A DISTRIBUTED PROGRAMMING MULTI-PLAYER RACING GAME

-A Java Application

-For ease of understanding, this has been divided into 3 parts

The goal of this assignment is to develop a racing game which allows one person to drive their kart on one workstation and the other person to drive their kart on a second workstation while both individuals can view both karts on their respective screens in real time as they move around a racetrack, the communication being managed by a multi-threaded server. However, the game can be tested on the same machine by launching two client instances and one server instance to achieve the same result.

Part 1: Design your karts

Using an off the shelf image editing tool of your choice (e.g. GIMP) create a JPEG image, 50 pixels by 50 pixels, of a kart viewed from above (i.e. the display will be a top-down 2D view). It will be important to choose an image where it is easy to tell the front from the back of the vehicle. You should create 16 versions of the image rotated around its centre, i.e. one for every 22.5 degrees around a circle. *Hint: create each new image from the original and NOT from a previously rotated image, as multiple rotations will have an accumulative effect on errors in the image.*

You will need two kart images (different colours?) with the same 16 views. Make sure you can distinguish between the two.

Write a Java application that spins one kart at a designated rate. To do this, create one class inheriting from a **JFrame**, and add another class which inherits from a **JPanel** implementing **ActionListener**. The customized **JPanel** will effectively be the 'palette' to which you add the image of the kart. An animation can be created by rapidly displaying slightly different views of an array of images (a 'sprite'). Recent versions of *Deitel & Deitel*'s book *Java: How to Program* for example include a section on '*Animating a Series of Images*'. General web searches should also provide sources.

One approach is to make each image a type of **ImageIcon** so that the **JPanel**'s **paintComponent()** method can render the image using method **paintIcon()**. The animation should be driven using a **Swing Timer** class (not to be confused with a **Timer** class from the **java.util** package) that can repeatedly delay execution for a designated number of milliseconds, automatically calling **paintComponent()** at the end of each cycle via the **ActionListener** whose **actionPerformed** method calls method **repaint()**.

Another approach is to wrap each image with a type of **JLabel** and use the timer action to change its location using **setLocation()**, without calling **repaint()**.

*NB Do not develop a threaded solution using the **Java Thread** class* – use of the **Swing Timer** class is all that is needed.

Part 2: Develop the racetrack

In the second phase, create a Java application for the racetrack where you can 'drive' your *two* karts around a central grassed area on *one* workstation. The racing arena is in a window 850 pixels by 650 pixels and uses the following code to create the arena:

```
Color c1 = Color.green;
g.setColor( c1 );
g.fillRect( 150, 200, 550, 300 ); // grass

Color c2 = Color.black;
g.setColor( c2 );
g.drawRect( 50, 100, 750, 500 );  // outer edge
g.drawRect( 150, 200, 550, 300 ); // inner edge

Color c3 = Color.yellow;
g.setColor( c3 );
g.drawRect( 100, 150, 650, 400 ); // mid-lane marker

Color c4 = Color.white;
g.setColor( c4 );
g.drawLine( 425, 500, 425, 600 ); // start line
```

Select four keyboard keys for each of the two players. Two keys are used to change the direction of each kart. A key press turns the kart 22.5 degrees left or right *by changing the current image – I recommend you do not arithmetically rotate a single image during execution as it will create latency issues and does not naturally follow from the part 1 development.* The other two keys are to change the speed of the kart. Since *speed = distance/time*, in this context the motion of the kart will be represented by pixel displacement/screen refresh. Use a scheme whereby speed is represented on a scale from 0 to 100. One press of a key either increases the speed by 10 or decreases the speed by 10. I recommend you develop *one* **JFrame** containing *one* **JPanel** which itself contains *both* the track and the two kart images, but you will also need a **KeyListener** to implement the key presses.

Write a Java application that allows two clients (players) to drive the two karts around the central reservation by sharing a common keyboard with different sets of keys. You may add additional graphical features to make it visually more interesting.

Factors to consider include how a change in direction is implemented, how a change in speed is implemented, how you might implement *collision detection* between a kart and the inner edge of the track or the outer edge of the track. Ideally, in the event of a collision with such a

barrier the kart's speed becomes 0 whilst the other kart proceeds, unless they collide with each other, in which case the game is ended. You could even associate sound effects with these events (e.g.: http://simplythebest.net/sounds/ contains some free sound clip files).

Part 3: Develop the ability to race the karts on two workstations

The idea here is to create a client/server application, i.e. a server and two clients where each player drives one kart on each client and races against the other player whose kart is also displayed on the corresponding client.

General issues:
It is preferable if at some stage you can test your application on a network; for example, you could access other machines by simultaneously logging into different machines, either via multiple instances of your own login or with help from a colleagues' login. Load the server software on one machine and a copy each of the client software onto two other machines

entirely possible to undertake all development and testing on one machine; for example, run two clients and the server from three separate console windows, or perhaps with one of the instances from the development environment (e.g. JGrasp, Eclipse, Netbeans).

Many issues must be considered when designing a client/server application; some *general* questions are listed below applicable to *any* client/server application (emboldened), followed by a comment specific to this project:

1. **What functionally is performed on the server? Does the server need to interact with a database or legacy software? Does it provide centralized services?**

   In this 3rd part you should use as much of part 2 code as possible. Therefore, the two clients perform most of the functionality. The server only provides the communication pipe between the two clients. Only the server needs to be threaded if you use a **Swing Timer** class in the client (which guarantees a regular screen update).

2. **What functionally is performed on the client? What user interface needs to be provided?**

   A user can only control their own kart with four keyboard keys. When one person drives their kart, the motion of the kart should appear on both workstations' screens. If the two karts collide, the game is over and both clients need to be notified. Allow either player to restart or exit the game from a menu.

