

Started on	Sunday, 17 March 2019, 5:47 PM
State	Finished
Completed on	Monday, 18 March 2019, 3:25 PM
Time taken	21 hours 37 mins
Marks	5.87/7.00
Grade	8.38 out of 10.00 (84%)

Information

## General Lab Information

To begin with, we are going to look more in depth at working with multiple classes, including visibility and equality. Then we'll look into how polymorphism is implemented in Java with inheritance and interfaces. Polymorphism is the idea where objects of several types can be accessed through the same interface. This is useful, as when we try to "model the real world", many objects in the world adhere to the the same interface.

The goals for this lab are:

- To understand visibility and equality
- To understand what polymorphism is and why we need it
- What an interface is, and when to use it
- How to create an interface
- What inheritance is, and when to use it
- How to create a subclass, and to assign to superclasses
- When to choose inheritance and when to choose an interface
- Abstract classes and abstract methods

### Response history

Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	17/03/19, 17:52	Seen	
3	18/03/19, 15:25	Attempt finished	

## Visibility

Let's talk about visibility. You probably see those public, private and protected tags floating around, and know that all the lab code so far has been public. But just what are public, private and protected even used for?

They are used to determine how "visible" an attribute or a method is, from another class, or even another instance.

Lets have an example. Say the Astronaut class had a int money; attribute, and Starman has a balance of 300. If we do what we have always done and placed public before it, to get

```
public int money;
```

Then any class or instance could come along and take Starman's money! A GreenAlien could simply do

```
public class GreenAlien {
    public int alienMoney;
    ...
    public void takeMoney(Astronaut astronaut) {
        stolenMoney = astronaut.money - 100;
        alienMoney += 100;
    }
    ....
}
```

And then the GreenAlien can go and spend Starman's money on even more candy! But how can we stop this? We need to look into different visibility tags.

- There is *public*: Public allows other classes to freely access and directly modify public attributes/methods of the class.
- *Protected*: Protected gives access to the current class, as well as classes inside the same package and any subclasses, regardless of where they are located (in the same or a different package).
- *No tag (package private)*: Not giving a tag is different from public and protected. It allows access to the current class, classes in the same package and the subclass, *only if* the subclass is located within the same package.
- *Private*: Like the name implies, private restricts access to only the class itself, and not any others. Private allows access to only the current class to modify attributes and call methods. Different classes (i.e. subclasses, and classes in other packages) won't be able to access attributes or methods.

This can be summarised in this table:

	Class	Package	Subclass	Subclass	World
			(same pkg)	(diff pkg)	
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				
+ : accessible					
blank : not accessible					

Now, when we are marking our class attributes and methods with a visibility tag, how do we decide what visibility everything should be?

**A good rule of thumb is to restrict visibility as much as possible.** No one ever got hurt because their access was too restricted right?

**This means that normally, all of your attributes will be private. Get into the habit of making attributes private by default from now on.**

But what about methods? Should all my methods be private too? Well, no. If all of your methods were private, then classes wouldn't be able to call any methods, and you would end up with a useless class. **Instead, think about the functionality you want to reveal to the outside. If a method is to be used outside the class, by other classes, then use the lowest visibility that will be appropriate** (i.e. in order of least visible to most: private, package-private, protected, public). If the method is only called from inside the class, and is rather useless when called from from other places, make that method private.

Response history			
Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	17/03/19, 17:52	Seen	
3	18/03/19, 15:25	Attempt finished	

Question **1**



Correct

Mark 0.17 out of 1.00

## GreenAlien visibility

Go ahead and change the attributes in your GreenAlien class to private, and indicate what happens in the boxes below.

Select one or more:

- ☒ a. When I change the class itself to private, I get a compile error.  You sure do. Can't have the main class as private right?
- ☒ b. Nothing changed, and the class still works as before. 
- ☐ c. The class changed and I can no longer use kloup.name = "Kloup"; in the main() method of GreenAlien
- ☐ d. When I change the class itself to private, everything still works.

Your answer is correct.

The correct answers are: Nothing changed, and the class still works as before., When I change the class itself to private, I get a compile error.

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.17/1.00**.

Response history					
Step	Time	Action	State	Marks	
<a href="#">1</a>	17/03/19, 17:47	Started	Not complete		
<a href="#">2</a>	17/03/19, 17:52	Submit: The class changed and I can no longer use kloup.name = "Kloup"; in the main() method of GreenAlien	Incorrect	0.00	
<a href="#">3</a>	17/03/19, 17:53	Submit: When I change the class itself to private, I get a compile error.	Partially correct	0.17	
<a href="#">4</a>	17/03/19, 17:54	Submit: When I change the class itself to private, I get a compile error. ; When I change the class itself to private, everything still works.	Incorrect	0.17	
<a href="#">5</a>	17/03/19, 17:54	Submit: When I change the class itself to private, I get a compile error. ; Nothing changed, and the class still works as before.	Correct	0.17	
6	18/03/19, 15:25	Attempt finished	Correct	0.17	

## Getters and Setters

So we have just made our class attributes private. Just how do we then set the values of our attributes now? How can other classes set values? How do we read values if we need to access them for a calculation from another class?

The answer is getter and setter methods. These are public methods that control access to reading and writing values of your attributes. This prevents a classes attributes from being modified in ways that you don't want them to be.

An example of basic getter and setter methods are:

```
public class Astronaut {
    // ...
    private int age;
    // ...

    public int getAge() {
        return age;
    }

    public void setAge(int newAge) {
        age = newAge;
    }

    // ...
}
```

Lets say we want to check that the caller is allowed to access the age of an astronaut. It might be top secret, so we have to check that the caller is allowed to access top secret things before we tell them. Or when we set the age of an astronaut, maybe astronauts are only allowed to be between 18-35.

Which would result in the class looking like this:

```
public class Astronaut {
    // ...
    /**
     * The astronauts age
     */
    private int age;
    // ...

    /**
     * Returns the astronauts age after checking access.
     * Only top-secret people can see the age.
     */
    public int getAge() {
        checkTopSecret();
        return age;
    }

    /**
     * Sets the age, only if the newAge is between 18 and 35
     * @param newAge The age of the astronaut
     */
    public void setAge(int newAge) {
        if (newAge >= 18 && newAge <= 35) {
            age = newAge;
        }
    }

    // ...
}
```

### Response history

Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	17/03/19, 17:52	Seen	
<b>3</b>	<b>18/03/19, 15:25</b>	<b>Attempt finished</b>	

## Equality

Sometimes its nice to be able to test for object equality. Say we have multiple astronauts, and there are possible duplicates in our pool of Astronauts. How do we find them?

If you try `==`, Java will return true if the two variables you are comparing reference the same instance of a variable. Much like something like this:

```
Astronaut person1 = new Astronaut();
Astronaut person2 = person1;
Astronaut person3 = new Astronaut();
...
if (person1 == person2) // will be true
...
if (person1 == person3) // will be false
```

person1 and person2 both refer to the same instance of an Astronaut object. Person1 and person3 however, refer to different instances, since person3 was a new instance of astronaut.

This is a little problematic, since person1 and person3 are both the "same" as in, their names are the same, their ages are the same, and their missions are the same.

Instead, to see object equality, we create a `.equals(Object object)` method.

Inside the `equals()` method, we can write various tests and return "true" if the objects are the same, or "false" if they are different.

Heres an example:

```
public class Astronaut {

    public String name;
    public int age;
    public String favouriteFood;
    public String spaceMission;

    public Astronaut() {
        name = "Starman";
        age = 23;
        favouriteFood = "Burgers";
        spaceMission = "0000 space exploration";
    }

    public Astronaut(String tempName, int tempAge, String tempFood, String tempMission) {
        name = tempName;
        age = tempAge;
        favouriteFood = tempFood;
        spaceMission = tempMission;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public String getFavouriteFood() {
        return favouriteFood;
    }

    public String getSpaceMission() {
        return spaceMission;
    }

    public boolean equals(Astronaut other) {
        if (this.name == other.getName() &&
            this.age == other.getAge() &&
            this.favouriteFood == other.getFavouriteFood() &&
            this.spaceMission == other.getSpaceMission()) {

            return true;
        }
        return false;
    }

    public String toString() {
        return "Hi, my name is " + name + " and I am " + age + " years old. " +
            "My favourite food is " + favouriteFood + " and I am on mission "
            + spaceMission + ".";
    }

    public static void main(String[] args) {
        Astronaut spaceman = new Astronaut("Spaceman", 20, "Hash browns", "0001 to Mars");
        Astronaut spacegirl = new Astronaut("Spacegirl", 21, "Pizza", "0002 to the Moon");
        Astronaut spacedog = new Astronaut("Spaceman", 20, "Hash browns", "0001 to Mars");

        System.out.println(spaceman.equals(spacegirl));
        System.out.println(spaceman.equals(spacedog));
    }
}
```

With the results:

```
false
true
```

Response history			
Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	17/03/19, 17:52	Seen	
3	18/03/19, 15:25	Attempt finished	

Question **2**

Correct

Mark 1.00 out of 1.00

## GreenAlien equality

Add an equals() method to your GreenAlien class which implements:

boolean equals(GreenAlien other), which checks equality with other GreenAliens.

