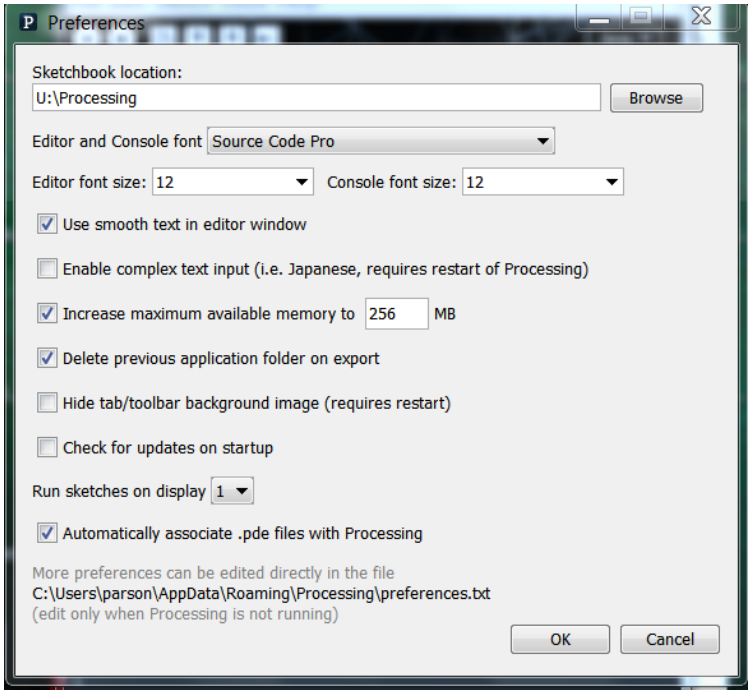
CSC 220 Object Oriented Multimedia Programming, Fall 2020

Dr. Dale E. Parson, Assignment 1, 2D automated avatars in 2D spaces in Processing.
This assignment is due via **D2L Assignments** **Assignment 1 CSC220F20AvatarClassInAvatarRoom** by **11:59 PM on 24 September**.

I recommend using your home machine as the Preferences -> Sketchbook location, or a USB thumb drive if you are moving from machine to machine. (FROM PRE-COVID SEMESTERS ONLY: When using Processing on the Kutztown campus Windows computers, make sure to start out **every time** by setting your Processing Preferences -> Sketchbook Location to U:\Processing. The U:\ drive is a networked drive that will save your work and make it accessible across campus. If you save it to your desktop or the lab PC you are using, you will lose your work when you log out. You must save it to the U:\ drive. If you do not have a folder[1] called Processing under U:\, you must create one using the Windows Explorer. Processing Preferences is under the File menu on Windows.) Alternatively, you can use a USB thumb drive with a folder for your sketchpad location, and use that on Windows, Mac, or Linux. Windows computer labs on the KU campus such as Rohrbach Library should allow you to run Processing from the networked S:\ComputerScience\processing subdirectory.



If you will be downloading Processing 3.X[2] and running it using an off-campus computer (do not use version 2.X for semester), you can copy your project sketch named **CSC220F20AvatarClassInAvatarRoom** to a flash drive on one machine, and then copy it from the flash drive to another Processing sketch folder.

You can copy and paste the text for my handout sketch into a new sketch, then save it, as a starting point. Create your **CSC220F20AvatarClassInAvatarRoom** folder by running File -> Save As ->

[1] Another name for a *folder* is a *directory*.
[2] https://processing.org/download/

CSC220F20AvatarClassInAvatarRoom after setting up your sketch folder.

The handout code is here:
https://faculty.kutztown.edu/parson/fall2020/CSC220F20AvatarClassInAvatarRoom.txt (handout code)
HOWEVER, to earn 100% you must complete all redesigns specified below. It must not be a small change to mine; it must be your own visual design. Nevertheless, starting with mine is necessary because substantial parts of the assignment remain intact, including the interface Avatar, the abstract class CollisionDetector (formerly ~~AvatarHelper~~), the setup() function, draw(), overlap(), and the keyPressed() function. The entire assignment including the Unified Modeling Language (UML) class diagram is here:
http://faculty.kutztown.edu/parson/fall2020/CSC220Fall2020Assn1.html (assignment web page)

NOTE: If you have only 1 display monitor and want to have the Processing editor and the graphical window available at the same time, do this:

Up in setup(), change this:

```
void setup() {
  // setup() runs once when the sketch starts, initializes sketch state.
  // size(1000, 800, P2D);
  fullScreen(P2D);
```

to this

```
void setup() {
  // setup() runs once when the sketch starts, initializes sketch state.
  size(1000, 800, P2D);
 // fullScreen(P2D);
```

and adjust the pixels in size() to work for your setup. Feel free to turn it in that way.

