P03 Toy Saga I

Overview

Not to be confused with *Toy Story*, P03 Toy Saga I involves developing a graphics application that will allow you to place and move various pieces of furniture and toy objects within a room. This assignment is similar to P02 Dorm Draw, but now you will develop instantiable classes for on-screen interactive objects yourself. Some of the features of this application are:

- **adding** furniture and toy elements to the layout of the room
- **dragging** and **rotating** toy elements using the mouse and keyboard
- **selecting** and **deleting** toy elements by using the mouse and keyboard, or by dragging it over a special toy box furniture element

Learning Objectives

The goals of this assignment include:

- Learning how to create and use instantiable classes
- Practicing using the `java.util.ArrayList` class
- Improving your skills with the Processing graphics library
- Writing tester methods for your own classes

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P03 pre-assignment quiz is accessible through Canvas before having access to this specification by **11:59PM CT on Sunday 02/18/2024**.
+2.5 points	**5% BONUS Points:** Students whose final submission to Gradescope has a timestamp earlier than **4:59PM CT on Wed 02/21/2024** 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.
20 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.
15 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.
10 points	**Manual Grading:** Human graders will review the commenting, style, and organization of your final submission. They will be checking whether it conforms to the requirements of the CS300 Course Style Guide. You will NOT be able to resubmit corrections for extra points, and should therefore carefully review the readability of your code with respect to the course style guide.
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 P03 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.util.ArrayList;
import java.io.File;
import processing.core.PImage;
```

P03 Assignment Requirements

- Identically to P02, 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.

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, 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

1.1 Create the Project

1. Create a new project in Eclipse, and name it something like "P03 Toy Saga I".
   a. Ensure this project uses Java 17. Select "JavaSE-17" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do **not** create a project-specific package; use the default package.
2. Download one (1) Java source file from the assignment page on Canvas:
   – `ToySagaTester.java`
3. Create three (3) Java source files within your project's src folder:
   – `ToySaga.java`
   – `Furniture.java`
   – `Toy.java`

   Only the `ToySaga` class should contain a `main()` method.

1.2 Download JAR File

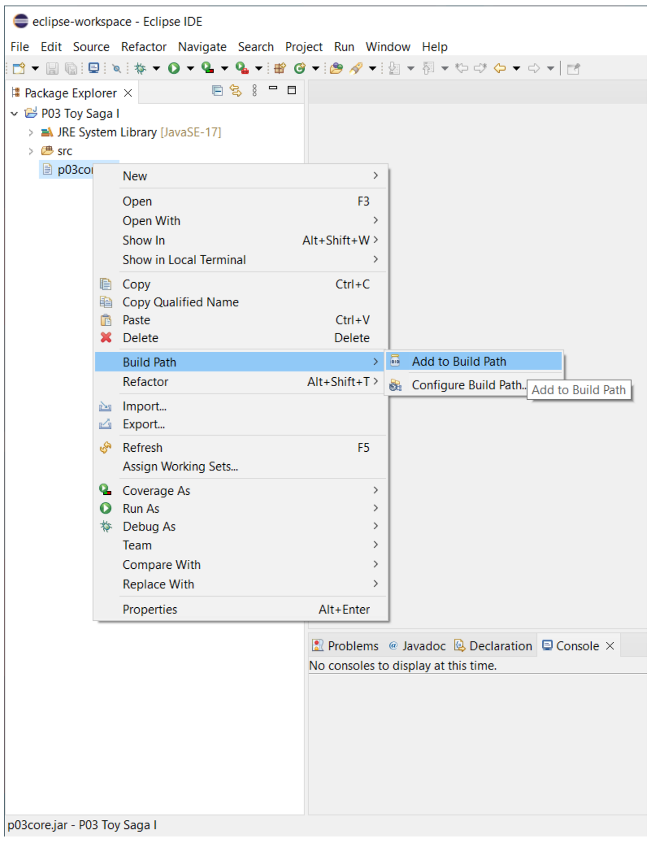
We prepared a jar file named `p03core.jar` that contains an interface with the Processing library to help you build this assignment.

