# P02 Dorm Draw

## Overview

This assignment involves developing a graphic application designed specifically for organizing and planning interior layouts within dorm rooms. This simple tool enables users, such as college students or dorm residents, to arrange and visualize furniture elements, enabling creative dorm room design. The key features of this first graphic application include:

- **adding** furniture elements: Users can add new dorm symbols such as beds, sofas, chairs, dressers, desks, rugs, and plants to the dorm room canvas.
- **rotating** dorm symbols to customize their orientation, ensuring a perfect fit within the dorm room layout.
- **dragging** dorm furniture symbols, allowing users to position elements with precision. Users will be able to move the dorm elements arbitrary within the dorm layout using the mouse.
- **selecting** and **deleting** undesirable dorm symbols.
- **saving** the design: save dorm room designs in a specific image format for future reference.

The Graphical User Interface (GUI) application will be written using the java processing library. Fig. 1 shows an example of a dorm room layout designed using our DormDraw application when it is done.

## Learning Objectives

The goals of this assignment include:

- Practice how to create and use objects,
- Develop the basis for creating an interactive graphical application,
- Give you experience working with callback methods to define how your program responds to mouse or key-based input.

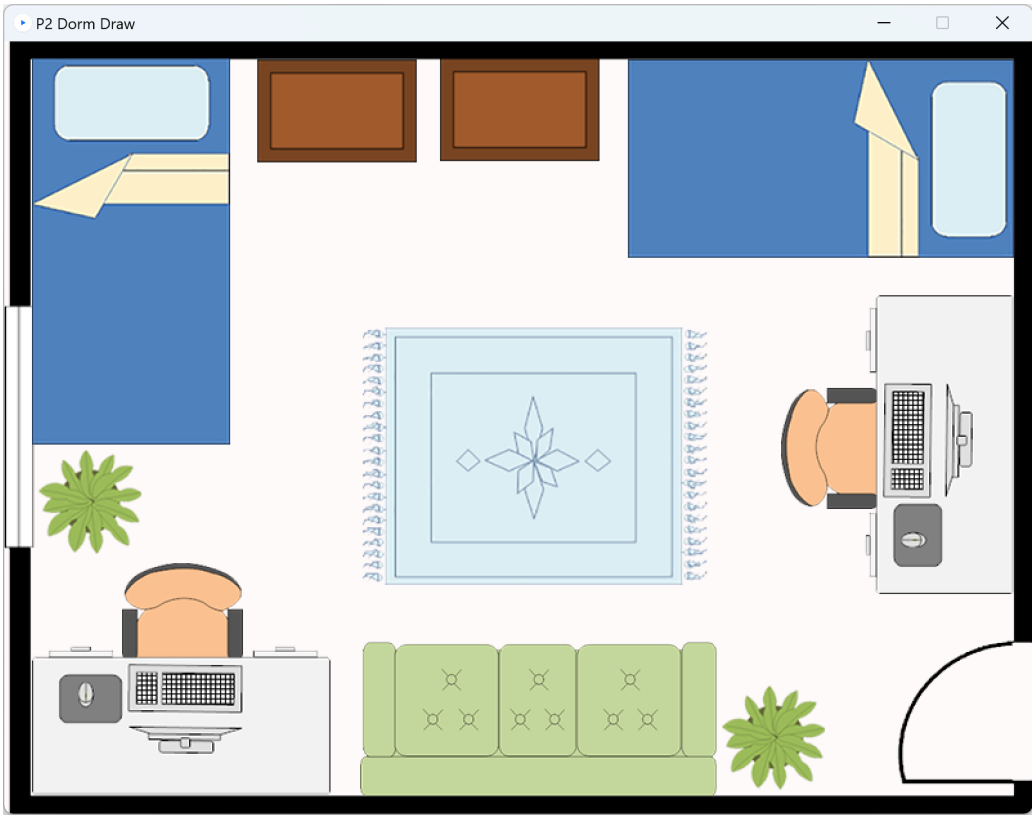



Figure 1: DormDraw Display Window

# Grading Rubric

| | |
|---|---|
| **5 points** | **Pre-Assignment Quiz:** The P02 pre-assignment quiz is accessible through Canvas before having access to this specification by **11:59PM CT on Sunday 02/04/2024**. |
| **+2.5 points** | **5% BONUS Points:** Students whose final submission to Gradescope has a timestamp earlier than **4:59PM CT on Wed 02/07/2024** and passes ALL the immediate tests will receive an additional 2.5 points toward this assignment's grade on gradescope, up to a maximum total of **50** points. |
| **25 points** | **Immediate Automated Tests:** Upon every submission of your assignment to Gradescope, you will receive immediate feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Passing all immediate automated tests does NOT guarantee full credit for the assignment. |
| **20 points** | **Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways. |
| **50 points** | **MAXIMUM Total Score** |

