

Department of Computer Science and Computer Engineering  
La Trobe University

**CSE100F/400F Semester 2, 2013 – Assignment 3  
and Progress Check Test 2**

**Due Date: Wednesday, 23<sup>th</sup> October, at 9.30 am**

**LAST DAY FOR LATE SUBMISSION Monday 28<sup>th</sup> October at 9.30 am**

*Delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. Students must plan their work to allow for both scheduled and unscheduled downtime. Penalties of 5% per day apply to late submissions (accepted only 3 working days after the due date as execution test marking will begin on Monday 28<sup>th</sup> October – in your normal lab (Week 13))*

**This is an individual Assignment. You are not permitted to work as a Pair Programming partnership or any other group when writing this assignment.**

**Copying, Plagiarism:** Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Computer Engineering treats academic misconduct seriously. When it is detected, penalties are strictly imposed. Refer to the subject guide for further information and strategies you can use to avoid a charge of academic misconduct.

**Assessment Objectives:**

- To design programs that conform to given specifications
- To implement classes and application programs in Java

**Submission Details:** Full instructions on how to submit electronic copies of your source code files from your latcs6 (or latcs5) account are given at the end. *If you have not been able to complete a program that compiles and executes containing all functionality, then you should submit a program that compiles and executes with as much functionality as you have completed. (You may comment out code that does not compile.)*

Note you must submit electronic copies of your source code files using the `submit` command on latcs6 or latcs5. Ensure you submit all required files, one file at a time. For example, the file `Rider.java` would be submitted with the command:

```
> submit OOF Rider.java
```

Do NOT use the LMS to submit your files, use latcs5/6 only

**PLEASE NOTE:** *While you are free to develop the code for this progress check on any operating system, your solution **must** run on the latcs6 (or latcs5) system.*

**Marking Scheme:** *This assignment is worth 13% of your final mark in this subject.*

*Implementation (Execution of code) 100%*

*You may be required to attend a viva voce (verbal) assessment (to be used as a weighting factor on the final mark).*

*Bonus questions 10% (A maximum of 100% can be obtained)*

## Return of Mark sheets:

Marking sheets will be returned on Tuesday 5<sup>th</sup> November (after your real-time programming exam).

## Problem Description

*Defence of Pern* (DOP) has asked you to write a program to manage their Riders and Dragons in their ongoing battle to defend Pern from the threat of Thread. This consists of a number of related tasks as detailed below.

### Task 1a – An Info Check Utility Class

*(Note: There may be better design decisions for this task but you are being asked to write a utility class for the purpose of practising class methods.)*

Each Rider of DOP has an information code that contains their personal information, formatted as follows (all letters in uppercase):

|   |   |   |
|---|---|---|
| M | } | F – female or M – male<br>three character Dragon identification code<br>(uppercase characters, will be unique)  |
| M |   |   |
| N |   |   |
| E |   |   |
| B | } | two character Dragon colour identification code<br>This corresponds to one of the enum values in the enum Colour.   |
| Z |   |   |
| 2 | } | one digit status level, (from 0 - 3, explained below)<br>two digit representation of the numbers of hours that the Rider has flown<br>on all tasks since their last break |
| 9 |   |   |
| 3 |   |   |

The status level values have the following meaning:

- 0: not trained, not available for any missions except **training** missions
- 1: trained, available for **transport and fall missions, not available for training missions.**
- 2: trained, available for **transport and fall missions, not available for training missions.**
- 3. unavailable for any missions (either injured or resting)

The DOP system requires the **optional** ability to check that an information code string is valid and the **required** ability to extract or update certain information contained in an information code.

Write a utility class in a file called `Info.java` that has separate class methods to perform the following functionality (assume all the String arguments are valid codes except if you implement the **bonus** optional method in Task 1b– in this task you are asked to check that the code argument is valid).

- A class method called `getGender()` that takes an information code as argument and returns a character 'M' or 'F' indicating the gender of the Rider.
- A class method called `getStatus()` that takes an information code as argument and returns the Rider's status (0, 1, 2, or 3) as an int.
- A class method called `getColour()` that takes an information code as argument and returns the corresponding String value of the 2 character colour code, which is one of the **enum** values from `Colour.java`.

