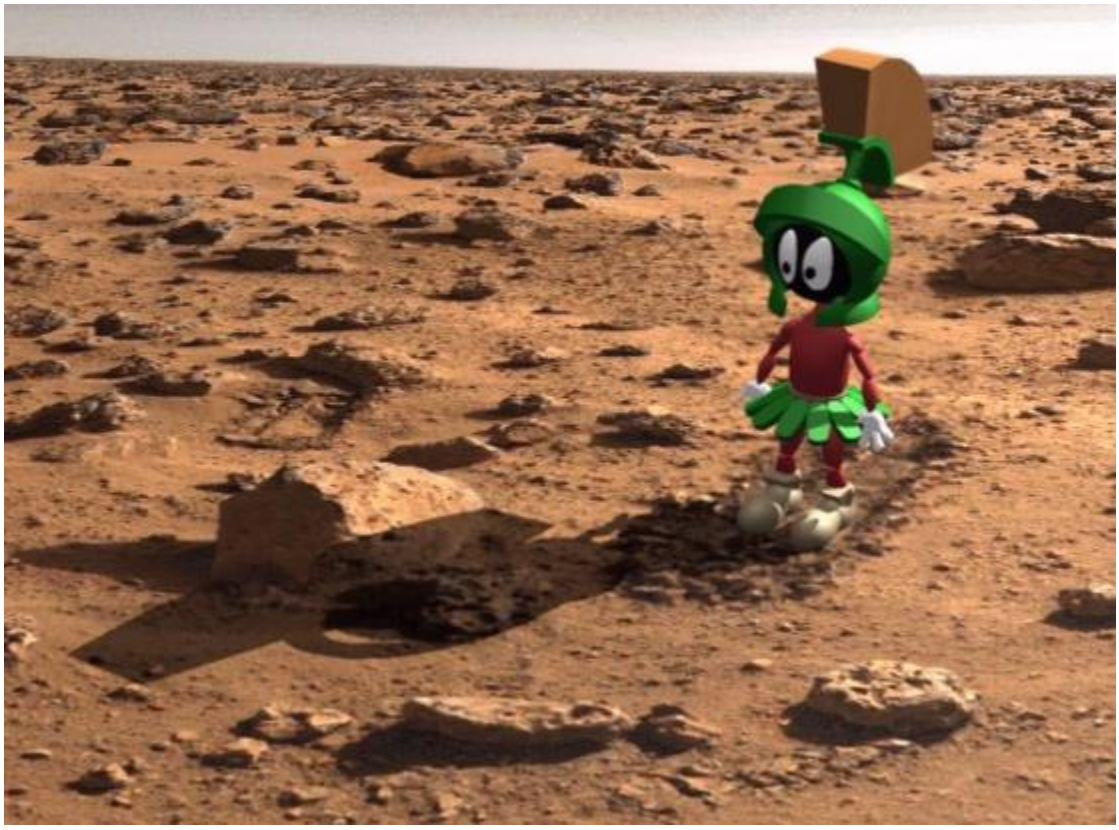
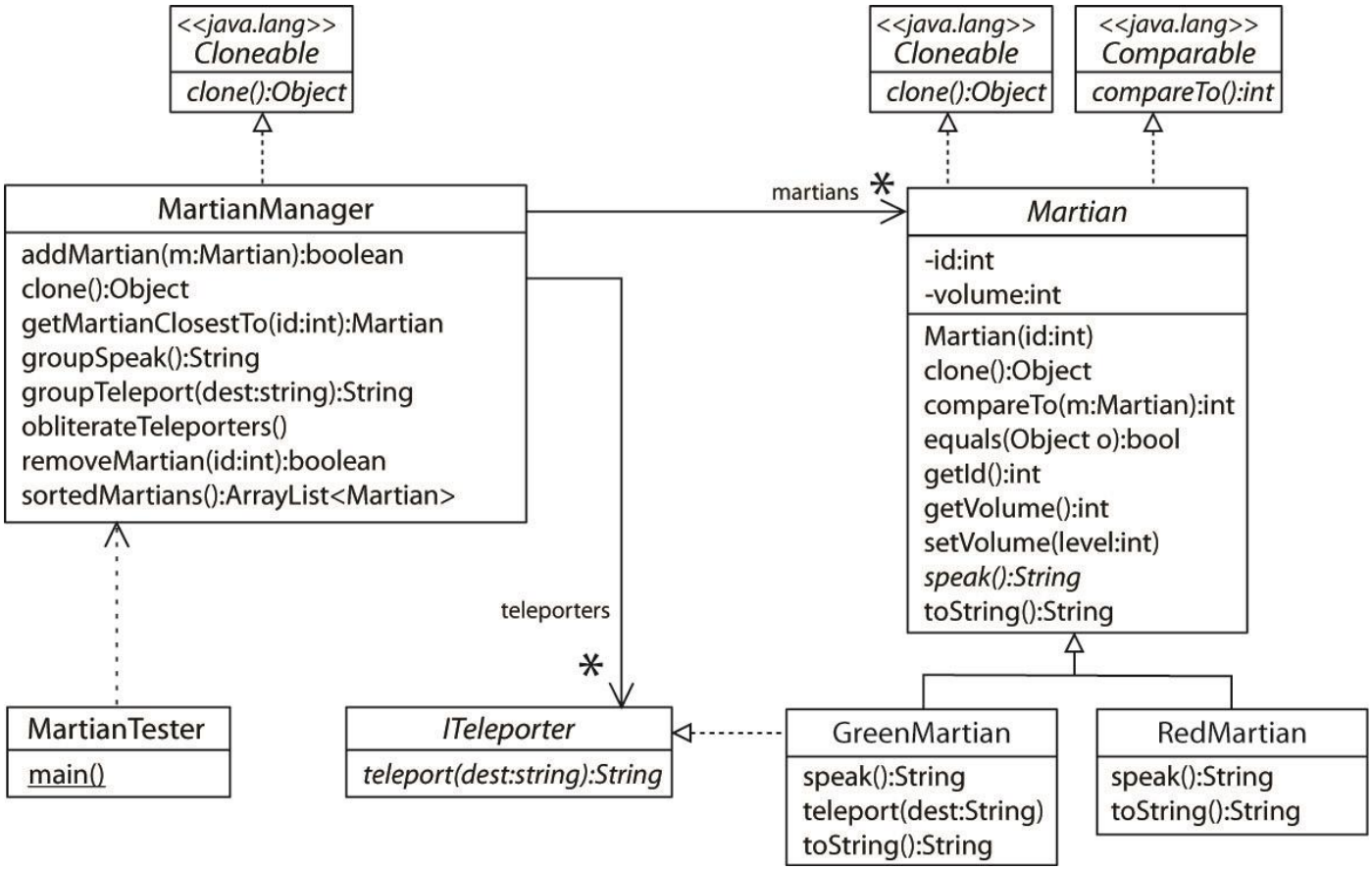
CS 1302A (Spring 2016)
Grand Challenge Project
Green Martian and Red Martian