For example:

Test	Result
GreenAlien kloup = new GreenAlien(); GreenAlien lesap = new GreenAlien(); GreenAlien gwerp = new GreenAlien("Gwerp", 4, "Marshmallows"); GreenAlien blarg = new GreenAlien("Kloup", 3, "Pop Rocks");  System.out.println(kloup.equals(lesap)); System.out.println(gwerp.equals(lesap)); System.out.println(kloup.equals(blarg));	true false false

Answer: (penalty regime: 0, 10, 20, ... %)

```
1 public class GreenAlien {
2     public String name="";
3     public int eyeCount=0;
4     public String favouriteCandy="";
5
6     public GreenAlien() {
7         name="Kloup";eyeCount=6;favouriteCandy="Lollypops";
8     }
9     public GreenAlien(String tempName, int tempEye, String tempCandy) {
10         name=tempName;eyeCount=tempEye;favouriteCandy=tempCandy;
11     }
12
13     public boolean equals(GreenAlien other) {
14         if (this.name == other.name && this.eyeCount == other.eyeCount && this.favouriteCandy == other.favouriteCandy) {
15
16             return true;
17         }
18         return false;
19     }
20
21     public String toString() {
22         return name + " " + eyeCount + " " + favouriteCandy;
23     }
24 }
```

	Test	Expected	Got	
✓	GreenAlien kloup = new GreenAlien(); GreenAlien lesap = new GreenAlien(); GreenAlien gwerp = new GreenAlien("Gwerp", 4, "Marshmallows"); GreenAlien blarg = new GreenAlien("Kloup", 3, "Pop Rocks");  System.out.println(kloup.equals(lesap)); System.out.println(gwerp.equals(lesap)); System.out.println(kloup.equals(blarg));	true false false	true false false	✓

Passed all tests! ✓

Question author's solution:



```
public class GreenAlien {
    private String name;
    private int eyeCount;
    private String favouriteCandy;

    public GreenAlien() {
        name = "Kloup";
        eyeCount = 6;
        favouriteCandy = "Lollypops";
    }

    public GreenAlien(String tempName, int tempEye, String tempCandy) {
        name = tempName;
        eyeCount = tempEye;
        favouriteCandy = tempCandy;
    }

    public String getName() {
        return name;
    }

    public void setName(String newName) {
        if (newName.length() == 5) {
            name = newName;
        }
    }

    public int getEyeCount() {
        return eyeCount;
    }

    public void setEyeCount(int newEyes) {
        eyeCount = newEyes;
    }

    public String getFavouriteCandy() {
        return favouriteCandy;
    }

    public void setFavouriteCandy(String newCandy) {
        favouriteCandy = newCandy;
    }

    public boolean equals(GreenAlien other) {
        if (this.name == other.getName() &&
            this.eyeCount == other.getEyeCount() &&
            this.favouriteCandy == other.getFavouriteCandy()) {

            return true;
        }
        return false;
    }

    public String toString() {
        return "This Alien is called " + name + " and has " + eyeCount + " eyes."
            + " Gross. " + "It seems to enjoy eating " + favouriteCandy;
    }

    public static void main(String[] args) {
        GreenAlien kloup = new GreenAlien();
        GreenAlien lesap = new GreenAlien();
        GreenAlien gwerp = new GreenAlien("Gwerp", 4, "Marshmallows");
        GreenAlien blarg = new GreenAlien("Kloup", 3, "Pop Rocks");

        System.out.println(kloup.equals(lesap));
        System.out.println(gwerp.equals(lesap));
        System.out.println(kloup.equals(blarg));

    }
}
```

Correct

Marks for this submission: 1.00/1.00.

Response history				
Step	Time	Action	State	Marks
<a href="#">1</a>	17/03/19, 17:47	Started	Not complete	
<a href="#">2</a>	17/03/19, 18:14	Submit: public class GreenAlien { public String name=""; public int eyeCount=0; public String favouriteCandy=""; public GreenAlien() { name="Kloup";eyeCount=6;favouriteCandy="Lollypops"; } public GreenAlien(String tempName, int tempEye, String tempCandy) { name=tempName;eyeCount=tempEye;favouriteCandy=tempCandy; } public boolean equals(GreenAlien other) { if (this.name == other.name && this.eyeCount == other.eyeCount && this.favouriteCandy == other.favouriteCandy) { return true; } return false; } public String toString() { return "This Alien is called "+name+" and has "+eyeCount+" eyes. Gross. It seems to enjoy eating "+favouriteCandy; } }	Correct	1.00
3	18/03/19, 15:25	Attempt finished submitting:	Correct	1.00

## ArrayLists

Its nice to know how to bring together lots of objects and store them in some sort of list. For this, we have basic Java arrays, and the ArrayList class (we will see other types of collections later, including class Collections and its sub-classes List and Set).

Basic arrays work very much like you would expect them to. Their syntax is

```
String[10] planets;  
...  
planets[0] = "Mercury";  
planets[1] = "Venus";
```

We can make arrays of any type. Simply place square brackets at the end of the type declaration, and ideally tell Java how big the array is, to make an array.

To index into an array, simply use the variables name and index inside square brackets like any other programming language. Basic arrays are closer to those found in C than Python. Python arrays grow as you add items, and you can shuffle index easily, but this is not the case in Java. Instead, class ArrayList is closer to what you are used to. Its syntax is of the form:

```
ArrayList<type> planets = new ArrayList<type>();
```

The "type" is very important, since it tells Java what type of object can be stored inside the list. Common methods for ArrayLists involve add(), get(), size(), remove(), contains().

```
import java.util.ArrayList;  
  
public class RedTeslaRoadster {  
    private ArrayList<Astronaut> passengers;  
  
    public RedTeslaRoadster() {  
        passengers = new ArrayList<Astronaut>();  
    }  
  
    public void addPassenger(Astronaut newPassenger) {  
        if (!passengers.contains(newPassenger)) {  
            passengers.add(newPassenger);  
        }  
    }  
  
    public boolean removePassenger(Astronaut toRemove) {  
        return passengers.remove(toRemove);  
    }  
  
    public int howManyPassengers() {  
        return passengers.size();  
    }  
  
    public String toString() {  
        String returnString = "There are " + howManyPassengers() + " in the car, and they are: ";  
        for (Astronaut passenger: passengers) {  
            returnString += passenger.getName();  
            returnString += ", ";  
        }  
        return returnString;  
    }  
  
    public static void main(String[] args) {  
        RedTeslaRoadster redCar = new RedTeslaRoadster();  
  
        Astronaut starman = new Astronaut();  
        Astronaut starDog = new Astronaut("Stardog", 4, "Bones", "0000 space exploration");  
  
        redCar.addPassenger(starman);  
        redCar.addPassenger(starDog);  
  
        System.out.println(redCar);  
    }  
}
```

With the output

```
There are 2 in the car, and they are: Starman, Stardog,
```

Response history			
Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	17/03/19, 21:13	Seen	
<b>3</b>	<b>18/03/19, 15:25</b>	<b>Attempt finished</b>	

Question **3**

Correct

Mark 0.80 out of 1.00

## GreenAlienTransporter class

Starman wants to start a GreenAlien transportation company. He wants to have multiple different transportation vehicles in action at a time, and he wants to be able to transport an unlimited number of GreenAliens at once, since GreenAliens can be squished down nicely.

Help Starman out by implementing the class GreenAlienTransporter which has the following methods:

- GreenAlienTransporter(String name), a constructor that sets the name of the transporter, and initialises ArrayLists
- boolean addPassenger(GreenAlien alien), a method which checks if the alien is already on the transporter. If not, the alien is added. Returns true on add, false otherwise.
- boolean removePassenger(GreenAlien alien), a method which check to see if an alien is on board, and removes the alien if it is. Return true on remove, false otherwise.
- void listPassengers(), a method which prints "The passengers on <<name>> are:" followed by a list of each of the aliens names
- int countEyes(), a method which counts the number of eyes of all passengers, and returns them as an int.
- ArrayList<String> shoppingList(), a method which places all of the aliens favourite candy into an ArrayList and is returned to the caller.
- void printDetails(), a method which calls listPassengers(), and then prints "The number of eyes on this transporter is: <<countEyes()>>".

Note: When we put <<something>> you need to put whatever we mean by something, not literally that as text.

**You need to submit your GreenAlien class as well as the GreenAlienTransporter class.**

You might find the following method useful: Collections.contains(Object o)

You will need to utilise ArrayLists in this question. Look them up in the Java API to learn about the methods you can use. In the midterm test there is no internet access. It's just you and the wonderful Java API.

