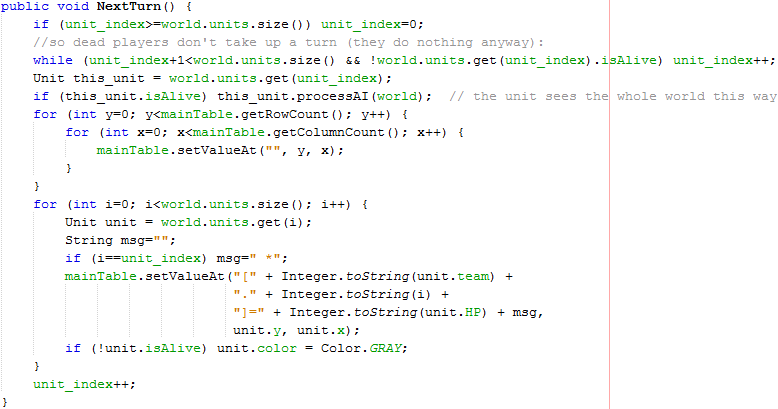
Advanced Programming Using Java - **acp3.6** - Lesson 6 (Chapter 3 Part B)

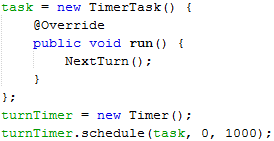
Abstract Classes

Some classes, like TimerTask, have one or more abstract methods that you must implement in order to use them. This makes the class abstract even though the class itself is not declared including the abstract keyword.

Create a NextTurn method that will make things happen in the world, make it into a TimerTask, and run that task using a Timer:

* Open your acp3 project
* Add a NextTurn method to the MainForm class:  
  
* Add Timer and TimerTask variables to the MainForm class:



* + Click the bulb by the turnTimer line and import for java.util.Timer
  + Click the bulb by the turnTask line and import java.util.TimerTask
* In the MainForm constructor, create an inline class declaration for two classes that will be only used once. It is a TimerTask class with your own override for the run method (the abstract method run makes the class abstract even though the class is not abstract; the abstract method must be implemented in order to use the TimerTask):   
  

Advanced Programming Using Java - **acp3.7** - Lesson 6 (Chapter 3 Part B)

Abstract Class

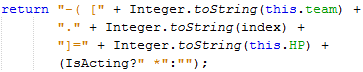
Make Unit abstract to ensure that a unit is never created (only Archer, Swordsman, and other subclasses, which implement the):

* In Unit.java, change public class Unit to public abstract class Unit

Create an abstract method to ensure that all units have a display method but must implement their own:

* In the Unit class in Unit.java, write the following abstract method:

Implement the abstract method in all subclasses:

* Open the Archer.java file
* Click the bulb by the class Archer line, then click “Implement all abstract methods”
* Find the new method that was added, erase the throw statement, then add the following code to the method:  
  
* Repeat these steps for Swordsman (after clicking implement all…, paste the code you typed above), but change "–( [" to "/ [" so it looks like a sword instead of a drawn bow.

Make sure that the unit’s own display method is used, instead of using the custom display string you wrote before in MainForm, so that Archer and Swordsman are actually displayed differently:

* Open MainForm
* Find where the old version of the code above was used (Ctrl F and type “Integer.toString(i” without quotes)
* Delete the String msg variable, i==unit\_index, and the first parameter of setValueAt, and change the call to look like the code below:   
  

(there is no more code required in this lesson)

Interfaces

Interfaces are already used in your program. Go to the MainForm. Find the interface by pressing Ctrl F then typing *Runnable*. An interface called Runnable is used for the form. An interface is purely abstract and doesn’t implement anything. Interfaces exist so that many objects can be used in the same way. Any object added to Java.awt.EventQueue must be runnable, which at minimum requires the object to have a run method that is called when it is your object’s turn in the event queue. This is just one example of an interface, but shows that many different classes that have nothing else in common can have interfaces that, when necessary, ensure that other parts of the program to know how to use them. If Unit were an interface, it would have no code except the class and empty method declarations. Creating an Archer would require typing class Archer implements Unit for the class declaration. However, interfaces are more helpful when the classes do have any contents of their methods in common, such as with lower level parts of the program such as where objects need to be processed in some way, such as being *serialized* (stored in some kind of data format for being stored or transmitted), or queued. Interfaces can also be used for interoperability, such as for writing a *plugin* interface for your software, so that you or other people can write plugins for your software, adding new functionality to your program without recompiling.

Changelog:

* (2017-04-17) Add teams\_count, and completely updated unit creation loop (top of 3.4)
* (2017-04-17) add the “name” line to Archer and Swordsman
* (2017-04-17) in Unit, Archer and Swordsman:
  + change import javafx.scene.paint.Color to import java.awt.Color
  + change Color.rgb to new Color
* (2017-04-25) The super function on 3.5 had row & column switched (the image is now correct)

Known Issues (full BONUS for fixing at least one):

* Instead of changing to gray on death, NextTurn could set unit color to lighter color, such as unit.color.brighter() when HP less than zero and isAlive (then isAlive set to false there instead of in the attack method).
* World should have a getAliveUnitAt method, used instead wherever getUnitAt has been used, so that units can walk over dead units