- A class method called `getDragonCode( )` that takes an information code as argument and returns a String which is the 3 character part of the information code that represents the Dragon code.
  - A class method called `getHours( )` that takes an information code as argument and returns the number of hours part of the information code passed in, as an int.
  - A class method called `updateStatus( )` that takes an information code and a new status as arguments. This new status replaces the old status in the information code String that was passed in as an argument. The method returns the updated information code, as a String.
  - A class method called `resetHours( )` that takes an information code as an argument. The method sets the hours part of the code to 0 and returns the updated information code as a String.
  - A class method called `updateHours( )` that takes an information code and an int as arguments. The int is the additional number of hours to **add** to the current hours in the information code. If, after adding the additional hours, the number of hours is greater than 99, then the number of hours is set to 0 and the status is set to 3.
- Keep in mind that hours is represented by 2 characters, so, for example, setting the hours to 0 means having to set **both** characters to '0'

### Task 1b –optional for bonus marks

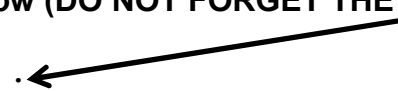
(Note: students should only try the bonus question if they have spare time – bonus marks up to 5% are added to your assignment mark but you cannot score more than 100%)

- A class method called `checkIdentity()` that takes an identity code as a String argument and returns a boolean value indicating whether the argument is a valid identity code (i.e. the right length and valid characters in each position).

The **Info** class and the other classes in this assignment make use of an enum, **Colour**, this has been fully coded for you. Some of the other parts of this assignment have also been coded for you.

**To start this assignment, make a separate directory in your latcs account and copy the files into that directory using the command below (DO NOT FORGET THE DOT)**

```
cp /home/1st/csilib/cse100f/assign3/* .
```



Note: this includes a **sample** text file named **a.dat** which you can use to carry out your program testing.

More classes needed by the DOP system are developed in the following tasks.

## Task 2 – A Rider Class

Details kept on each rider available to send on missions include a **name** and an information **code** (formatted as discussed above on page 2). A boolean **onMission** indicates whether the person is currently on a mission, a **missions** array stores the number of missions of the type *training*, *transport* and *fall* in the first, second and third elements of the array respectively and a **missType** array stores **true** or **false** to indicate the type of mission (if any) the Rider is on, with the mission type represented by *training*, *transport* and *fall* in the first, second and third elements of the array respectively.

Begin coding your **Rider** class, providing attributes as indicated in the class diagram. Then provide the following functionality for the **Rider** class:

- A constructor that takes a string name and string code (assumed valid) as arguments. Initialise **onMission** to false, set the elements of the **missions** array to 0 and set the elements of the **missType** array to **false**.
- Add another constructor that takes a string name and string code as arguments as well as a boolean **onMission**, three integer arguments to initialise the *training*, *transport* and *fall* elements (indexes 0, 1 and 2 respectively) of the **missions** array and three boolean arguments to initialise the *training*, *transport* and *fall* elements (indexes 0, 1 and 2 respectively) of the **missType** array.
- Provide accessor (get) methods for all four attributes, **a get method to return the dragon colour as a String and a get method to return the Rider status as an int.** (Consider privacy leaks with **getMissions ( )**.)
- The ability to update a name is NOT required and any changes to the code should be done by the instance methods **setMission( int missionIndex ), endMission( ), endMission( int missionIndex), updateHours( )** and **setStatus( int newStatus )** (these methods are explained below)

Use your utility class *Info.java* when you code these methods.

- Provide a mutator (set) method called **setMission( int missionType )** This method must check that the Rider is not already on a mission, that the **missionType** is valid (0, 1 or 2), the Rider's status is not 3 and that the **missionType** is appropriate for the Rider.

| Rider  |
|--|
| - String name<br>- String code<br>- boolean onMission<br>- int [ ] missions = new int [3]<br>- boolean [ ] missType = new boolean [ 3 ]  |
| + Rider(String name, String code)<br>+ Rider(String name, String code, boolean onMission, int training, int transport, int fall, boolean onTrainMiss, boolean onTransportMiss, boolean onFallMiss)<br>+ String getName()<br>+ String getCode()<br>+ boolean getOnMission()<br>+ int [ ] getMissions()<br>+ String getDragonColour ( )<br>+ int getStatus( )<br>+ boolean availForMission( )<br>+ void setMission( int missionType )<br>- void addMission( int missionType )<br>+ void endMission( )<br>+ void endMission( int missionType )<br>+ void updateHours( )<br>+ boolean availForMission( )<br>+ void setStatus( int newStatus )<br>+ String toString() |

(This means that a **missionType** of 0 (training) can only be undertaken by a Rider of status 0, a **missionType** of 1 (transport) or 2 (fall) can only be undertaken by a Rider with status of 1 or 2)

If any of these conditions are **not** met, then this method displays an appropriate message to the screen and nothing is changed.

If the Rider can undertake the mission then the attribute **onMission** is set to **true** and the private method **addMission** is called.

