sometimes makes custom-ordered arrows, but these are rare. Most of the time, he sells one
of the following standard arrows:
- The Elite Arrow, made from a steel arrowhead, plastic fletching, and a 95 cm shaft.
- The Beginner Arrow, made from a wood arrowhead, goose feathers, and a 75 cm shaft.
- The Marksman Arrow, made from a steel arrowhead, goose feathers, and a 65 cm shaft.
You can make static methods to make these specific variations of arrows easy.

Objectives:

- Modify your Arrow class one final time to include static methods of the form public static
Arrow createEliteArrow() { ... } for each of the three above arrow types.
- Modify the program to allow users to choose one of these pre-defined types or a custom arrow. If
they select one of the predefined styles, produce an Arrow instance using one of the new static
methods. If
they select one of the predefined styles, produce an Arrow instance using one of the new static
methods. If they choose to make a custom arrow, use your earlier code to get their custom data
about the desired arrow.

Narrative 2: Entering the Catacombs
You arrive at the Catacombs of the Class, the place that will reveal the path to the Fountain of Objects.
The Catacombs lie inside a mountain, with a wide stone entrance leading you into a series of three
chambers. In the first chamber, you find five pedestals with the remnants of a class definition and specific
instructions by each. Etched above a sealed doorway at the back of the room is the text, “Only the True
Programmer who can remake the Five Prototypes can proceed.” Each pedestal appears to have
instructions for crafting a class. These are the Five Prototypes that you must reassemble.

Day 23 Challenge: The Point
The first pedestal asks you to create a Point class to store a point in two dimensions. Each point is
represented by an x-coordinate (x), a side-to-side distance from a special central point called the origin,
and a y-coordinate (y), an up-and-down distance away from the origin.

Objectives:

- Define a new Point class with fields and getter methods for x and y.
- Add a constructor to create a point from a specific x- and y-coordinate.
- Add a parameterless constructor to create a point at the origin (0, 0).
- In your main method, create a point at (2, 3) and another at (-4, 0). Display these points on the
console window in the format (x, y) to illustrate that the class works.
- Answer this question: Are your x and y immutable? Why did you choose what you did?

To study if you find this difficult yet:
- [extra about constructors in Java](https://linuxhint.com/java_constructor_tutorial/)
- Handy to print a nice representation of a class: overriding the [toString method](https://www.w3schools.blog/tostring-method-in-java)

Day 24 Challenge: The Color
The second pedestal asks you to create a Color class to represent a color. The pedestal includes an
etching of this diagram that illustrates its potential usage:
The color consists of three parts or channels: red, green, and blue, which indicate how much those
channels are lit up. Each channel can be from 0 to 255. 0 means completely off; 255 means completely
on.
The pedestal also includes some color names, with a set of numbers indicating their specific values for
each channel. These are commonly used colors: White (255, 255, 255), Black (0, 0, 0), Red (255, 0, 0),
Orange (255,165, 0), Yellow (255, 255, 0), Green (0, 128, 0), Blue (0, 0, 255), Purple (128, 0, 128).

Objectives:

- Define a new Color class with fields for its red, green, and blue channels.
- Add appropriate constructors that you feel make sense for creating new Color objects.
- Create final static fields to define the eight commonly used colors for easy access.
- In your main method, make two Color-typed variables. Use a constructor to create a color instance
and use a static field for the other. Display each of their red, green, and blue channel values.

To study if you find this difficult yet:
- static final fields are the generally only fields that you can and want to make public. A bit about those fields on [GeeksForGeeks](https://www.geeksforgeeks.org/final-static-variable-java/)

Day 25 Challenge: The Card
The digital Realms of Java have playing cards like ours but with some differences. Each card has a color (red, green, blue, yellow) and a rank (the numbers 1 through 10, followed by the symbols $, %, ^, and &).
The third pedestal requires that you create a class to represent a card of this nature.

Objectives:

- Define enumerations for card colors and card ranks.
- Define a Card class to represent a card with a color and a rank, as described above.
- Add methods that tell you if a card is a number or symbol card (the equivalent of a face card).
- Create a main method that will create a card instance for the whole deck (every color with every
rank) and display each (for example, “The Red Ampersand” and “The Blue Seven”).
- Answer this question: Why do you think we used a color enumeration here but made a color class
in the previous challenge?

Day 26 Challenge: The Locked Door
The fourth pedestal demands constructing a door class with a locking mechanism that requires a unique
numeric code to unlock. You have done something similar before without using a class, but the locking
mechanism is new. The door should only unlock if the passcode is the right one. The following statements
describe how the door works.
- An open door can always be closed.
- A closed (but not locked) door can always be opened.
- A closed door can always be locked.
- A locked door can be unlocked, but a numeric passcode is needed, and the door will only unlock if
the code supplied matches the door’s current passcode.
- When a door is created, it must be given an initial passcode.
- Additionally, you should be able to change the passcode by supplying the current code and a new
one. The passcode should only change if the correct, current code is given.

Objectives:

- Define a Door class that can keep track of whether it is locked, open, or closed.
- Make it so you can perform the four transitions defined above with methods.
- Build a constructor that requires the starting numeric passcode.
- Build a method that will allow you to change the passcode for an existing door by supplying the
current passcode and new passcode. Only change the passcode if the current passcode is correct.
- Make your main method ask the user for a starting passcode, then create a new Door instance. Allow
the user to attempt the four transitions described above (open, close, lock, unlock) and change the
code by typing in text commands

Day 27 Challenge: The Password Validator
The fifth and final pedestal describes a class that represents a concept more abstract than the first four:
a password validator. You must create a class that can determine if a password is valid (meets the rules

defined for a legitimate password). The pedestal initially doesn’t describe any rules, but as you brush the
dust off the pedestal, it vibrates for a moment, and the following rules appear:
- Passwords must be at least 6 letters long and no more than 13 letters long.
- Passwords must contain at least one uppercase letter, one lowercase letter, and one number.
- Passwords cannot contain a capital T or an ampersand (&) because Ingelmar in IT has decreed it.
That last rule seems random, and you wonder if the pedestal is just tormenting you with obscure rules.
You ponder for a moment about how to decide if a character is uppercase, lowercase, or a number, but
while scratching your head, you notice a piece of folded parchment on the ground near your feet. You
pick it up, unfold it, and read it:
foreach with a string lets you get its characters!

> *for (char ch: word.toCharArray()) { if (Character.isUpperCase(ch)) } Character has static methods to categorize letters! char.isUpperCase('A'), char.isLowerCase('a'), char.isDigit('0')*

That might be useful information! You are grateful to whoever left it behind. It is signed simply “A.”

Objectives:

- Define a new PasswordValidator class that can be given a password and determine if the
password follows the rules above.
- Make your main method loop forever, asking for a password and reporting whether the password is
allowed using an instance of the PasswordValidator class.

To study if you find this difficult yet:
- You can find out about methods in Character by either typing in "Character." in IntelliJ or by looking at [the Java API documentation for Character](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Character.html)

- since char is a primitive type (in contrast to a String), you can use '==' for comparisons :)

Narrative 3: The Chamber of Design
As you finish the final class and place its complete definition back on its pedestal, the writing on each
pedestal begins to glow a reddish-orange. A beam forms from each pedestal, extending upward towards
the high cavernous ceiling. Additional runes on the wall begin to shine as well, and the far walls slide
apart, revealing an opening further into the Catacombs.
You pass through to the next chamber and find three more pedestals with etched text.

On the floor, in
a ring running around the three pedestals, lie the words, “Only a True Programmer can design a system
of objects for the ancient games of the people.”
You must make an object-oriented design (not a complete program) for each game described on the
three pedestals in the room’s center to continue further.

IMPORTANT NOTE
For the next three assignments, do _not_ write a program, only make a design on paper about the objects you need. You can use [CRC cards]
(http://agilemodeling.com/artifacts/crcModel.htm) or [UML class diagrams]
(https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/) or any other method to express the global design and functionality of a class on paper -_without writing any code!!!_.

Day 28 Design Challenge: Rock-Paper-Scissors
The first design pedestal requires you to provide an object-oriented design—a set of objects, classes,
and how they interact—for the game of Rock-Paper-Scissors, described below:
- Two human players compete against each other.
- Each player picks Rock, Paper, or Scissors.
- Depending on the players’ choices, a winner is determined: Rock beats Scissors, Scissors beats Paper,
Paper beats Rock. If both players pick the same option, it is a draw.
- The game must display who won the round.
- The game will keep running rounds until the window is closed but must remember the historical
record of how many times each player won and how many draws there were.

Objectives:

- Use CRC cards (or a suitable alternative) to outline the objects and classes that may be needed to
make the game of Rock-Paper-Scissors. You do not need to create this full game; just come up
with a potential design as a starting point.

Day 29 Design Challenge: 15-Puzzle
The second pedestal requires you to provide an object-oriented design for the game of 15-Puzzle.
The game of 15-Puzzle contains a set of numbered tiles on a board with a single open slot. The goal is to
rearrange the tiles to put the numbers in order, with the empty space in the bottom-right corner.
- The player needs to be able to manipulate the board to rearrange it.
- The current state of the game needs to be displayed to the user.
- The game needs to detect when it has been solved and tell the player they won.
- The game needs to be able to generate random puzzles to solve.
- The game needs to track and display how many moves the player has made.