# Assignment Requirements and Notes

**(Please read carefully!)**

## Pair Programming and Use of External Libraries and Sources

- Pair programming is **NOT ALLOWED** for this assignment. You MUST complete and submit your p02 individually.
- Any source code provided in this specification may be included verbatim in your program without attribution.
- All other sources must be cited explicitly in your program comments, in accordance with the Appropriate Academic Conduct guidelines.
- Any use of `ChatGPT` or other large language models (LLM) must be cited AND your submission MUST include a file called log.txt containing the full transcript of your usage of the tool. Failure to cite or include your logs is considered academic misconduct and will be handled accordingly.
- You are only allowed to **import** or **use** the libraries listed below in their respective files **only**.
- **The ONLY external libraries** you may use in your submitted file are:

```
import java.io.File;
import processing.core.PImage;
```

## P02 Assignment Requirements

- This may be your first experience with developing a graphic application using the java processing library. Make sure to read carefully through the specification provided in this write-up. This assignment requires clear understanding of its instructions. Read TWICE the instructions and do not hesitate to ask for clarification on piazza if you find any ambiguity.
- The Utility class is a wrapper class that provides you access to ALL the methods related to the processing library that you might need to draw an image to the screen, set the background color, get the dimensions (width, height) of the display window, get the mouse position (mouseX, mouseY), and get a key pressed.
- **NONE** of the callback methods `(setup(), draw(), mousePressed(), mouseReleased(), and keyPressed()` defined later in this write-up should be called explicitly in this program.

- You MUST NOT add any additional fields either instance or static to your program, and any public methods either static or instance to your program, other than those defined in this write-up.
- DO NOT add any statement to the provided *main()* method, other than the call to `Utility.runApplication()` method.
- The array used in this assignment is a NON-compact PERFECT size array. Null references can be at any index within this array.
- Do NOT hard code the length of the array in any loop. The length of an array is given by the arrayReference**.length**
- All the methods in this programming assignment MUST be static.
- There is NO tester class to be implemented for this assignment. You can print some messages to the console to help you debugging some methods.

# CS300 Assignment Requirements

**This section is VALID for ALL the CS300 assignments**

- If you need assistance, please check the list of our Resources.
- You MUST NOT add any additional fields either instance or static to your program, and any public methods either static or instance to your program, other than those defined in this write-up.
- You CAN define local variables (declared inside a method's body) that you may need to implement the methods defined in this program.
- You CAN define **private** methods to help implement the different public methods defined in this program, if needed.
- Your assignment submission must conform to the CS300 Course Style Guide. Please review ALL the commenting, naming, and style requirements.
  - Every submitted file MUST contain a complete file header, with accordance to the CS300 Course Style Guide.
  - All your classes MUST have a javadoc-style class header.
  - All implemented methods including the main method MUST have their own javadoc-style method headers, with accordance to the CS300 Course Style Guide.
  - Indentation should be 2 spaces and NOT 4 spaces.
  - The maximum width line should be 100.

- If starter code files to download are provided, be sure to remove the comments including the `TODO` tags from your last submission to gradescope.
- Avoid submitting code which does not compile. Make sure that ALL of your submitted files ALWAYS compile. A submission which contains compile errors won't pass any of the automated tests on gradescope.
- Run your program locally before you submit to Gradescope. If it doesn't work on your computer, it will not work on Gradescope.
- You are responsible for maintaining secure back-ups of your progress as you work. The OneDrive and GoogleDrive accounts associated with your UW NetID are often convenient and secure places to store such backups. Aspiring students may try their hands at version control.
- Be sure to submit your code (work in progress) of this assignment on Gradescope both early and often. This will 1) give you time before the deadline to fix any defects that are detected by the tests, 2) provide you with an additional backup of your work, and 3) help track your progress through the implementation of the assignment. These tests are designed to detect and provide feedback about only very specific kinds of defects. **It is your responsibility to implement additional testing to verify that the rest of your code is functioning in accordance with this write-up.**
- You can submit your **work in progress (incomplete work) multiple times** on gradescope. Your submission may include methods not implemented or with partial implementation or with a default return statement.

# 1 GETTING STARTED

To get started, let's first create a new **Java17** project within Eclipse. You can name this project whatever you like, but "p02 DormDraw" is a descriptive choice. Then, create a new class named `DormDraw` with a public static void main(String[] args) method stub. This class represents the main class in your program.

DO NOT include a package statement at the top of your `DormDraw` class (leave it in the default package). The *DormDraw.java* source file will be the only file that you submit for grading through Gradescope.

## 1.1 Download p2core.jar file and add it to your project build path

We prepared a jar file named `p2core.jar` that contains the processing library, along with a few extra object types to help you build this assignment.

**Download** the `p2core.jar` file available on the p02 assignment page on canvas and copy it into the project folder that you just created.

Then, **add** this jar file to your p02 project's Java build path with the following steps, provided for `eclipse` users. Instructions on how to add a jar file to the build path of a java project on `IntelliJ` can be found here.