The **addMission** adds 1 to the total number of missions of that type that the Rider has undertaken and sets the corresponding index of the **missType** array to **true**.

- Add an **endMission** method that takes no arguments. This method checks that the Rider is on a mission and if the Rider is on a mission, then this method searches the **missType** array for the **true** value (there should only be one) and sets that value to **false**.
- Add an overloaded **endMission** method that takes an integer argument **type** (0 for *training*, 1 for *transport* and 2 for *fall*) and sets the appropriate element in the **missType** array to **false**. Note that this overloaded method can be called from the **updateHours** method described below
- Add an **updateHours** method that takes no arguments. If the Rider is on a mission then this method adds the following hours to the total hours that the Rider has flown without a break
  - 4 hours if the Rider is on a training mission ( **missType**[ 0 ] )
  - 6 hours if the Rider is on a transport mission ( **missType**[ 1 ] )
  - 2 hours if the Rider is on a fall mission ( **missType**[ 2 ] )

If this new number of hours is greater than 99, then this method does the following:  
sets the number of hours flown to 0  
ends the current mission of the Rider  
sets the Rider's status to 3 (unavailable)

**If the number of hours has changed (and possibly the status level), then this is updated in the code attribute.**

- Add an **availForMission( )** method. This method returns **true** if the Rider is not currently on a mission and Rider's status is not 3
- Add a **setStatus** method that takes an **int newStatus** as an argument. This method checks that the **newStatus** passed in is valid ( 0 - 3 ). Provided that the **newStatus** is valid then the method updates the status part of the **code** attribute.  
Note: checking other conditions for the **newStatus** is done in **class Weyr**
- Add a **toString()** method that returns a string containing all the current values of the attributes formatted as in the following example for a Rider named F'lar, with code MMNEBZ295, currently on a mission, with mission total values of 23, 56, 104 in the **missions** array and **false false true** in the **missType** array.

```
Rider[ name: F'lar is male
rides Bronze dragon MNE is currently on a mission
Mission type: Thread fall, is of status level: 2
currently has flown for 97 hours without a break
and has flown a total of
23 training missions, 56 transport missions, 104 fall missions ]
```

Before continuing to Task 3, you should run some tests in a driver to verify that your **Rider** class is correct.

-----

### Task 3 – Adding and displaying

A menu-based interactive driver program *Pern.java* and an array-based collection of Riders in *Weyr.java* have been partly coded.

*See the end of this assignment for skeleton code in Pern.java and the array-based collection of Rider objects in Weyr.java. Instructions for copying the files from the csilib area are given.*

You will have already copied all the required files into your assignment directory. Understand the code, compile and see how the menu works. The menu displays as follows and currently outputs messages that a selected option is not yet implemented.

#### Main Menu

```
a - Add Rider
s - Show All Riders
r - Read from file
m - Set mission
u - Update status
e - End Mission
q - Quit
    Enter choice >>
```

This menu repeatedly displays after each (case-insensitive) user selection, until the user chooses 'Q' or 'q' to quit the program.

The **Weyr** class is designed to contain an array of references to **Rider** objects as one of its attributes. You may assume no more than 1500 Riders are on the DOP system. Required functionality for this class includes a constructor (already coded). Write methods to:

#### In Weyr.java

- Add a new Rider to the array (with the new Rider and their code sent as parameters to the method). *You may assume the name of each Rider added is unique.*
- Display the details of all Riders in the array to screen. (You may use the `toString()` method of each **Rider** object.)

#### In Pern.java

- Option 'a' requires the user to be prompted to supply the rider's name (which could be more than one word) and their code. Then call the appropriate method in **Weyr** to add the rider.
  - Option 's' calls the appropriate method in **Weyr** to show details of all Riders' on the screen.
-

#### Task 4 – Reading from file

You are required to add more functionality to the **Weyr** class and then complete more of the **Pern** class.

- In the **Weyr** class write a method to do the following:

Given an input file name as parameter, append Riders from the input file into the array. You may assume that the file exists and contains the details (or record) for each Rider in 5 lines; that is, name, code, whether they are “*on a mission*” or “*available*”, a line containing three integers separated by a space (indicating total number of missions flown for *training*, *transport* and *fall*) and a line containing three boolean values separated by a space (indicating *true/false* for the mission type *training*, *transport* and *fall*).

A sample input file follows (*Note you can use this option any time during the program*):

```
F'lar
MMNEBZ295
true
23 56 104
false false true
Lessa
FRAMGD288
true
34 9 90
false false true
F'nor
MCANBN356
false
33 67 89
false false false
```

- Now in the **Pern** class implement menu option ‘r’:

Option ‘r’ prompts the user for the name of the file and calls the appropriate method in the **Weyr** class.