Objectives:

- Use CRC cards (or a suitable alternative) to outline the objects and classes that may be needed to
make the game of 15-Puzzle. You do not need to create this full game; just come up with a
potential design as a starting point.
- Answer this question: Would your design need to change if we also wanted 3×3 or 5×5 boards?

Day 30 Design Challenge: Hangman
The third pedestal in this room requires you to provide an object-oriented design for the game of
Hangman. In Hangman, the computer picks a random word for the player to guess. The player then
proceeds to guess the word by selecting letters from the alphabet, which get filled in, progressively
revealing the word. The player can only get so many letters wrong (a letter not found in the word) before
losing the game. An example run of this game could look like this:

Word: _ _ _ _ _ _ _ _ _ | Remaining: 5 | Incorrect: | Guess: e Word: _ _ _ _ _ _ _ _ E | Remaining: 5 | Incorrect: | Guess: i Word: I _ _ _ _ _ _ _ E | Remaining: 5 | Incorrect: | Guess: u Word: I _ _ U _ _ _ _ E | Remaining: 5 | Incorrect: | Guess: o Word: I _ _ U _ _ _ _ E | Remaining: 4 | Incorrect: O | Guess: a Word: I _ _ U _ A _ _ E | Remaining: 4 | Incorrect: O | Guess: t Word: I _ _ U T A _ _ E | Remaining: 4 | Incorrect: O | Guess: s Word: I _ _ U T A _ _ E | Remaining: 3 | Incorrect: OS | Guess: r Word: I _ _ U T A _ _ E | Remaining: 2 | Incorrect: OSR | Guess: m Word: I M M U T A _ _ E | Remaining: 2 | Incorrect: OSR | Guess: l Word: I M M U T A _ L E | Remaining: 2 | Incorrect: OSR | Guess: b Word: I M M U T A B L E You won!

- The game picks a word at random from a list of words.
- The game’s state is displayed to the player, as shown above.
- The player can pick a letter. If they pick a letter they already chose, pick again.
- The game should update its state based on the letter the player picked.
- The game needs to detect a win for the player (all letters have been guessed).
- The game needs to detect a loss for the player (out of incorrect guesses).

Objectives:

- Use CRC cards (or a suitable alternative) to outline the objects and classes that may be needed to
make the game of Hangman. You do not need to create this full game; just come up with a
potential design as a starting point.

Day 31 Design Challenge: Tic-Tac-Toe
Completing designs for the three games in the Chamber of Design causes the pedestals to light up red
again, and another door opens, letting you into the final chamber. This chamber has only a single large,
broad pedestal. Inscribed on the stone floor in a circle around the pedestal are the engraved words,“Only

a True Programmer can build object-oriented programs.”
More text engraved on the pedestal describes what you recognize as the game of Tic-Tac-Toe, stating
that in ancient times, inhabitants of the land would use this as a Battle of Wits to determine the outcome
of political strife. Instead of fighting wars, they would battle it out in a game of Tic-Tac-Toe.
Your job is to recreate the game of Tic-Tac-Toe, allowing two players to compete against each other. The
following features are required:
- Two human players first enter their names. Then they take turns entering their choice using the same keyboard.
- The players designate which square they want to play in. Hint: You might consider using the number
pad as a guide. For example, if they enter 7, they have chosen the top left corner of the board.
- The game should prevent players from choosing squares that are already occupied. If such a move
is attempted, the player should be told of the problem and given another chance.
- The game must detect when a player wins or when the board is full with no winner (draw/”cat”).
- When the game is over, the outcome is displayed to the players.
- The state of the board must be displayed to the player after each play. Hint: One possible way to
show the board could be like this:

It is X's turn. | X | ---+---+--- | O | X ---+---+--- O | | What square do you want to play in?

Objectives:

- Build the game of Tic-Tac-Toe as described in the requirements above. Starting with CRC cards is
recommended, but the goal is to make working software, not CRC cards.
- Answer this question: How might you modify your completed program if running multiple rounds
was a requirement (for example, a best-out-of-five series)?

To study if you find this difficult yet:
- Consider 2-dimensional arrays, or a [Map]
(https://www.w3schools.com/java/java_hashmap.asp) to represent the board

Narrative 4: The Gift of Object Sight
As you place the finished Tic-Tac-Toe program onto the pedestal, writing etched into the stone walls
begins to glow reddish-orange. The glow is bright enough that you have to shield your eyes with your
hand for a moment before the glowing dims to a more manageable intensity.
Suddenly, you realize that you are no longer the only thing in the room. Thousands of faintly glowing,
bluish objects of various shapes and sizes float in the air around you.

You hear a resounding, booming voice echo through the chamber: “We are the Guardians of the
Catacombs. We have seen your creations and know that you are a True Programmer. We have deemed
you worthy of the Gift of Object Sight—the ability to see objects in code and requirements and craft
solutions from objects and types.
“We need your help. The Fountain of Objects—the lifeblood of this island—has been destroyed by the
vile Uncoded One. Use the Gift of Object Sight to reforge the Fountain of Objects. Without the Fountain,
this land will crumble and fade into oblivion. Object Sight will lead you to the Fountain. Depart now and
save this land!”
As you leave the Catacombs of the Class, you discover that your new Object Sight ability has made
countless code objects visible in the world around you. You also see a distinct trail, marked with a faint
blue line heading into the rugged, distant mountains where the Fountain of Objects supposedly lies.
Though the journey ahead is still long, the pathway to the Fountain of Objects is now clear!

Day 32 Challenge: Packing Inventory
You know you have a long, dangerous journey ahead of you to travel to and repair the Fountain of
Objects. You decide to build some classes and objects to manage your inventory to prepare for the trip.
You decide to create a Pack class to help in holding your items. Each pack has three limits: the total
number of items it can hold, the weight it can carry, and the volume it can hold. Each item has a weight
and volume, and you must not overload a pack by adding too many items, too much weight, or too much
volume.
There are many item types that you might add to your inventory, each their own class in the inventory
system.
1. An arrow has a weight of 0.1 and a volume of 0.05.
2. A bow has a weight of 1 and a volume of 4.
3. Rope has a weight of 1 and a volume of 1.5.
4. Water has a weight of 2 and a volume of 3.
5. Food rations have a weight of 1 and a volume of 0.5.
6. A sword has a weight of 5 and a volume of 3.