REQUIREMENTS:

1. Write another direct subclass of abstract class CollisionDetector (formerly ~~AvatarHelper~~) that implements your custom avatar. You can use Processing shape-drawing primitives arc/ellipse/line/point/quad/rect/triangle[3], or a .svg vector graphics file, or include at most one .jpg or .png image file in your class. You must use at least one Processing shape-drawing primitive. Your avatar class must be mobile and support creation of multiple objects. You may copy/paste/rename my Professor class as your starting point, and you may keep and construct a small number of mobile Professor(s), or you may delete class Professor if you do not want to keep it. Name your Avatar-derived class whatever you want. Tell me its name in a comment at the top of the file.
2. Your class' display() method must use **push()** and **translate()** as the first two instructions. Make sure to have a well-defined 0,0 reference point within the body of your class objects. Denote the well-defined 0,0 reference point with a comment in the code. Also, **pop()** must be the final instruction in your display function.
3. Your move() method **must** change one or more of **scaleX**, **scaleY**, **shearX**, or **shearY**[4]; you may

[3] https://processing.org/reference/

[4] Note 9/12/2019: rotateBB will not work correctly in some cases of shearX or shearY. It must work in the absence of shearX or shearY (I'll turn them off while testing your sketch), but you are free to use them if you want.

have to add class data fields for this requirement. You must perform **rotation**. You may change color or other custom properties of your avatar. Make sure to call detectCollisions() as the final step of move().

4. Redefine method **getBoundingBox**() in your class to accurately depict the bounding box for your object. Use my **rotateBB**(…) helper function to compensate for rotation; one of my supplied Avatar-derived classes provides an example. Test this using the 'b' key command to show the bounding box. The box must closely encompass your avatar.
   NOTE: You can copy functions rotatePoint() and rotateBB() from this example code that we went over 9/8:
   https://faculty.kutztown.edu/parson/fall2019/CSC220F19DemoG_ClassInterfaceInheritance.txt
   You do not need to understand how I implemented them, but you do need to use them. Look at function getBoundingBox() inside class Avatar2 in that example sketch. It simplifies to this:

```
}
int [] getBoundingBox() {
  int [] result = new int[4];
  result = rotateBB(-ewidth/2, -eheight/2, ewidth/2, eheight/2,
             rotateFactorDegrees, XscaleFactor, YscaleFactor, elx, ely);
  return result ;
}
```

   where you must replace the first 4 arguments with your LEFT, TOP, RIGHT, BOTTOM extents (bounds) of the shape drawn in your display() function, relative to your 0,0 point at elx, ely. Get the graphics for your Avatar correct, then work on its bounding box.
5. Construct some number of your avatars and store them in the avatars[] array within setup(). Make sure your constructor initializes any data fields you add, and use super(…) to call its base class constructor.
6. Change the immobile Furniture class' methods to create obstacles (they work more like membranes) that use Processing's arc or ellipse or quad or triangle shapes instead of (or in addition to) my Furniture's use of rect. Make sure that getBoundingBox() is correct for your class. Place some of your immobile Furniture objects in your scene.
7. Create a mobile peer class to Paddle that moves in a periodic manner using Processing shape(s). Make sure that getBoundingBox() is correct for your class. Place some of your mobile class objects in your sketch. Modifying class Paddle to meet this requirement is OK.
8. Update your sketch documentation comments at the top to include your name and any other appropriate changes. Identify the classes you have added or changed in comments at the top; put the add/change detail comments within the classes themselves. Add documentation comments to your new classes. Make especially sure to document the body parts for your avatar similar to comments in Professor.display(). Make sure to document your avatar's internal 0,0 reference point within display().
9. Test thoroughly. Each substantial bug results in a 10% deduction. Get with me during lab time or office hours if you get stuck.
10. Do not change any code other than that specified here without checking with me first. Interface Avatar and the abstract class CollisionDetector (formerly ~~AvatarHelper)~~ **MUST** remain unchanged. Function setup() requires some new constructor calls for your classes and space in the avatars[] array. You will need a new global variable or two if you load a .svg, .jpg, or .png file (no extra credit and no requirement for using such files); load them within setup(), not within your class, in order to avoid duplicate files in memory. Functions draw(), keyPressed(), and overlap() should work correctly in the handout; do not change them without checking with me. The reason for these restrictions is that assignment 2 may use teams of 2 or 3 students to create a 3D counterpart to this sketch, and if you change the non-class framework too much, your sketch will be incompatible with others.

We will have a class period for working on this project. If you do not get it done in class, you will have to

complete it as homework. I expect it to be to me by the due date via D2L. **I will deduct 10% for each day it is late**. Also, re-read the above requirements when you turn it in, to ensure that you don't miss anything. If you make changes after turning it in, just turn in another copy of your sketch via D2L. I will look at the last one that you turn in.

TURNING IT IN: When your work is completed, and you have re-read and satisfied the project requirements, you can use the Windows Explorer to find the file **CSC220F20AvatarClassInAvatarRoom.pde** in your **U:\Processing\CSC220F20AvatarClassInAvatarRoom** folder. Drag **CSC220F20AvatarClassInAvatarRoom.pde** into the **Assignment 1 dropbox** under our course's D2L account by the due date. If you find you have created an error, you can drop an updated CSC220F20AvatarClassInAvatarRoom.pde into the dropbox. **Assignment 1** is under **Assessments -> Assignments** in our D2L account. If you are working on a laptop or your machine at home, turn it in via D2L in the same way. Finally, if your sketch loads any image or .svg vector graphics files, turn in a .zip archive of the entire CSC220F20AvatarClassInAvatarRoom/ directory instead of the individual files. The assignment web page shows how to zip of a Processing sketch. I cannot run a sketch that depends on additional files without receiving those files.