GREENFOOT QUICK REFERENCE

Origin and grid: In the upper left (0, 0). Max x and y are (size – 1). So a World with 600 by 400 cells has a max x of 599 and a max y of 399.

Constructors: Every object has a hidden default constructor. You can create optional (overloading) constructors the same way you would in any other Java class.

Javadocs: There are complete Javadocs for the Greenfoot classes (Greenfoot, Actor, World, GreenfootImage, GreenfootSound, MouseInfo) is available within the program and online.

Important Actor Methods	Description
`public void act ()`	Method called by Greenfoot for each act the program
`public void addedToWorld (World w)`	Method called by Greenfoot when a new instance of the Actor is added to the World
`public void setLocation(int x, int y)`	Set the Actor's current cell location to x, y. This is one way of achieving movement.
`public int getX()`	Returns the Actor's current X position as an integer
`public int getY()`	eturns the Actor's current Y position as an integer
`public World getWorld()`	Returns the World that the Actor currently exists in. The returned World object can be used to call methods from the World. You can cast the resulting World into a specific type of World (I.e. MyWorld) in order to call the methods you have written in your World.
`public void setRotation(int)`	Set the rotational angle of the current Actor
`public void turn(int)`	Turn the actor clockwise by *int* degrees. Negative values will turn counter-clock
`public void move(int)`	Move the actor forward *int* cells. Negative values will cause the Actor to move backward

Useful Notes on Actors:

Your World can add an Actor (in this case called Ball) as follows:

```
addObject (new Ball(), xPosition, yPosition);
```

Or, if you want to store a permanent reference, create that reference as follows:

```
// Ball inherits from Actor
private Ball myBall;
```

```
…
myBall = new Ball ();
addObject (myBall, xPosition, yPosition);
```

Actor removing itself from the World:

```
getWorld().removeObject(this);
```

(Note: After having an Actor remove itself from the World, make sure that no more code within that Actor gets executed.)

Important World Methods	Description
`public void act ()`	Method called by Greenfoot for each act the program
`Public void addObject (object, int x, int y)`	Add a specified object to the current World
`public void removeObject (object)`	Remove the specified object from the World
`public void setActOrder (Class, Class, [Class,…])`	Specify act order of objects in the World
`public void setPaintOrder (Class, Class, [Class,…])`	Specify the order in which objects in the World get painted on the screen
`public int getWidth ()`	Returns the width of the current World
`public int getHeight ()`	Returns the height of the current World

Calling user-defined methods from your own World from an Actor:

It is commonly useful to be able to call your own methods from an Actor (for example, to signal the World to increase score, change game flow, etc). Here is an example of how to do this:

```
MyWorld m = (MyWorld)getWorld();
m.addToScore (10);
```

Getting Keyboard Input

There are two ways of checking which key(s) a user is pressing:

Single Keystroke(prevents "spamming" by holding key down):

```
String k = Greenfoot.getKey();
```

Check if a key is currently being held down:

```
if (Greenfoot.isKeyDown("right"))
     setLocation (getX() + 1, getY());
```

This example would be placed in the act() method of your main character, and will check if the right arrow button is being pressed, and if so it will move this Actor 1 cell to the right.