Objectives:
- Create an InventoryItem class that represents any of the different item types. This class must
represent the item’s weight and volume, which it needs at creation time (constructor).
- Create derived classes for each of the types of items above. Each class should pass the correct weight
and volume to the base class constructor but should be creatable themselves with a

```
parameterless
constructor (for example, new Rope() or new Sword()).
- Build a Pack class that can store an array of items. The total number of items, the
maximum weight,
and the maximum volume are provided at creation time and cannot change afterward.
- Make a public bool add(InventoryItem item) method to Pack that allows you to add
items
of any type to the pack’s contents. This method should fail (return false and not
modify the pack’s
fields) if adding the item would cause it to exceed the pack’s item, weight, or volume
limit.
- Add properties to Pack that allow it to report the current item count, weight, and
volume, and the
limits of each.
- Create a program that creates a new pack and then allow the user to add (or attempt
to add) items
chosen from a menu.

**To study if you find this difficult yet:**
- Inheritance in Java can be done via [extends]
(https://www.w3schools.com/java/ref_keyword_extends.asp)
- You may need to call the superclass constructor using [super]
(https://www.programiz.com/java-programming/super-keyword)

## Day 33 Challenge: Labeling Inventory
You realize that your inventory items are not easy to sort through. If you could make
it easy to label all
of your inventory items, it would be easier to know what items you have in your pack.
Modify your inventory program from the previous level as described below.

**Objectives**:

- Override the existing toString method (from the object base class) on all of your
inventory item
subclasses to give them a name. For example, new Rope().toString() should return
"Rope".
- Override toString on the Pack class to display the contents of the pack. If a pack
contains water,
rope, and two arrows, then calling toString on that Pack object could look like "Pack
containing Water Rope Arrow Arrow".
- Before the user chooses the next item to add, display the pack’s current contents
via its new
toString method.

## Day 34 Challenge: The Old Robot
You spot something shiny, half-buried in the mud. You pull it out and realize that it
seems to be some
mechanical automaton with the words “Property of Dynamak” etched into it. As you knock
off the caked-on mud, you realize that it seems like this old automaton might even be
programmable if you can give it
the proper commands. The automaton seems to be structured like this:
```

```
class Robot { private int x;

    private int y;

    private boolean isPowered;

    private RobotCommand[] commands = new RobotCommand[3];

    public void run() {
        for (RobotCommand command : commands) {
            command.run(this);
            System.out.printf("[%d %d %b]\n", x, y, isPowered);
        }
    }

}

You don’t see a definition of that RobotCommand class. Still, you think you might be
able to recreate it
(a class with only an abstract `run` command) and then make derived classes that
extend RobotCommand
that move it in each of the four directions and power it on and off. (You wish you
could manufacture a
whole army of these!)

**Objectives**:

- Copy the code above into a new project.
- Create a RobotCommand class with a public and abstract void run(Robot robot) method.
(The
code above should compile after this step.)
- Make OnCommand and OffCommand classes that inherit from RobotCommand and turn the
robot
on or off by overriding the `run` method.
- Make a NorthCommand, SouthCommand, WestCommand, and EastCommand that move the robot
1
unit in the +y direction, 1 unit in the -y direction, 1 unit in the -x direction, and
1 unit in the +x
direction, respectively. Also, ensure that these commands only work if the robot’s
isPowered is true.
- Make your main method able to collect three commands from the console window.
Generate new
RobotCommand objects based on the text entered. After filling the robot’s command set
with these
new RobotCommand objects, use the robot’s `run` method to execute them. For example:

on north west

[0 0 True] [0 1 True] [-1 1 True]

- Note: You might find this strategy for making commands that update other objects
useful in some
of the larger challenges in the coming levels.
```

```
**To study if you find this difficult yet:**
- [W3 schools abstract class](https://www.w3schools.com/java/java_abstract.asp)

## Day 35 Challenge: Robotic Interface
With your knowledge of interfaces, you realize you can refine the old robot you found
in the mud to use
interfaces instead of the original design. Instead of an abstract RobotCommand base
class, it could
become an RobotCommand interface!
Building on your solution to the Old Robot challenge, perform the changes below:

**Objectives**:

- Change your abstract RobotCommand class into an RobotCommand interface.
- Remove the unnecessary public and abstract keywords from the run method.
- Change the Robot class to use the interface
- Make all of your commands implement this new interface instead of extending the
RobotCommand
class that no longer exists. You will also want to remove the @Override keywords in
these classes.
- Ensure your program still compiles and runs.
- Answer this question: Do you feel this is an improvement over using an abstract base
class? Why
or why not?

**To study if you find this difficult yet:**
- [W3 schools interface](https://www.w3schools.com/java/java_interface.asp)

## Day 36 Challenge: Room Coordinates
The time to enter the Fountain of Objects draws closer. While you don’t know what to
expect, you have
found some scrolls that describe the area in ancient times. It seems to be structured
as a set of rooms
in a grid-like arrangement.
Locations of the room may be represented as a row and column, and you take it upon
yourself to try to
capture this concept with a new record definition.

**Objectives**:

- Create a Coordinate record that can represent a room coordinate with a row and
column.
- Ensure Coordinate is immutable.
- Make a method to determine if one coordinate is adjacent to another (differing only
by a single row
or column).
- Write a main method that creates a few coordinates and determines if they are
adjacent to each
other to prove that it is working correctly.

**To study if you find this difficult yet:**
```

```
- [W3 schools record](https://www.w3schools.io/java/java16-record-type/?
utm_content=cmp-true)

## Day 37 Challenge: War Preparations
As you pass through the city of Rocaard, two blacksmiths, Cygnus and Lyra, approach
you. “We know
where this is headed. A confrontation with the Uncoded One’s forces,” Lyra says.
Cygnus continues,
“You’re going to need an army at your side—one prepared to do battle. We forge
enchanted swords and
will do everything we can to support this cause. We need the Power of Programming to
flow unfettered
too. We want to help, but we can’t equip an entire army without the help of a program
to aid in crafting
swords.” They describe the program they need, and you dive in to help.

**Objectives**:

- Swords can be made out of any of the following materials: wood, bronze, iron, steel,
and the rare
binarium. Create an enumeration to represent the material type.
- Gemstones can be attached to a sword, which gives them strange powers through Cygnus
and Lyra’s
touch. Gemstone types include emerald, amber, sapphire, diamond, and the rare
bitstone. Or no
gemstone at all. Create an enumeration to represent a gemstone type.
- Create a Sword record with a material, gemstone, length, and crossguard width.
- In your main program, create a basic Sword instance made out of iron and with no
gemstone. Then
create two variations on the basic sword, defining methods in Sword that produce a
modified copy, like `Sword withMaterial(Material material)`.
- Give sword a toString method that produces output like "a steel sword of 85
centimetres long, a crossguard width of 19 cm, and an embedded ruby" or "a plain
binarium sword of 55 centimetres long and a crossguard width of 15 cm"
- Display all three sword instances with code like System.out.println(original);.

## Day 38 Challenge : Colored Items
You have a sword, a bow, and an axe in front of you, defined like this:
```

```
public class Sword { } public class Bow { } public class Axe { }
```

```
You want to associate a color with these items (or any item type). You could make
ColoredSword
derived from Sword that adds a Color property, but doing this for all three item types
will be
painstaking. Instead, you define a new generic ColoredItem class that does this for
any item.

**Objectives**:

- Put the three class definitions above into a new package.
- Define a generic class to represent a colored item. It must have properties for the
```

item itself (generic
in type) and a ConsoleColor associated with it.
- Add a void display() method to your colored item type that changes the console’s foreground
color to the item’s color and displays the item in that color.
- In your main method, create a new colored item containing a blue sword, a red bow, and a green axe.
Display all three items to see each item displayed in its color.

To study if you find this difficult yet:
- [W3 schools generic classes](https://www.w3schools.blog/generics-class-java)

Narrative 5: Arrival at the Caverns
You have made your way to the Cavern of Objects, high atop jagged mountains. Within these caverns
lies the Fountain of Objects, the one-time source of the River of Objects that gave life to this entire
island. By returning the Heart of Object-Oriented Programming—the gem you received from Simula
after arriving on this island—to the Fountain of Objects, you can repair and restore the fountain to its
former glory.
The cavern is a grid of rooms, and no natural or human-made light works within due to unnatural
darkness. You can see nothing, but you can hear and smell your way through the caverns to find the
Fountain of Objects, restore it, and escape to the exit.
The cavern is full of dangers. Bottomless pits and monsters lurk in the caverns, placed here by the
Uncoded One to prevent you from restoring the Fountain of Objects and the land to its former glory.
By returning the Heart of Object-Oriented Programming to the Fountain of Objects, you can save the
Island of Object-Oriented Programming!

Day 39 Challenge The Fountain of Objects

The Fountain of Objects game is a 2D grid-based world full of rooms. Most rooms are empty, but a few
are unique rooms. One room is the cavern entrance. Another is the fountain room, containing the
Fountain of Objects.
The player moves through the cavern system one room at a time to find the Fountain of Objects. They
activate it and then return to the entrance room. If they do this without falling into a pit, they win the
game.
Unnatural darkness pervades the caverns, preventing both natural and human-made light. The player
must navigate the caverns in the dark, relying on their sense of smell and hearing to determine what

room they are in and what dangers lurk in nearby rooms.
This challenge serves as the basis for the other challenges in this level. It must be completed before the
others can be started. The requirements of this game are listed below.

Objectives:

- The world consists of a grid of rooms, where each room can be referenced by its row and column. Initially, the grid consists of 4 x 4 is 16 rooms.
North is up, east is right, south is down, and west is left:
- The game’s flow works like this: The player is told what they can sense in the dark (see, hear, smell).
Then the player gets a chance to perform some action by typing it in. Their chosen action is resolved
(the player moves, state of things in the game changes, checking for a win or a loss, etc.). Then the
loop repeats.
- Most rooms are empty rooms, and there is nothing to sense.
- The player is in one of the rooms and can move between them by typing commands like the
following: “move north”, “move south”, “move east”, and “move west”. The player should not be able
to move past the end of the map.
- The room at (Row=0, Column=0) is the cavern entrance (and exit). The player should start here.The
player can sense light coming from outside the cavern when in this room. (“You see light in this room
coming from outside the cavern. This is the entrance.”)
- The room at (Row=0, Column=2) is the fountain room, containing the Fountain of Objects itself. The
Fountain can be either enabled or disabled. The player can hear the fountain but hears different
things depending on if it is on or not. (“You hear water dripping in this room. The Fountain of Objects
is here!” or “You hear the rushing waters from the Fountain of Objects. It has been reactivated!”) The
fountain is off initially. In the fountain room, the player can type “enable fountain” to enable it. If the
player is not in the fountain room and runs this, there should be no effect, and the player should be
told so.
- The player wins by moving to the fountain room, enabling the Fountain of Objects, and moving back
to the cavern entrance. If the player is in the entrance and the fountain is on, the player wins.
- Use different colors to display the different types of text in the console window. For example,
narrative items (intro, ending, etc.) may be magenta, descriptive text in white, input from the user
in cyan, text describing entrance light in yellow, messages about the fountain in blue.
- An example of what the program might look like is shown below:

```
```----------------------------------------------------------------------------
You are in the room at (Row=0, Column=0).
You see light coming from the cavern entrance.
What do you want to do? move east

You are in the room at (Row=0, Column=1).
What do you want to do? move east

You are in the room at (Row=0, Column=2).
You hear water dripping in this room. The Fountain of Objects is here!
What do you want to do? enable fountain

You are in the room at (Row=0, Column=2).
You hear the rushing waters from the Fountain of Objects. It has been reactivated!
What do you want to do? move west

You are in the room at (Row=0, Column=1).
What do you want to do? move west

You are in the room at (Row=0, Column=0).
The Fountain of Objects has been reactivated, and you have escaped with your life!
You win!
```

- Hint: You may find two-dimensional arrays helpful in representing a 2D grid-based game world.
- Hint: Remember your training! You do not need to solve this entire problem all at once, and you do not have to get it right in your first attempt. Pick an item or two to start and solve just those items. Rework until you are happy with it, then add the next item or two.

## Day 40 Challenge 1/2: Small, Medium, or Large

The larger the Cavern of Objects is, the more difficult the game becomes. The basic game only requires a small 4×4 world, but we will add a medium 6×6 world and a large 8×8 world for this challenge.

**Objectives**:

- Before the game begins, ask the player whether they want to play a small, medium, or large game. Create a 4×4 world if they choose a small world, a 6×6 world if they choose a medium world, and an 8×8 world if they choose a large world.
- Pick an appropriate location for both the Fountain Room and the Entrance room.
- Note: When combined with the Amaroks, Maelstroms, or Pits challenges, you will need to adapt the game by adding amaroks, maelstroms, and pits to all three sizes.

## Day 40 Challenge 2/2: Pits

The Cavern of Objects is a dangerous place. Some rooms open up to bottomless pits. Entering a pit means death. The player can sense a pit is in an adjacent room because a draft of air pushes through the pits into adjacent rooms. Add pit rooms to the game. End the game if the player stumbles into one.
```

Objectives:

- Add a pit room to your 4×4 cavern anywhere that isn’t the fountain or entrance room.
- Players can sense the draft blowing out of pits in adjacent rooms (all eight directions): “You feel a draft. There is a pit in a nearby room.”
- If a player ends their turn in a room with a pit, they lose the game.
- Note: When combined with the Small, Medium, or Large challenge, add one pit to the 4×4 world,two pits to the 6×6 world, and four pits to the 8×8 world, in locations of your choice.

