

# Chapter 3: The Driver Program

## The Driver Program

Let me say just a word here about how we'll be coding programs in this course. The `Team` class isn't a complete program because there's no `main()` method. We've created this class without one. Its purpose is to build a team roster and to provide the roster in an output format, not to interact with a user. For that purpose, we'll create another class called a *driver* class, and we'll name it *TeamDriver*. This class will provide the `main()` method we need to have a complete program, and it will allow us to interact with the `Team` class and its capabilities. We can also use it to test the `Team` class very conveniently.

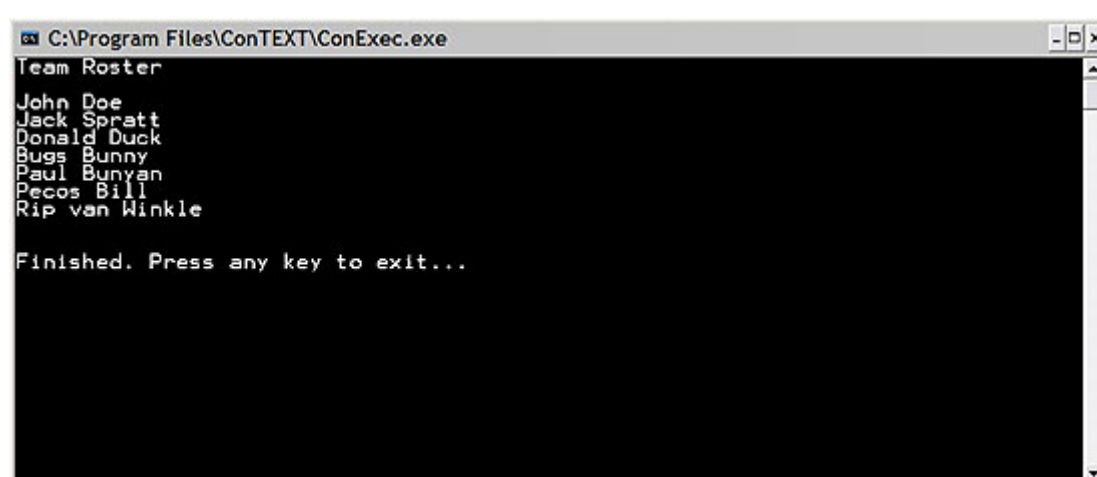
Here's the driver program that will use and test our `Team` class. After I show you the driver, we'll look at the output from running it.

```
public class TeamDriver
{
    public static void main(String[] args)
    {
        Team t = new Team();

        t.addPlayer("John Doe");
        t.addPlayer("Jack Spratt");
        t.addPlayer("Donald Duck");
        t.addPlayer("Bugs Bunny");
        t.addPlayer("Paul Bunyan");
        t.addPlayer("Pecos Bill");
        t.addPlayer("Rip van Winkle");

        System.out.println(t.toString());
    }
}
```

This simple driver does three things. First, it declares a `Team` reference named *t* and creates a default `Team` object for it to refer to. Second, it adds several players to the team using repeated calls to `addPlayer()`. Finally, it displays the roster built by the `toString()` method. As promised, here's the output of a run of this driver:

A screenshot of a Windows command window titled "C:\Program Files\ConTEXT\ConExec.exe". The window has a black background with white text. The output shows a team roster with seven players: John Doe, Jack Spratt, Donald Duck, Bugs Bunny, Paul Bunyan, Pecos Bill, and Rip van Winkle. Below the roster, it says "Finished. Press any key to exit...".

```
C:\Program Files\ConTEXT\ConExec.exe
Team Roster
John Doe
Jack Spratt
Donald Duck
Bugs Bunny
Paul Bunyan
Pecos Bill
Rip van Winkle

Finished. Press any key to exit...
```

Creating a driver like this has several advantages. If we change the Team class for any reason, we can run this test again to make sure it still works. If we want to test other aspects of the class, like what happens if we try to add more players than the roster will hold, all we have to do is add more tests to the driver program. For example, we could create a smaller roster of five spots and add the same seven names to it. And if we want to build a set of tests to test every aspect of the class, we can put them all into one driver so we can rerun them whenever we want. Drivers are very common tools when testing classes, and we'll use them extensively in this course.

Okay! We have a class that uses an array to store and display a roster, and a driver program to test it. Now what?

Being the avid basketball fan that I am, I just decided that a roster of names is not enough. I want to store more information about the players on the team and be able to display it. There are at least two basic ways we could approach this. The first is to create a separate array for each piece of information about the players. Just like we have an array of their names, we could have a second array with their positions, a third one with their heights, a fourth one with average points per game, and so on. The advantage of doing it this way is that it's quick—all we have to do to add another statistic is add another array to the Team class along with ways to get the information in and out.

