Lab 11 – CodeRally

Objectives:

- Understanding the concept of object-oriented
- Using inheritance

CodeRally

CodeRally gives you the opportunity to pit your Java programming skills against other teams in a world of car rally racing. Each team will write a Java class which represents and controls a `RallyCar`. Your car (class) will be placed on a simulated race track along with cars from other teams.

Rally cars can move around on a track and obtain important information such as the location of various objects on the track and the current capabilities of other cars. Cars can collide with other cars, throw spare tires to distract other cars, and they can enter a protected mode to (temporarily) protect themselves from being hurt by other cars.

The rally pits cars against each other in a series of matches. A match consists of up to six cars competing with each other. Each car starts a match positioned in a random location and facing a random direction on a finite two-dimensional grid, with the same amount of fuel and spare tires as the other cars. Driving around the track uses fuel. The track contains places where cars can go to get additional fuel and different places they can go to pick up spare tires. If a car runs out of fuel it can no longer move, so it is important to constantly check your fuel and refuel if necessary.

During each match, a car can accumulate "points." Cars can earn points in three ways:

- by successfully throwing a spare tire at another car,
- by passing checkpoints in order or out of order around the track,
- and by the amount of fuel remaining at the end of a match.

Cars with the highest point totals from each match advance to subsequent rounds.

Your turn (1)

Let's start.

1. Goto File > New > Project

2. Select Other > Game Project. Click on Next as shown in Figure 1.

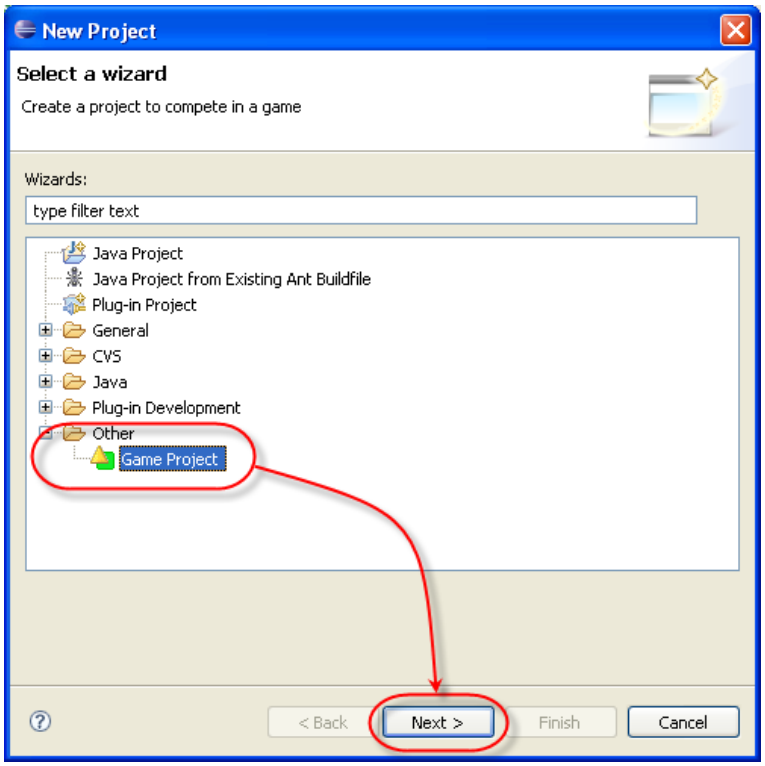

Figure 1 Create new Game Project

3. Make sure that the game is CodeRally. Enter your student ID. as your project name, and click Finish as shown in Figure 2.

