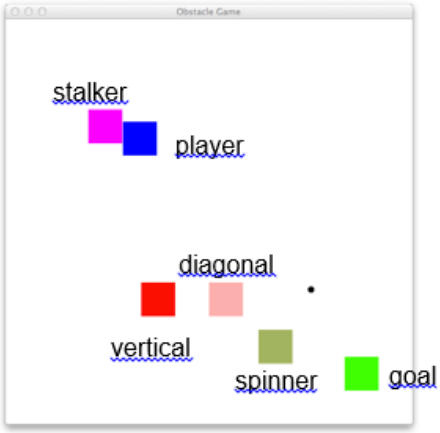
AP Computer Science Name: _____

Obstacle Game: Credit Mr. Fahrenbacher NWHS



The concept of this obstacle game is fairly simple – you (The Blue Rectangle) want to reach the goal (The Green Rectangle). As usual, there are enemies that are trying to stop you. We are going to represent the bodies of things in the game world by the built in **Rectangle** java class. Rectangles are convenient objects, but they are not **encapsulated**. That means clients of the Rectangle class have direct access to the four fields - **x**, **y**, **width**, and **height**. This is generally bad programming style. But for the purposes of our game, it will be okay.

Part 0 - Project Setup: Open double click on "package" to open BlueJ and this project. Compile the project if necessary.

Part 1 - ObstacleGame Class: Look at the ObstacleGame class. Take some time to read through the comments and the code - try to make connections between what the comments say and what code is written. Part of the code is related to the graphics aspect of the game and to the game controls. While we have only spent a little time on this see if you can make sense of how it works (it may come in handy for your 4th quarter project). Next, focus on how the Player and Enemy objects are being instantiated and called upon.

Part 2 - **Enemy Class**: Look at the Enemy class. Please answer these questions about the class:

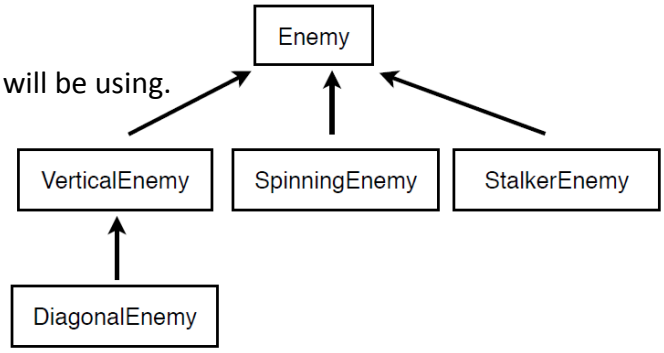
1. What does it mean for the Enemy class to be abstract?

2. What are the benefit(s) of making Enemy an abstract class as opposed to a concrete class or an interface?

3. What methods are abstract? How do you know? Why do you think those were made abstract?

4. What does the Enemy class' move() method **currently** do? Why do you think that is?

Part 3 - Enemy Hierarchy: Here is the inheritance hierarchy we will be using.



Based on this hierarchy...

5. What is the superclass of VerticalEnemy? _____
6. What is the subclass of VerticalEnemy? _____
7. How many methods will the Enemy subclasses inherit from Enemy? _____
   List them and categorize them as accessors, mutators, or something else.

8. How many instance variables (fields) will the Enemy subclasses inherit from Enemy? List it/them.

9. How many constructors will the Enemy subclasses inherit from Enemy? What are the parameters?

10. Why would someone bother to create this class hierarchy as opposed to totally separate class or just writing everything in a main method?

Part 4 - Spinning Enemy: Look at the SpinningEnemy class. I have already finished SpinningEnemy for you. Look through the code and see if you can make sense of what is happening. Use this code to help you figure out how to write the implementation for the other enemies.

11. Go back to the ObstacleGame class and uncomment the line of code that adds a SpinningEnemy object to the Enemies array in setupGame(). Hint: Look for //enemies[0] = new SpinningEnemy(450, 400, 50, 50, 100);. Play the game again and now the spinning enemy should appear. Spend some time understanding how that worked.

Part 5 - Vertical Enemy: Look at the VerticalEnemy class. Follow these steps to set up the VerticalEnemy class.

12. How do you let java know that you want VerticalEnemy to be a subclass of Enemy? **Make it so.** You will have an error, why?