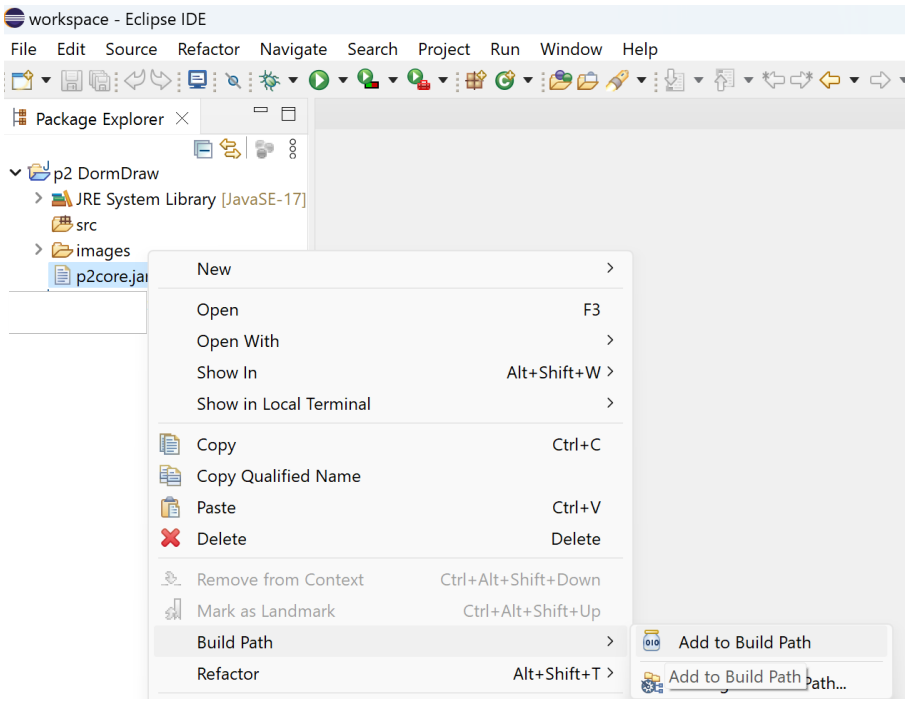
- Right-click on this file in the "Package Explorer" within Eclipse, choose "Build Path" and then "Add to Build Path" from the menu. If the .jar file is not immediately visible within Eclipse's Package Explorer, try right-clicking on your project folder and selecting "Refresh".
- **For Chrome users on MAC**, Chrome may block the the jar file and incorrectly reports it as a malicious file. To be able to copy the downloaded jar file, Go to "chrome://downloads/" and click on "Show in folder" to open the folder where your jar file is located.
- **If the "Build Path" entry is missing** when you right click on the jar file in the "Package Explorer", follow the next set of instructions to add the jar file to the build path:

  1. Right-click on the project and choose "Properties".
  2. Click on the "Java Build Path" option in the left side menu.
  3. From the Java Build Path window, click on the "Libraries" Tab.
  4. You can add the "p2core.jar" file located in your project folder by clicking "Add JARs..." from the right side menu.
  5. Click on "Apply" button.

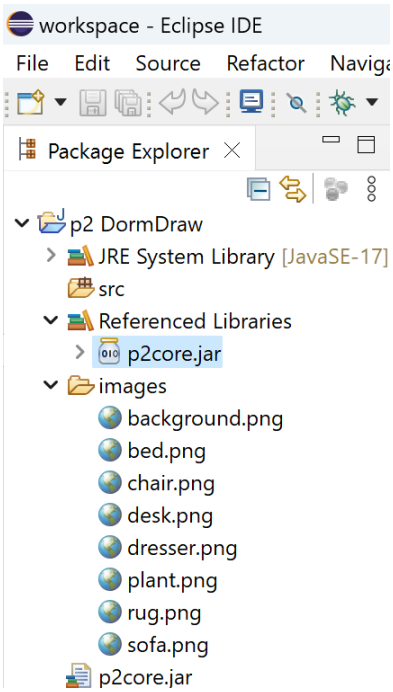
  This operation is illustrated in Fig. 2 (a).

## 1.2 Download images for your DormDraw application

- Now, download the `images.zip` file from the p02 assignment page on canvas and unzip it. It contains 8 images: the background image and the images of seven Dorm Symbols (bed.png, chair.png, desk.png, dresser.png, plant.png, rug.png, and sofa.png).
- Add the unzipped folder to your project folder in Eclipse, either by importing it or drag-and-dropping it into the Package Explorer directly. (Make SURE you unzip the file before continuing; if you try to use the zip file directly, it won't work.)
- The organization of your p02 project through eclipse's package explorer after completing this step is illustrated by Fig. 2 (b).




(a) Adding p2core.jar to build path




(b) P02 package explorer

Figure 2: Getting Started - P02 Package Explorer

## 1.3 Check your project setup

Now, to test that the p2core.jar file library is added appropriately to the build path of your project, try running your program with the following method being called from the main() method.

```
Utility.runApplication(); // starts the application
```

Note that you MUST NOT add any additional statements to your `DormDraw.main()` method. It should contain the only above statement (a call of *Utility.runApplication* method).