Day 41 Challenge: Maelstroms

The Uncoded One knows the significance of the Fountain of Objects and has placed minions in the caverns to defend it. One of these is the maelstrom—a sentient, malevolent wind. Encountering a maelstrom does not result in instant death, but entering a room containing a maelstrom causes the player to be swept away to another room. The maelstrom also moves to a new location. If the player is moved to another dangerous location, such as a pit, that room’s effects will happen upon landing in that room. A player can hear the growling and groaning of a maelstrom from a neighboring room (including diagonals), which gives them a clue to be careful. Modify the basic Fountain of Objects game in the ways below to add maelstroms to the game.

Objectives:

- Add a maelstrom to the small 4×4 game in a location of your choice.
- The player can sense maelstroms by hearing them in adjacent rooms. (“You hear the growling and groaning of a maelstrom nearby.”)
- If a player enters a room with a maelstrom, the player moves one space north and two spaces east, while the maelstrom moves one space south and two spaces west. When the player is moved like this, tell them so. If this would move the player or maelstrom beyond the map’s edge, ensure they stay on the map. (Clamp them to the map, wrap around to the other side, or any other strategy.)
- Note: When combined with the Small, Medium, or Large challenge, place one maelstrom into the medium-sized game and two into the large-sized game.

Day 42 Challenge 1/2: Amaroks

The Uncoded One has also placed amaroks in the caverns to protect the fountain from people like you. Amaroks are giant, rotting, wolf-like creatures that stalk the caverns.When players enter a room with an amarok, they are instantly killed, and the game is over. Players can smell an amarok in any adjacent room (all eight directions), which tells them that an amarok is nearby. Modify the basic Fountain of Objects game as described below.

Objectives:

- Amarok locations are up to you. Pick a room to place an amarok aside from the entrance or fountain room in the small 4×4 world.
- When a player is in one of the eight spaces adjacent to an amarok, a message should be displayed when sensing surroundings that indicate that the player can smell the amarok nearby. For example, “You can smell the rotten stench of an amarok in a nearby room.”
- When a player enters a room with an amarok, the player dies and loses the game.

- Note: When combined with the Small, Medium, or Large challenge, place two amaroks in the medium level and three in the large level in locations of your choosing.

Day 42 Challenge 2/2: Getting Armed

Note: Requires doing the Maelstroms or Amaroks challenge first. The player brings a bow and several arrows with them into the Caverns.The player can shoot arrows into the rooms around them, and if they hit a monster, they kill it, and it should no longer impact the game.

Objectives:

- Add the following commands that allow a player to shoot in any of the four directions: shoot north, shoot east, shoot south, and shoot west.When the player shoots in one of the four directions, an arrow is fired into the room in that direction. If a monster is in that room, it is killed and should not affect the game anymore. They can no longer sense it, and it should not affect the player.
- The player only has five arrows and cannot shoot when they are out of arrows. Display the number of arrows the player has when displaying the game’s status before asking for their action.

Day 43 Challenge: Getting Help

The player should not be left guessing about how to play the game. This challenge requires adding two key elements that make playing the Fountain of Objects easier: introductory text that explains the game and a help command that lists all available commands and what they each do.

Objectives:

- When the game starts, display text that describes the game shown below: You enter the Cavern of Objects, a maze of rooms filled with dangerous pits in search of the Fountain of Objects. Light is visible only in the entrance, and no other light is seen anywhere in the caverns. You must navigate the Caverns with your other senses. Find the Fountain of Objects, activate it, and return to the entrance.
- If you chose to do the Pits challenge, add the following to the description: “Look out for pits. You will feel a breeze if a pit is in an adjacent room. If you enter a room with a pit, you will die.”
- If you chose to do the Maelstroms challenge, add the following to the description: “Maelstroms are violent forces of sentient wind. Entering a room with one could transport you to any other location in the caverns. You will be able to hear their growling and groaning in nearby rooms.”
- If you chose to do the Amaroks challenge, add the following to the description: “Amaroks roam the caverns. Encountering one is certain death, but you can smell their rotten stench in nearby rooms.”
- If you chose to do the Getting Armed challenge, add the following to the description: “You carry with you a bow and a quiver of arrows. You can use them to shoot monsters in the caverns but be warned: you have a limited supply.”
- When the player types the command help, display all available commands and a short description of what each does. The complete list of commands will depend on what challenges you complete.

Narrative 6: The Fountain Remade

You scramble through the dark Cavern of Objects, crawling and feeling your way to the Fountain of Objects. The dripping sound that you hear is a giveaway that you have found it. You pull Simula's green gem—the Heart of Object-Oriented Programming—out of your pack and hold it in the palm of your hand, contemplating the journey you have taken to get here.You slide your hand along the Fountain until you find a small recess, just big enough for the Heart to be placed. You slide the green gem in, and the fountain immediately comes to life. The water in the fountain, previously still, suddenly begins churning and overflowing onto the ground around you. You make a hasty escape to the cavern entrance. Within minutes, water rushes out the entrance and through a thousand other holes in the mountainside, collecting into a raging waterfall down into the valley below. Within days, the newly restored River of Objects will flow to the sea, restoring its life-giving power to the entire island. With the River of Objects flowing again, the land will become bountiful with objects of every class, interface, and struct imaginable. The island has been saved. You turn your attention towards the scattered islands on the horizon and your final destination beyond: a confrontation with The Uncoded One.

Narrative 7: The Harvest of Objects

A few days have passed since the Fountain of Objects was restored, but the land has already become more vibrant and lush. New objects and classes, unseen for thousands of clock cycles, have been found again. The classes described in this level represent a collection of some of the most versatile and interesting ones you have seen, and you gather some up for the rest of your journey.

Day 44 Challenge 1/2: The Robot Pilot

When we first made the Hunting the Manticore game in Level 14, we required two human players: one to set up the Manticore's range from the city and the other to destroy it. With Random, we can turn this into a single-player game by randomly picking the range for the Manticore.

Objectives:

- Modify your Hunting the Manticore game to be a single-player game by having the computer pick a random range between 0 and 100 (you can use the `Random` class for that).
- Answer this question: How might you use inheritance, polymorphism, or interfaces to allow the game to be either a single player (the computer randomly chooses the starting location and direction) or two players (the second human determines the starting location and direction)?

Day 44 Challenge 2/2: Time in the Cavern

With LocalDateTime and Duration, you can track how much time a player spends in the Cavern of Objects to beat the game. With these tools, modify your Fountain of Objects game to display how much time a player spent exploring the caverns.

Objectives:

- When a new game begins, capture the current time using LocalDateTime.

- When a game finishes (win or loss), capture the current time.
- Use Duration to compute how much time elapsed and display that to the player.

Day 45 Challenge 1/2: Lists of Commands

In Level 27, we encountered a robot with an array to hold commands to run. But we could make the robot have as many commands as we want by turning the array into a list. Revisit that challenge to make the robot use a list instead of an array, and add commands to run until the user says to stop.

Objectives:

• Change the Robot class to use a `List<RobotCommand>` instead of an array for its Commands (you can use an `ArrayList` as the concrete class to base the List on). • Instead of looping three times, go until the user types stop.Then run all of the commands created.

Narrative 8: Gathering Medallions

You stand on the east coast of the vast island of Object-Oriented Programming. A strong breeze blows salty air across your face as the sun rises above the watery horizon. You study your maps. Ahead of you lies the scattered Islands of Advanced Features, and beyond that, the Domain of the Uncoded One— your final destination. Scattered across the islands are the ancient Medallions of Code, made of nearly indestructible binarium, each of which grants True Programmers additional powers. Each medallion is guarded by the islands' inhabitants, who serve as protectors and stewards. Without being Programmers, they are unable to use them themselves. By visiting these islands, you can acquire these medallions, gain the powers they provide, and maybe even enlist these guardians to help in the final assault at the Uncoded One's domain. Yet time is short; every clock cycle you delay gives the Uncoded One more time to reign destruction and may even give it the time needed to uncode and unravel the world itself. You grab a pencil and begin making tentative plans on your map about the final leg of your journey through the Islands of Advanced Features.