1. **Download** the `p03core.jar` file available on the P03 assignment page on Canvas and copy it into the project folder that you just created.
2. Then, **add** this jar file to your 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.
   a. 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".
   b. **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.

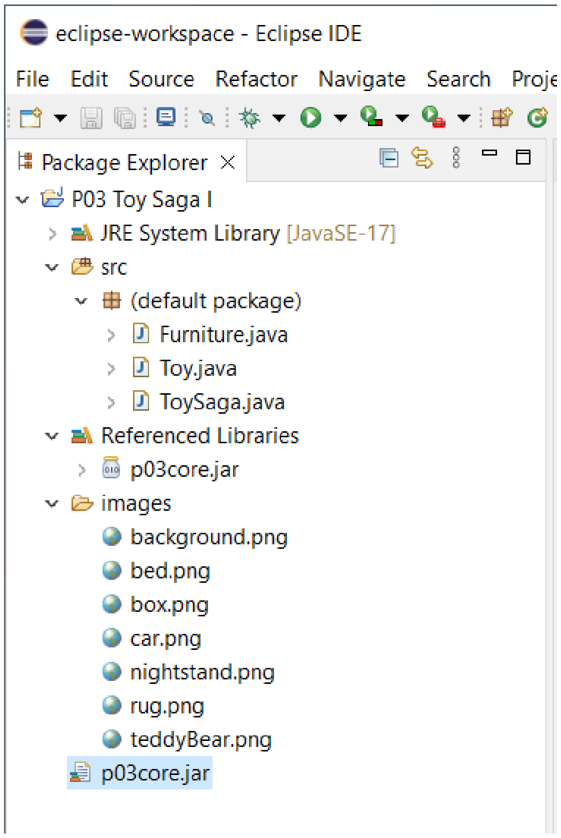
c. **If the "Build Path" entry is missing** when you right click on the jar file in the "Package Explorer", follow the next set of instructions to add the jar file to the build path:

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 `p03core.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. 1 (a).



(a) Adding `p03core.jar` to build path



(b) P03 package explorer

Figure 1: Getting Started - P03 Package Explorer

1.3 Download Image Files

1. Download the `images.zip` file from the P03 assignment page on Canvas and unzip it. It contains 8 images: the background image (`background.png`) and the images of seven furniture and toy objects (`bed.png`, `box.png`, `car.png`, `nightstand.png`, `playmat.png`, `rug.png`, and `teddyBear.png`).

2. 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.

3. The organization of your P03 project through Eclipse's package explorer after completing this step is illustrated by Fig. 1 (b).

1.4 Test Project Setup

To test that the `p03core.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 `ToySaga.main()` method.

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

Note that you MUST NOT add any additional statements to your `ToySaga.main()` method. It should contain the only above statement.

If everything is working correctly, you should see a blank window that appears with the title, "P03 Toy Saga I" as shown in Fig. 2. You should also notice the following error message showing up in the console: `ERROR: Could not find method named setup that can take arguments [] in class ToySaga.` We will resolve this error in the next steps.

If you have any problems with this setup, please consult Piazza or one of the course staff before proceeding. Note that the Processing library provided within the `p03core.jar` file only works with Java 17. If you work with another version of Java, you must switch to Java 17 for this assignment.

Note: The `runApplication()` method from the `Utility` class, provided in `p03core.jar`, 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.