If everything is working correctly, you should see a blank window that appears with the title, "P2 Dorm Draw" as shown in Fig. 3. You can also notice an error message showing up in the console: `ERROR: Could not find method named setup that can take arguments [] in class DormDraw.` We are going to resolve this error in the next steps.

Please consult Piazza or one of the course staff if you have any problems with this setup before proceeding. Note that the Processing library provided within the jar file (p2core.jar) works with Java 17. If you work with another version of Java, you must switch to Java 17 for this assignment.

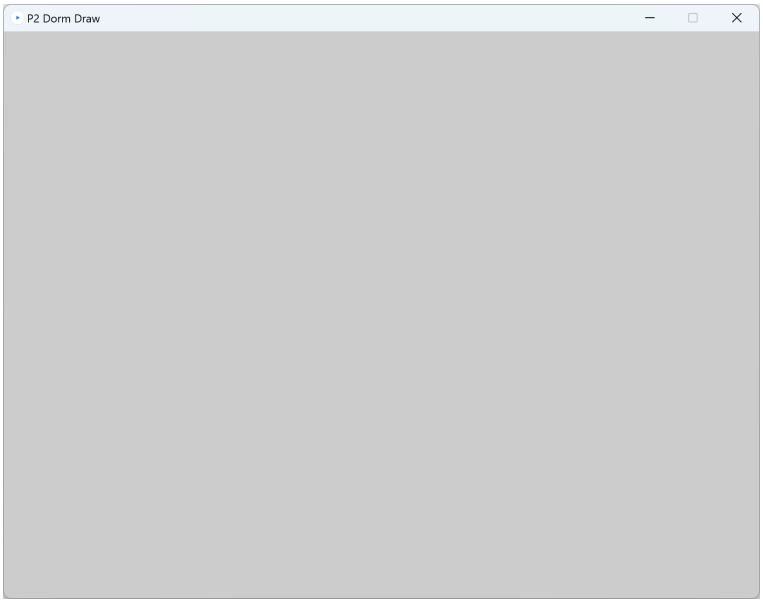



Figure 3: DormDraw - Blank Screen Window

**Note** that the runApplication() method from the provided Utility class, provided in the p2core jar file, creates the main window for the application and then repeatedly updates its appearance and checks for user input. It also handles callback methods running in the background. Callback methods specify additional computation that should happen when the program begins, the user pressed a key or a mouse button, and every time the display window is repeatedly redrawn to the screen.

# 2 Utility Framework and Overview of the class Symbol

All of the methods within the provided p2core.jar file are documented in these javadocs. Note that the `Utility.runApplication()` creates the display window, sets its dimension, and checks for callback methods.

## 2.1 Overview of callback methods

In this assignment, we are going to implement the following callback methods.

- `setup()` method: This method is automatically called by Utility.runApplication() when the program begins. It creates and initializes the different data fields defined in your program, and configures the different graphical settings of your application, such as loading the background image, setting the dimensions of the display window, etc.
- `draw()` method: This method continuously executes the lines of code contained inside its block until the program is stopped. It continuously draws the application display

window and updates its content with respect to any change or any event which affects its appearance.

- `mousePressed()`: The block of code defined in this method is automatically executed each time the mouse bottom is pressed.
- `mouseReleased()`: The block of code defined in this method is automatically executed each time the mouse bottom is released.
- `keyPressed()`: The block of code defined in this method is automatically executed each time a key is pressed.

Note that NONE of the above callback methods should be called explicitly in your program. They are automatically called by every processing program in response to specific events as described above. You can read through the documentation of the Utility class and have an idea about these callback methods and the other methods which can be used to draw images to the screen. You can also read through some of the other methods that are available through this `Utility` class.

## 2.2 Overview of the class Symbol

The Symbol class represents the data type for symbols related to Dorm item objects that will be added, moved around, and organized to build the design of the dorm room layout in our `DormDraw` application. The documentation for this class is provided these javadocs. Be sure to carefully read through the description of the constructors, and the different methods implemented in the Symbol class. You do not need to implement this class or any of its declared methods. This class is entirely provided for you in the p2core.jar file. Its implementation details are hidden for you. All the important information required to use its public methods appropriately are provided in the javadocs.

# 3 Visualizing the DormDraw Display Window

## 3.1 Declare DormDraw Static Fields

Add the following static fields (class variables) to your `DormDraw` class. Put them outside of any method including, the main(). The top of the class body is be a good placement where to declare them.

```
private static PImage backgroundImage; // PImage object that represents the
                                       // background image
private static Symbol[] symbols; // non-compact perfect size array storing
                                 // dorm symbols added to the display window
```

To avoid compile without errors, **import** the `processing.core.PImage` class at the top of your `DormDraw.java` source file.

Recall that you should not add any other variable (instance or static data field) to the `DormDraw` class. You can define and declare local variables, as needed, inside the methods that you will later define and implement in this program.

## 3.2 Define the setup and draw callback methods

At the end of the last step we saw an error related to the fact that our `DormDraw` class was missing a method called setup(). Let's solve this problem.