Day 46 Challenge 2/2: The Feud

On the Island of Packages, two families of ranchers are caretakers of the Medallion of Packages. They are in a feud. They are the iFields and the McDroids. The iFields ranch sheep and pigs and the McDroids ranch pigs and cows. Since both have pigs, they keep having conflicts. The two families will give you the Medallion of Packages if you can resolve the dispute and help them track their animals.

Objectives:

- Create a Sheep class in the ifield package (fully qualified name of ifield.Sheep).
- Create a Pig class in the ifield package (fully qualified name of ifield.Pig).
- Create a Cow class in the mcdroid package (fully qualified name of mcdroid.Cow).
- Create a Pig class in the mcdroid package (fully qualified name of mcdroid.Pig).
- For your main method, add import statement for both the ifield and mcdroid packages. Make new instances of all four classes. There are no conflicts with Sheep and Cow, so make sure you can create new instances of those with new

Sheep() and new Cow(). Resolve the conflicting Pig classes with either an alias or fully qualified names.

Day 47 Challenge 1/2: Dueling Traditions

The inhabitants of Programain, guardians of the Medallion of Organization, seem to be hiding from you, peering at you through shuttered windows, leaving you alone on the dusty streets. The only other people on the road stand in front of you—a gray-haired wrinkle-faced woman and two toughs who stand just behind her. “We heard a Programmer might be headed our way. But you’re no True Programmer. In the Age Before, programmers used packages and split their programs into multiple files. You probably don’t even know what those things are. Bah.” She spits on the ground and demands you leave, but you know you can win her and the townspeople over—and acquire the Medallion of Organization—if you can show you know how to use the tools she named. Do the following with one of the larger programs you have created in another challenge.

Objectives:

- Use a package for all the code of the chosen challenge (if you haven't already done so)
- Place each type in its own file. (Small types like enumerations or records can be an exception.)
- Answer this question: Having used both a single file and multi-file program, (and possibly no package versus an explicitly declared package) which do you prefer and why?

Day 47 Challenge 2/2: Safer Number Crunching

“Master Programmer! We need your help! We are but humble number crunchers. We read numbers in, work with them for a bit, then display the results. But not everybody enters good numbers. Sometimes, we type in wrong things by accident. And sometimes, somebody does it on purpose. Trolls, looking to cause trouble, I tell ya! “We’ve heard about those methods that return so-called 'Optionals' that cannot fail or break. We know you’re here looking for Medallions and allies. If you can help us with this, the Medallion of Optional is yours, and we will join you at the final battle.”

Objectives:

- Create a program that asks the user for an int value. Using Integer.parseInt(), and a try-catch block create a method that returns an `Optional<Integer>`
- Let the program call that method and loop until the user enters a valid value.
- Extend the program to do the same for both double and boolean. Note: a boolean is a bit tricky, as it will almost always return false, also for inputs like `null` or "very very true!". Make it only return false when it is really "False" or such (ignoring case).

Day 48 Challenge: Better Random

The villagers of Randetherin often use the Random class but struggle with its limited capabilities. They have asked for your help to make Random better. They offer you the Medallion of Powerful Methods in exchange. Their complaints are as follows:

- Random.nextDouble() only returns values between 0 and 1, and they often need to be able to produce random double values between 0 and another number, such as 0 to 10. • They need to randomly choose from one of several strings, such as

"up", "down", "left", and "right", with each having an equal probability of being chosen. • They need to do a coin toss, randomly picking a bool, and usually want it to be a fair coin toss (50% heads and 50% tails) but occasionally want unequal probabilities. For example, a 75% chance of true and a 25% chance of false.

Objectives:

- Create a new class to that contains methods that extends those of Random. Call it `RandomUtils`, for example.
- Add a static nextDouble method that gives a maximum value for a randomly generated double.
- Add a static nextString method that allows you to pass in any number of string values (using varargs) and randomly pick one of them.
- Add a static coinFlip method that randomly picks a boolean value.
- Made an overload of the coinFlip method that has a parameter that indicates the frequency of heads coming up, which should default to 0.5, or 50% of the time.
- Bonus: try to make one coinFlip method refer to the other coinFlip method to reduce code duplication. • Answer this question: In your opinion, would it be better to make a derived AdvancedRandom class that adds these methods or use static methods in the current kind of utility class, and why?

Day 49 Exepti’s Game

On the Island of Exceptions, you find the village of Excepti, which has seen little happiness and joy since the arrival of The Uncoded One. The Exceptians used to have a game that they played called Cookie Exception. The village leader, Noit Pecxe, promises the warriors of Excepti will join you against the Uncoded One if you can recreate their ancient tradition in a program. Noit offers you the Medallion of Exceptions as well. Cookie Exception is played by gathering nine chocolate chip cookies and one oatmeal raisin cookie. The cookies are mixed and put in a dark room with two players who can’t see the cookies. Each player takes a turn picking a cookie randomly and shoving it in their mouth without seeing whether it is a delicious chocolate chip cookie or an awful oatmeal raisin cookie. If they pick wrong and eat the oatmeal raisin cookie, they lose. If their opponent eats the oatmeal raisin cookie, then they win.

Objectives:

- The game will pick a random number between 0 and 9 (inclusive) to represent the oatmeal raisin cookie.
- The game will allow players to take turns picking numbers between 0 and 9.
- If a player repeats a number that has been already used, the program should make them select another. Hint: If you use a `List<Integer>` (or `Set<Integer>`) to store previously chosen numbers, you can use its contains method to see if a number has been used before.
- If the number matches the one the game picked initially, an exception should be thrown, terminating the program. Run the program at least once like this to see it crash.
- Put in a try/catch block to handle the exception and display the results.
- Answer this question: Did you make a custom exception type or use an existing one, and why did you choose what you did?
- Answer this question: You could write this program without exceptions, but the requirements demanded an exception for learning purposes. If you didn’t have

that requirement, would you have used an exception? Why or why not?

Day 50 Challenge: The Sieve

The Island of Predicata is home to the Numeromechanical Sieve, a machine that takes numbers and judges them as good or bad numbers. In ancient times, the sieve could be supplied with a single method to use as a filter by the island's rulers, making the sieve adaptable as leadership changed over time. The Predicatans will give you the Medallion of Predicates if you can reforge their Numeromechanical Sieve.

Objectives:

- Create a Sieve class with a public boolean isGood(int number) method. This class needs a constructor with a Predicate parameter that can be invoked later within the isGood method.
- Define methods with an int parameter and a boolean return type for the following: (1) returns true for even numbers, (2) returns true for positive numbers, and (3) returns true for multiples of 10.
- Create a program that asks the user to pick one of those three filters, constructs a new Sieve instance by passing in one of those methods as a parameter, and then ask the user to enter numbers repeatedly, displaying whether the number is good or bad depending on the filter in use.
- Answer this question: Describe how you could have also solved this problem with inheritance and polymorphism. Which solution seems more straightforward to you, and why?

Day 51 Challenge: Charberry Trees

The Island of Observia survives by harvesting and eating the fruit of the native charberry trees. Harvesting charberry fruit requires three people and an afternoon, but two is enough to feed a family for a week. Charberry trees fruit randomly, requiring somebody to frequently check in on the plants to notice one has fruited. Observia will give you the Medallion of Observers if you can help them with two things: (1) automatically notify people as soon as a tree ripens and (2) automatically harvest the fruit. Their tree looks like this:

```
class Observia {

    public static void main(String[] args ) {
        CharberryTree tree = new CharberryTree();
        while (true) tree.maybeGrow();
    }
}

class CharberryTree {
    private Random random = new Random();
    private boolean ripe = false;

    public void maybeGrow() {

        // Only a tiny chance of ripening each time, but we try a lot!
        if (random.nextDouble() < 0.00000001 && !ripe)
            ripe = true;
```

```
    }
}
```

Objectives:

- Make a new project that includes the above code.
- Make a Notifier class with a `handle` method that displays something like “A charberry fruit has ripened!” to the console window.
- Make a Harvester class that knows about the tree. When it is called, it sets the tree's "ripe" field back to false
- make the CharberryTree class implement the Listener (also known as Observer) pattern. (If you don't know it, Google "observer pattern java")
- Hint: for ease of programming, it may help to have the `handle` method in the Notifier class also take an argument of type CharberryTree
- Update your main method to create a tree, notifier, and harvester, and get them to work together to grow, notify, and harvest forever.

Day 52 Challenge 1/2: The Lambda Sieve

The city of Lambdan, also on the Island of Predicata, believes that the great Numeromechanical Sieve, which you worked on in Level 36, could be made better by using lambda expressions instead of regular, named methods. If you can help them convince island leadership to make this change, they will give you the Lambda Medallion and pledge the Lambdani Fleet’s assistance in the coming final battle.

Objectives:

- Modify your The Sieve program from Level 36 to use lambda expressions for the constructor instead of named methods for each of the three filters.
- Answer this question: Does this change make the program shorter or longer?
- Answer this question: Does this change make the program easier to read or harder?

Day 52 Challenge 2/2: The Long Game

The island of Io has a long-running tradition that was destroyed when the Uncoded One arrived. The inhabitants of Io would compete over a long period of time to see who could press the most keys on the keyboard. But the Uncoded One’s arrival destroyed the island’s ability to use the Medallion of Files, and the historic competitions spanning days, weeks, and months have become impossible. As a True Programmer, you can use the Medallion of Files to bring back these long-running games to the island.

