Student name: _____ Revision #:_____

Program Development:

1. The program Race begins with two cars on a straight race track. A blue car is in the left lane (top) and a red car is in the right lane (bottom). Once the program is run the cars will move forward to the end of the race track. The blue car wins the race by a landslide and displays the image “Blue Car Wins!!”

2. **Car** – is the subclass of Actor. It has no methods but acts to categorize the two cars. This sort of superclass will be essential for larger projects.

   RedCar – is a subclass of Car. It has two methods; act and race. Race contains the source code move which is inherited from the Actor. The act method calls on the race method once the program is run.

```
/**
   * Act - Race the blue car
   */
  public void act()
  {
     race();
  }
```

```
/**
   * Method race moves the car to the finish line slower than the blue car
   */
  public void race()
  {
     move(1); //moves the car forward
  }
```

BlueCar – is a subclass of Car. Class BlueCar moves to the end of the world (RaceTrack) and displays a winning message. It has three methods; act, race, and declareWinner. Race contains the source code move which is inherited from the Actor. The method declareWinner sets a new image for the class BlueCar that says “Blue Car Wins!!” The act method simply calls these two methods.

```
/**
    * Act - Race the red car
    */
   public void act()
   {
      race();
      declareWinner();
   }
```

```
/**
    * Method declareWinner, once the car reaches the edge, changes the car image to an image that says Blue Car Wins!!
    *
    */
   public void declareWinner()
   {
      if (isAtEdge())
      {
         setImage("winner.png");//sets image to "Blue Car Wins!!"
      }
   }
```

```
/**
    * Method race moves the car to the finish line before the red car
    *
    */
   public void race()
   {
      move(2);//moves the car forward
   }
```

RaceTrack – is a subclass of World. RaceTrack is a straight race track that starts and ends at the left and right edge, respectively, of the world. It has one method which creates the size and pixels and adds the two cars.

```
/**
   * Create the blue and green car's world (the race track).
   * The size is 700x175 with every cell being one pixel.
   */
  public RaceTrack()
  {
     super(700, 175, 1);
     addObject(new RedCar(),50,120); //adds the red car to the right lane at startup
     addObject(new BlueCar(),50,50); //adds the blue car to the left lane at startup
  }
```

3. **Method signatures –**
   - public void race() – This method is public so it can be used by other objects and programs. Void means that it will not return a value and I named it race because the cars are racing each other. It is called by inserting race() into the act method.
   - public void declareWinner() – This method is public so it can be used by other objects and programs. Void means that it will not return a value and I named it declareWinner because it displays a message on the winning car (BlueCar). It is called by inserting declareWinner() into the act method.

4. **Inherited method signatures –**
   - public boolean isAtEdge() - Detect whether the actor has reached the edge of the world. The actor is at the edge of the world if their position is at, or beyond, the cells at the very edge of the world. For example, if your world is 640 by 480 pixels, an actor is at the edge if its X position is <= 0 or >= 639, or its Y position is <= 0 or >= 479. It is called with if isAtEdge().
   - public void move(int distance) - Move this actor the specified distance in the direction it is currently facing. It is called with move()
   - public void setImage(GreenfootImage image) - Set the image for this actor to the specified image. It is called with setImage("filename").