From this project, you will learn how to apply what you were taught in classroom so far to solve real-world problems. You will gain hands-on experience on how to code using concrete class, abstract class, interface, and inheritance for problem solving. This project covers (almost all) materials from chapters 10, 11 and 13. Please read carefully to understand the problem. You are strongly encouraged to try it on your own before we work on it together in class. That is the way you can get the most out of this project!



Problem Description

Consider the class diagram shown below. Write all six classes/interfaces (Martian, GreenMartian, RedMartian, MartianManager, ITeleporter, MartianTester). Details follow below the class diagram.



1. Martian Class:

    a. Martian(id:int) – Creates a Martian with an *id* and sets the *volume* to 1.
    b. clone():object – Creates and returns a clone of the martian.
    c. compareTo(Martian m):int – implements Java's compareTo so that Martians (either red or green) are compared based on their *id*'s which results in the usual ascending sorted order.
    d. equals(o:Object):bool – Two martians are equal if their *id*'s are equal, regardless of whether they are Green or Red.
    e. getId():int – Returns the *id*.
    f. getVolume():int – Returns the *volume.*
    g. setVolume(level:int) – Sets the *volume* to *level*.
    h. *speak():string* – Abstract.
    i. toString():string – Returns a string like: "Martian id=xxx vol=yyy", where *xxx* is the *id* and *yyy* is the volume.

