2D Arrays and Inheritance

Robot Outfit Matcher

# Main Objectives

* Work with 2D arrays.
* Work with Inheritance between Classes.

# Prior Knowledge

* Students should know how to concatenate Strings.
* Students should know how to build and add objects to Greenfoot.

# Greenfoot Background

This lab uses the Greenfoot API in the following ways: Students will need to build, place, and remove Actors to/from the screen, be able to play sounds and add a small delay in to keep the program from running too fast, place text on the screen, and check to see if specific keyboard keys are pressed.

# Getting Started

Install the Greenfoot IDE: <https://www.greenfoot.org/download>

* Greenfoot went through many changes beginning with the 3.0.0 release. Version 2.4.2, although older, tends to be one of the most stable versions but is missing a few of the newer additions.

In your Robot Outfit Matcher folder run the project file with the Greenfoot icon to begin.

# Mini-Goal for Students

Throughout this lab there will be many numbers used to keep track various values. Which row number represents the masks versus the robes versus the dresses, how many of each type of clothing are there, etcetera. As you are writing this lab try not to use these numbers directly into your code, instead make variables to represent the numbers. These types of numbers are often referred to as ‘magic numbers’ and should be avoided in actual code. Instead of writing the numbers directly into your code it is better practice to make static final fields to represent the number by giving the number a name. The word ‘final’ prevents the number from changing after compilation. The word ‘static’ means only one copy of the number will be stored in memory, instead of each individual object storing a separate copy of the number. The practice of making almost every field private is a good one, but static final variables are a bit more open ended on when they should be private, public, or protected. The more experience you get with large scale projects in groups the more you will see the variety of access types used on these types of variables.

Example: private static final int BEES\_PER\_HIVE = 10;

As a mini-goal, try to avoid using as many numbers as possible. Keeping numbers like 0 or 1 for counting purposes is usually okay, or a / 2 for cutting a value in half is also okay, but having the number 450 in your code instead of using a static final variable should be avoided.

# Demonstration

View a sample demonstration of what the Robot Outfit Matcher lab could look like here:

<https://www.greenfoot.org/scenarios/23663>

# Instructions

The instructions this time will be somewhat vague to allow the student to figure out how to approach the situations themselves. Each class will be identified with its main goals and students are asked to figure out how to make those goals come to fruition.

### The World (Wardrobe):

The World is responsible for the following tasks:

* Building and store the initial two robots, one a generic Robot and the other a RobotPlayer. The generic Robot will need to be told to start with a Random outfit.
* The World can keep track of the score, although a separate class could also do this. Each act the World should determine if the two robots have matched outfits, and if so, should update the score and tell the generic robot to randomize again. A small sound effect could be used to represent a matching set.
* The World can keep track of the timer left in the game and update the timer. Due to the speed of Greenfoot’s running acts, it falls onto the programmer to determine the best way to make the time go down so it is not moving so quickly that the numbers cannot be read. When the timer ends, the program should stop, an end game sound should be heard, and the final score should be shown.

### The Robot:

The robots, as currently designed, will wear a costume consisting of two pieces: a mask and a torso item. There are two different types of torso items, a robe or a dress, and neither can be worn simultaneously as the other.

Each Robot should keep a 2D Array of CostumePieces. When a Robot is constructor, this 2D array needs to be built with the correct set of Costume pieces, each of the same type in the same row. For instance, if Row 0 represents the masks, then all masks should be placed in row 0. The current images for the lab include 3 different types and 8 individual items of each type.

The following accessor methods (or similar) should be available for all robots:

int getTorsoNumber(), int getHeadNumber(), booelean isRobed, boolean equals(Robot)

The following mutator methods (or similar) should be available for all robots:

void setTorsoNumber(int), void setHeadNumber(int), void setRobed( boolean ), removeAll(), randomize(), wearOutfit().

The equals method should determine if the current robot and the incoming parameter robot have the same outfit or not. The removeAll method should remove all current costume pieces from the screen (it is okay to try to remove an object that is not on the screen, so it is okay to loop through the entire set of CostumePieces in the 2D array and tell the world to remove each of them even if it is not on the screen). The randomize() method will generate a random outfit the robot to currently wear, and the wearOutfit will put the current outfit, one mask and one torso item, on the screen at the same locations as the robot itself.

### The RobotPlayer:

The main responsibility of the RobotPlayer is to allow user input to control what the RobotPlayer is currently wearing.

The RobotPlayer is a Robot and thus gains all public or protected methods from Robot. It should not re-define any of the items Robot has already defined.

The RobotPlayer needs to keep track of which ‘row’ its keyboard input will currently be manipulating. When the up or down arrows are pressed, the row should update in correspondingly and should stay in bounds of the possible row numbers in a Robot’s 2D array.

When the left or right arrows are selected, the current Torso or Head number should be adjusted either positively or negatively. Make sure the numbers stay in bounds of the possible column numbers in a Robot’s 2D array. A RobotPlayer should not re-define a torso or head number.

After one of the up/down/left/right presses, the Robot should remove its old outfit and put on its new outfit.

* The RobotPlayer could also build and control a Selector and tell the Selector to adjust each time the current row is changed. This will show a visual representation of which current Costume type would be changed by using the arrow keys.

### The Selector and CustomePiece

Both of these classes have no modifications needed unless you wish to adjust how they work. When a CostumePiece is built it must be given two values: the type it is (dress, mask, or robe) and the number it will use (1-8). A Selector, when built, must be told the Y coordinates (as an array) of where it will move between when it moves. The changeLoc method will allow the Selector to move to the Y coordinate of the given index of its array.