For example:

Test	Result
<pre>GreenAlienTransporter transporter = new GreenAlienTransporter("Fun Club");  GreenAlien kloup = new GreenAlien("Kloup", 9, "Biscuits"); GreenAlien gwerp = new GreenAlien("Gwerp", 4, "Marshmallows"); GreenAlien blarg = new GreenAlien("Blarg", 3, "Pop Rocks"); GreenAlien lesap = new GreenAlien("Lesap", 5, "Chocolate"); GreenAlien hugso = new GreenAlien("Hugso", 2, "Pop Rocks");  transporter.addPassenger(kloup); transporter.addPassenger(gwerp); transporter.addPassenger(blarg); transporter.addPassenger(lesap); transporter.addPassenger(hugso);  transporter.printDetails();</pre>	<p>The passengers on Fun Club are: Kloup, Gwerp, Blarg, Lesap, Hugso, The number of eyes on this transporter is: 23</p>

**Answer:** (penalty regime: 0, 10, 20, ... %)

```
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.TreeSet;
4
5 public class GreenAlienTransporter {
6     private ArrayList<GreenAlien> passengers;
7     private String nameTransport;
8
9     public GreenAlienTransporter(String name) {
10         // TODO Auto-generated constructor stub
11         passengers = new ArrayList<GreenAlien>();
12         nameTransport = name;
13     }
14
15     public boolean addPassenger(GreenAlien newPassenger) {
16         if (!passengers.contains(newPassenger)) {
17             return passengers.add(newPassenger);
18         }
19         else{
20             return false;
21         }
22     },
23 }
```

	Test	Expected	Got	
✓	<pre> GreenAlienTransporter transporter = new GreenAlienTransporter("Fun Club");  GreenAlien kloup = new GreenAlien("Kloup", 9, "Biscuits"); GreenAlien gwerp = new GreenAlien("Gwerp", 4, "Marshmallows"); GreenAlien blarg = new GreenAlien("Blarg", 3, "Pop Rocks"); GreenAlien lesap = new GreenAlien("Lesap", 5, "Chocolate"); GreenAlien hugso = new GreenAlien("Hugso", 2, "Pop Rocks");  transporter.addPassenger(kloup); transporter.addPassenger(gwerp); transporter.addPassenger(blarg); transporter.addPassenger(lesap); transporter.addPassenger(hugso);  transporter.printDetails(); </pre>	<p>The passengers on Fun Club are: Kloup, Gwerp, Blarg, Lesap, Hugso, The number of eyes on this transporter is: 23 The favourites of this group are: Biscuits(1), Chocolate(1), Marshmallows(1), Pop Rocks(2),</p>	<p>The passengers on Fun Club are: Kloup, Gwerp, Blarg, Lesap, Hugso, The number of eyes on this transporter is: 23 The favourites of this group are: Biscuits(1), Chocolate(1), Marshmallows(1), Pop Rocks(2),</p>	✓
✓	<pre> GreenAlienTransporter transporter = new GreenAlienTransporter("Fun Club");  GreenAlien kloup = new GreenAlien("Kloup", 2, "Biscuits"); GreenAlien gwerp = new GreenAlien("Gwerp", 99, "Marshmallows"); GreenAlien blarg = new GreenAlien("Blarg", 5, "Pop Rocks"); GreenAlien lesap = new GreenAlien("Lesap", 3, "Chocolate"); GreenAlien hugso = new GreenAlien("Hugso", 8, "Pop Rocks");  transporter.addPassenger(kloup); transporter.addPassenger(gwerp); transporter.addPassenger(blarg); transporter.addPassenger(lesap); transporter.addPassenger(hugso);  System.out.println(transporter.countEyes()); </pre>	117	117	✓
✓	<pre> GreenAlienTransporter transporter = new GreenAlienTransporter("Fun Club");  GreenAlien kloup = new GreenAlien("Yvest", 2, "Biscuits"); GreenAlien gwerp = new GreenAlien("Lmona", 99, "Marshmallows"); GreenAlien blarg = new GreenAlien("Troll", 5, "Pop Rocks"); GreenAlien lesap = new GreenAlien("Nemoa", 3, "Chocolate"); GreenAlien hugso = new GreenAlien("Hugso", 8, "Pop Rocks");  transporter.addPassenger(kloup); transporter.addPassenger(gwerp); transporter.addPassenger(blarg); transporter.addPassenger(lesap); transporter.addPassenger(hugso);  transporter.listPassengers(); </pre>	<p>The passengers on Fun Club are: Yvest, Lmona, Troll, Nemoa, Hugso,</p>	<p>The passengers on Fun Club are: Yvest, Lmona, Troll, Nemoa, Hugso,</p>	✓

	Test	Expected	Got	
✓	<pre> GreenAlienTransporter transporter = new GreenAlienTransporter("Fun Club");  GreenAlien kloup = new GreenAlien("Yvest", 2, "Biscuits"); GreenAlien gwerp = new GreenAlien("Lmona", 99, "Marshmallows"); GreenAlien blarg = new GreenAlien("Troll", 5, "Pop Rocks"); GreenAlien lesap = new GreenAlien("Nemoa", 3, "Marshmallows"); GreenAlien hugso = new GreenAlien("Hugso", 8, "Pop Rocks");  transporter.addPassenger(kloup); transporter.addPassenger(gwerp); transporter.addPassenger(blarg); transporter.addPassenger(lesap); transporter.addPassenger(hugso);  transporter.printDetails(); </pre>	<pre> The passengers on Fun Club are: Yvest, Lmona, Troll, Nemoa, Hugso, The number of eyes on this transporter is: 117 The favourites of this group are: Biscuits(1), Marshmallows(2), Pop Rocks(2), </pre>	<pre> The passengers on Fun Club are: Yvest, Lmona, Troll, Nemoa, Hugso, The number of eyes on this transporter is: 117 The favourites of this group are: Biscuits(1), Marshmallows(2), Pop Rocks(2), </pre>	✓
✓	<pre> GreenAlienTransporter transporter = new GreenAlienTransporter("Fun Club");  GreenAlien kloup = new GreenAlien("Yvest", 2, "Biscuits"); GreenAlien gwerp = new GreenAlien("Lmona", 99, "Marshmallows"); GreenAlien blarg = new GreenAlien("Troll", 5, "Pop Rocks"); GreenAlien lesap = new GreenAlien("Nemoa", 3, "Marshmallows"); GreenAlien hugso = new GreenAlien("Hugso", 8, "Pop Rocks");  transporter.addPassenger(kloup); System.out.println(transporter.addPassenger(kloup)); transporter.removePassenger(kloup); System.out.println(transporter.addPassenger(kloup)); transporter.addPassenger(gwerp); transporter.addPassenger(blarg); transporter.addPassenger(lesap); transporter.addPassenger(hugso);  transporter.printDetails(); </pre>	<pre> false true The passengers on Fun Club are: Yvest, Lmona, Troll, Nemoa, Hugso, The number of eyes on this transporter is: 117 The favourites of this group are: Biscuits(1), Marshmallows(2), Pop Rocks(2), </pre>	<pre> false true The passengers on Fun Club are: Yvest, Lmona, Troll, Nemoa, Hugso, The number of eyes on this transporter is: 117 The favourites of this group are: Biscuits(1), Marshmallows(2), Pop Rocks(2), </pre>	✓

Passed all tests! ✓

Question author's solution:

```
import java.util.Collections;
import java.util.ArrayList;
import java.util.TreeSet;

public class GreenAlien {
    private String name;
    private int eyeCount;
    private String favouriteCandy;

    public GreenAlien() {
        name = "Kloup";
        eyeCount = 6;
        favouriteCandy = "Lollypops";
    }

    public GreenAlien(String tempName, int tempEye, String tempCandy) {
        name = tempName;
        eyeCount = tempEye;
        favouriteCandy = tempCandy;
    }

    public String getName() {
        return name;
    }

    public void setName(String newName) {
        if (newName.length() == 5) {
            name = newName;
        }
    }

    public int getEyeCount() {
        return eyeCount;
    }

    public void setEyeCount(int newEyes) {
        eyeCount = newEyes;
    }

    public String getFavouriteCandy() {
        return favouriteCandy;
    }

    public void setFavouriteCandy(String newCandy) {
        favouriteCandy = newCandy;
    }

    public boolean equals(GreenAlien other) {
        if (this.name == other.getName() &&
            this.eyeCount == other.getEyeCount() &&
            this.favouriteCandy == other.getFavouriteCandy()) {

            return true;
        }
        return false;
    }

    public String toString() {
        return "This Alien is called " + name + " and has " + eyeCount + " eyes."
            + " Gross. " + "It seems to enjoy eating " + favouriteCandy;
    }

    public static void main(String[] args) {
        GreenAlien kloup = new GreenAlien();
        GreenAlien lesap = new GreenAlien();
        GreenAlien gwerp = new GreenAlien("Gwerp", 4, "Marshmallows");
        GreenAlien blarg = new GreenAlien("Kloup", 3, "Pop Rocks");

        System.out.println(kloup.equals(lesap));
        System.out.println(gwerp.equals(lesap));
        System.out.println(kloup.equals(blarg));

    }
}

public class GreenAlienTransporter {

    private ArrayList<GreenAlien> passengers;
```



```
private String name;

public GreenAlienTransporter(String newName) {
    name = newName;
    passengers = new ArrayList<GreenAlien>();
}

public boolean addPassenger(GreenAlien alien) {
    if (passengers.contains(alien)) {
        return false;
    } else {
        passengers.add(alien);
        return true;
    }
}

public boolean removePassenger(GreenAlien alien) {
    if (passengers.contains(alien)) {
        passengers.remove(alien);
        return true;
    } else {
        return false;
    }
}

public void listPassengers() {
    System.out.println("The passengers on " + name + " are: ");
    for (GreenAlien alien: passengers) {
        System.out.print(alien.getName() + ", ");
    }
    System.out.println();
}

public int countEyes() {
    int eyes = 0;

    for (GreenAlien alien: passengers) {
        eyes += alien.getEyeCount();
    }

    return eyes;
}

public ArrayList<String> shoppingList() {
    ArrayList<String> candy = new ArrayList<String>();

    for (GreenAlien alien: passengers) {
        candy.add(alien.getFavouriteCandy());
    }

    return candy;
}

public void printDetails() {
    listPassengers();
    System.out.println("The number of eyes on this transporter is: " + countEyes());
}

public static void main(String[] args) {
    GreenAlienTransporter transporter = new GreenAlienTransporter("Fun Club");

    GreenAlien kloup = new GreenAlien("Yyest", 2, "Biscuits");
    GreenAlien gwerp = new GreenAlien("Lmona", 99, "Marshmallows");
    GreenAlien blarg = new GreenAlien("Troll", 5, "Pop Rocks");
    GreenAlien lesap = new GreenAlien("Nemoa", 3, "Marshmallows");
    GreenAlien hugso = new GreenAlien("Hugso", 8, "Pop Rocks");

    transporter.addPassenger(kloup);
    System.out.println(transporter.addPassenger(kloup));
    transporter.removePassenger(kloup);
    System.out.println(transporter.addPassenger(kloup));
    transporter.addPassenger(gwerp);
    transporter.addPassenger(blarg);
    transporter.addPassenger(lesap);
    transporter.addPassenger(hugso);
}
```

```
        transporter.printDetails();
    }
}
```

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.80/1.00**.