2. RedMartian Class:

    a. speak():string – Returns a string like: "id=xxx, Rubldy Rock", where *xxx* is the *id*.
    b. toString():string – Returns a string like: "Red Martian id=xxx vol=yyy"

3. GreenMartian Class:

    a. speak():string – Returns a string like: “id=xxx, Grobldy Grock”, where *xxx* is the *id*.
    b. teleport(dest:string):string – The GreenMartian implements the *teleport* method by returning a string like this: “id=xxx teleporting to *dest*”
    c. toString():string – Returns a string like: “Green Martian id=xxx vol=yyy”

4. MartianManager Class:

    a. martians – an arraylist of all martians, Green and Red.
    b. teleporters – an arraylist of all martians that implement the ITeleport interface.
    c. addMartian(m:Martian):boolean – adds *m* (either red or green) to *martians*, only when it doesn’t already exist in the list of *martians*. If *m* is a GreenMartian (and unique), then it is also added to the *teleporters* list. If the add is successful, then return true, otherwise return false.
    d. *clone():Object – Creates a clone of the MartianManager class. Warning: this requires a deep copy.
    e. getMartianClosestToId(id:int):Martian – Return the martian that has *id* closest to the input *id*. Note: “closest” doesn’t mean the *next* martian. For example, if you have *ids:* 6, 9, 13, 18 and you call the method with 10, then the martian with *id=9* should be returned.
    f. groupSpeak():String – Returns a string with each martian speaking (line break between each).
    g. groupTeleport(dest:string):String – Returns a string showing the result of all *teleporters* teleporting to the *dest* (line break between each).
    h. *obliterateTeleporters() – Remove all *teleporters* and the corresponding GreenMartians in *martians.*
    i. *removeMartian(id:int):boolean – Remove the martian with *id*. Must also remove from *teleporters* if it is a GreenMartian. Martians are unique so there is at most 1 to remove. This method should return *true* if remove was successful, and *false* if not found. Hint: (1) Create a “dummy” GreenMartian with the *id*. (2) use *indexOf* to get index of Martian, (3) use *get* to get the Martian, save him for a minute, (4) use *remove* to remove from *martians,* (5) if martian from step 3 is a GreenMartian then remove from *teleporters*.
    j. sortedMartians(): ArrayList<Martian> - Return a list of martians sorted on their *id*’s. This should **not** sort the *martians.* Hint: clone *martians* and then sort the clone using a predefined method in Java API *Collections.sort(). Collections.sort() takes an ArrayList instance as argument and sort all the elements in that list, based on the compareTo() method defined on each element.*

        * These are challenging!