- Create a **public static** method with named `setup` that **takes no input arguments** and **has no return value**.
- Next, run your program. This should lead to a slightly different error message being displayed: `"ERROR: Could not find method named draw that can take arguments [] in class DormDraw."`
- We can now fix this error in a similar way, by creating a new **public static** method named `draw`, which also **takes no input** arguments and **has no return** value.
- If you look at the documentation for the Utility class, you can see how the `runApplication` method is making use of the setup and draw methods that we have just created. Recall that these methods should not be called directly from your code. They are called by the Utility class, as a result of your calling `Utility.runApplication()`.
- To convince yourself that the `setup()` method is being called once when your program starts up, try adding a print statement to the `setup()` method, then run your program. Note that the output of your System.out.println() print statement will be displayed in the console and not on the graphic display window.
- To convince yourself that the `draw()` method is being called after the setup method executes and continuously executes in an infinite loop until the program stops, try adding a print statement to the `draw()` method. Note that the text message will be repeatedly printed to the console until you terminate the program. After this test, you can remove the print statements from both the `setup` and the `draw` methods before proceeding.

## 3.3 Set the background color and draw the background image at the top of the display window

To set the background color of this application, call the `Utility.background()` from the top of the `draw()` method. We'd like to set the background color of this application to the `snow` color as follows. You can try different other colors at your choice.

```
Utility.background(Utility.color(255,250,250)); // snow color
```

To draw a background image at the top of the display window, follow the next instructions.

- First of all, make sure that you have a folder called "**images**" in your "Package Explorer" and that it contains the background.png image and seven dorm symbols. If it is not the case, go up to "1.2 Download images for your DormDraw application" subsection in the Getting Started section.
- We'd first like to initialize the `backgroundImage` variable of type `PImage` that we defined earlier. Any data field initialization should be done within the `setup() method`.
- In the `setup()` method, load the background image and store its reference into the `backgroundImage` variable of type `PImage` by calling the method `Utility.loadImage()` as follows.

```
// set the background image
backgroundImage = Utility.loadImage("images" + File.separator + "background.png");
```

- To avoid compile without errors, check that `java.io.File` and `processing.core.PImage` classes are already imported at the top of your DormDraw.java source file.
- Note that we'd like our application to be platform-independent. That's the reason why we use `File.separator` in the file path passed as input to the `Utility.loadImage()` method call. Our application can thus correctly run for any OS system (Windows, MAC OS, or Unix).
- If you run your program now, you may notice that even though the background image was loaded in the `setup()` method, it is not yet drawn at the center of the screen.
- To draw the background image to the screen, we need to call the `Utility.image()` method from the `draw()` method. Note all images or graphic objects should be drawn from the `draw()` callback method.
- The `Utility.image()` method draws a PImage object at a given position of the screen. Please read the javadocs of the Utility class for more details. Below is an example of code that draws an image at the center of the display window.

```
// Draw the background image at the center of the screen
Utility.image(backgroundImage, Utility.width()/2, Utility.height()/2);
```

  - The position `(0, 0)` refers to the upper left corner of the display window.

  - The position `(Utility.width(), Utility.height())` refers to the bottom right corner of the display window.
  - The position `(Utility.width()/2, Utility.height()/2)` refers to the center of the display window.

- Notice the importance of running the above line of code within the `draw()` method **after** calling the `Utility.background()` method, instead of before. `Utility.background()` clears the display window each time it is called as first line of code within the `draw()` method.

- Running your program now should result in a window that looks like the one shown in Fig.4.