Response history

Step	Time	Action	State	Marks
<a href="#">1</a>	17/03/19, 17:47	Started	Not complete	
<a href="#">2</a>	17/03/19, 21:13	Submit: import java.util.ArrayList; import java.util.Collections; public class GreenAlienTransporter { private ArrayList<GreenAlien> passengers; private String nameTransport; public GreenAlienTransporter(String name) { // TODO Auto-generated constructor stub passengers = new ArrayList<GreenAlien>(); nameTransport = name; } public boolean addPassenger(GreenAlien newPassenger) { if (!passengers.contains(newPassenger)) { return passengers.add(newPassenger); } else{ return false; } } public boolean removePassenger(GreenAlien toRemove) { return passengers.remove(toRemove); } public int howManyPassengers() { return passengers.size(); } public void listPassengers() { String returnString = "The passengers on "+nameTransport+" are:\n"; for (GreenAlien passenger: passengers) { returnString += passenger.getName(); returnString += ", "; } System.out.println(returnString); } public int countEyes() { int eyes = 0; for (GreenAlien passenger: passengers) { eyes += passenger.eyeCount; } return eyes; } public ArrayList<String> shoppingList(){ ArrayList<String> shoppingList = new ArrayList<String>(); for (GreenAlien passenger: passengers) { shoppingList.add(passenger.getCan()); } return shoppingList; } public void printDetails() { listPassengers(); System.out.println("The number of eyes on this transporter is: "+countEyes()); //Set<String> mySet = new HashSet<String>(shoppingList()); ArrayList<String> shoppingList = shoppingList(); String prtstr = "The favourites of this group are:\n"; for (String candy:shoppingList) { prtstr += candy + "(" + Collections.frequency(shoppingList,candy) + ")"; prtstr += ", "; } System.out.print(prtstr); } } class GreenAlien { private String name=""; public int eyeCount=0; private String favouriteCandy=""; public GreenAlien() { name="Kloup";eyeCount=6;favouriteCandy="Lollypops"; } public GreenAlien(String tempName, int tempEye, String tempCandy) { name=tempName;eyeCount=tempEye;favouriteCandy=tempCandy; } public String getName() { return name; } public String getCan() { return name; } public boolean equals(GreenAlien other) { if (this.name == other.name && this.eyeCount == other.eyeCount && this.favouriteCandy == other.favouriteCandy) { return true; } return false; } public String toString() { return "This Alien is called "+name+" and has "+eyeCount+" eyes. Gross. It seems to enjoy eating "+favouriteCandy; } }	Incorrect	0.00

Step	Time	Action	State	Marks
<a href="#">3</a>	17/03/19, 21:28	<div>Submit: import java.util.ArrayList; import java.util.Collections; public class GreenAlienTransporter { private ArrayList&lt;GreenAlien&gt; passengers; private String nameTransport; public GreenAlienTransporter(String name) { // TODO Auto-generated constructor stub passengers = new ArrayList&lt;GreenAlien&gt;(); nameTransport = name; } public boolean addPassenger(GreenAlien newPassenger) { if (!passengers.contains(newPassenger)) { return passengers.add(newPassenger); } else{ return false; } } public boolean removePassenger(GreenAlien toRemove) { return passengers.remove(toRemove); } public int howManyPassengers() { return passengers.size(); } public void listPassengers() { String returnString = "The passengers on "+nameTransport+" are:\n"; for (GreenAlien passenger: passengers) { returnString += passenger.getName(); returnString += ", "; } System.out.println(returnString); } public int countEyes() { int eyes = 0; for (GreenAlien passenger: passengers) { eyes += passenger.eyeCount; } return eyes; } public ArrayList&lt;String&gt; shoppingList(){ ArrayList&lt;String&gt; shoppingList = new ArrayList&lt;String&gt;(); for (GreenAlien passenger: passengers) { shoppingList.add(passenger.getCan()); } return shoppingList; } public void printDetails() { listPassengers(); System.out.println("The number of eyes on this transporter is: "+countEyes()); ArrayList&lt;String&gt; shoppingList = shoppingList(); HashSet&lt;String&gt; mySet = new HashSet&lt;String&gt;(shoppingList()); String prtstr = "The favourites of this group are:\n"; for (String candy:mySet) { prtstr += candy + "(" + Collections.frequency(shoppingList,candy) + ")"; prtstr += ", "; } System.out.print(prtstr); } } class GreenAlien { private String name=""; public int eyeCount=0; private String favouriteCandy=""; public GreenAlien() { name="Kloup";eyeCount=6;favouriteCandy="Lollypops"; } public GreenAlien(String tempName, int tempEye, String tempCandy) { name=tempName;eyeCount=tempEye;favouriteCandy=tempCandy; } public String getName() { return name; } public String getCan() { return name; } public boolean equals(GreenAlien other) { if (this.name == other.name &amp;&amp; this.eyeCount == other.eyeCount &amp;&amp; this.favouriteCandy == other.favouriteCandy) { return true; } return false; } public String toString() { return "This Alien is called "+name+" and has "+eyeCount+" eyes. Gross. It seems to enjoy eating "+favouriteCandy; } }</div>	Incorrect	0.00

Step	Time	Action	State	Marks
<a href="#">4</a>	17/03/19, 21:29	<div>Submit: import java.util.ArrayList; import java.util.Collections; public class GreenAlienTransporter { private ArrayList&lt;GreenAlien&gt; passengers; private String nameTransport; public GreenAlienTransporter(String name) { // TODO Auto-generated constructor stub passengers = new ArrayList&lt;GreenAlien&gt;(); nameTransport = name; } public boolean addPassenger(GreenAlien newPassenger) { if (!passengers.contains(newPassenger)) { return passengers.add(newPassenger); } else{ return false; } } public boolean removePassenger(GreenAlien toRemove) { return passengers.remove(toRemove); } public int howManyPassengers() { return passengers.size(); } public void listPassengers() { String returnString = "The passengers on "+nameTransport+" are:\n"; for (GreenAlien passenger: passengers) { returnString += passenger.getName(); returnString += ", "; } System.out.println(returnString); } public int countEyes() { int eyes = 0; for (GreenAlien passenger: passengers) { eyes += passenger.eyeCount; } return eyes; } public ArrayList&lt;String&gt; shoppingList(){ ArrayList&lt;String&gt; shoppingList = new ArrayList&lt;String&gt;(); for (GreenAlien passenger: passengers) { shoppingList.add(passenger.getCan()); } return shoppingList; } public void printDetails() { listPassengers(); System.out.println("The number of eyes on this transporter is: "+countEyes()); ArrayList&lt;String&gt; shoppingList = shoppingList(); HashSet&lt;String&gt; mySet = new HashSet&lt;String&gt;(shoppingList()); String prtstr = "The favourites of this group are:\n"; for (String candy:mySet) { prtstr += candy + "(" + Collections.frequency(shoppingList,candy) + ")"; prtstr += ", "; } System.out.print(prtstr); } } class GreenAlien { private String name=""; public int eyeCount=0; private String favouriteCandy=""; public GreenAlien() { name="Kloup";eyeCount=6;favouriteCandy="Lollypops"; } public GreenAlien(String tempName, int tempEye, String tempCandy) { name=tempName;eyeCount=tempEye;favouriteCandy=tempCandy; } public String getName() { return name; } public String getCan() { return favouriteCandy; } public boolean equals(GreenAlien other) { if (this.name == other.name &amp;&amp; this.eyeCount == other.eyeCount &amp;&amp; this.favouriteCandy == other.favouriteCandy) { return true; } return false; } public String toString() { return "This Alien is called "+name+" and has "+eyeCount+" eyes. Gross. It seems to enjoy eating "+favouriteCandy; } }</div>	Incorrect	0.00