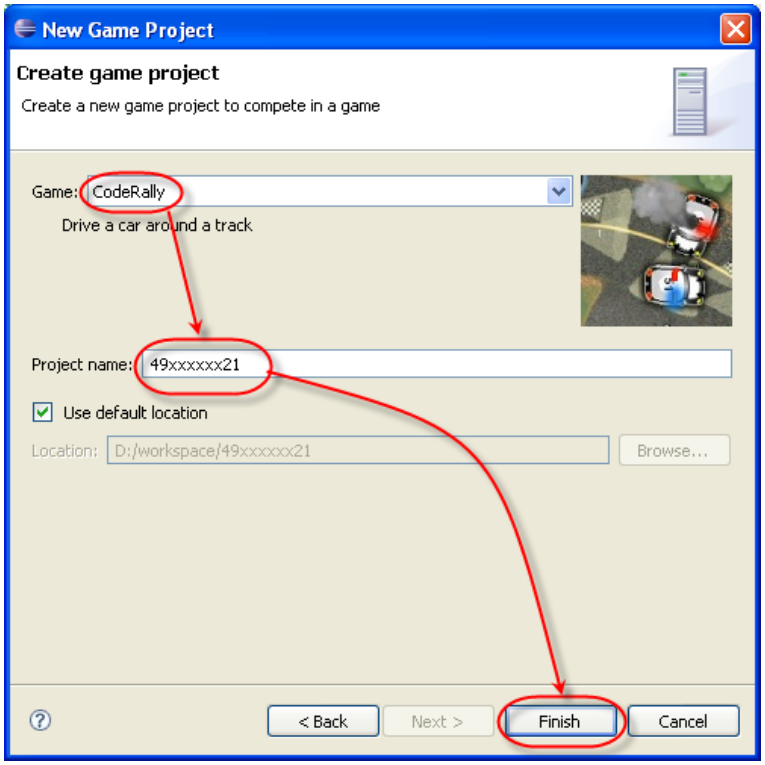

Figure 2 New Game Project name

4. Your project will have the content as shown in Figure 3.

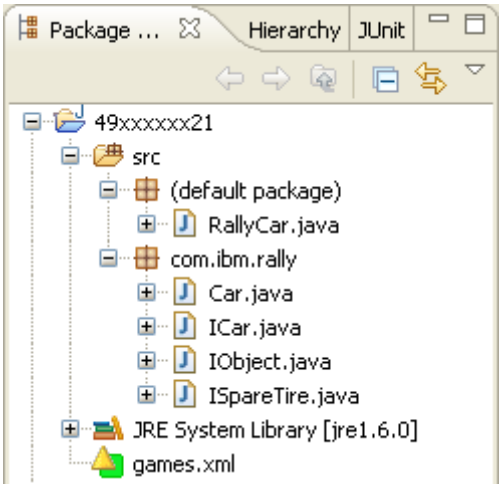

Figure 3 Your game project

5. Select `RallyCar.java`. This will be the source file that you must modify to make your car be able to run on the simulated track. In method `getColor()`, change the returned color from *CAR_BLUE* to other colors specified by *CAR_XXX* in `Car.java`.

6. Try to run your car in the simulated track by select `games.xml`.
   6.1. Enter your name and organization.
   6.2. Click Add My Team to add your car into the track.
   6.3. Select other players, i.e. Crazy driver, Right and turn and click Add. To run the rally, click Run as shown in Figure 4.