-----

#### Task 5 – Adding a mission, ending a mission, updating Rider status

- For all of the following tasks

You may wish to write a **search()** helper method in **Weyr**. It could return the index of the Rider or -1 if the Rider does not exist. The method heading could be:

```
private int search(String name)
```

- Adding a mission

In the **Weyr** class write a method to do the following:

Given a String parameter corresponding to a Rider’s name, an integer parameter (value of 0, 1 or 2 corresponding to *training*, *transport* or *fall*), update that Rider's mission status.

(Note: The **Weyr** class finds the appropriate index of the Rider in the array of Riders, then calls the **setMission** method of the Rider class for that particular Rider)

If the Rider does not exist, output an error message to screen. The method heading could be:

```
public void mission( String name, int type )
```

Now in the **Pern** class implement menu option 'm':

Option 'm' should prompt the user for the Rider's name and whether the mission type is *training*, *transport* or *fall*

The method that implements this menu option, in **Pern**, must check that the Rider is available for a mission and that the Rider can undertake that type of mission

Recall that a Rider with status 3 cannot undertake any missions, a Rider with status 0 can only undertake missions of type 0 (*training*) and *transport* or *fall* missions can only be undertaken by Riders with status 1 or 2.

- **Ending a mission**

In the **Weyr** class write a method to do the following:

Given a String parameter corresponding to a Rider's name, check that the Rider exists, and if they do, call the Rider method **endMission** to update the appropriate attributes in the Rider object.

(Note: Once again, the **Weyr** class finds the appropriate index of the Rider in the array of Riders, then calls the **endMission** method of the Rider class for that particular Rider)

If the Rider does not exist, output an error message to screen. The method heading (in **Weyr.java** ) could be:

```
public void endMission( String name )
```

Now in the **Pern** class implement menu option 'e':

Option 'e' should prompt the user for the Rider's name and call the appropriate method in **Weyr** class, after first checking that the Rider is in fact on a mission.

- **Updating the status**

In the **Weyr** class write a method to do the following:

Given a String parameter corresponding to a Rider's name and an int parameter corresponding to the new status, update the status of the Rider.

Once again, the **Weyr** class finds the appropriate index of the Rider in the array of Riders. If the Rider does not exist, output an error message to screen.

This method checks the current status of the Rider. If the Rider already has the same status as the new status passed in, then this method returns a String with the appropriate message.



If the Rider currently has a status of 1 or 2, then they cannot be given a new status of 0, neither can a Rider with a current status of 2 be given a new status of 1, nor can a Rider with a current status of 0 be given a new status of 2, they can only be given a new status of 1.

If any of these above error conditions are `true` the method returns a String with the appropriate message.

**Regardless of the conditions above, any Rider can be given a status of 3, it is assumed that a Rider with current status of 3 is being returned to their previous correct status.** (The previous status does NOT have to be stored, assume that the user knows the previous status)

Provide that all of the above conditions are met, then this method calls the `setStatus` method of the Rider class for that particular Rider)

The method heading (in `Weyr.java` ) could be:

```
public String setStatus( String name, int newStatus )
```

Now in the `Pern` class implement menu option 'u':

Option 'u' should prompt the user for the Rider's name and the new status, call the appropriate method in `Weyr` class and **display the String that is returned**.

## Task 6 – Updating the hours

- In the `Weyr` class write a method that updates the hours of all Riders currently on a mission. This method calls the `updateHours` method of the Rider class for each Rider on a mission.

The method header could look like this

```
public void updateHours( )
```

- In the `Pern` class

write a method that will call the above `updateHours` method in the `Weyr` class.

This method to be written in the `Pern` class, will be called every time through the loop, **except when the user selects the quit option OR makes an invalid menu choice.**

---

## CSE400F Task 7 – Optional bonus for CSE100F

- In the above specifications, Riders with status 0 may only missions of type 0, that is, *training* missions. Riders of status 1 or 2 may undertake any *transport* or *fall* mission. **Status 1 or 2 DOES NOT correspond to mission type 1 or 2.**

Extend the program so that when a Rider is to assigned to a *fall* mission, the user is asked for another integer. This integer is the level of the *fall*, either 1 or 2. A Rider with status 2 may be assigned a *fall* mission of either type 1 or 2, but a Rider with status 1 may only be assigned a *fall* mission of type 1.