Objectives:

- When the program starts, ask the user to enter their name.
- By default, the player starts with a score of 0.
- Add 1 point to their score for every character they type. Hint: it appears the Ioans of yore did not accept anything less than a Holy Whole Line that was ended with an Enter key press, so the characters will only be counted after the Enter key has been pressed. The enter key does not count towards the score itself, however, as one should not overuse sacred keys...
- Display the player’s updated score after each press of the Enter key.
- End the game when the player enters the holy word IO (and then of course Enter)
- When the game ends, save the user's score in a file. Hint: Consider saving this to a file named [username].txt. For this challenge, you can assume the user

doesn’t enter a name that would produce an illegal file name (such as *).
- When a user enters a name at the start, if they already have a previously saved score, start with that score instead.

Day 53 Challenge: The Potion Masters of Statem

The island of Statem is home to skilled potion masters in need of some help. Potions are mixed by adding one ingredient at a time until they produce a valuable potion type. The potion masters will give you the Statem Medallion if you help them make a program to build potions according to the rules below:

- All potions start as water.
- Adding stardust to water turns it into an elixir.
- Adding snake venom to an elixir turns it into a poison potion.
- Adding dragon breath to an elixir turns it into a flying potion.
- Adding shadow glass to an elixir turns it into an invisibility potion.
- Adding an eyeshine gem to an elixir turns it into a night sight potion.
- Adding shadow glass to a night sight potion turns it into a cloudy brew.
- Adding an eyeshine gem to an invisibility potion turns it into a cloudy brew.
- Adding stardust to a cloudy brew turns it into a wraith potion.
- Anything else results in a ruined potion.

Objectives:

- Create enumerations for the potion and ingredient types.
- Tell the user what type of potion they currently have and what ingredient choices are available.
- Allow them to enter an ingredient choice.
- Change the current potion type according to the rules above.
- Allow them to choose whether to complete the potion or continue before adding an ingredient. If the user decides to complete the potion, end the program.
- When the user creates a ruined potion, tell them and start over with water.
- Bonus: some people in the outside world would compare the potions to a 'finite state machine'. Read a bit about it, and think of the other options you would have to implement it.

Day 54 Challenge: The Two Lenses

The Guardian of the Medallion of Lambdas, Lennik, has long awaited when he can return the Medallion to a worthy programmer. But he only wants to give it to somebody who truly understands the value of lambdas. He requires you to build a solution to a simple problem two times over. Lennik gives you the following array of positive numbers: [1, 9, 2, 8, 3, 7, 4, 6, 5]. He asks you to make a new collection from this data where:

- All the odd numbers are filtered out, and only the even should be considered.
- The numbers are in order.
- The numbers are doubled.

For example, with the array above, the odd/even filter should result in 2, 8, 4, 6. The ordering step should result in 2, 4, 6, 8. The doubling step should result in 4, 8, 12, 16 as the final answer.

Objectives:

- Write a method that will take an int[] as input and produces a `List<Integer>` that meets all three of the conditions above using only procedural code— if statements, switches, and loops. Hint: the static Arrays.sort or Collections.sort method might be a useful tool here.
- Write a method that will take an int[] as input and produces a `List<Integer>` that meets the three above conditions using a lambda expression.
- Run both methods and display the results to ensure they all produce good answers.
- Answer this question: Compare the size and understandability of these approaches. Does any stand out as being particularly good or particularly bad?
- Answer this question: Of the two approaches, which is your personal favorite, and why?

Day 55 Challenge: The Repeating Stream

In Threadia, there is a stream that generates numbers once a second. The numbers are randomly generated, between 0 and 9. Occasionally, the stream generates the same number more than once in a row. A repeat number like this is significant—an omen of good things to come. Unfortunately, since the Uncoded One's arrival, Threadians haven't been able to monitor the stream while it produces numbers. Either the stream generates numbers while nobody watches, or they watch while the stream produces no numbers. The Threadians offer you the Medallion of Threads willingly and ask you to use it to make both possible at the same time. Build a program to generate numbers while simultaneously allowing a user to flag repeats.

Objectives:

- Make a RecentNumbers class that holds at least the two most recent numbers. • Make a method that loops forever, generating random numbers from 0 to 9 once a second. Hint: Thread.sleep can help you wait.
- Write the numbers to the console window, put the generated numbers in a RecentNumbers object, and update it as new numbers are generated.
- Make a thread that runs the above method.
- In a second loop, wait for the user to push the Enter key (on the main thread or another new thread). When the user presses the Enter key, check if the last two numbers are the same. If they are, tell the user that they correctly identified the repeat. If they are not, indicate that they got it wrong.
- Use a lock to ensure that only one thread accesses the shared data at a time.

Day 56 Challenge: Asynchronous Random Words - the Start

On the Island of Callus, you meet Awat, who tells you that being a True Programmer can't be all that hard. His ancestors have been the stewards of the Asynchronous Medallion, yet Awat uses it as a food dish for his cat. "A thousand monoids with a thousand random generators will also eventually produce 'hello world'!" he claims. Indeed, they could, but you know it would take a while. With tasks, you can allow a human to pick a word and randomly generate the word asynchronously. Doing this will show Awat how long it will take to randomly generate the words "hello" and "world," convincing him that a Programmer's skills mean something.

Objectives:

- Make a class that implements `Callable<Integer>`, add a constructor that takes a string. Its call method should take the string's length and generate an equal

number of random characters. It is okay to assume all words only use lowercase letters. One way to randomly generate a lowercase letter is (char)('a' + random.nextInt(26)). This method should loop until it randomly generates the target word, counting the required attempts. The return value is the number of attempts.
- Schedule it using an ExecutorService, the service should return a 'Future'. Show the result when it is obtained.
- Have your main method ask the user for a word. Run the Callable object and await its result and display it. Note: Be careful about long words! For me, a five-letter word took several seconds, and my math indicates that a 10-letter word may take nearly two years.
- Use System.currentTimeMillis() before and after the async task runs to measure the wall clock time it took. Display the time elapsed.

Day 57 Challenge: Many Random Words

Awat is impressed with what you did in the last challenge but thinks it could be better. “Why not generate ‘hello’ and ‘world’ in parallel?” he asks.“You do that, and I’ll let you take this medallion off of me.”

Objectives:

- Modify your program from the previous challenge to allow the main thread to keep waiting for the user to enter more words. For every new word entered, create and run a task to compute the attempt count and the time elapsed and display the result, but then let that run asynchronously while you wait for the next word.You can generate many words in parallel this way.

Day 58 Challenge: Altar of Publication

To acquire the Medallion of Publishing, you must place a published program on the Altar of Publication.

Objectives:

- Select a program of yours for publication. This can be anything from Hello World to the most complex program you have made.
- Check out how to make an "executable jar" using IntelliJ IDEA (or the standard Java compiler)
- Create the executable jar, and test using the command line that it runs.
- Move the program to the target computer.
- Run the program successfully.
- Note: You will learn much by actually moving this to another computer. If you only have one, send it to a friend by email or the Internet. But if all of this fails, you can call the challenge done anyway.
- Bonus: if your project has its source code on GitHub, try make a Release on GitHub.

Narrative 9: The Uncoded One

The final battle has arrived. On a volcanic island, enshrouded in a cloud of ash, you have reached the lair of the Uncoded One. You have prepared for this fight, and you will return Programming to the lands. Your allies have gathered to engage the Uncoded One’s minions on the volcano’s slopes while you (and maybe a companion or two) strike

into the heart of the Uncoded One's lair to battle and destroy it. Only a True Programmer will be able to survive the encounter, defeat the Uncoded One, and escape!

Day 59 Challenge: Building Character

This challenge is deceptively complex: it will require building out enough of the game's foundation to get two characters taking turns in a loop. (Sure, they'll be doing nothing, but that's still a big step forward!)

Objectives:

- The game needs to be able to represent characters with a name and able to take a turn. (We'll change this a little in the challenge Characters, Actions, and Players.)
- The game should be able to have skeleton characters with the name SKELETON.
- The game should be able to represent a party with a collection of characters.
- The game should be able to run a battle composed of two parties—heroes and monsters. A battle needs to run a series of rounds where each character in each party (heroes first) can take a turn.
- Before a character takes their turn, the game should report to the user whose turn it is. For example, "It is SKELETON's turn…."
- The only action the game needs to support at this point is the action of doing nothing (skipping a turn). This action is done by displaying text about doing nothing, resting, or skipping a turn in the console window. For example, "SKELETON did NOTHING."
- The game must run a battle with a single skeleton in both the hero and the monster party. At this point, the two skeletons should do nothing repeatedly. The game might look like the following:

```
It is SKELETON's turn...
SKELETON did NOTHING.
It is SKELETON's turn...
SKELETON did NOTHING.
...
```

- Optional: Put a blank line between each character's turn to differentiate one turn from another.
- Optional: At this point, the game will run automatically. Consider adding a Thread.sleep(500); to slow the game down enough to allow the user to see what is happening over time.

Day 60 Challenge: The True Programmer

Our skeletons need a hero to fight. We'll do that by adding in the focal character of the game, the player character, which represents the player directly, called the True Programmer by in-game role. The player will choose the True Programmer's name.