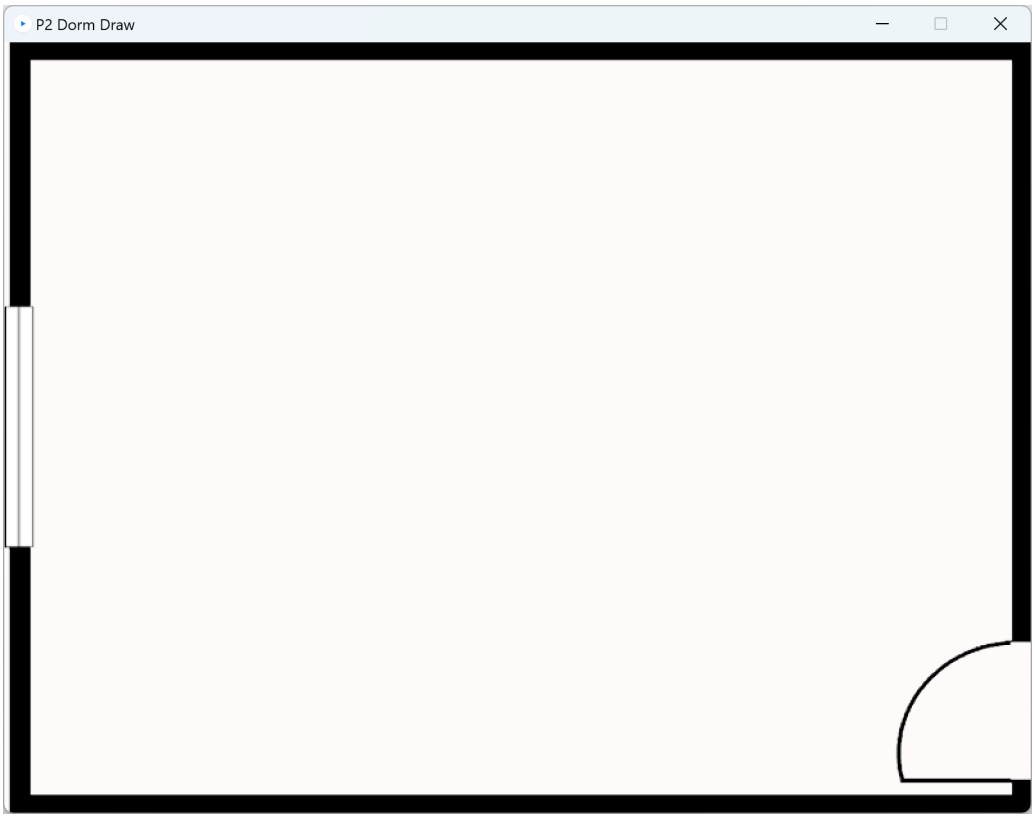



Figure 4: Background Image at the top of the screen

The processing library defines the PImage class as a datatype used for storing images. A PImage object is used to represent an image that can be loaded from a file by calling loadImage() method, and then drawn to the screen at a given position by calling image() method. Processing can display .gif, .jpg, .tga, and .png images.

# 4 Adding Dorm Symbols to the display window

Dorm Symbols will be stored in the `DormDraw.symbols` array and then displayed to the dorm layout. Notice that the array `symbols` has not been yet initialized. Its reference is `null`.

In the `setup()` method, set the `symbols` data field to a one-dimensional array that can store up to **12** Symbol references. This array must begin with 12 null references. Note that the array `symbols` **DOES NOT** refer to an oversize array. It is a non-compact **perfect size** array defined by its reference, only. Null references can be at any index position within this array.

## 4.1 Add Dorm Symbols to Specific Positions of the Screen

- Let's now create some dorm Symbols and draw them to the display window (dorm layout). For instance, create four symbols located at different x,y-positions, and set them at different indexes within the array `symbols`. Read through the details provided javadocs of the Symbol class to learn how to use its constructors and other defined methods.

- Within the `setup()` method and after initializing the `symbols` array, you can add the following lines of code.

```
symbols = new Symbol[12];
symbols[0] = new Symbol("bed.png", 200, 300);
symbols[1] = new Symbol("sofa.png", 700, 300);
symbols[4] = new Symbol("dresser.png", 100, 100);
symbols[9] = new Symbol("plant.png", 100, 500);
```

- Note the importance to add the above statements after the array *symbols* is created. Otherwise, a NullPointerException will be thrown.

- Next, to draw the created symbol objects to the dorm layout display window, you need to call the Symbol object's *draw()* method on each non null reference within the array `symbols` from the `DormDraw.draw()` method. An example of invoking the Symbol object's *draw()* method is provided in the following line of code.

```
symbols[i].draw(); // where i is a valid index within the array symbols.
```

- To do so, from within the `DormDraw.draw()` method, add a `for-loop` to traverse the array `symbols` and call the `Symbol.draw()` method of each of the `non-null` references. Be sure to add this `for-loop` below the statement drawing the background image.

- Running your program at this step will result in a window that looks like the one illustrated by Fig.5.


Figure 5: Some Dorm Symbols Added to the Screen

## 4.2 Adding Dorm Symbols at mouse position by pressing specific keys

Now, remove the lines of code creating new Symbol objects from the `setup()` method. The `symbols` array should be empty (contains only null references) in the `setup` method.

Instead of adding the Dorm Symbols manually in the `setup()` method, we'd like to allow the user to add new Symbols to the Dorm layout each time a specific `key` is pressed. Such behavior should be implemented in the `keyPressed()` callback method.

To help implement the `keyPressed` behavior related to adding new dorm symbols to the dorm layout, you should first implement the `addSymbol()` method. This method must have the exact following signature.

```
public static void addSymbol(Symbol[] symbols, Symbol toAdd) {}
// Adds a new Symbol (toAdd) to the perfect size array symbols.
// The toAdd Symbol must be added to the first null position in the array.
// If the array is full, the method does nothing.
```

Then, add a new method called `keyPressed` to your `DormDraw` class with the exact following signature. The detailed expected behavior of this method is described in the javadocs of the DormDraw class.

```
public static void keyPressed() {}
```

Note that at this step, we'd like to only implement the behavior of adding a new dorm symbol **at the mouse position** on the display window when a specific key is pressed, as follows.