Step	Time	Action	State	Marks
<a href="#">5</a>	17/03/19, 21:43	Submit: import java.util.ArrayList; import java.util.Collections; import java.util.TreeSet; public class GreenAlienTransporter { private ArrayList<GreenAlien> passengers; private String nameTransport; public GreenAlienTransporter(String name) { // TODO Auto-generated constructor stub passengers = new ArrayList<GreenAlien>(); nameTransport = name; } public boolean addPassenger(GreenAlien newPassenger) { if (!passengers.contains(newPassenger)) { return passengers.add(newPassenger); } else{ return false; } } public boolean removePassenger(GreenAlien toRemove) { return passengers.remove(toRemove); } public int howManyPassengers() { return passengers.size(); } public void listPassengers() { String returnString = "The passengers on "+nameTransport+" are:\n"; for (GreenAlien passenger: passengers) { returnString += passenger.getName(); returnString += ", "; } System.out.println(returnString); } public int countEyes() { int eyes = 0; for (GreenAlien passenger: passengers) { eyes += passenger.eyeCount; } return eyes; } public ArrayList<String> shoppingList(){ ArrayList<String> shoppingList = new ArrayList<String>(); for (GreenAlien passenger: passengers) { shoppingList.add(passenger.getCan()); } return shoppingList; } public void printDetails() { listPassengers(); System.out.println("The number of eyes on this transporter is: "+countEyes()); ArrayList<String> shoppingList = shoppingList(); TreeSet<String> mySet = new TreeSet<String>(shoppingList);//Set sort and has un-duplicate values String prtstr = "The favourites of this group are:\n"; for (String candy:mySet) { prtstr += candy + "(" + Collections.frequency(shoppingList,candy) + ")"; prtstr += ", "; } System.out.print(prtstr); } } class GreenAlien { private String name=""; public int eyeCount=0; private String favouriteCandy=""; public GreenAlien() { name="Kloup";eyeCount=6;favouriteCandy="Lollypops"; } public GreenAlien(String tempName, int tempEye, String tempCandy) { name=tempName;eyeCount=tempEye;favouriteCandy=tempCandy; } public String getName() { return name; } public String getCan() { return favouriteCandy; } public boolean equals(GreenAlien other) { if (this.name == other.name && this.eyeCount == other.eyeCount && this.favouriteCandy == other.favouriteCandy) { return true; } return false; } public String toString() { return "This Alien is called "+name+" and has "+eyeCount+" eyes. Gross. It seems to enjoy eating "+favouriteCandy; } }	Correct	0.80
6	18/03/19, 15:25	Attempt finished submitting:	Correct	0.80

## On to Interfaces...

Lets talk about interfaces. What do the objects Rocket, SoccerBall, Bird, Plane have in common? For a start, they are all very different objects. A Rocket is absolutely not some type of SoccerBall, and neither is a Bird some type of Rocket. But they have something in common: they can all fly.

Being able to fly makes them connected in some way. I can call `Rocket.fly()` just as easily as `Bird.fly()`. If I was a scientist who studied aerodynamics, then maybe I would be interested in collections of objects that are able to fly, and not interested in anything else. But how can I see if an object provides such functionality? Do I go around and see if it has a `fly()` method? How do I know in the context of `fly()`, that it is moving through the air / sky / space? What happens if I get a new shiny object, and call the `fly()` method only to hear a song "Sugar how you get so fly" playing? That wasn't what I wanted or expected.

Interfaces let seemingly unrelated objects provide some common functionality / service with an agreed upon contract. The "contract" is the method signature, while the common functionality is implemented methods.

Interfaces are created with the "public interface <name>" syntax, and can contain method signatures, variable definitions and more. Methods are implicitly assumed to be *abstract* by default.

```
public interface Flyable {
    public void fly();
}
```

Now, interfaces cannot be instantiated on their own. Instead they need to be built into classes that want to conform to the contract specified in the interface. We use the *implements* keyword for this. Any class that implements the Flyable interface must have an implementation of the `fly()` method with a method signature that matches the one above.

```
public class soccerBall implements Flyable {
    public void fly() {
        // implementation
    }
}
```

You can also include constants in your interface. Remember, interfaces are stateless, and cannot hold any mutable variables. In this case, simply declare them normally. Note, "final" does not need to be used, as it is implicitly assumed.

```
public interface Flyable {
    public static final String DOMAIN = "Air";

    public void fly();
}
```

Response history			
Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	17/03/19, 23:18	Seen	
3	18/03/19, 15:25	Attempt finished	



Question **4**

Correct

Mark 0.90 out of 1.00

## RemoteControllable Class

Starman is having a hard time remotely controlling a number of devices he has around the galaxy. He has a few rovers, satellites and space stations that he wants to keep tabs on, but they are all written with different exposed APIs. Starman wants to make them more consistent, and his needs are very simple.

Write an interface called **RemoteControllable** that has the following abstract methods:

- String getStatusReport()
- void updateMission(String newMission)

Write two classes, **Rover** and **SpaceStation** that implement the **RemoteControllable** interface.

**Rover** has the following methods:

- void setLocation(double latitude, double longitude), which sets latitude and longitude variables.
- String getLocation(), which returns a String of "The rover is located <latitude>, <longitude> on the planet."
- String getStatusReport(), which returns the value of getLocation(), concatenated with the String: "The rover is driving to: <mission>" on a new line (you might remember you can add a new line to a string by adding a \n).
- void updateMission(String mission), which updates value of mission.

**SpaceStation** has the following methods:

- SpaceStation(String planet), which stores the planet the station is floating around.
- String getLocation(), which returns a String of "The space station floats around the planet <planet>"
- String getStatusReport(), which returns a String of getLocation() concatenated with "The station is on a mission to: <mission>" on a new line.
- void updateMission(String mission), which updates the value of mission.

Submit all classes and interfaces. RemoteControllable, Rover and SpaceStation.

For example:

Test	Result
SpaceStation station = new SpaceStation("Mars"); System.out.println(station instanceof RemoteControllable); station.updateMission("Store food rations for hungry space travelers"); System.out.println(station.getStatusReport());	true The space station floats around the planet Mars The station is on a mission to: Store food rations for hungry space travelers
Rover rover = new Rover(); System.out.println(rover instanceof RemoteControllable); rover.setLocation(100.0, 42.0); rover.updateMission("Find water"); System.out.println(rover.getStatusReport());	true The rover is located 100.0, 42.0 on the planet. The rover is driving to: Find water

Answer: (penalty regime: 10, 20, ... %)

```
1 public interface RemoteControllable {
2     public static final String DOMAIN = "You can have a static attribute within an interface";
3
4     public String getStatusReport();
5     public void updateMission(String newMission);
6 }
7
8 class SpaceStation implements RemoteControllable{
9     private String Planet="";
10    private String currMission="";
11
12    public SpaceStation(String planet) {
13        Planet = planet;
14    }
15    public void updateMission(String mission) {
16        currMission = mission;
17    }
18    public String getLocation() {
19        return "The space station floats around the planet "+Planet;
20    }
21    public String getStatusReport() {
22        return getLocation()+"\nThe station is on a mission to: "+currMission;
```

	Test	Expected	Got	
--	------	----------	-----	--



	Test	Expected	Got	
✓	<pre>SpaceStation station = new SpaceStation("Mars"); System.out.println(station instanceof RemoteControllable); station.updateMission("Store food rations for hungry space travelers"); System.out.println(station.getStatusReport());</pre>	<pre>true The space station floats around the planet Mars The station is on a mission to: Store food rations for hungry space travelers</pre>	<pre>true The space station floats around the planet Mars The station is on a mission to: Store food rations for hungry space travelers</pre>	✓
✓	<pre>Rover rover = new Rover(); System.out.println(rover instanceof RemoteControllable); rover.setLocation(100.0, 42.0); rover.updateMission("Find water"); System.out.println(rover.getStatusReport());</pre>	<pre>true The rover is located 100.0, 42.0 on the planet. The rover is driving to: Find water</pre>	<pre>true The rover is located 100.0, 42.0 on the planet. The rover is driving to: Find water</pre>	✓

Passed all tests! ✓

### Question author's solution:

```
public interface RemoteControllable {
    String getStatusReport();
    void updateMission(String newMission);
}

public class Rover implements RemoteControllable {

    String mission;
    double latitude;
    double longitude;

    public void setLocation(double latitude, double longitude) {
        this.latitude = latitude;
        this.longitude = longitude;
    }

    public String getLocation() {
        return "The rover is located " + latitude + ", " + longitude + " on the planet.";
    }

    public String getStatusReport() {
        return getLocation() + "\nThe rover is driving to: " + mission;
    }

    public void updateMission(String mission) {
        this.mission = mission;
    }
}

public class SpaceStation implements RemoteControllable {
    String mission;
    String planet;

    public SpaceStation(String planet) {
        this.planet = planet;
    }

    public String getLocation() {
        return "The space station floats around the planet " + planet;
    }

    public String getStatusReport() {
        return getLocation() + "\nThe station is on a mission to: " + mission;
    }

    public void updateMission(String mission) {
        this.mission = mission;
    }
}
```

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.90/1.00**.

Response history				
Step	Time	Action	State	Marks
<a href="#">1</a>	17/03/19, 17:47	Started	Not complete	
<a href="#">2</a>	17/03/19, 23:18	Submit: public interface RemoteControllable { public static final String DOMAIN = "You can have a static attribute within an interface (final means cannot be override)"; public String getStatusReport(); public void updateMission(String newMission); } class SpaceStation implements RemoteControllable{ private String Planet=""; private String currMission=""; public SpaceStation(String planet) { Planet = planet; } public void updateMission(String mission) { currMission = mission; } public String getLocation() { return "The space station floats around the planet "+Planet; } public String getStatusReport() { return getLocation()+"\nThe rover is driving to: "+currMission; } } class Rover implements RemoteControllable{ private double latitude; private double longitude; private String currMission; public void setLocation(double latitude, double longitude) { this.latitude = latitude; this.longitude = longitude; } public void updateMission(String mission) { currMission = mission; } public String getLocation() { return String.format("The rover is located %.1f, %.1f on the planet.", latitude, longitude); } public String getStatusReport() { return getLocation()+"\nThe rover is driving to: "+currMission; } }	Incorrect	0.00
<a href="#">3</a>	17/03/19, 23:19	Submit: public interface RemoteControllable { public static final String DOMAIN = "You can have a static attribute within an interface (final means cannot be override)"; public String getStatusReport(); public void updateMission(String newMission); } class SpaceStation implements RemoteControllable{ private String Planet=""; private String currMission=""; public SpaceStation(String planet) { Planet = planet; } public void updateMission(String mission) { currMission = mission; } public String getLocation() { return "The space station floats around the planet "+Planet; } public String getStatusReport() { return getLocation()+"\nThe station is on a mission to: "+currMission; } } class Rover implements RemoteControllable{ private double latitude; private double longitude; private String currMission; public void setLocation(double latitude, double longitude) { this.latitude = latitude; this.longitude = longitude; } public void updateMission(String mission) { currMission = mission; } public String getLocation() { return String.format("The rover is located %.1f, %.1f on the planet.", latitude, longitude); } public String getStatusReport() { return getLocation()+"\nThe rover is driving to: "+currMission; } }	Correct	0.90
4	18/03/19, 15:25	Attempt finished submitting:	Correct	0.90

## ArrayLists and Interfaces

You can use interfaces as the raw types for things like ArrayLists, if you want to have many different objects in one place that implement a particular interface.