Objectives:

- The game must represent a True Programmer character with a name supplied by the user.
- Before getting started, ask the user for a name for this character.
- The game should run a battle with the True Programmer in the hero party vs. a skeleton in the monster party. The game might look like the following this

challenge:

```
It is TOG's turn...
TOG did NOTHING.
It is SKELETON's turn...
SKELETON did NOTHING.
...
```

Day 61 Challenge: Actions and Players

The previous two challenges have had the characters taking turns directly. But instead of characters deciding actions, the player controlling the character's team should pick the action for each character. Eventually, there will be several action types to choose from (do nothing, attack, use item, etc.). There will also be multiple player types (computer/AI and human input from the console window). A player is responsible for picking an action for each character in their party. The game should ask the players to choose the action rather than asking the characters to act for themselves. For now, the only action type will be a do-nothing action, and the only player type will be a computer player. This challenge does not demand that you add new externally visible capabilities but make any needed changes to allow the game to work as described above, with players choosing actions instead of characters. If you are confident your design already supports this, claim the XP now and move on.

Objectives:

- The game needs to be able to represent action types. Each action should be able to run when asked.
- The game needs to include a do-nothing action, which displays the same text as in previous challenges (for example, "SKELETON did NOTHING.")
- The game needs to be able to represent different player types. A player needs the ability to pick an action for a character they control, given the battle's current state.
- The game needs a sole player type: a computer player (a simple AI of sorts). For now, the computer player will simply pick the one available option: do nothing (and optionally wait a bit first with Thread.sleep).
- The game must know which player controls each party to ask the correct player to pick each character's action. Set up the game to ask the player to select an action for each of their characters and then run the chosen action.
- Put a simple computer player in charge of each party.
- Note: To somebody watching, the end result of this challenge may look identical to before this challenge.

Day 62 Challenge: Attacks

In this challenge, we will extend our game by giving characters attacks and allowing players to pick an attack instead of doing nothing. We won't address tracking or dealing out damage yet.

Objectives:

- The game needs to be able to represent specific types of attacks. Attacks all have a name (and will have other capabilities later).
- Each character has a standard attack, but this varies from character to character. The True Programmer's (the player character's) attack is called

punch. The Skeleton’s attack is called bone crunch.
- The program must be capable of representing an attack action, our second action type. An attack action must represent which attack is being used and the target of the attack. When this action runs, it should state the attacker, the attack, and the target. For example: “TOG used PUNCH on SKELETON.”
- Our computer player should pick an attack action instead of a do-nothing action. The attack action can be simple for now: always use the character’s standard attack and always target the other party’s first character. (If you want to choose a random target or some other logic, you can.)
- The game should now run more like this:

```
It is TOG's turn...
TOG used PUNCH on SKELETON.
It is SKELETON's turn...
SKELETON used BONE CRUNCH on TOG.
...
```

- Hint: The player will need access to the list of characters that are potential targets. In my case, I passed my Battle object (which represents the whole battle and had access to both parties and all their characters) to the player. I then added methods to Battle where I could give it a character, and it would return the character’s party (getPartyFor(GameCharacter)) or the opposing party (getEnemyPartyFor(GameCharacter)).

Day 63 Challenge: Damage and HP

Now that our characters are attacking each other, it is time to let those attacks matter. In this challenge, we will enhance the game to give characters hit points (HP). Attacking should reduce the HP of the target down to 0 but not past it. Reaching 0 HP means death, which we will deal with in the next challenge.

Objectives:

- Characters should be able to track both their initial/maximum HP and their current HP. The True Programmer should have 25 HP, while skeletons should have 5.
- Attacks should be able to produce attack data for a specific use of the attack. For now, this is simply the amount of damage that they will deal this time, though keep in mind that other challenges will add more data to this, including things like the frequency of hitting or missing and damage types.
- The punch attack should deliver 1 point of damage every time, while the bone crunch attack should randomly deal 0 or 1. Hint: Remember that Random can be used to generate random numbers. random.Next(2) will generate a 0 or 1 with equal probability.
- The attack action should ask the attack to determine how much damage it caused this time and then reduce the target’s HP by that amount. A character’s HP should not be lowered below 0.
- The attack action should report how much damage the attack did and what the target’s HP is now at. For example, “PUNCH dealt 1 damage to SKELETON.” “SKELETON is now at 4/5 HP.”
- When the game runs after this challenge, the output might look something like this:

```
It is TOG's turn...
TOG used PUNCH on SKELETON.
PUNCH dealt 1 damage to SKELETON.
SKELETON is now at 4/5 HP.
It is SKELETON's turn...
SKELETON used BONE CRUNCH on TOG.
BONE CRUNCH dealt 0 damage to TOG.
TOG is now at 25/25 HP.
...
```

Day 64 Challenge 1/2: Death

When a character’s HP reaches 0, it has been defeated and should be removed from its party. If a party has no characters left, the battle is over.

Objectives:

- After an attack deals damage, if the target’s HP has reached 0, remove them from the game.
- When you remove a character from the game, display text to illustrate this. For example, “SKELETON has been defeated!”
- Between rounds (or between character turns), the game should see if a party has no more living characters. If so, the battle (and the game) should end.
- After the battle is over, if the heroes won (there are still surviving characters in the hero party),then display a message stating that the heroes won, and the Uncoded One was defeated. If the monsters won, then display a message saying that the heroes lost and the Uncoded One’s forces have prevailed.

Day 64 Challenge 2/2: Battle Series

The game runs as a series of battles, not just one. The heroes do not win until every battle has been won, while the monsters win if they can stop the heroes in any battle.

Objectives:

- There is one party of heroes but multiple parties of monsters. For now, build two monster parties: the first should have one skeleton. The second has two skeletons.
- Start a battle with the heroes and the first party of monsters. When the heroes win, advance to the next party of monsters. If the heroes lose a battle, end the game. If the monsters lose a battle, move to the next battle unless it is the last.

Day 65 Challenge: The Uncoded One

It is time to put the final boss into the game: The Uncoded One itself. We will add this in as a third battle.

Objectives:

- Define a new type of monster: The Uncoded One. It should have 15 HP and an unraveling attack that randomly deals between 0 and 2 damage when used. (The Uncoded One ought to have more HP than the True Programmer, but much more than

15 HP means the Uncoded One wins every time. We can adjust these numbers later.)
- Add a third battle to the series that contains the Uncoded One.

Day 66 Challenge: The Player Decides

We have one critical missing piece to add before our core game is done: letting a human play it.

Objectives:

- The game should allow a human player to play it by retrieving their action choices through the console window. For a human-controlled character, the human can use that character's standard attack or do nothing. It is acceptable for all attacks selected by the human to target the first (or random) target without allowing the player to pick one specifically. (You can let the player pick if you want, but it is not required.) The following is one possible approach:

```
It is TOG's turn...
1 – Standard Attack (PUNCH)
2 – Do Nothing
What do you want to do? 2
```

- As the game is starting, allow the user to choose from the three following gameplay modes: player vs. computer (the human in charge of the heroes and the computer controlling the monsters), computer vs. computer (a computer player running each team, as we have done so far), and human vs. human, where a human picks actions for both sides.
- Hint: There are many ways you could approach this. My solution was to build a MenuItem record that held information about options in the menu. It included the string description, boolean isEnabled, and Action ActionToPerform. Action is my interface representing any of the action types, with implementations like DoNothingAction and AttackAction. I have a method that creates the list of menu items, and it produces a new MenuItem instance for each choice. The code that draws the menu iterates through the MenuItem instances and displays them with a number. After getting the number, I find the right MenuItem, extract the Action, and return it. That means I create many Action objects that don't get used, but it is a system that is easy to extend in future challenges.

Day 67 Challenge: The Game's Status

This challenge gives us a clearer representation of the status of the game.

Objectives:

- Before a character gets their turn, display the overall status of the battle. This status must include all characters in both parties with their current and total HP.
- You must also somehow distinguish the character whose turn it is from the others with color, a text marker, or something similar.
- Note: You have flexibility in how you approach this challenge, but the following shows one possibility (with the current character colored yellow instead of white): ``` == BATTLE == TOG (25/25)

- ---------- VS ---------- SKELETON (5/5)

SKELETON (5/5)

Day 68 Challenge: Items

Each party has a shared inventory of items. Players can choose to use an item as an action. We will add a health potion item that players can use as an action.

Objectives:

- The game must support adding consumable items with the ability to use one as an action. Item types could be potentially very broad (keep that in mind when choosing your design), but all that is required now is a health potion item type. Items are usable, and when used, the reaction depends on the item type.
- A health potion is the only item type we need to add here. It should increase the user's HP by 10 points when used. In doing so, the HP should not rise above the character's maximum HP.
- The entire party shares inventory. Ensure parties can hold a collection of items.
- Start the hero party with three health potions. Give each monster party one health potion.
- The game must support the inclusion of a use item action, along with the item to use. When this action runs, it should cause the item's effect to occur and remove the item from the inventory.
- The computer player should consider using a potion when (a) the team has a potion in their inventory and (b) the character's health is under half. Under these conditions, use a potion 25% of the time.
- The console player should have the option to use a potion if the party has one.
- Note: Digging through the party inventory for potions is a little tricky. It is reasonable to assume (for now) that all items in the inventory are health potions. That will be a correct assumption until you make other item types. This assumption simplifies the changes you need to make to the different player types.