13. A VerticalEnemy will be described by 3 things - a rectangle (inherited from Enemy), a **ySpeed** integer (how fast the Enemy moves), and **screenHeight** integer (how tall the screen is). Create instance variables for the two **non-inherited instance variables**. IMPORTANT NOTE: Remember that you SHOULD NOT redeclare fields that are declared in the superclass. Doing so can break things. What instance variables did you create?

14. The VerticalEnemy constructor takes 6 parameters - the first 4 represent the rectangular bounds of the object, the final 2 represent the speed and height of the screen. To finish the constructor correctly, you must do the following:

    i) Call the superclass constructor using the keyword _____ - ALWAYS do this in all of your constructors when using inheritance. You should pass some of the parameters of your constructor to the superclass constructor. Specifically, the Enemy constructor requires the _____, _____, _____, and _____ as parameters.

    ii) Initialiaze your **ySpeed** and **screenHeight** instance variables to the other two parameters.

15. You will need to override a method of the Enemy class to make the VerticalEnemy have its own look. Specifically, you need to return what color you want it to be. For example: return Color.RED;. It should still be drawn as a rectangle. What method should you override?

16. Go back to the ObstacleGame class and find the setUpGame() method. You should see a line like the following in comments:

    ```
    //enemies[1] = new VerticalEnemy(200, 300, 50, 50, gameHeight, 5);
    ```

    Uncomment the line of code and run your game - you should see your VerticalEnemy in red

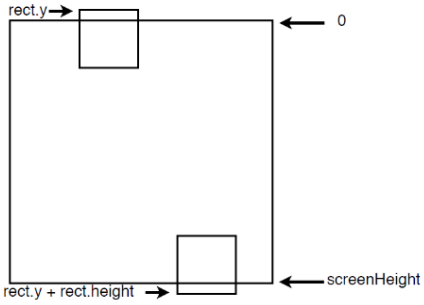
17. You will need to override another method of the Enemy class to make the VerticalEnemy move in its own special way. That method is _____. Override that method by first adding this line to the method:

    ```
    Rectangle rect = getRectangle();
    ```

18. The line above is calling a method not in the VerticalEnemy class. How did VerticalEnemy get such a method?

19. Increase the y-coordinate of the **rect** variable by the **ySpeed**. (Recall that Rectangle variables have a public field **y**)

20. If you run your game now, you should notice that the VerticalEnemy is now moving! Unfortunately, it is not bouncing off the walls. To fix this, you need a condition that checks if the top of the rectangle goes off the top of the screen **OR** if the bottom of the rectangle goes off the bottom of the screen. In either case the ySpeed should be negated (take on the opposite sign). **Notice how Player does not go through the walls, how was that accomplished? Where could you look for a hint?**



21. If you run your project now, your VerticalEnemy should be working correctly!

Part 6 - Diagonal Enemy: Look at the DiagonalEnemy class. Follow these steps to set up the DiagonalEnemy class.

22. Add a line of code so that DiagonalEnemy is a subclass of VerticalEnemy. You will have an error - this is ok!

23. A DiagonalEnemy will be described by 5 things - a rectangle (inherited from _____), a **ySpeed** integer (inherited from _____), a **screenHeight** integer (inherited from), an **xSpeed**, and a **screenWidth**. Create fields for the two **non-inherited fields**. IMPORTANT NOTE: Remember that you SHOULD NOT redeclare fields that are declared in the superclass. Doing so can break things.

24. The DiagonalEnemy constructor takes 8 parameters - the first 4 represent the rectangular bounds of the object, the next 2 represent the y speed and height of the screen, and the final 2 represent the x speed and width of the screen. To finish the constructor correctly, you must do the following:

    i) Call the superclass constructor using the keyword _____ - ALWAYS do this in all of your constructors when using inheritance. You should pass some of the parameters of your constructor to the superclass constructor. Specifically, the VerticalEnemy constructor requires the _____, _____, _____, _____, _____, and _____ as parameters.

    ii) Initialize your **xSpeed** and **screenWidth** fields to the other two parameters.

25. Override a method so that the DiagonalEnemy has a color of your choice. Go back to the ObstacleGame class and uncomment the line of code that adds a DiagonalEnemy object to the Enemies array in setUpGame(). If you run your project, you should now see your DiagonalEnemy (although it is moving like a VerticalEnemy currently...)