But this approach also has disadvantages: The number of arrays quickly becomes pretty clumsy, and we have the problem of coordinating the arrays to make sure that each player's information is in the same position in each array. Of course, these problems aren't critical, and this straightforward approach will work.

There's another approach, however, that will only use one array and keep the Team class much simpler. It will also make our lives easier, since we won't have to deal with a bunch of different arrays. (Have you seen how many statistics teams keep these days? We could easily end up with 15 or 20 arrays to keep straight!)

This second approach is simply to create another class to hold all the information about a single player and change the Team class to use an array of these new objects instead of an array of strings. Doesn't that sound *so* much easier?

The first thing to do is set up our new class, which we'll call *Player*. We need to decide what information to store about each player. Most teams track position, average points, rebounds, and assists per game, so let's use those. Here's a *minimal* Player class:

```
public class Player
{
    private String name;

    private char position;

    private double avgPoints;

    private double avgRebounds;

    private double avgAssists;

    public Player(String pName, char pPos, double pPoints, double pRebounds, double pAssists)
    {
        name = pName;

        position = pPos;

        avgPoints = pPoints;

        avgRebounds = pRebounds;

        avgAssists = pAssists;
    }

    public String toString()
    {
        return "Player: " + name +

            "\n   Position:      " + position +

            "\n   Points/Game:    " + avgPoints +

            "\n   Rebounds/Game:  " + avgRebounds +

            "\n   Assists/Game:   " + avgAssists;
    }
}
```

This is a minimal class because all it does is let us create and display player objects. If this were a class for real use, we would have edits on all the fields to make sure they were valid (no negative averages, no positions that don't exist, and so on). We would also have methods to allow changes as the season progressed. This class will do for our purposes, though. If you have any questions about what I didn't include, please ask me in the Discussion Area.

Now that we have a Player class, we need to change our Team class to use it. Let's go over the changes we need to make to Team, and then we'll look at a new listing of the class.

First, we need to change the type of the variable roster from `String[]` to `Player[]`.

Second, in both constructors, we need to change the `new String` declaration to `new Player`.

Next, in the `addPlayer()` method, we need to change the parameter type from `String` to `Player`. Since the parameter is now a `Player` object instead of just a player's name, let's change the parameter name to *player* to better describe its contents. That means we need to change it in the method's header line and also four lines later, where it's used in the assignment statement.

The last change we need to make is in the `toString()` method, to the while loop where each array entry is concatenated to the roster text string. Instead of concatenating the array element (`roster[i]`), we want to concatenate the string returned by its `toString()` method, like this: `roster[i].toString()` .

Our `Team` class now looks like this:

```
public class Team

{

    private Player[] roster; // array for roster

    private int teamSize;    // number of players in roster


    public Team()

    {

        roster = new Player[20]; // create array

        teamSize = 0;            // initialize team size

    }


    public Team(int arraySize)

    {

        roster = new Player[arraySize]; // create array

        teamSize = 0;            // initialize team size

    }


    public void addPlayer(Player player)

    {

        if (teamSize < roster.length)

        {

            roster[teamSize] = player; // add player to roster

            teamSize++;                // increment team size

        }

    }


    public String toString()

    {

        String teamRoster = "Team Roster\n\n"; // output String

        int i = 0;                            // loop counter


        while (i < teamSize)                  // while more players

        {

            teamRoster = teamRoster + roster[i].toString() + "\n"; // add to roster

            i++;                            // increment loop counter

        }


        return teamRoster;                    // return roster

    }

}
```