1.5 Overview of Callback Methods

All of the methods within the provided `p03core.jar` file are documented in the `Utility` class JavaDocs. Note that the `Utility.runApplication()` creates the display window, sets its

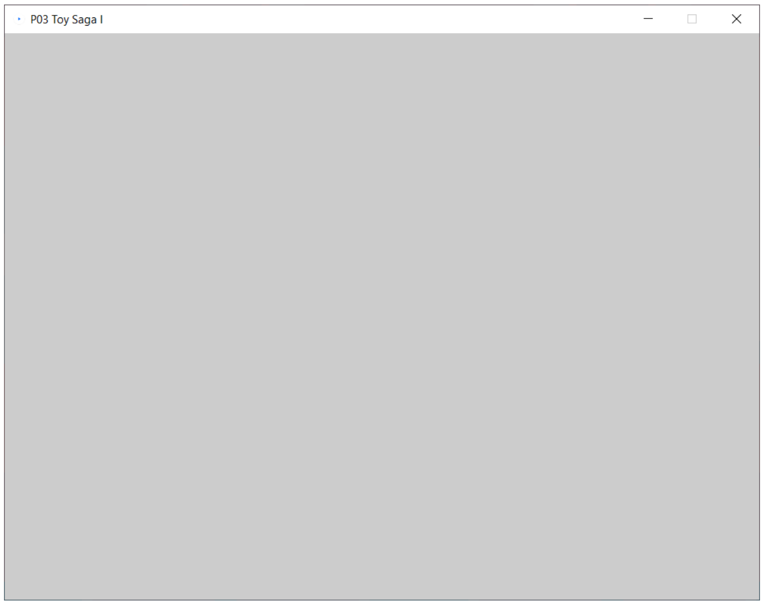


Figure 2: ToySaga - Blank Screen Window

dimension, and checks for 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.

2 Furniture Class

There are two instantiable classes, `Furniture` and `Toy`, that you will create in this assignment to represent interactive objects which can be drawn to the screen. **NO JavaDocs** will be provided for these classes; you should read through the following description of the two classes and implement them to best match the required functionality.
`Furniture` objects represent on-screen items which the user cannot interact with, and `Toy` objects represent on-screen items which the user can interact with by adding, deleting, moving, or rotating them.

2.1 Furniture Class Fields

The `Furniture` class should have four (4) **instance** fields:

1. A **public** constant (**final**) field named `IMAGE` that will store a `PImage` which will be drawn to represent this `Furniture` item.
2. A **private** field named `name` that stores a `String` value, and holds the name of this `Furniture` item.
3. A **private** field named `x` that stores an `int` representing the x-position of this `Furniture` object in the display window.
4. A **private** field named `y` that stores an `int` representing the y-position of this `Furniture` object in the display window.

2.2 Furniture Class Constructors

The `Furniture` class should have two (2) **public** constructors:

1. A constructor which takes as argument a `String name`, an `int x`, and an `int y`, in that order, and sets the corresponding instance variables.
   The `IMAGE` instance field should be initialized by loading the `PImage` located at the path `"images" + File.separator + name + ".png"` using the `Utility.loadImage()` method.
2. A constructor which takes as argument a `String name`. This constructor should have the same functionality as the first constructor, but the position of the object should be initialized to the center of the screen (use `Utility.width()` and `Utility.height()` to find the center).

2.3 Furniture Class Methods

The `Furniture` class should have four (4) **public instance** methods:

1. A *getter/accessor* method for the `x` field named `getX()`.
2. A *getter/accessor* method for the `y` field named `getY()`.
3. A *getter/accessor* method for the `name` field named `name()`.
4. A method named `draw()` which calls `Utility.image()` to draw the `Furniture` object at its current position. This method has no return value.

3 Toy Class

3.1 Toy Class Fields

The `Toy` class should have five (5) **instance** fields:

1. A **public** constant (**final**) field named `IMAGE` that will store a `PImage` which will be drawn to represent this `Toy` item.
2. A **private** field named `x` that stores an `int` representing the x-position of this `Toy` object in the display window.
3. A **private** field named `y` that stores an `int` representing the y-position of this `Toy` object in the display window.
4. A **private** field named `isDragging` that stores a `boolean` indicating whether this object is being dragged or not.
5. A **private** field named `rotations` that stores an `int` counting the number of times this object has been rotated 90° clockwise.

3.2 Toy Class Constructors

The `Toy` class should have two (2) **public** constructors:

1. A constructor which takes as argument a `String name`, an `int x`, and an `int y`, in that order, and sets the corresponding instance variables.
   The `isDragging` and `rotations` fields should be initialized to `false` and `0`, respectively.