26. You will now need to override the move method.

i) When overriding the move method, what happens when you run your program if below is the only thing you write? _____

```
public void move() {
}
```

ii) We still want the DiagonalEnemy to move like a VerticalEnemy (up and down) but with the added ability to also move side to side. So we are not trying to replace the contents of the move() method we are inheriting, we want to add on to it. To make the DiagonalEnemy still move like a VerticalEnemy, you should write what ONE command as the first line of the method?

_____.

iii) To get the DiagonalEnemy to move horizontally, we need code similar to what VerticalEnemy had, but for the x direction. **See if you can come up with that code with your partner.**

iv) If you run your project now, you should have a working DiagonalEnemy!

Part 7 - Stalker Enemy: Look at the StalkerEnemy class. Follow these steps to set up the StalkerEnemy class.

27. Add a line of code so that StalkerEnemy is a subclass of Enemy. You will have an error - this is ok!

28. A StalkerEnemy will be described by 5 things - a rectangle (inherited from Enemy) and the playerRect (the rectangle that represents the player). Create fields for the **non-inherited field**. IMPORTANT NOTE: Remember that you SHOULD NOT redeclare fields that are declared in the superclass. What did you add?

29. The StalkerEnemy constructor takes 5 parameters - the first 4 represent the rectangular bounds of the object, and the last one represents the rectangle of the player. To finish the constructor correctly, you must do the following:

i) Call the superclass constructor using the keyword _____ - ALWAYS do this in all of your constructors when using inheritance. You should pass some of the parameters of your constructor to the superclass constructor. Specifically, the Enemy constructor requires the _____, _____, _____, as parameters.

ii) Initialize your **playerRect** fields to the other parameter.

30. Override the appropriate method so that the StalkerEnemy is another color. Go back to the ObstacleGame class and uncomment the line of code that adds a StalkerEnemy object to the Enemies array in setUpGame(). If you run your project, you should now see your StalkerEnemy.

31. Now we need to make the StalkerEnemy always move toward the player.

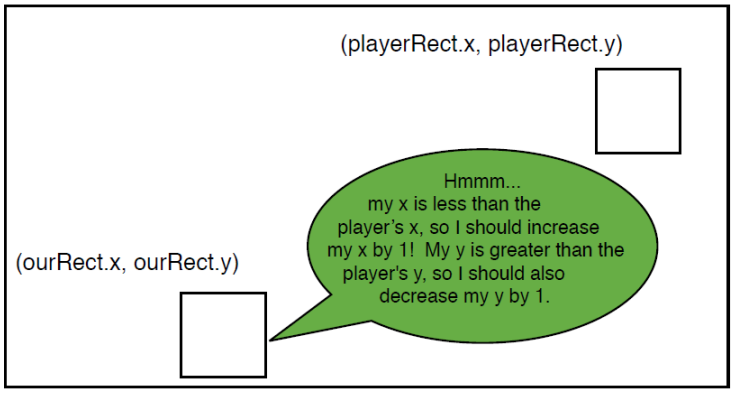
i) Override the _____ method.

ii) Write this line of code to access the Stalker's rectangle object

Rectangle ourRect = getRectangle();

iii) Now the movement of the Stalker should depend on its position relative to the player.

iv) If you run your project now, your StalkerEnemy should work properly!



Extras: Games always provide lots of room for extras. Here are some ideas for you!

New Enemies – what other types of enemies can you think of?
Random Starting Locations – Can you make the enemies start in random locations?
Levels - add levels to the game that progressively get harder.
Timer - make it so the player has to reach the goal in a set amount of time.
Score - perhaps give the player a score for completing the game quickly?

Pictures - You have to change a few things to get this to work, but here are the essentials

A) You need to find pictures to use in your game (preferably png's with transparent backgrounds) - add them to the src folder of your project as well - be sure to refresh!
B) Add this method to Enemy class

```
public Image getImage() {
        return ImageLoader.loadCompatibleImage("nameOfDefaultImage.png");
 }
```

Where nameOfDefaultImage.png is a default image for your Enemies. This needs to be a picture file you put in the src folder.

C) Change the Enemy class' draw method accordingly:

```
public void draw(Graphics g) {
        g.drawImage(getImage(), rect.x, rect.y, rect.width, rect.height, null);
}
```

D) In the Enemy subclasses, override the getImage() method so that it returns the image you want

Packing a Project to Send to a Friend - http://www.youtube.com/watch?v=xTHum2n2igs