Now all we have to do is update our driver to match the new classes and try it out! To do that, we need to replace each player's name in the calls to **addPlayer()** with a complete player object. Since we'll only be using them to pass as arguments, we don't really need to save or name them, so we can put the **new Player** right into the method calls, like this:

```
public class TeamDriver

{

    public static void main(String[] args)

    {

        Team t = new Team();

        t.addPlayer(new Player("John Doe", 'C', 19.19, 15.15, 4.4));

        t.addPlayer(new Player("Jack Spratt", 'F', 11.11, 7.7, 3.3));

        t.addPlayer(new Player("Donald Duck", 'G', 13.13, 4.4, 6.6));

        t.addPlayer(new Player("Bugs Bunny", 'G', 8.8, 2.2, 1.1));

        t.addPlayer(new Player("Paul Bunyan", 'C', 5.5, 1.1, 2.2));

        t.addPlayer(new Player("Pecos Bill", 'G', 9.9, 3.3, 6.6));

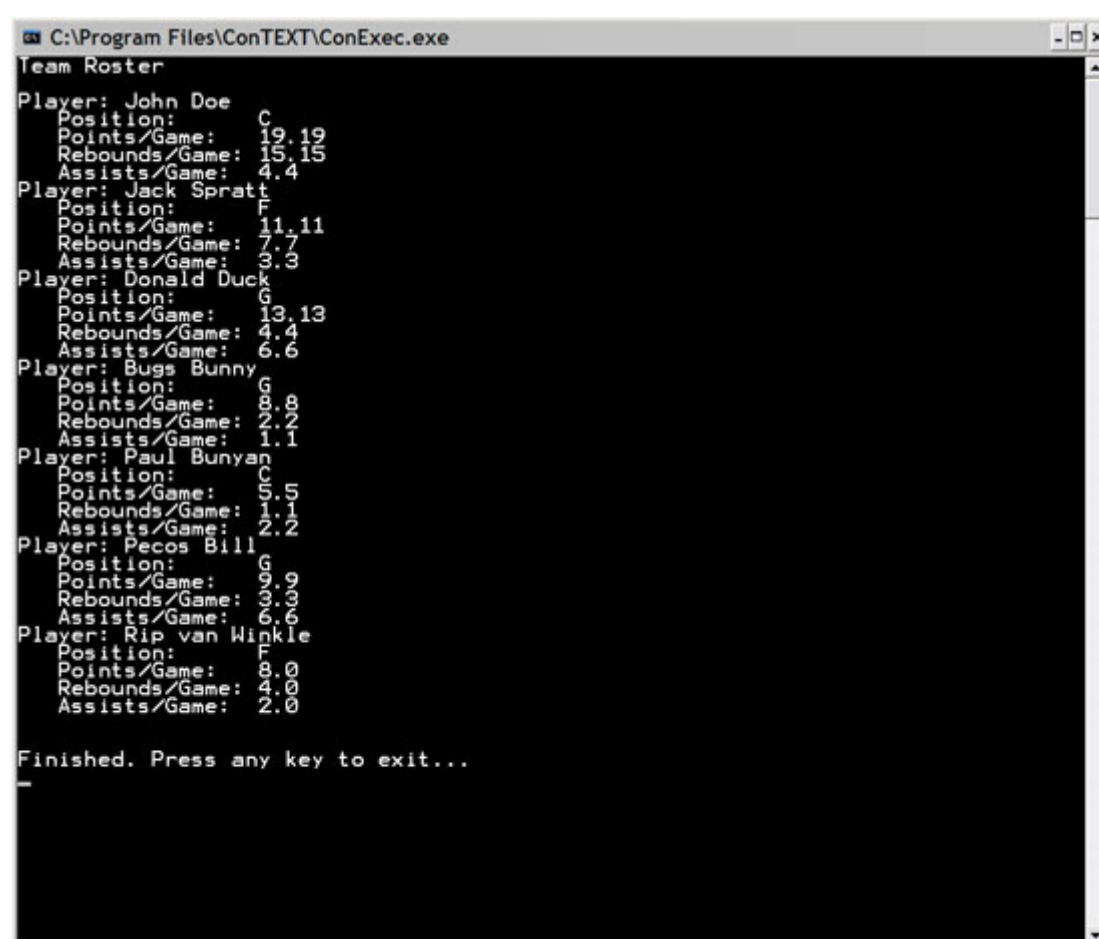
        t.addPlayer(new Player("Rip van Winkle", 'F', 8.0, 4.0, 2.0));

        System.out.println(t.toString());

    }

}
```

Finally, here's what the output looks like from a run of our new driver using our new classes:



```
C:\Program Files\ConTEXT\ConExec.exe
Team Roster
Player: John Doe
  Position: C
  Points/Game: 19.19
  Rebounds/Game: 15.15
  Assists/Game: 4.4
Player: Jack Spratt
  Position: F
  Points/Game: 11.11
  Rebounds/Game: 7.7
  Assists/Game: 3.3
Player: Donald Duck
  Position: G
  Points/Game: 13.13
  Rebounds/Game: 4.4
  Assists/Game: 6.6
Player: Bugs Bunny
  Position: G
  Points/Game: 8.8
  Rebounds/Game: 2.2
  Assists/Game: 1.1
Player: Paul Bunyan
  Position: C
  Points/Game: 5.5
  Rebounds/Game: 1.1
  Assists/Game: 2.2
Player: Pecos Bill
  Position: G
  Points/Game: 9.9
  Rebounds/Game: 3.3
  Assists/Game: 6.6
Player: Rip van Winkle
  Position: F
  Points/Game: 8.0
  Rebounds/Game: 4.0
  Assists/Game: 2.0

Finished. Press any key to exit...
_
```

New TeamDriver output