3. **How many clients? And what categories of clients?**

   In this scenario we have two clients.

4. **What message protocols are needed? What does each type of message look like? What events cause a message? Does the application need a secure channel?**

   You will need to give careful thought on how to do this. Ignore the issue of security.

5. **How to get the application started? Is authentication of clients/users needed?**

   You will need to start the server first. Then allow either client to connect one after the other. After both clients have connected and the images of the karts are loaded, you will need a way to start the race.

6. **How to close the application down?**

   If a user exits in the middle of a race, the client should inform the server then the server can inform the other client to exit as well.

7. **How many applications can run simultaneously?**

   Here we assume only one game with two clients (2 karts racing) at a time.

8. **How to coordinate the server and the clients?**

   You cannot have a client send a request and the server *not* respond. If you do, the application will lockup.

THE PROJECT MUST:

-Be handed in one zip file containing source and compiled code for all three parts in three separate folders.

-Have a word document explaining each 3 parts of the project.

-In each three sections present a
brief description (about 600 words) evaluating any items of interest related to your development (e.g.: kart graphics, arena/map detail, display update, implementation of networked version, etc.), including a critical evaluation of the various distributed technologies, principles, and protocols you have used, and why. Even if any part was not fully completed or is unsuccessful the work should be critically appraised (what worked, what did not work so well). Each section should also contain a copy of the relevant code.
It is likely some of the code will be duplicated between parts 1, 2 and 3, that is to be expected. The word count for each section could exceed 1000 words if needed

-A README document containing instructions for set up and running each of the three applications.

PROJECT HELP (PART 1)

Part 1: Spinning your cars

- Find a top-down simple image of a racing car
- Edit it to be 50x50, ideally save as png, tiff, or jpeg (ie supports α <alpha> channel)
- Copy and rotate 16 times (22.5° intervals), name them kartRed0.png, kartRed1.png, etc
- View section in *Deitel & Deitel 'Java:How To Program'* – *'Animating a Series of Images'* and utilise code listing. This uses a **JFrame** and a **JPanel**

JPanel coding:

- Define a class that extends a **JPanel** implements **ActionListener**
- In constructor
  - Load images (eg into an array or alternative approach).
  - Create a **Swing Timer** class with eg 100ms delay and start it
- In **actionPerformed** specify one line '**repaint();**'
- In **paintComponent()**
  - Display the current image
  - Re-assign the current image to next sequential image

JFrame coding:

- Define a class that extends a **JFrame**
- In constructor
  - Configure frame properties
  - Create an object of the panel class and add it to the frame
- In **Main()**
  - Create an object of the frame
  - Make the frame visible

The kart will animate because the **Timer** clicks force a call to **actionPerformed()** which in turn calls **paintComponent()** and refreshes the screen with a new image every time. i.e the timer ensures a regular screen refresh

PROJECT HELP (PART 2)

Implementing Speed

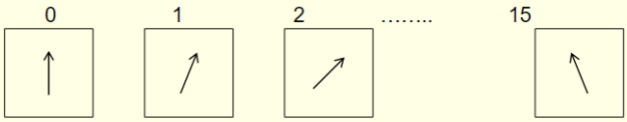
One approach (e.g. timer, listener(s), code location can change)

1. Define a class that extends **JFrame**.
2. Define a class extends **JPanel** implements **ActionListener**, **KeyListener**.
3. Create an object of the **Swing Timer** class. Use **paintComponent()** to load and draw a current image. Call **repaint()** in the **actionPerformed()** method (regularly updates the screen).

4. Consider defining a class to define properties of a car (location, speed, image, etc)

Sole purpose of **Swing Timer** class in client is to ensure regular screen refresh – it is *not* used to change in-game kart 'speed'

```
paintComponent()
{
        … draw track, update car image and position
}
```



visible change = fixed change * displacement

14	15	0	1	2
13				3
12		*		4
11				5
10	9	8	7	6

```
if( direction == 3 )
{
   // displace image
   x = x + 2 * speedf;
   y = y - 1 * speedf;
}
else if( direction == 4 )
{
      // displace image
      x = x + 2 * speedf;
} else if .....
```

PROJECT HELP (PART 3)

Part 3: Implementing a threaded server for connection to two clients

1. Remember you need to develop TWO programs: one client and one server. The client program will be run twice, either on separate machines or in separate console windows on the same machine. The server will be run once either on a separate machine or in a 3rd console window.

This advice assumes you have Assignment Part 2 working.

2. Remove the client code for keyboard keys controlling one of the karts, so that only one set of 4-keys can control one kart (the other kart will still be visible but initially motionless).

3. Add Socket code that enables this program (the client) to connect to a server (base your solution on the practical on developing a multi-client echo server).

4. In principle, each client will send its local kart information to the server (e.g.: x, y, location, direction, speed/displacement) and receive the remote kart information to update the remote image on the client.

5. Client reading and writing should be undertaken as discrete activities (no loop) at the point at which the karts are displayed (e.g.: in **paintComponent()**) as timer will ensure regular updates.
6. Develop the multi-threaded server program. More than one solution is possible, but here is one: The

server program needs two user-defined objects for each connection to the clients, each of which is passed to a thread, this *may* include input/output streams. Each server object's threaded **run()** method sends kart data to one client via one outputstream and receives the kart data via one inputstream from another client.

7. Overall, it's important to distinguish which operations in the server code are undertaken once (e.g.: initial client-server connections) and which are constantly undertaken <e.g.: I/O input via a loop in the **run()** method>).

NB. The suggestions listed here are part of just ONE possible approach; other approaches are possible and quite valid.