Just:

```
public class TestFlying {
    public static void main(String[] args) {
        ArrayList<Flyable> flyingThings = new ArrayList<Flyable>();
        flyingThings.add(new SoccerBall());
        flyingThings.add(new FalconHeavy());
        flyingThings.add(new Bird());
        flyingThings.add(new FalconHeavy());

        for (Flyable canFly: flyingThings) {
            canFly.fly();
        }
    }
}
```

In the example above, we can keep objects of different classes together in an ArrayList<Flyable>. We can safely call the fly() method of each member of flyingThings, as we know they all implement the interface Flyable, and so must have such a method.

Response history			
Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	17/03/19, 23:18	Seen	
<b>3</b>	<b>18/03/19, 15:25</b>	<b>Attempt finished</b>	

Question **5**

Correct

Mark 1.00 out of 1.00

Starman wants to bring all of his **Rovers** and **SpaceStations** into one place, where he can easily manage things. This class will store all the instances of any object that implements the interface **RemoteControllable**, and reject any that do not adhere to the interface.

Implement the class **StarmanOrganises** which has the following methods:

- void addControllable(Object o), a method that determines if the object implements RemoteControllable, and if it does, adds it to an ArrayList.
- void getAllStatusReports(), a method which prints the status report for all RemoteControllable objects.

You might want to use the **instanceof** keyword somewhere. Learning about a **cast** might be useful too, as you will need to cast the Object o to type RemoteControllable in addControllable() after the instanceof check.

Submit all classes and interfaces. Aka, RemoteControllable, Rover, SpaceStation and StarmanOrganises

For example:

Test	Result
<pre>SpaceStation station = new SpaceStation("Mars"); station.updateMission("Store food rations for hungry space travelers");  Rover rover = new Rover(); rover.setLocation(100.0, 42.0); rover.updateMission("Find water");  StarmanOrganises organise = new StarmanOrganises(); organise.addControllable(station); organise.addControllable(rover); organise.getAllStatusReports();</pre>	<pre>The space station floats around the planet Mars The station is on a mission to: Store food rations for hungry space travelers The rover is located 100.0, 42.0 on the planet. The rover is driving to: Find water</pre>

Answer: (penalty regime: 10, 20, ... %)

```
1 public interface RemoteControllable {
2     public static final String DOMAIN = "You can have a static attribute within an interface";
3
4     public String getStatusReport();
5     public void updateMission(String newMission);
6 }
7
8 class SpaceStation implements RemoteControllable{
9     private String Planet="";
10    private String currMission="";
11
12    public SpaceStation(String planet) {
13        Planet = planet;
14    }
15    public void updateMission(String mission) {
16        currMission = mission;
17    }
18    public String getLocation() {
19        return "The space station floats around the planet "+Planet;
20    }
21    public String getStatusReport() {
22        return getLocation()+"\nThe station is on a mission to: "+currMission;
```

	Test	Expected	Got	
✓	<pre>SpaceStation station = new SpaceStation("Mars"); station.updateMission("Store food rations for hungry space travelers");  Rover rover = new Rover(); rover.setLocation(100.0, 42.0); rover.updateMission("Find water");  StarmanOrganises organise = new StarmanOrganises(); organise.addControllable(station); organise.addControllable(rover); organise.getAllStatusReports();</pre>	<pre>The space station floats around the planet Mars The station is on a mission to: Store food rations for hungry space travelers The rover is located 100.0, 42.0 on the planet. The rover is driving to: Find water</pre>	<pre>The space station floats around the planet Mars The station is on a mission to: Store food rations for hungry space travelers The rover is located 100.0, 42.0 on the planet. The rover is driving to: Find water</pre>	✓

	Test	Expected	Got	
✓	<pre>SpaceStation station = new SpaceStation("Mars"); station.updateMission("Store food rations for hungry space travelers");  Object obj = new Object();  StarmanOrganises organise = new StarmanOrganises(); organise.addControllable(station); organise.addControllable(obj); organise.getAllStatusReports();</pre>	<pre>The space station floats around the planet Mars The station is on a mission to: Store food rations for hungry space travelers</pre>	<pre>The space station floats around the planet Mars The station is on a mission to: Store food rations for hungry space travelers</pre>	✓

Passed all tests! ✓

Question author's solution:

```
import java.util.ArrayList;

public interface RemoteControllable {
    String mission = "Follow Starmans orders";

    String getStatusReport();
    void updateMission(String newMission);
}

public class Rover implements RemoteControllable {

    String mission;
    double latitude;
    double longitude;

    public void setLocation(double latitude, double longitude) {
        this.latitude = latitude;
        this.longitude = longitude;
    }

    public String getLocation() {
        return "The rover is located " + latitude + ", " + longitude + " on the planet.";
    }

    public String getStatusReport() {
        return getLocation() + "\nThe rover is driving to: " + mission;
    }

    public void updateMission(String mission) {
        this.mission = mission;
    }

}

public class SpaceStation implements RemoteControllable {
    String mission;
    String planet;

    public SpaceStation(String planet) {
        this.planet = planet;
    }

    public String getLocation() {
        return "The space station floats around the planet " + planet;
    }

    public String getStatusReport() {
        return getLocation() + "\nThe station is on a mission to: " + mission;
    }

    public void updateMission(String mission) {
        this.mission = mission;
    }

}

public class StarmanOrganises {
    ArrayList<RemoteControllable> remote = new ArrayList<RemoteControllable>();

    void addControllable(Object o) {
        if (o instanceof RemoteControllable) {
            remote.add((RemoteControllable) o);
        }
    }

    void getAllStatusReports() {
        for (RemoteControllable item: remote) {
            System.out.println(item.getStatusReport());
        }
    }

}
```

Correct

Marks for this submission: 1.00/1.00.

Response history				
Step	Time	Action	State	Marks
<a href="#">1</a>	17/03/19, 17:47	Started	Not complete	
<a href="#">2</a>	18/03/19, 00:04	Submit: public interface RemoteControllable { public static final String DOMAIN = "You can have a static attribute within an interface (final means cannot be override)"; public String getStatusReport(); public void updateMission(String newMission); } class SpaceStation implements RemoteControllable{ private String Planet=""; private String currMission=""; public SpaceStation(String planet) { Planet = planet; } public void updateMission(String mission) { currMission = mission; } public String getLocation() { return "The space station floats around the planet "+Planet; } public String getStatusReport() { return getLocation()+"\nThe station is on a mission to: "+currMission; } } class Rover implements RemoteControllable{ private double latitude; private double longitude; private String currMission; public void setLocation(double latitude, double longitude) { this.latitude = latitude; this.longitude = longitude; } public void updateMission(String mission) { currMission = mission; } public String getLocation() { return String.format("The rover is located %.1f, %.1f on the planet.", latitude, longitude); } public String getStatusReport() { return getLocation()+"\nThe rover is driving to: "+currMission; } } class StarmanOrganises { //This class store data of objects implements RemoteControlable interface private ArrayList<RemoteControllable> controlObj = new ArrayList<RemoteControllable>(); public void addControllable(Object obj) { if (obj instanceof RemoteControllable) { controlObj.add((RemoteControllable) obj);} //casting - aka explicit } } public void getAllStatusReports() { for (RemoteControllable obj:controlObj) { System.out.println(obj.getStatusReport()); } } }	Correct	1.00
3	18/03/19, 15:25	Attempt finished submitting:	Correct	1.00

# Inheritance

As we mentioned above, *polymorphism* is the idea where objects of several types can be accessed through the same interface. If we think of a real world object, such as a rocket, a rocket is a very *generalised* concept. A rocket is some form of spacecraft or vehicle, that obtains thrust from a rocket engine. Rockets burn propellant which is within the rocket before launch, and is ejected in the opposite direction to the motion of the rocket.

Just from this, we get the idea of what a rocket is, and some of the interfaces it provides. A rocket might have the following methods:

```
public class Rocket {
    private boolean engineRunning;
    private double currentThrust;
    private double propellant;

    public Rocket(double initialThrust, double propellant) {
        engineRunning = false;
        currentThrust = initialThrust;
        currentPropellant = propellant;
    }

    public double getPropellantLevels() {
        // implementation
    }
    public double getThrust() {
        // implementation
    }
    public double increaseThrust() {
        // implementation
    }
    public double lowerThrust() {
        // implementation
    }
    public boolean engineStartup() {
        // implementation
    }
    public boolean engineShutdown() {
        // implementation
    }
    // and so on...
}
```

These methods are common to all rockets, and to an extent, can offer generic implementations. But what happens if I need something more *specific*? My rocket might have multiple engines, or special guidance systems, or laser beams. I could go and write a new class, but I would still need to include the same generic functions, and I would be duplicating work. How can object-orientation help us?

Well, this is where inheritance becomes very useful. We can create a *subclass*, where all of the *parent* class methods and attributes are carried over to the subclass, but we can extend the subclass to have more functionality and even override some functionality from the parent class.

How can we do this? By using the *extends* keyword. When you declare the subclass, write the name, followed with "extends parent" like in the following example:



```
public class FalconHeavy extends Rocket {
    // these attributes belong to FalconHeavy only
    int num boosterRockets;
    ArrayList<String> payload;

    public FalconHeavy(String initialPayload) {
        super(100000, 100);
        boosterRockets = 2;
        payload = new ArrayList<String>();
        payload.add(initialPayload);
    }

    public boolean startBoosters() {
        // implementation
    }
    public boolean stopBoosters() {
        // implementation
    }
    public boolean boosterSelfLand() {
        // implementation
    }
    public boolean ejectStage(int stage) {
        // implementation
    }

    public static void int main(String[] args) {
        String redTeslaRoadster = "Red Tesla Roadster";
        FalconHeavy falcon = new FalconHeavy(redTeslaRoadster);
        falcon.startBooster();
        falcon.engineStartup();
        falcon.increaseThrust();
    }
}
```

The idea of inheritance is that the further down the inheritance chain, the more specific your classes get. In the other direction, when you move up the inheritance chain, the classes get more generalised. You can have as many subclasses as you wish, and there is no limit to depth. The process of making something that is general, and moving it down the inheritance tree to more specific classes is *specialisation*. The process of taking something specific and moving it up the tree to be more general, is *generalisation*.

Did you see in the main() method of FalconHeavy, that you are free to call methods of parent as you wish? You can also reference class attributes, but this can get dicey. We recommend sticking to getters and setters for that.

In the constructor of FalconHeavy, did you see the use of *super()*? *super()* allows you to call the parents constructor to set other attributes.

Response history

Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	18/03/19, 00:54	Seen	
<b>3</b>	<b>18/03/19, 15:25</b>	<b>Attempt finished</b>	

Question **6**

Correct

Mark 1.00 out of 1.00

## Planet Class (Inheritance)

Starman wants to write some code to formalise the universe that he sees around him, and he wants to start with Planets. Planets interest Starman greatly, and he especially likes taking long holidays to take in the vistas that Earth dwellers seldom get to see.

Implement a Planet class which has the following methods:

- Planet(String name, int order, String temperature), the constructor.
- String orderFromSun(), a method which returns the String "<planetName> is number <orderNum> from the Sun"
- String getName(), which returns the name of the planet as a String.
- String getTemperature(), which returns a String which indicates the temperature. "Cold", "Warm" and "Hot" are possible.
- String toString(), which returns the String "orderFromSun() and is <temperature>"

Then go and make subclasses for each Planet in the Milky Way Galaxy that we treat as a full planet. You don't have to do dwarf planets.

We don't want to bore you, so you only need to implement Mercury, Venus, Earth, and Saturn.

They should all have default constructors that set their names, order from the sun, and their temperatures. Hint - you can use use the super() method.

Temperatures:

Cold: Saturn

Warm: Earth

Hot: Mercury, Venus

The Earth class should have the following additional methods:

- String home(), which returns a String "Home to every one of us"

The Saturn class should have the following methods:

- String largeRings, which returns the String "Saturn has massive rings"

The other planets do not have any additional methods.

You need to submit ALL of the classes involved. Planet, Mercury, Venus, Earth, Saturn

For example:

Test	Result
Mercury mercury = new Mercury(); System.out.println(mercury instanceof Planet); System.out.println(mercury);	true Mercury is number 1 from the Sun and is Hot
Earth earth = new Earth(); System.out.println(earth instanceof Planet); System.out.println(earth); System.out.println(earth.home());	true Earth is number 3 from the Sun and is Warm Home to every one of us

Answer: (penalty regime: 10, 20, ... %)

```
1 public class Planet {
2     //Super class
3     private String Name;
4     private int Order;
5     private String Temp;
6
7     //public Planet() {}
8     public Planet(String name, int order, String temperature){
9         Name = name;
10        Order = order;
11        Temp = temperature;
12    }
13    public String getName() {
14        return Name;
15    }
16    public String getTemp() {
17        return Temp;
18    }
19
20    public String orderFromSun() {
21        return String.format("%s is number %d from the Sun", Name, Order);
22    }
23 }
```

	Test	Expected	Got	
--	------	----------	-----	--

	Test	Expected	Got	
✓	Mercury mercury = new Mercury(); System.out.println(mercury instanceof Planet); System.out.println(mercury);	true Mercury is number 1 from the Sun and is Hot	true Mercury is number 1 from the Sun and is Hot	✓
✓	Venus venus = new Venus(); System.out.println(venus instanceof Planet); System.out.println(venus);	true Venus is number 2 from the Sun and is Hot	true Venus is number 2 from the Sun and is Hot	✓
✓	Earth earth = new Earth(); System.out.println(earth instanceof Planet); System.out.println(earth); System.out.println(earth.home());	true Earth is number 3 from the Sun and is Warm Home to every one of us	true Earth is number 3 from the Sun and is Warm Home to every one of us	✓
✓	Saturn saturn = new Saturn(); System.out.println(saturn instanceof Planet); System.out.println(saturn); System.out.println(saturn.largeRings());	true Saturn is number 6 from the Sun and is Cold Saturn has massive rings	true Saturn is number 6 from the Sun and is Cold Saturn has massive rings	✓

Passed all tests! ✓

Question author's solution:

```
import java.util.ArrayList;

class Planet {
    private String planetName;
    private String temperature;
    private int orderNum;

    public Planet(String name, int order, String temperature) {
        planetName = name;
        orderNum = order;
        this.temperature = temperature;
    }

    public String orderFromSun() {
        return planetName + " is number " + orderNum + " from the Sun";
    }

    public String getName() {
        return planetName;
    }

    public String getTemperature() {
        return temperature;
    }

    public String toString() {
        return orderFromSun() + " and is " + getTemperature();
    }
}

class Mercury extends Planet {
    public Mercury() {
        super("Mercury", 1, "Hot");
    }
}

class Venus extends Planet {
    public Venus() {
        super("Venus", 2, "Hot");
    }
}

class Earth extends Planet {
    public Earth() {
        super("Earth", 3, "Warm");
    }

    public String home() {
        return "Home to every one of us";
    }
}

class Saturn extends Planet {
    public Saturn() {
        super("Saturn", 6, "Cold");
    }

    public String largeRings() {
        return "Saturn has massive rings";
    }
}
```

Correct

Marks for this submission: 1.00/1.00.

Response history				
Step	Time	Action	State	Marks
<a href="#">1</a>	17/03/19, 17:47	Started	Not complete	
<a href="#">2</a>	18/03/19, 00:54	Submit: public class Planet { //Super class private String Name; private int Order; private String Temp; //public Planet() {} public Planet(String name, int order, String temperature){ Name = name; Order = order; Temp = temperature; } public String getName() { return Name; } public String getTemp() { return Temp; } public String orderFromSun() { return String.format("%s is number %d from the Sun", Name, Order); } public String toString() { return orderFromSun() + " and is " + getTemp(); } } class Earth extends Planet{ public Earth() { super("Earth", 3, "Warm"); } public String home() { return "Home to every one of us"; } } class Mercury extends Planet{ public Mercury() { super("Mercury", 1, "Hot"); } } class Venus extends Planet { public Venus() { super("Venus", 2, "Hot"); } } class Saturn extends Planet{ public Saturn() { super("Saturn", 6, "Cold"); } public String largeRings() { return "Saturn has massive rings"; } }	Correct	1.00
3	18/03/19, 15:25	Attempt finished submitting:	Correct	1.00

Information

## Assignment and Inheritance

Now that we have some experience with inheritance and the types they convey, lets see how we can use inheritance for variable assignment and collections.

Everything you have seen so far in Java has been:

```
Type object = new Type();
```

And this is still the same with Inheritance.

```
Rocket rocketship = new Rocket(1000, 100);
FalconHeavy falcon = new FalconHeavy("Red Tesla Roadster");
```

Now, FalconHeavy is a type of Rocket, and this allows us to directly assign subclasses to the parent type. Like:

```
Rocket falcon = new FalconHeavy("Red Tesla Roadster");
```

This works for any subclass, and at any level of the inheritance tree. As long as the type on the left is a parent type, any subclass will work. This enables us to have ArrayLists, Sets and Maps or anything Collection related to hold subclasses.

```
ArrayList<Rocket> rocketList = new ArrayList<Rocket>();
rocketList.add(new Rocket(100, 100));
rocketList.add(new FalconHeavy("Red Tesla Roadster"));
rocketList.add(new SpaceShuttle());
```

**Note, that if the object is a subclass, and is stored as a parent type, only the methods of the parent are available to the object.** This is because Java treats the type as the parent type only, and so the parent interface is what you must stick to. If you want to use methods of the subclass, you must restore its type by making a new assignment.

While we are at it, **this relationship does not work the other way. You cannot put an object of the parent into a subclass type.** This is because the object of the parent does not conform to the interface offered by the subclass. A standard rocket cannot fire laser beams, if it has none in the first place right?

Response history			
Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	18/03/19, 00:54	Seen	
3	18/03/19, 15:25	Attempt finished	

Question **7**

Correct

Mark 1.00 out of 1.00

## SolarSystem Class

Starman wants to keep all of his Planets in a nice little Array, so he can perform some calculations in the future. Help Starman implement the class **SolarSystem** which has the following methods:

- void addPlanet(Planet planet), which adds a planet to an ArrayList
- void printAllPlanets(), which calls the toString() method of each planet in the ArrayList.
- int getPlanetCount(), which returns the number of planets currently contained in the ArrayList.

You need to submit all classes, aka, SolarSystem, Planet, Mercury, Venus, Earth, Saturn.

For example:

Test	Result
<pre>Planet mercury = new Mercury(); Planet venus = new Venus(); Planet earth = new Earth(); Planet saturn = new Saturn();  SolarSystem solarsystem = new SolarSystem(); solarsystem.addPlanet(mercury); solarsystem.addPlanet(venus); solarsystem.addPlanet(earth); solarsystem.addPlanet(saturn);  solarsystem.printAllPlanets(); System.out.println(solarsystem.getPlanetCount());</pre>	<pre>Mercury is number 1 from the Sun and is Hot Venus is number 2 from the Sun and is Hot Earth is number 3 from the Sun and is Warm Saturn is number 6 from the Sun and is Cold 4</pre>

Answer: (penalty regime: 10, 20, ... %)

```
1 import java.util.ArrayList;
2
3 public class SolarSystem {
4     private ArrayList<Planet> solarList = new ArrayList<Planet>();
5
6     public void addPlanet(Planet planet) {
7         solarList.add(planet);
8     }
9
10    public int getPlanetCount() {
11        return solarList.size();
12    }
13
14    public void printAllPlanets() {
15        for (Planet planet:solarList) {
16            System.out.println(planet);
17        }
18    }
19 }
20
21 class Planet {
22     //Sun class
```

	Test	Expected	Got	
✓	<pre>Planet mercury = new Mercury(); Planet venus = new Venus(); Planet earth = new Earth(); Planet saturn = new Saturn();  SolarSystem solarsystem = new SolarSystem(); solarsystem.addPlanet(mercury); solarsystem.addPlanet(venus); solarsystem.addPlanet(earth); solarsystem.addPlanet(saturn);  solarsystem.printAllPlanets(); System.out.println(solarsystem.getPlanetCount());</pre>	<pre>Mercury is number 1 from the Sun and is Hot Venus is number 2 from the Sun and is Hot Earth is number 3 from the Sun and is Warm Saturn is number 6 from the Sun and is Cold 4</pre>	<pre>Mercury is number 1 from the Sun and is Hot Venus is number 2 from the Sun and is Hot Earth is number 3 from the Sun and is Warm Saturn is number 6 from the Sun and is Cold 4</pre>	✓

	Test	Expected	Got	
✔	<pre>Planet venus = new Venus(); Planet saturn = new Saturn();  SolarSystem solarsystem = new SolarSystem(); solarsystem.addPlanet(venus); solarsystem.addPlanet(saturn);  solarsystem.printAllPlanets(); System.out.println(solarsystem.getPlanetCount());</pre>	Venus is number 2 from the Sun and is Hot Saturn is number 6 from the Sun and is Cold 2	Venus is number 2 from the Sun and is Hot Saturn is number 6 from the Sun and is Cold 2	✔

Passed all tests! ✔

Question author's solution:

```
import java.util.ArrayList;

class Planet {
    private String planetName;
    private String temperature;
    private int orderNum;

    public Planet(String name, int order, String temperature) {
        planetName = name;
        orderNum = order;
        this.temperature = temperature;
    }

    public String orderFromSun() {
        return planetName + " is number " + orderNum + " from the Sun";
    }

    public String getName() {
        return planetName;
    }

    public String getTemperature() {
        return temperature;
    }

    public String toString() {
        return orderFromSun() + " and is " + getTemperature();
    }
}

class Mercury extends Planet {
    public Mercury() {
        super("Mercury", 1, "Hot");
    }
}

class Venus extends Planet {
    public Venus() {
        super("Venus", 2, "Hot");
    }
}

class Earth extends Planet {
    public Earth() {
        super("Earth", 3, "Warm");
    }

    public String home() {
        return "Home to every one of us";
    }
}

class Saturn extends Planet {
    public Saturn() {
        super("Saturn", 6, "Cold");
    }

    public String largeRings() {
        return "Saturn has massive rings";
    }
}

class SolarSystem {
    private ArrayList<Planet> solarsystem = new ArrayList<Planet>();

    public void addPlanet(Planet planet) {
        solarsystem.add(planet);
    }

    public int getPlanetCount() {
        return solarsystem.size();
    }

    public void printAllPlanets() {
        for (Planet planet: solarsystem) {
            System.out.println(planet);
        }
    }
}
```



Correct

Marks for this submission: 1.00/1.00.

Response history				
Step	Time	Action	State	Marks
<a href="#">1</a>	17/03/19, 17:47	Started	Not complete	
<a href="#">2</a>	18/03/19, 15:22	Submit: import java.util.ArrayList; public class SolarSystem { private ArrayList<Planet> solarList = new ArrayList<Planet>(); public void addPlanet(Planet planet) { solarList.add(planet); } public int getPlanetCount() { return solarList.size(); } public void printAllPlanets() { for (Planet planet:solarList) { System.out.println(planet); } } } class Planet { //Super class private String Name; private int Order; private String Temp; //public Planet() { } public Planet(String name, int order, String temperature){ Name = name; Order = order; Temp = temperature; } public String getName() { return Name; } public String getTemp() { return Temp; } public String orderFromSun() { return String.format("%s is number %d from the Sun", Name, Order); } public String toString() { return orderFromSun() + " and is " + getTemp(); } } class Earth extends Planet{ public Earth() { super("Earth", 3, "Warm"); } public String home() { return "Home to every one of us"; } } class Mercury extends Planet{ public Mercury() { super("Mercury", 1, "Hot"); } } class Venus extends Planet { public Venus() { super("Venus", 2, "Hot"); } } class Saturn extends Planet{ public Saturn() { super("Saturn", 6, "Cold"); } public String largeRings() { return "Saturn has massive rings"; } } }	Correct	1.00
3	18/03/19, 15:25	Attempt finished submitting:	Correct	1.00

## Abstract Classes

Sometimes a method needs to be provided that applies to all classes, but the parent cannot provide a generic implementation because it may not make sense. This does not change the fact that all classes need their own implementation, and it also suggests that each subclass will have their own, slightly different implementation of the method.

Lets take something like *void emergencyEscape()* in the context of a Rocket class, or subclass. All Rockets need an escape plan in case things don't go to plan, but the design of different types of Rockets make it hard to offer a generic implementation. Some Rockets may offer a purpose-built escape craft that blasts off, some may require astronauts to jump out of windows and parachute, or one might just simply turn the engines off.

Now, we could simply ask our users of the Rocket class to override the method with a new implementation, but this wouldn't always be followed, and it would not be good in an emergency for a subclass to call emergencyEscape() and find a light blinks on the dashboard and nothing else.

This is where *abstract* comes in.

An abstract class contains one or more abstract methods. An abstract method is a blank, empty method that is given a signature and is marked abstract. Subclasses must implement the abstract method, or the class cannot be instantiated. This applies to the parent abstract class too. If a class is marked abstract, then it cannot be instantiated, but only extended.

What does it look like?

```
public abstract class Rocket {
    // ...

    public abstract void emergencyEscape();

    // ...
}

public class FalconHeavy extends Rocket {
    // ...

    public void emergencyEscape() {
        // implementation
    }

    // ...
}
```

### Response history

Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	18/03/19, 00:54	Seen	
3	18/03/19, 15:25	Attempt finished	

Information

## Interfaces + Inheritance

Now, if you really need to, you can use both interfaces and inheritance.

We know a Rocket can fly, so it is a candidate for the Interface Flyable. We also know there are different types of Rockets, and they can all fly, so the interface could be implemented in the parent. It could also be implemented in the child, since not all Rockets may necessarily fly, such as failed prototype rockets.

To use both, simply place their keywords in the correct places.

```
// mixing things up with abstract classes
public abstract class Rocket implements Flyable {
    // ...
    public void emergencyEscape();
    // ...
    public void fly() {
        // implementation
    }
    // ...
}
// OR IF ROCKET DID NOT ALREADY IMPLEMENT Flyable
// mixing things up with inheritance
public class FalconHeavy extends Rocket implements Flyable {
    // ...
    public void fly() {
        // implementation
    }
}
```

Response history			
Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	18/03/19, 15:23	Seen	
3	18/03/19, 15:25	Attempt finished	

Information

## Interface or Inheritance?

After learning about Inheritance and Interfaces, what one is the best to choose?

If you can answer the "is a kind of" or "is a type of" question, you are after inheritance.

A SpaceShuttle is a type of Rocket. A Lion is a type of Cat. Pine is a kind of Tree.

There may be multiple "children" for each parent, and they are directly related. Laptops and Desktops are kinds of Computers.

This means that if things are not directly related, then you are after interfaces.

A Cat, Dog and Human all walk, but a Cat is not a Dog, and neither is a Human a Cat.

Response history			
Step	Time	Action	State
<a href="#">1</a>	17/03/19, 17:47	Started	
<a href="#">2</a>	18/03/19, 15:23	Seen	
3	18/03/19, 15:25	Attempt finished	

◀ Quiz 3 - Decisions, Loops, and Multiple Classes

Jump to...

Quiz 4 - Inheritance, Interfaces, and Collections ▶