   The `IMAGE` instance field should be set by loading the `PImage` located at the path `"images" + File.separator + name + ".png"` using the `Utility.loadImage()` method.
2. A constructor which takes as argument a `String name`. This constructor should have the same functionality as the first constructor, but the position of the object should be initialized to the center of the screen.

3.3 Toy Class Methods

The `Toy` class should have thirteen (13) **public instance** methods:

1-2. Getters for the `x` and `y` fields named `getX()` and `getY()`.

3-4. Setters for the `x` and `y` fields named `setX()` and `setY()`.

5. A getter named `getRotationsCount()` for the `rotations` field.

6. A getter named `isDragging()` for the `isDragging` field.

7-8. Two setters, `startDragging()` and `stopDragging()`, for the `isDragging` field. These setters should set `isDragging` to `true` and `false`, respectively.

9. A method `rotate()` which increments the `rotations` field when called.

10. A method `move()` which takes as arguments an `int dx` and an `int dy`, and adds these values to the `x` and `y` fields, respectively.
    If the updated `x` or `y` value is outside the bounds of the window, it should be reset to the closest value which is inside the window. For instance, if `x` is `-5` after adding `dx`, it should be reset to `0`. Use `Utility.width()` and `Utility.height()` to access the width and height of the display window.

11. A method named `draw()`, which first updates the position of the `Toy` using `move()` if it is dragging, and then draws the `Toy` at the updated position. If the `Toy` is dragging, you should move it by the **difference** between the current and previous positions (current - previous) of the mouse. These positions can be accessed using the following methods:

    – `Utility.mouseX()` (current mouse x-position),

    – `Utility.pmouseX()` (previous mouse x-position),

    – `Utility.mouseY()` (current mouse y-position), and

    – `Utility.pmouseY()` (previous mouse y-position).

    Once the position has been updated, you can draw the `Toy` by calling the following provided private helper method `drawToyImage()`.

```java
/**
 * Helper method to draw an image accounting for any rotations to the screen.
 * The implementation of this method is fully provided in the write-up.
 */
private void drawToyImage() {
  Utility.pushMatrix();
  Utility.translate(x, y);
  Utility.rotate(this.rotations * Utility.PI / 2);
  Utility.image(IMAGE, 0.0f, 0.0f);
  Utility.popMatrix();
}
```

12. A method named `isOver()` which has parameters `int x` and `int y`, and returns a `boolean` indicating whether the position (x,y) is within the rectangle of the `PImage` representing this `Toy`, including the boundary. You should have a similar method in P02, which you may re-use here.

    Important: Make sure to account for the fact that the `Toy` can now rotate, which can change the width and height of the image. Use the `rotations` field to determine the width and height of the rotated image. If there is an **even** number of rotations, the width and height of the rotated image are the **same** as the width and height of the original image. If there is an **odd** number of rotations, the width and height of the rotated image are **swapped** from the width and height of the original image.

13. An *overloaded* method named `isOver()` which has a single parameter `Furniture other`, and returns a `boolean` indicating whether the image of this `Toy` overlaps with the image of the given `Furniture` object. That is, it should return true if any point within the `Toy` image is within the `Furniture` image, including the boundary.

    Feel free to reference the P03 Pre-Assignment Quiz or this GeeksForGeeks article if you need help with the logic (remember to cite the article properly if you do so). Note that the article has (0,0) centered at the bottom right, whereas our graphical application has (0,0) centered at the top left. Be sure to adjust the visual diagram accordingly when calculating the points.

    Important: Remember to account for possible rotation in the `Toy` object when calculating its width and height, as in the previous `isOver()` method.

4 Tester Class

Make sure you have the `ToySagaTester.java` file downloaded from Canvas and added to your `src` folder. Using the above descriptions of the methods in the `Toy` and `Furniture` classes, and the comments in the tester methods of `ToySagaTester`, complete each of the incomplete tester methods which have a `TODO` listed as an inline comment.