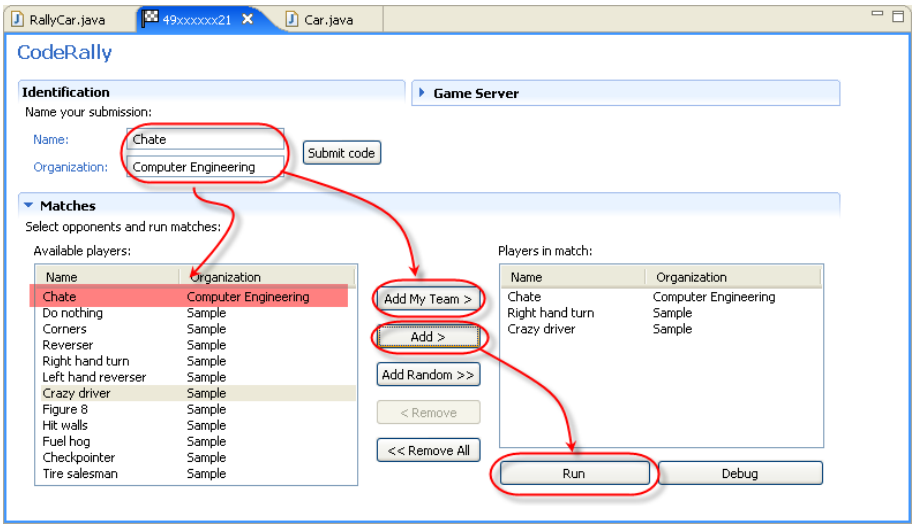

Figure 4 Setup your car to the race

7. The simulated track will have the race with the cars you added to the match as shown in Figure 5.

8. To get the document for the classes in the project,
   8.1. click on the project name and select menu **File** > **Export**.
   8.2. select **Java** > **Javadoc**

8.3. click **Next** and the **Finish**.

9. You will see that the doc subfolder will be added to your project. Explore the document to see the Javadoc of all classes.
10. Your job is to add statements to the method move() in `RallyCar.java` to make your car get as much points as possible. The following section will help you in coding your car.

Coding Your Car – Overview

When you start Eclipse on your machine and open the `CodeRally` project, you will find a skeleton for the class `RallyCar.java`. This is the class that will contain the code making up your car. You may add fields and declare additional methods, as well as create other Java classes.

The `RallyCar` class contains stubs for certain methods required in cars; you will have to fill in the code in these methods. Modifying these methods is the primary manner in which you create the "personality" of your car. You may also add other fields and methods to the `RallyCar` class to further define its characteristics.

When you have a version of the `RallyCar` class that you wish to test it, save your `RallyCar` code, select the `games.xml` in your project, and repeat the step (6) and (7) in exercise 1.

The class `Car.java` is the superclass of `RallyCar`. It defines a number of methods that are inherited by `RallyCar`. These methods can be extremely useful in controlling your car.

You should **NOT** modify anything in the `Car` class. In fact, when you run your car, it will actually run with a different version of the `Car` class from the one you see in your environment. In particular, the Car class you will see contains some dummy initialization and return value code that will be replaced when you run in the simulated track.

In addition to the above classes, your environment will contain three Java interfaces that define the interfaces presented by various components of the track:

- **IObject.java**
  This is the interface of all objects in the simulated track. Every object implements this interface, which declares methods `getX()` and `getY()` that return the location of the object on the track. All track coordinates are non-negative values of type double.

- **ISpareTire.java**
  This interface extends `IObject` and defines the interface of all spare tires that are currently active in the simulated track. Every spare tire implements this interface, which declares methods `getHeading()` and `getSpeed()`. Thus every spare tire contains methods that allow cars to determine important characteristics of the spare tire as the tire moves across the track.

- **ICar.java**

  This interface extends `IObject` and defines the interface for all cars on the simulated track. Thus it defines methods your car can invoke either on itself or on an opponent's car. These methods are described in further detail below and in the JavaDocs for the `CodeRally` environment.

The CodeRally Track Simulation

Identification

There is a method stubs in the `RallyCar` class that you must fill to identify your car. This method is getColor(), which must return a byte constant chosen from the predefined car colors given in the `Car` class. You can use this method to assign a color to your car, which determines its appearance in the graphical display of the `CodeRally` environment. The default value returned by `getColor()` is `CAR_BLUE`.

The identification method must not do any computations other than returning the specified constant values.

Initialization

When your car is placed into a track, the simulator invokes the `initialize()` method in your car. Put any initialization code you want to have executed into this method. You may make use of the entire API at this time. Be aware that the simulator will provide only a limited amount of time (1 second) for your initialization code to execute before it begins the game. If your initialization code fails to complete within the time limit, your car will enter the track in an uninitialized state, with unpredictable results.