| **Key-Pressed** | **Action** |
|---|---|
| b-key: 'b' or 'B' | adds a new bed at the mouse position if the `symbols` array is NOT full |
| c-key: 'c' or 'C' | adds a new chair at the mouse position if the `symbols` array is NOT full |
| d-key: 'd' or 'D' | adds a new dresser at the mouse position if the `symbols` array is NOT full |
| k-key: 'k' or 'K' | adds a new desk at the mouse position if the `symbols` array is NOT full |
| f-key: 'f' or 'F' | adds a new sofa at the mouse position if the `symbols` array is NOT full |
| g-key: 'g' or 'G' | adds a new rug at the mouse position if the `symbols` array is NOT full |
| p-key; 'p' or 'P' | adds a new plant at the mouse position if the `symbols` array is NOT full |

- You can access the mouse position by calling **Utility.mouseX()** and **Utility.mouseY()**. These methods return the current horizontal and vertical position coordinates of the mouse, respectively.
- The `Utility.key()` method returns the char value of the most recent key on the keyboard that was pressed.
- We recommend using a `switch` statement to implement the `keyPressed()` method behavior. This zybook's section provides a review of the syntax of the `switch` statement if you are not familiar with Java.

# 5 Dragging Dorm Symbols

## 5.1 Is the mouse over a Dorm Symbol?

- We'd like to allow the user to move a dorm symbol across the dorm layout when it is clicked and being dragged. This behavior allows for interactive repositioning of dorm symbols, providing a dynamic and user-friendly experience. We'll need first to implement the `isMouseOver()` with the exact following signature.

```
public static boolean isMouseOver(Symbol symbol) {}
// This method returns true if the method is over the input symbol
```

- We can check if the mouse is over an image by calculating where the each side of the symbol is and seeing if the mouse's X position is between the left and right sides of the symbol and if the mouse's Y position is between the symbol's top and bottom.
- Read carefully through the methods defined in the javadocs of the class Symbol to determine which methods return the `x`, `y` positions, and the `width`, `height` of a `Symbol` object.

- Recall that the display window is oriented such that the `position (0,0)` is at the `upper-left` corner of the screen, and the `position (Utility.width(), Utility.height())` is at the `bottom-right` corner of the display window. The positive `x-axis` goes towards `the right` and the `positive y-axis goes down`.
- Fig. 6 shows the dimensions of a `Symbol` object within the display window. The mouse shown in this figure is NOT over the symbol.