5 ToySaga Class

Now it's time to actually create the GUI using the `Furniture` and `Toy` classes to represent our on-screen objects. Add the following fields and methds to the `ToySaga` class:

5.1 Static Fields

1. Create a `private static PImage` field named `backgroundImage`. This variable will store the application's background image.

2. Create a `private static ArrayList` of `Furniture` objects named `furnitureList`, and a `private static ArrayList` of `Toy` objects named `toyList`. These two `ArrayLists` will hold references to the objects that are on the screen.
3. Create a `private static final` field named `BOX_NAME` which holds the `String` value `"box"`. This is the constant name of the toy box furniture.
4. Lastly, create a `private static final` field named `MAX_TOYS_COUNT` which holds the `int` value `8`. This is the maximum number of visible toys that can be stored in `toyList`.

5.2 `setup()` Method

1. Create a `public static void` method named `setup()`. This method is run once by the provided library when the graphics application starts.
2. Now 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 ToySaga.` This error will be fixed once we add the `draw()` method.
3. In the `setup()` method, instantiate the `backgroundImage` static field using the `Utility.loadImage()` method to load the `PImage` from the path `"images" + File.separator + "background.png"`.
4. Instantiate the `furnitureList` and `toyList` variables as new, empty `ArrayLists`.
5. Add the following four new `Furniture` items to the `furnitureList`:
   - A `Furniture` object with name `"bed"` centered at `520, 270`
   - A `Furniture` object with name `"rug"` centered at `220, 370`
   - A `Furniture` object with name `"nightstand"` centered at `325, 240`
   - A `Furniture` object with name `BOX_NAME` centered at `90, 230`

5.3 Remaining `ToySaga` Methods

1. Now you will complete the remaining six (6) static methods in the `ToySaga` class: `draw()`, `getToyBox()`, `getDraggingToy()`, `mousePressed()`, `mouseReleased()`, and `keyPressed()`. The description of each of these methods is provided in the JavaDocs.
2. After completing these methods, you should see the background image and furniture items appear, and be able to add toys using the 'C' and 'T' keys, rotate toys using the 'R' key, and delete toys by dragging them over the toy box furniture object labeled "Toys" on the left-hand side. A picture of the application after adding some toys is shown in Fig. 3, and a demo video is linked on the Canvas assignment page under **Video Demonstration**.

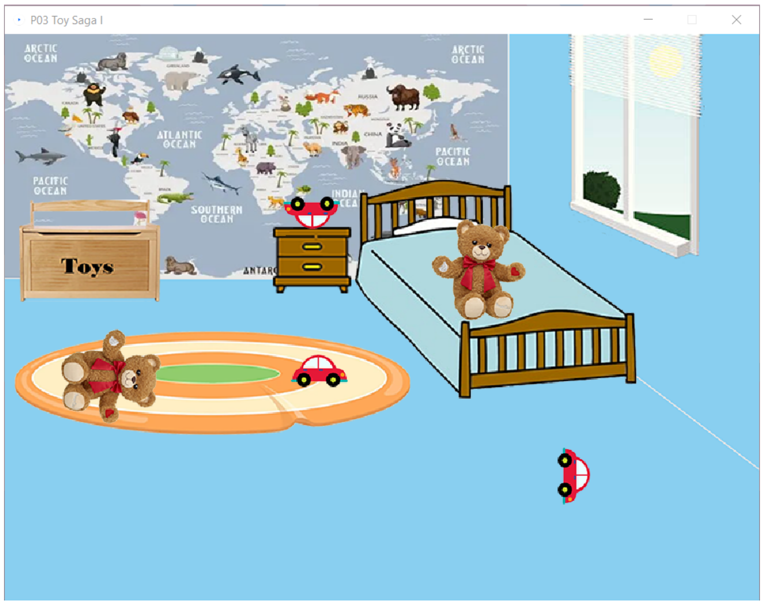


Figure 3: ToySaga - Finished Product

6 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 four (4) files that you must submit are

- `Furniture.java`
- `Toy.java`
- `ToySaga.java`
- `ToySagaTester.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 22nd**. 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.