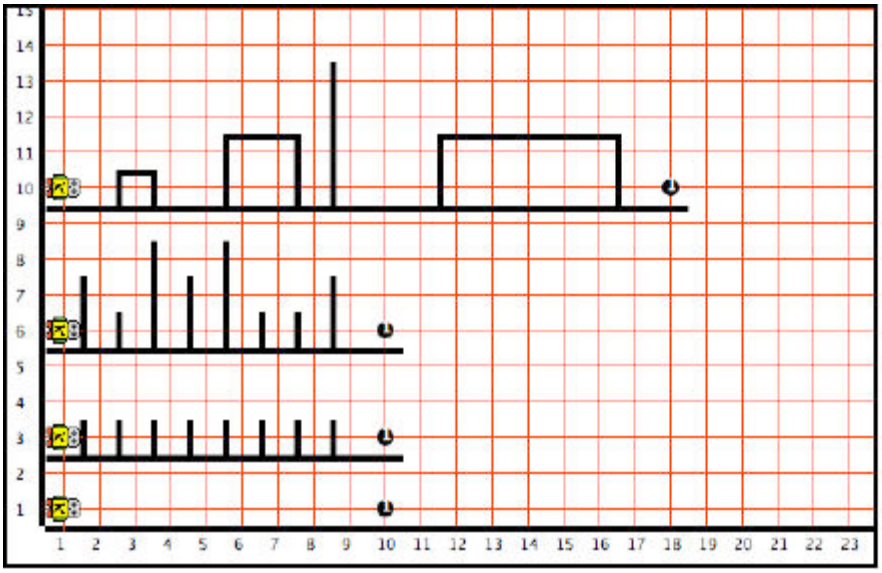
Racer Robots Lab – Lab 2



(note: for the hurdles and high-hurdles in the world, they are not necessarily on every avenue – as the graphic above might falsely lead you to believe – we'll run this lab in class before you get started so you are clear)

Each robot races to the finish line (a beeper). Call the method `runRace()`.

Each robot encounters obstacles along the way. Call the method `raceStride()`.

There will be an abstract class called `AbstractRacerRobot`. The two methods above, `runRace()` and `raceStride()`, go in AbstractRacerRobot. You decide where they each are to be implemented - ask yourself if the method would be implemented the same regardless of subclass – if the answer is yes, that tells you one thing – if the answer is no, that tells you another.

There will be another abstract class called `AbstractHurdlerRobot`. It will contain at least 3 methods (you decide whether the methods are abstract or not) – `up()`, `over()`, and `down()`. You figure out what `AbstractHurdlerRobot` should have as a superclass and which classes should inherit from `AbstractHurdlerRobot`.

Some of the concrete subclasses (there are 4) will need other methods (because, of course, you're going to re-factor/stepwise-refine your algorithms – create other methods and necessary).

The 4 concrete classes will be `SteepleChaserBot`, `HighHurdlerBot`, `HurdlerBot`, and `SprinterBot`.

Step 1: draw the **complete** Inheritance structure. Your diagram should make it clear as to what methods there will be and where they will go (use the diagramming method we've been using in class). At this point, you are making your best guess as to the structure. You'll probably not be correct. What will happen is, as you code you'll find a better OO organization of the methods and possibly the inheritance structure – then you'll adjust. In order to do this step correctly, you will have to stop and think a bit about the algorithms so you have an idea how you will refactor and stepwise-refine (i.e., what helper methods will you create).

Step 2: write the `AbstractRacerRobot` abstract class (what's its superclass?)

Step 3: write `SprinterBot` – test it by itself and don't go on until it works

Step 4: write the `AbstractHurdlerRobot` abstract class (what's its superclass?)

Step 5: write `SteepleChaserBot` – test it by itself and don't go on until it works

Step 6: write `HighHurdlerBot` – test it by itself and don't go on until it works

Step 7: write `HurdlerBot` – test it by itself and don't go on until it works

Step 8: Look for any code you may have written in more than one method – if so, pull it out and create one method. Next, look for any methods that you may have written in more than one class – if so, look for a better place to put those methods

Step 9: make sure your client code demonstrates polymorphism - you should explain what code is polymorphic and why (using English sentences)

Step 10: Draw the object relationships – use Paint to create the diagram and then save as a graphic file in your project folder.

Step 11: Justify your choices for OO/class/method design. There is no one perfect organization of the classes/methods – some are, however, better than others. I'm interested in your justification more than whether you found the best organization. This is a very important part of your learning. You should use the terminology we've been focusing on the last few weeks. You should use English sentences, refer to your object diagrams, and incorporate OO terminology into your design justification. You justifications should be part of your class comments.

In particular, make sure to explain:

- where in your code you are using polymorphism
- for the helper/auxiliary methods you chose to write, why did you write them and why did you put them where you did
- where in your code you are overriding methods and why

Ch4 Problem 2 – Racer Robot Class Diagram