5. MartianTester Class:

   a. main() – Use the main() below to test your code.

```
public static void main(String[] args) throws CloneNotSupportedException {

        MartianManager mm = new MartianManager();

        RedMartian r1 = new RedMartian(8);
        RedMartian r2 = new RedMartian(18);
        RedMartian r3 = new RedMartian(2);

        GreenMartian g1 = new GreenMartian(11);
        GreenMartian g2 = new GreenMartian(4);
        GreenMartian g3 = new GreenMartian(10);
        GreenMartian g4 = new GreenMartian(2);

        // Test 1 - Martian.toString()
        System.out.println("***Test 1 - Martian.toString()");
        System.out.print(r1 + ", ");
        System.out.println(g1 + "\n");

        // Test 2 - Martian.equals()
        System.out.println("***Test 2 - Martian.equals()");
        System.out.println("r3.equals(g1)= " + (r3.equals(g1)));
        System.out.println("r3.equals(g4)= " + (r3.equals(g4)) + "\n");

        // Test 3 - Martian.compareTo()
        System.out.println("***Test 3 - Martian.compareTo()");
        System.out.println("r3.compareTo(g1)= " + (r3.compareTo(g1)));
        System.out.println("r3.compareTo(g4)= " + (r3.compareTo(g4)));
        System.out.println("r2.compareTo(g3)= " + (r2.compareTo(g3)) + "\n");

        // Test 4 - Martian.clone()
        System.out.println("***Test 4 - Martian.clone()");
        Martian m = new RedMartian(83);
        m.setVolume(44);
        Martian mClone = (Martian)m.clone();
        mClone.setVolume(77);
        System.out.print("m.getVolume()= " + m.getVolume() + ", ");
        System.out.print("mClone.getVolume()= " + mClone.getVolume() + "\n\n");

        // Test 5 - MartianManager.addMartian()
        System.out.println("***Test 5 - MartianManager.addMartian()");

        System.out.print(mm.addMartian(r1) + ", ");
        System.out.print(mm.addMartian(r2) + ", ");
        System.out.print(mm.addMartian(r3) + ", ");

        System.out.print(mm.addMartian(g1) + ", ");
        System.out.print(mm.addMartian(g2) + ", ");
        System.out.print(mm.addMartian(g3) + ", ");
        System.out.println(mm.addMartian(g4) + "\n");

        // Test 6 - MartianManager.groupSpeak()
        System.out.println("***Test 6 - MartianManager.groupSpeak()");
```

```
        System.out.println( mm.groupSpeak() );

        // Test 7 - MartianManager.groupTeleport()
        System.out.println("***Test 7 - MartianManager.groupTeleport()");
        System.out.println( mm.groupTeleport( "Orck") );

        // Test 8 - MartianManager.getMartianClosestToID()
        System.out.println("***Test 8 - MartianManager.getMartianClosestToID()");
        System.out.println( "closest to 7 is: " + mm.getMartianClosestToId(7) );
        System.out.println( "closest to 12 is: " + mm.getMartianClosestToId(12) + "\n" );

        // Test 9 - MartianManager.removeMartian()
        System.out.println("***Test 9 - MartianManager.removeMartian()");
        System.out.print( mm.removeMartian(4) + ", " );
        System.out.print( mm.removeMartian(18) + ", " );
        System.out.println( mm.removeMartian(99) );
        System.out.println( mm.groupSpeak() );
        System.out.println( mm.groupTeleport( "Orck") );


        // Test 10 - MartianManager.sortedMartians()
        System.out.println("***Test 10 - MartianManager.sortedMartians()");
        ArrayList<Martian> sortedMartians = mm.sortedMartians();
        System.out.println("sorted:");
        System.out.println( sortedMartians );
        System.out.println("original order:");
        System.out.println( mm.groupSpeak() );


        // Test 11 - MartianManager.obliterateTeleporters()
        System.out.println("***Test 11 - MartianManager.obliterateTeleporters()");
        mm.obliterateTeleporters();

        System.out.println( mm.groupSpeak() );
        System.out.println( mm.groupTeleport( "Orck") );

        // Test 12 - MartianManager.clone()
        System.out.println("***Test 12 - MartianManager.clone()");
        g1 = new GreenMartian(51);
        System.out.print(mm.addMartian(g1) + "\n");
        MartianManager mmClone = (MartianManager)mm.clone();
        System.out.print( mmClone.removeMartian(8) + "\n" );
        System.out.println( "Cloned MartianManager-with id=8 removed" );
        System.out.println( mmClone.groupSpeak() );
        System.out.println( "Original MartianManager" );
        System.out.println( mm.groupSpeak() );

        System.out.println( "Get id=51 from mm, change vol to 999" );
        Martian m2 = mm.getMartianClosestToId(51);
        m2.setVolume(999);
        System.out.println( "mm, should see 999" );
        System.out.println( mm.sortedMartians());
        System.out.println( "mmClone, should not see 999" );
        System.out.println( mmClone.sortedMartians());

}
```

Expected Result:

Your code should print out the following result:

```
***Test 1 - Martian.toString()
Red Martian id=8 vol=0, Green Martian id=11 vol=0

***Test 2 - Martian.equals()
r3.equals(g1)= false
r3.equals(g4)= true

***Test 3 - Martian.compareTo()
r3.compareTo(g1)= -9
r3.compareTo(g4)= 0
r2.compareTo(g3)= 8

***Test 4 - Martian.clone()
m.getVolume()= 44, mClone.getVolume()= 77

***Test 5 - MartianManager.addMartian()
true, true, true, true, true, true, false

***Test 6 - MartianManager.groupSpeak()
8 Rubldy Rock
18 Rubldy Rock
2 Rubldy Rock
11 Grubldy Grock
4 Grubldy Grock
10 Grubldy Grock

***Test 7 - MartianManager.groupTeleport()
11 Teleporting to Orck
4 Teleporting to Orck
10 Teleporting to Orck

***Test 8 - MartianManager.getMartianClosestToID()
closest to 7 is: Red Martian id=8 vol=0
closest to 12 is: Green Martian id=11 vol=0

***Test 9 - MartianManager.removeMartian()
true, true, false
8 Rubldy Rock
2 Rubldy Rock
11 Grubldy Grock
10 Grubldy Grock

11 Teleporting to Orck
10 Teleporting to Orck

***Test 10 - MartianManager.sortedMartians()
```

```
sorted:
[Red Martian id=2 vol=0, Red Martian id=8 vol=0, Green Martian id=10 vol=0, Green Martian id=11 vol=0]
original order:
8 Rubldy Rock
2 Rubldy Rock
11 Grubldy Grock
10 Grubldy Grock

***Test 11 - MartianManager.obliterateTeleporters()
8 Rubldy Rock
2 Rubldy Rock


***Test 12 - MartianManager.clone()
true
true
Cloned MartianManager-with id=8 removed
2 Rubldy Rock
51 Grubldy Grock

Original MartianManager
8 Rubldy Rock
2 Rubldy Rock
51 Grubldy Grock

Get id=51 from mm, change vol to 999
mm, should see 999
[Red Martian id=2 vol=0, Red Martian id=8 vol=0, Green Martian id=51 vol=999]
mmClone, should not see 999
[Red Martian id=2 vol=0, Green Martian id=51 vol=0]
```

Suggested Steps to Complete is on the next page!

Try It on Your Own: Suggested Steps to Complete

1. Code Martian class with only these members: id, vol, constructor, getId, getVolume, setVolume, speak (abstract), and toString.
2. Code RedMartian class completely: constructor, speak method, and toString.
3. Write a driver to test.
4. Code ITeleporter interface.
5. Code GreenMartian class completely: constructor, speak, teleport, toString
6. Write a driver to test:
   a. Create an arrayList of Martians, put a few in, iterate over calling speak
   b. Create an arrayList of ITeleporters, put a few in, iterate over calling teleport
7. Override the equals method in the Martian class.
8. Write a driver to test:
   a. Create two red martians with the same ID and test equals, *e.g. m1.equals(m2)*. Then try two with different IDs. Then try with a red and green.
   b. Put a few martians in an array list of Martians. Then, create a martian with the same ID as one in the array list. Use either the indexOf or contains method of the array list to see whether the array list already contains a martian with the same ID.
9. Code the compareTo method in the Martian class. Don't forget to implement the Comparable interface.
10. Write a driver to test:
    a. Create two red martians with different ID's and test compareTo, *e.g. m1.compareTo(m2)*. Then try with a red and green.
    b. Create an array list of martians, put in a few with random ID's. Sort and print the array to show that it is sorted.
11. Code the clone method in the Martian class. Don't forget to implement the Cloneable interface.
12. Write a driver to test: Create a red (or green) martian, set the volume, clone it, set the volume of the clone to a different value, then print both martians out and verify that their ID's are the same, but have different volumes.
13. Write the MartianManager class this way:
    a. Add the instance variable to hold the arraylist of martians
    b. Write the addMartian method to simply add the input martian (in other words, don't check for unique ID yet, nor deal with the list of teleporters). Don't forget to return a Boolean (true, since we are not doing any checking yet)
    c. Write the groupSpeak method completely.
14. Write a driver to test:
    a. Create the martian manager
    b. Create red and green martians adding them using the add method
    c. Call the groupSpeak method and print the result verifying that things are correct.
15. Modify addMartian to make sure not to add a martian with the same ID as one that already exists. Hint: see 8b above. Be sure and return false if you detect a duplicate ID.
16. Write a driver to test:
    a. Create the martian manager
    b. Create red and green martians adding them using the add method.
    c. Try to add one with a duplicate ID. Verify that the return is false.
    d. Call the groupSpeak method and print the result verifying that no martians have the same ID.
17. Add the array list of ITeleporters to the martian manager class.
18. Modify the addMartian method to add green martains to the list of teleporters. Hint: How do you tell if a martian is Green?
19. Write groupTeleport method
20. Write a driver to test: add some red and green, call groupTeleport

21. Write and test the getMartianClosestToId method
22. Write any of the other methods not written
23. Write the MartianTest class and put in the supplied tester code, commenting out what you have not implemented. The tester code includes 12 tests. Use the printout result to verify your code.