Moving Your Car

Once the simulator finishes its timed calls to each car's `initialize()` routine, it calls the `move()` method in each car in sequential order. This happens once every clock tick. The code in your car's `move()` method determines what actions it takes during the course of a game. In addition to the input parameters to `move()`, which give some status information, methods are available to your car to query its own status, to change variables such as the desired direction and speed at which it should move, to query the status of other cars, to find the location of objects on the track (for example, the gas stations which can be used to refuel or places where you can pick up spare tires), and to throw spare tires from your car.

`move()` has four parameters that provide information about what happened during the previous movement cycle. These parameters specify:

(1) how much time (in milliseconds) your `move()` method used the previous time it was called;
(2) whether your car hit a wall during the previous cycle,
(3) whether your car collided with another car during the previous cycle, and
(4) whether your car was hit by a spare tire from another car during the previous cycle.

The first parameter is an `int`, the second is a `boolean`, the third and fourth parameters are an `ICar` reference to the corresponding car (or null, if no collision or hit occurred). The first

parameter is useful in determining whether your car is in danger of exceeding the maximum amount of time allowed to complete a move.

CodeRally Track Details

A `CodeRally` track is a two-dimensional world of 1010 units in X by 580 units in Y, with the origin in the top left corner. There is a wall around the outside edge of the track, and cars cannot go beyond the wall. There are no walls on the interior of the world. Cars can move freely about the world, unless they bump into another car. Objects move in directions called headings, which are measured in integer degrees. Zero degrees is "straight up". All headings are positive numbers in the range 0..359 and increase in the clockwise direction.

The figure 5 below describes the world:

Figure 5 Simulated track world

The world has the following characteristics:

- The world is driven by a ticking clock whose value can be read using `getClockTicks()`.
- Each car starts the match with 100 fuel units and 3 spare tires.

- Setting steering and throttle causes a car to move continuously with those settings until it is instructed to do otherwise, although it may be blocked by walls or other cars.
- Cars can change throttle and steering instantaneously. Speed and direction will not change instantaneously because the cars have inertia.
- The minimum throttle of a car (`MIN_THROTTLE`) is `-50` units and the maximum throttle (`MAX_THROTTLE`) is `100`; the maximum rate of change of speed from a stopped position (except in collisions) is 8 units per tick.
- The minimum steering setting (`MAX_STEER_LEFT`) is `-10`, and the maximum steering setting (`MAX_STEER_RIGHT`) is `10`. The rate of change of heading is dependant on speed and can be found via the `getChangeInHeading()` method.
- The location of a car on the track is a point. Cars are 60 units long and 40 units wide, centered at their location.
- To keep the detection of a round spare tire hitting a rectangular car simple, a spare tire will hit a car when its location passes within 40 units of the car's location.
- Spare tires thrown by a car move at a constant velocity of 12 units per tick until they hit a car or a wall, at which time they disappear. Checkpoints, fuel depots, and spare tire depots do not affect spare tires as they move across the world. Spare tires do not hit each other if they pass over the same location.
- The maximum amount of fuel a car can have is 100 units.
- The maximum number of spare tires a car can have is 5.
- Whenever a car's location is within 25 units of a fuel depot, the car's fuel is increased at a constant rate of 1 unit per clock tick up to the maximum. There are 3 randomly placed fuel depots during each match.
- Whenever a car's location is within 25 units of a spare tire depot, and the car has less than 5 spare tires, the car will pick up a spare tire every 5 clock ticks. There are 3 randomly placed spare tire depots during each match.
- Cars can protect themselves against collisions with spare tires or other cars by entering "protect mode". A car moving in protect mode consumes fuel at twice the normal rate. Protect mode lasts for 50 clock ticks.
- Colliding with another car will cause your momentum to be transferred to the other car and both cars will loose 10 fuel units.
- Throwing a spare tire and successfully hitting another car will increase your points by 10. The car that is hit will loose 10 fuel units, its `move()` method will not be called for 10 clock ticks, and it will be pushed in the direction that the spare tire was traveling. You will not get any points for hitting a car which is in protected mode, nor will it affect that other car.
- Once a car throws a spare tire, that car will be unable to throw another spare tire for 25 clock ticks from the time the tire was thrown.
- The time limit to complete a single move is 500 milliseconds. If a car's `move()` method does not return within 500 milliseconds of the time when it is called, the `move()` will not be called again for the rest of the match and the most recent steering and throttle settings will be maintained.
- There are a number of ordered checkpoints placed in a route on each track. Passing within 25 units of any checkpoint will give you 2 points, but going to the next