Day 69 Challenge: Gear

Characters can equip gear that allows them to have a second special attack. A party can have gear in their shared inventory, but unlike items, gear is not usable until a character equips it, and it takes a turn to equip gear from inventory.

Objectives:

- The game must support the concept of gear. All gear has a name and an attack they provide.
- Each character can equip one piece of gear.
- Each party also has a collection of unequipped gear.
- Add the ability to perform an equip action, which knows what gear is being equipped. When this action runs, it should move the gear from the party's inventory to the character.

- If a character already has something equipped, the previously equipped gear should be unequipped and moved back to the party’s shared inventory.
- The computer player should equip gear. If a character has nothing equipped but the party has equipable gear, the computer player should choose to equip the gear 50% of the time.
- Note: If you also did the Items challenge, using potions should be a priority over equipping gear.
- The console player should also have the option to equip gear. If there is more than one thing to equip, allow the player to choose from all available options.
- The computer player should prefer the attack provided by equipped gear when one is available. Gearbased attacks are typically stronger.
- If gear is equipped, the human player should be able to pick either the standard attack or the gearbased attack.
- The True Programmer character should start the game with a sword item equipped. The sword should have a slash attack that deals two damage.
- Create a dagger with the attack stab that reliably deals 1 point of damage.
- Start the first battle’s skeleton with a dagger equipped. This one was prepared for battle.
- Put two daggers in the team inventory for the second battle. Both skeletons will be able to use a dagger, but they will have to equip it first.These two were less prepared.
- Optional: If you did The Game’s Status, consider showing what gear each character has equipped.

Day 70 Challenge: Stolen Inventory

Requires that you have done either Items or Gear. This feature will allow us to have a basic loot system. When a character dies, the opposing side should immediately recover the dead character’s equipped gear. When the monster party is eliminated, the monster party’s unequipped gear and items should be transferred to the hero party. If you only did Items or Gear, some of the objectives below will not apply to you. Do the ones that apply.

Objectives:

- If you did the Items challenge, when a party is eliminated, transfer all items from the losing party’s inventory items to the winning party. Display a message that indicates which items have been acquired. Note: It is okay only to do this when the hero party wins. When the monster party wins a battle, the game ends.
- If you did the Gear challenge, when a party is eliminated, transfer all unequipped gear from the losing party’s inventory to the winning party. Display a message that indicates when gear has been acquired.
- If you did the Gear challenge, when a character is eliminated, transfer any equipped gear to the winning party’s inventory. Display a message that states gear that was acquired.

Day 71 Challenge: Vin Fletcher

The True Programmer does not have to fight the Uncoded One alone! The hero party can have other heroes (companions) that should each get their turn to fight. In this challenge, we will add our favorite arrow maker, Vin Fletcher, to the game. This challenge will also add in the possibility for an attack to sometimes miss.

Objectives:

- When an attack generates attack data, it must also include a probability of success. 0 is guaranteed failure, 1 is guaranteed success, 0.5 is 50/50, etc.
- Modify your attack action to account for the possibility of missing. If an attack misses, don't damage the target and instead report that the attack missed. For example, "VIN FLETCHER MISSED!"
- Create a new character type to represent Vin Fletcher. He starts with 15 HP. If you did the Gear challenge, Vin should have the same standard punch attack the True Programmer has and equip him with a Vin's Bow gear with an attack called quick shot that deals 3 damage but only succeeds 50% of the time. If you did not do the Gear challenge, give Vin quick shot as his standard attack.

Day 72 Challenge: Attack Modifiers

An attack modifier is a character attribute that adjusts attacks involving them. We will make only defensive attack modifiers for this challenge, which apply when a character is on the receiving end of an attack. You can add offensive attack modifiers as a part of the Making It Yours challenge.Attack modifiers are applied when an attack action is being resolved. An attack modifier takes the current attack data as input and produces new, potentially altered attack data. For example, it could reduce the damage done (a resistance to the attack) or increase it (a weakness to the attack). This challenge will add a new monster type with resistance to all attacks, reducing incoming damage by 1.

Objectives:

- The game must be able to represent attack modifiers. Attack modifiers take attack data as an input and produce new attack data as a result, possibly tweaking the attack data in the process. Attack modifiers also have names.
- All characters should be able to have a defensive attack modifier.
- When performing an attack, see if the target character has a defensive attack modifier before delivering damage to the target. If it does, allow the modifier to manipulate the attack data and use the modified results instead.
- When attack modifiers adjust damage, they should report that they changed the damage and by how much. For example, "STONE ARMOR reduced the attack by 1 point."
- The game must support a new stone amarok character. Stone amaroks have 4 HP and a standard bite attack that deals 1 damage. They also have a defensive attack modifier called stone armor that reduces all attacks by 1 damage. If your heroes still only have a 1-point punch attack, change something so that the heroes can survive an encounter with stone amaroks.
- Add a battle that includes two stone amaroks as monsters.

Day 73 Challenge: Damage Types

Requires that you have done Attack Modifiers. Attacks should have a damage type that may affect how it works. For now, our damage types must include at least normal and decoding, but you can add additional damage types (such as fire or ice) if you want. The damage type primarily adds flavor to the game but also leads to additional game mechanics. For example, we will give the True Programmer a defensive attack modifier called Object Sight, which gives the character resistance to decoding damage.

Objectives:

- Attack data should include a damage type. You are free to define any damage types that you think make sense, but include the damage types of normal (or similar) and decoding. Hint:An enumeration might be a good choice for this.
- Have each attack use one of these types when producing damage data. Use whatever damage types you wish, but make The Uncoded One's unraveling attack be decoding damage.
- Give the True Programmer character a defensive attack modifier called Object Sight that reduces decoding damage by 2.
- Increase the unraveling attack to deal 0 to 4 damage randomly, instead of 0 to 2, ensuring that this attack is the single most powerful attack in the game (but one that the hero has resistance to).

Day 74 Challenge: Making it Yours

This is perhaps the most exciting challenge: the one where you get to do whatever you want to the game. This challenge is also repeatable. If you'd rather make four of your own enhancements instead of the other challenges listed above, go for it. Use your best judgment when awarding yourself XP, comparing what you've chosen with the other challenges here. The possibilities are limitless, but here are some ideas:

- More heroes. You could add Mylara and Skorin, whose gear (or standard attack) is the Cannon of Consolas, which deals 1 damage most of the time, 2 damage every multiple of 3 or 5, and 5 damage every multiple of 3 and 5. (This adds some continuity from the early part of the book!)
- Add more item types. Maybe Simula's soup is a full heal, restoring HP to its maximum.
- Add offensive attack modifiers.
- Let characters equip more than one piece of gear.
- Let gear provide offensive and defensive attack modifiers. (A Binary Helm that reduces all damage done by 1 when equipped, for example.)
- Experience Points. As heroes defeat monsters, give them XP to track their accomplishments.
- More monster types.
- Add attack side effects, which allow an attack to run other logic when an attack happens. Maybe a Shadow Octopoid with a grapple attack that has a chance of stealing equipped gear.
- Load levels from a file instead of hardcoding them in the game.
- Display a small indicator in the status display that gives you a warning if a character is close to death. Perhaps a yellow [!] if the character is under 25% and a red [!] if the character is under 10%.
- Allow characters to taunt their enemies with messages like the Uncoded One saying <> and a skeleton saying, "We will repel your spineless assault!"
- Allow characters to have temporary effects applied to them. For example, a poison effect that deals 1 damage per turn for three turns.
- Strip out all Player's Guide-specific elements and re-theme the game into your own creation.

Day 75 Challenge: Restoring Balance

Depending on the challenges you completed, the game is likely not very balanced anymore. Either the heroes or the monsters win all the time. Without changing logic and mechanics, adjust things like potion counts, attack strengths and probabilities,

the number of battles and monsters they contain to produce a version of the game where if you play wisely, you are likely to win, but if you play poorly, you will lose.

Objectives:

- Tweak aspects of the game (no need to write new logic, just change parameters and character counts, etc.) until you have something challenging and fun.
- Do what you can to ensure that The Uncoded One feels formidable.

Narrative 10: The True Programmer

You slice your sword through the smoky form of the Uncoded One for the last time. It begins to disintegrate, binary streams flowing out of it until it bursts apart in a dazzling blue light. You have done it. You have defeated the Uncoded One. You step out of the cave’s entrance and back onto the slopes of the volcano as ash falls like snow from the sky. Your allies have turned the tide of the battle against the Uncoded One’s minions, and they surge through their final ranks as the remnants begin to scatter in retreat. You and your allies have saved these realms from the grasp of the Uncoded One. As the sun sets, alighting the ash cloud in a crimson glow, you meet up with your old friends to celebrate with a feast. They’re all there. Vin Fletcher with his arrows. Simula with her soups. Mylara and Skorin, with an improved cannon that helped break the Uncoded One’s lines during the battle. Everyone else that you met as you voyaged through the Realms. As the feast begins to wind down—and after you have had three full bowls of stew—Simula speaks to you. “You have shown that you are, without question, a True Programmer. To me. To us. To the Uncoded One. The power of Programming can return to these islands once again. But tell me, brave Programmer, what will you do next?” You take a slow, deep breath as you ponder the question before you respond.