## CSE4OOF Task 8 – Optional bonus for CSE1OOF

- Ensure the names are unique, whether added interactively or from a file. If a name is not unique, do not add that person
- 

### Optional Tasks –

*Please note: This optional work should NOT be attempted until all other tasks have been successfully completed. It should only be attempted if you have time after attending lectures, labs and doing assignments in all your other subjects.*

*Good attempts at these extensions are worth up to an additional 10%. However, it is not possible to end up with more than 100% in your assignment mark.*

#### Optional Task 1:

- Redesign menu option 's' so that a submenu is displayed giving options for how to show Riders on the screen. Allow options that:
  - Sort all Riders by name so that they are displayed in alphabetical order
  - Sort all Riders by gender with a sub sort on name
  - Sort all Riders by Dragon colour, in the order Gold, Bronze, Brown, Blue then Green with, or without, a sub sort on name
  - Ask the user for a mission type (*training*, *transport* or *fall*) and only display those Riders who are available for that type of mission.

#### Optional Task 2:

- Menu option '?'  
Design a two- dimensional board game to amuse the user during their rest time. For example, a player could start in the top left hand corner of the board and proceed left to right along each row, until the bottom right-hand corner is reached. Certain board positions could be lucky or unlucky in some way.

Display the board, player(s), and other information using characters output to screen, and redisplay the board after each move. Perhaps the ability to stop playing part way through the game (to return to work) would be useful (without saving the game state).

On completion of the game, the user is returned to the main menu.

**Note** `Math.random( )` is a useful method for generating random numbers.

---

### Electronic Submission of the Source Code

- Submit all the Java files that you have developed in the tasks above.
- The code has to run under Unix on the latcs6 (or latcs5) machine.
- You submit your files from your latcs6 (or latcs5) account. Make sure you are in the same directory as the files you are submitting. Submit each file separately using the **submit** command. For example:

```
submit OOF Colour.java
submit OOF Info.java
submit OOF Rider.java
submit OOF Weyr.java
submit OOF Pern.java
```

After submitting the files, you can run the following command that lists the files submitted from your account:

```
verify
```

You can submit the same filename as many times as you like before the assignment deadline; the previously submitted copy will be replaced by the latest one.

---

### Skeleton Code (to start your answer)

Copy the following two file skeletons and a sample text input file from the library area into your Assignment 3 directory, using the following command (when in that directory):

```
cp /home/1st/csilib/cseloof/assign3/* .
```



Don't forget the DOT

```
/*
 * Class Name:      Weyr
 */
import java.util.Scanner;
import java.io.File;
import java.io.IOException;

public class Weyr
{
    private final int SIZE = 1500;
    private Rider [ ] riders;
    private int count;

    public Weyr( )
    {
        riders = new Rider[ SIZE ];
        count = 0;
    }

    public void addRider( Rider newRider )
    {
        // to be implemented
    }
}
```

```
-----
/*
 * Class Name:      Pern
 */
import java.util.Scanner;
import java.io.IOException;

public class Pern
{
    private Scanner kb = new Scanner( System.in );
    private Weyr benden;

    public Pern( )
    {
        benden = new Weyr( );
    }
}
```

```

private void run( ) throws IOException
{
    char choice;
    do
    {
        menu( );
        choice = kb.nextLine( ).toLowerCase( ).charAt( 0 );
        switch( choice )
        {
            case 'a':
                addRider( );
                break;

            case 's' :
                System.out.println("To be implemented");
                break;

            case 'm' :
                System.out.println("To be implemented");
                break;

            case 'u' :
                System.out.println("To be implemented");
                break;

            case 'e' :
                System.out.println("To be implemented");
                break;

            case 'r' :
                System.out.println("To be implemented");
                break;

            case 'q':
                System.out.println("To be implemented");
                break;

            default:
                System.out.println("Invalid choice");
                break;
        }
    }while( choice != 'q' );
}

private void menu( )
{
    System.out.println( "\n\tMain Menu\n");
    System.out.println( "\t a - Add Rider");
    System.out.println( "\t s - Show All Riders ");
    System.out.println( "\t r - Read from file");
    System.out.println( "\t m - Set mission");
    System.out.println( "\t u - Update status");
    System.out.println( "\t e - End Mission");
    System.out.println( "\t q - Quit");
    System.out.print( "\t\tEnter choice >> ");
}

```

```

private void addRider( )
{
    System.out.print("To be implemented");
    /*
     * Ask the user for the name and code of a Rider
     * Create the Rider object
     * call benden.addRider( ... );
     *
     */
}

public static void main( String [ ] args ) throws IOException
{
    Pern p = new Pern( );
    p.run( );
}
}

```