successive checkpoint will give you 6 points. Returning to the same checkpoint twice in a row will not give you any points.

- Points are earned according to the following table. Remember that it is total points earned that determines which cars advance during elimination rounds.

Action	Points Earned
Passing any checkpoint	2
Passing a checkpoint in successive order	6
Hitting another car with a spare tire	10
For each 10 units of fuel left after a match	1

General Information, Caveats, Constraints, and Restrictions

- Cars may not define any constructors.
- Cars may not use static initialization blocks to initialize their cars.
- All the Java code for your car must be contained within the `RallyCar` class.
- Cars may not create their own threads, processes, print jobs, files, or other similar system functions.
- Cars may use `System.out.println()` to display information on the Eclipse console, but the time it takes to do this is charged against the car's `move()` time limit. (Actions like this are relatively time-consuming.)
- Any player that submits a car containing code deemed to be intentionally designed to damage the `CodeRally` environment will be disqualified.

Example RallyCar Code

The following code snippets show simple examples of various operations which might be used inside a `move()` method. Note that these are separate code snippets, not a single complete `move()` method. Note also that these are only examples, intended to give you an idea of how to do things within your car. Winning cars will undoubtedly utilize sophisticated strategies which take full advantage of the range of method calls available to them.

The list of methods available to cars is documented in the JavaDoc descriptions of the classes and interfaces of the `CodeRally` environment. The primary challenge in the `CodeRally` is for you to decide on a strategy which uses the available methods to optimum advantage for your car.

```
/**
 * Go toward the first spare tire depot.
 */
public void move(int lastMoveTime, boolean hitWall,
                 ICar collidedWithCar, ICar hitBySpareTire) {
  // pick a spare tire depot
  IObject st = getSpareTireDepot()[0];
  // go toward the checkpoint
  int h = getHeadingTo(st);
  if (getHeading() > h)
```

```
    setSteeringSetting(MAX_STEER_LEFT);
  else
    setSteeringSetting(MAX_STEER_RIGHT);
  setThrottle(MAX_THROTTLE);
}
```

```
/**
 * Put the car in reverse for a few moves if you collide with another car.
 */
protected int wait;
public void move(int lastMoveTime, boolean hitWall,
                 ICar collidedWithCar, ICar hitBySpareTire) {
  if (collidedWithCar != null)
    wait = 10;
  if (wait > 0)
    setThrottle(MIN_THROTTLE);
  else
    setThrottle(MAX_THROTTLE);
  if (wait > 0)
    wait--;
}
```

Your turn (2)

Let's race

1. Make your car move pass the checkpoints in order.
2. You can run your car alone or select your competitors. Try to make as much points as you can.

To be continued next week. We may have a tournament. Modify your car at home. Please bring this lab document with you for your reference. To download goto http://www.alphaworks.ibm.com/tech/coderally

Reference:

http://www.alphaworks.ibm.com/tech/coderally

Lab 11 – CodeRally

Task	Description	Result	Note
1	Change color		
2	Car can move.		
3	**`Racing`**		
4			
5			
6			
7			