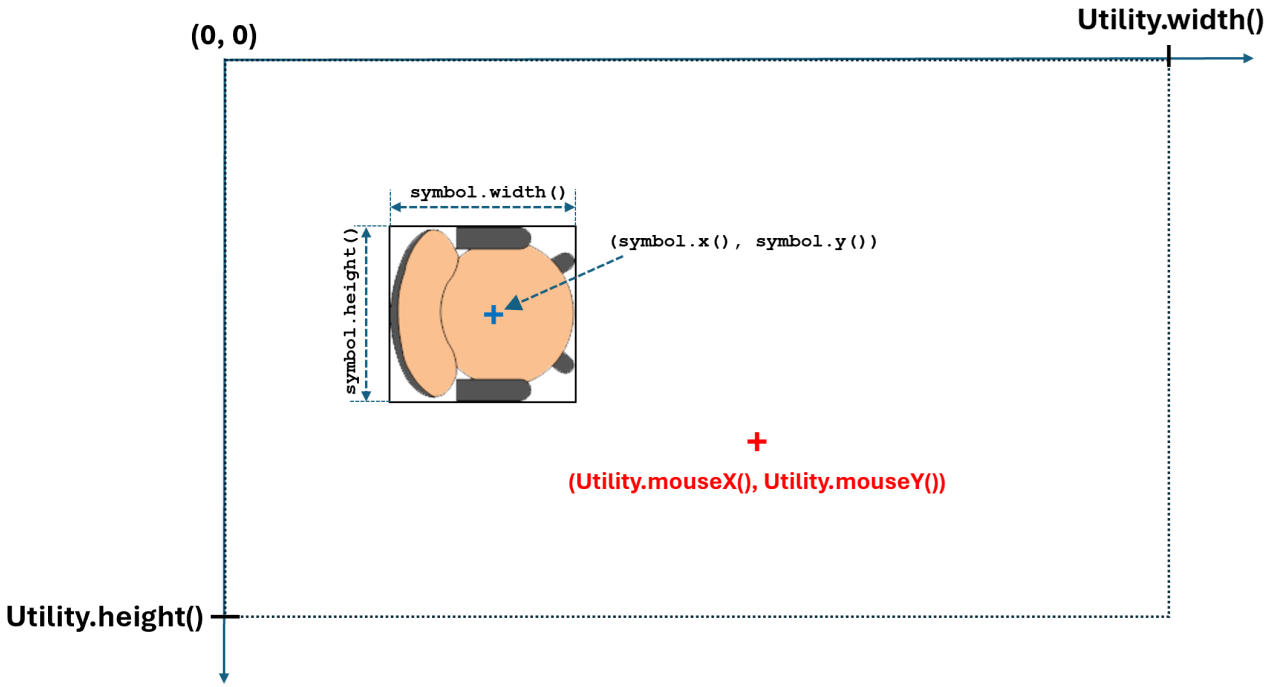



Figure 6: Dorm Symbol Dimensions Within Display Window

## 5.2 Move a dorm Symbol when it is being dragged

We'd like to move a Dorm Symbol following the mouse moves when it is being dragged. This allows the user to better organize the layout design of their dorm.
In the `DormDraw` class, define the `mousePressed()` and `mouseReleased()` callback methods with the exact following signatures.

```
/**
 * Callback method called each time the user presses the mouse
 */
public static void mousePressed() {}
```

```
/**
 * Callback method called each time the mouse is released
 */
public static void mouseReleased() {}
```

Recall that the Utility class will automatically call the `mousePressed()` method once every time the mouse button is pressed down. The `mouseReleased()` method will be automatically called once when the mouse button is later released.

- Now, implement the `mousePressed()` method. This method should check if the mouse is over one of the Symbol objects stored in the `symbols` array and start dragging it. Use a `loop` to traverse the `symbols` array. If the mouse is over a non-null Symbol reference, call `startDragging()` method on that Symbol (see the javadocs of the Symbol class for details). Only the Symbol stored at the lowest index within the `symbols` array will be dragged. This means that you should break the loop and do not check for next Symbol objects in the array, as soon as you found a Symbol being clicked on by the mouse.
- When dragging a Symbol object, the other symbols have to remain visible in the application display window.
- Try checking the correctness of the `isMouseOver()` method here by printing out a message to the console each time a Symbol is clicked. Try checking this behavior considering only one Symbol object in the screen. Try clicking on different positions of the screen.
- Try checking the correctness of the `mousePressed()` method by trying dragging symbols around the dorm design layout. Do not add many symbols. Checking this behavior given three symbol objects will be enough.
- Next, implement the `mouseReleased()` method. No Symbol object must be dragged when the mouse is released. Within the `mouseReleased()` method, call the `stopDragging()` method on every Symbol stored in the `symbols` array. Again, you need to skip the null references while traversing the array to avoid NullPointerExceptions.
- With all of this in place, the user should be able to drag and drop up to `symbols.length` Symbol objects within the application area. If a Symbol is being dragged, the draw() method defined in the Symbol class sets the position of that Symbol to the position of the mouse and draws the Symbol's image to the screen window accordingly.

Recall that `draw()` method will redraw the display window each time an event registered occurs, *such as the mouse is pressed down or released or dragged*. If the mouse is over more than one Symbol object, the one at the lowest index within the `symbols` array will be dragged.

Since only the Symbol stored at the lowest index within the `symbols` array will be dragged, a rather counterintuitive behavior will happen: Suppose that the `symbols` array is `[Bed, Sofa, null, null, ...]`, and currently the sofa (`symbols[1]`) is being dragged. Now, if the mouse moves into the bed, the dragged object should be transferred to the bed (`symbols[0]`) due to its smaller index. It's not as intuitive, but it's the expected behavior! We wanted the `mousePressed()` method to be implemented as simple as possible.

# 6 KeyPressed: More actions

## 6.1 Rotate a dorm Symbol

You may have noticed that all of our symbols have a fixed orientation. We're about to fix that! The following behavior must be implemented in the `keyPressed()` method.If the `r-key` ('r' or 'R') is pressed *and* the **mouse is over** a Symbol, then we need to call `Symbol.rotate()`. We only need to call `rotate()` on the first Symbol in the symbols array being clicked. Writing a private helper method for this is highly encouraged.

## 6.2 Removing Symbols by pressing the Backspace-key

Now we'll implement deleting symbols (after all, we might decide we don't actually want 10 desks and a bed in our dorm). We will do this similar to how we rotated a symbol, but instead of calling `rotate()`, we will remove the Symbol from symbols. Note that we can check for the `Backspace` character by using `Utility.BACKSPACE` the same way we used 'b', 'c', etc. This method is similar to rotate in that it only applies to the first symbol in the symbols array (the one with the smallest index) if the mouse is over it. **Deleting** an element from a non-compact perfect size array is as simple as setting its reference to **null**!

## 6.3 Save a screenshot of the Dorm design pressing S-key

Finally, we'd like to allow the user to save the layout design of the dorm in an image file named **dormDraw.png** each time the `s-key` is pressed. To do so, update the `keyPressed()` method to call Utility.`save()` method each time 's' or 'S' key is pressed. The *dormDraw.png* file should be saved in the current directory. Do NOT add any specific path.You should only provide the filename as input to the `Utility.save()` method call. Try refreshing the view on your package explorer to show the `dormDraw.png` saved file.

# 7 Assignment Submission

**Congratulations on finishing this CS300 assignment!** After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through Gradescope. The only ONE file that you must submit is `DormDraw.java`. Your score for this assignment will be based on your **"active"** submission made prior to the assignment due date of Due: **9:59PM CT on February 8**$^{th}$. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline.